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.

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` *Arbitrary Dimensional Clifford Algebras*

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.

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@gm
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. Can

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                    Conjugate of a Clifford object
Ops.clifford            Arithmetic Ops Group Methods for 'clifford'
                        objects
[.clifford              Extract or Replace Parts of a clifford
allcliff                Clifford object containing all possible terms
antivector              Antivectors or pseudovectors
as.vector               Coerce a clifford vector to a numeric vector
c_identity              Low-level helper functions for 'clifford'
                        objects
clifford                Create, coerce, and test for 'clifford' objects
clifford-package        Arbitrary Dimensional Clifford Algebras
drop                    Drop redundant information
dual                    The dual of a clifford object
even                    Even and odd clifford objects
getcoeffs               Get coefficients of a Clifford object
grade                   The grade of a clifford object
homog                   Homogenous Clifford objects
magnitude               Magnitude of a clifford object
neg                     Grade negation
numeric_to_clifford     Coercion from numeric to Clifford form
print.clifford          Print clifford objects
quaternion              Quaternions using Clifford algebras
rcliff                  Random clifford objects
rev                     Reverse of a Clifford object
signature               The signature of the Clifford algebra
summary.clifford        Summary methods for clifford objects
term                    Deal with terms
zap                     Zap small values in a clifford object
zero                    The zero Clifford object
```

Author(s)

NA

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

References

- J. Snygg (2012). *A new approach to differential geometry using Clifford's geometric Algebra*, Birkhauser. ISBN 978-0-8176-8282-8
- D. Hestenes (1987). *Clifford algebra to geometric calculus*, Kluwer. ISBN 90-277-1673-0
- C. Perwass (2009). *Geometric algebra with applications in engineering*, Springer. ISBN 978-3-540-89068-3
- D. Hildenbrand (2013). *Foundations of geometric algebra computing*. Springer, ISBN 978-3-642-31794-1

See Also

`clifford`

Examples

```
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 $e_1, e_2, \ldots, e_n$ has 2^n independent multivectors. Function `allcliff()` generates a clifford object with a nonzero coefficient for each multivector.

Usage

```
allcliff(n)
```

Arguments

`n`	Integer

Author(s)

Robin K. S. Hankin

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

Wikipedia contributors. (2018, July 20). "Antivector". In *Wikipedia, The Free Encyclopedia*. Retrieved 19:06, January 27, 2020, from https://en.wikipedia.org/w/index.php?title=Antivector&oldid=851094060

Examples

```
antivector(1:5)
```

---

as.vector *Coerce a clifford vector to a numeric vector*

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

References

Snygg 2012. “A new approach to differential geometry using Clifford’s geometric algebra”. Birkhauser; Springer Science+Business.

See Also

Ops.clifford

Examples

```
(x <- clifford(list(1,2,1:4),1:3))   # Formal creation method
(y <- as.1vector(4:2))
(z <- rcliff(include.fewer=TRUE))

terms(x+100)
coeffs(z)

## Clifford objects may be added and multiplied:

x + y
x*y



## They are associative and distributive:

(x*y)*z == x*(y*z)  # should be true
x*(y+z) == x*y + x*z  # should be true


## Other forms of manipulation are included:

coeffs(z) <- 1999
```

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

$$A^\dagger_{\langle k\rangle}=(a_1\wedge\ldots\wedge a_k)^\dagger=a^\dagger_k\wedge\ldots\wedge a^\dagger_1=\bigwedge_{i=1}^{k}a^\dagger_i$$

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

$$\left(A_{\langle k\rangle}\wedge B_{\langle l\rangle}\right)^\dagger=B^\dagger_{\langle l\rangle}\wedge A^\dagger_{\langle k\rangle}$$

and

$$\left(A_{\langle k\rangle}B_{\langle l\rangle}\right)^\dagger=B^\dagger_{\langle l\rangle}A^\dagger_{\langle k\rangle}$$

See examples for package idiom.

Author(s)

Robin K. S. Hankin

References

C. Perwass 2009. “Geometric algebra with applications in engineering”. Springer.

See Also

grade,rev

Examples

```
signature(2)

A <- rblade(g=3)
B <- rblade(g=4)

Conj(A %^% B) - Conj(B) %^% Conj(A)      # should be small
```

```
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)
```

---

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

```
drop(as.clifford(5))

const(rcliff())
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	Clifford object
n	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

C. Perwass. “Geometric algebra with applications in engineering”. Springer, 2009.

See Also

pseudoscalar

Examples

```
a <- rcliff()
dual(dual(dual(dual(a,8),8),8),8) == a # should be TRUE
```

`even` *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

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

Arguments

`C`	Clifford object

Author(s)

Robin K. S. Hankin

See Also

grade

Examples

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

`Extract.clifford` *Extract or Replace Parts of a clifford*

Description

Extract or replace subsets of cliffords.

Usage

```
## 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

```
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[2:3, 4] <- 99
A[] <- B
```

getcoeffs — *Get coefficients of a Clifford object*

Description

Access specific coefficents 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
```

```
const(X) <- 0.6
const(X) # should be 0.6, no numerical error
```

---

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 $e_{125} = e_1e_2e_5$ is 3. The grade operator $\langle\cdot\rangle_r$ is used to extract terms of a particular grade, with

$$A = \langle A\rangle_0 + \langle A\rangle_1 + \langle A\rangle_2 + \cdots = \sum_r \langle A\rangle_r$$

for any Clifford object A. Thus $\langle A\rangle_r$ is said to be homogenous of grade r. Hestenes sometimes writes subscripts that specify grades using an overbar as in $\langle A\rangle_{\overline{r}}$. It is conventional to denote the zero-grade object $\langle A\rangle_0$ as simply $\langle A\rangle$.

We have

$$\langle A+B\rangle_r = \langle A\rangle_r \qquad \langle \lambda A\rangle_r = \lambda\langle A\rangle_r \qquad \langle\langle A\rangle_r\rangle_s = \langle A\rangle_r\,\delta_{rs}.$$

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

C. Perwass 2009. “Geometric algebra with applications in engineering”. Springer.

Examples

```
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

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

Arguments

`C`	Object of class clifford

Details

Homogenous clifford objects have a multiplicative inverse.

Author(s)

Robin K. S. Hankin

Examples

```
is.homog(rcliff())
is.homog(rcliff(include.fewer=FALSE))
```

lowlevel *Low-level helper functions for* `clifford` *objects*

Description

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

Usage

```
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

$$||A|| = \sqrt{A * A}$$

Where $A * A$ denotes the Euclidean scalar product eucprod(). Recall that the Euclidean scalar product is never negative.

Usage

```
## 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

```
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 $A - 2\left\langle A\right\rangle_r$.

Usage

```
neg(C, n)
```

Arguments

C	Clifford object
n	Integer vector indicating grades to negate

Details

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

Author(s)

Robin K. S. Hankin

References

A. Acus and A. Dargys 2018. "The inverse of a multivector: beyond the threshold $p+q=5$". *Advances in applied Clifford Algebras*, 28:65

See Also

Conj

Examples

```
A <- rcliff()
neg(A,1:2) == 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

getcoeffs

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
```

Ops.clifford *Arithmetic Ops Group Methods for* clifford *objects*

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)
clifford_negative(C)
geoprod(C1,C2)
clifford_times_scalar(C,x)
clifford_plus_clifford(C1,C2)
clifford_eq_clifford(C1,C2)
clifford_inverse(C)
cliffdotprod(C1,C2)
fatdot(C1,C2)
lefttick(C1,C2)
righttick(C1,C2)
wedge(C1,C2)
scalprod(C1,C2=rev(C1),drop=TRUE)
eucprod(C1,C2=C1,drop=TRUE)
maxyterm(C1,C2=as.clifford(0))
C1 %.% C2
C1 %^% C2
C1 %X% C2
C1 %star% C2
C1 % % C2
```

```
C1 %euc% C2
C1 %o% C2
C1 %_|% C2
C1 %|_% C2
```

Arguments

`e1,e2,C,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*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)`	$A \star B = A * B^\dagger$

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 function is.blade() is not yet implemented. Blades (including null blades) have a pseudoinverse, but this is not implemented yet either.

The *scalar product* of two clifford objects is defined as the zero-grade component of their geometric product:

$$A * B = \langle AB \rangle_0 \qquad \text{NB: notation used by both Perwass and Hestenes}$$

In package idiom the scalar product is given by A %star% B or scalprod(A,B). Hestenes and Perwass both use an asterisk for scalar product as in "$A * B$", but in package idiom, the asterisk is reserved for geometric product.

Note: in the package, A*B is the geometric product.

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

$$A \star B = A * B^\dagger = \left\langle AB^\dagger \right\rangle_0 \qquad \text{Perwass}$$

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

Note that the scalar product $A * A$ can be positive or negative [that is, A %star% A may be any sign], but the Euclidean product is guaranteed to be non-negative [that is, A %euc% A is always positive or zero].

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

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

scalprod

Examples

```
u <- rcliff(5)
v <- rcliff(5)
w <- rcliff(5)

u*v

u^3
```

```
u+(v+w) == (u+v)+w              # should be TRUE
u*(v*w) == (u*v)*w              # should be TRUE
u %^% v == (u*v-v*u)/2        # should be TRUE

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

x <- as.1vector(5)
y <- as.1vector(5)
x*y == x%.%y + x%^%y        # should be TRUE
x %^% y == x %^% (y + 3*x)  # should be TRUE

## Inner product is "%.%" is not associative:
 rcliff(5,g=2) -> x
 rcliff(5,g=2) -> y
 rcliff(5,g=2) -> z
x %.% (y %.% z)
(x %.% y) %.% z

## Geometric product *is* associative:
x * (y * z)
(x * y) * z
```

---

print *Print clifford objects*

Description

Print methods for Clifford algebra

Usage

```
## S3 method for class 'clifford'
print(x,...)
## S3 method for class 'clifford'
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

`clifford`

Examples

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


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


options("separate" = FALSE)
```

`quaternion` *Quaternions using Clifford algebras*

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

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

# 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

```
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 comprisig 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 $A_{\langle r\rangle}, B_{\langle s\rangle}, C_{\langle t\rangle}$, then

$$\langle A_{\langle r\rangle} B_{\langle s\rangle} C_{\langle t\rangle}\rangle_0 = \langle C_{\langle t\rangle} A_{\langle r\rangle} B_{\langle s\rangle}\rangle_0$$

In the proof he notes in an intermediate step that

$$\langle A_{\langle r\rangle} B_{\langle s\rangle}\rangle_t * C_{\langle t\rangle} = C_{\langle t\rangle} * \langle A_{\langle r\rangle} B_{\langle s\rangle}\rangle_t = \langle C_{\langle t\rangle} A_{\langle r\rangle} B_{\langle s\rangle}\rangle_0.$$

Package idiom is shown in the examples.

Author(s)

Robin K. S. Hankin

Examples

```
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 <- rblade(g=4)  # r=4
B <- rblade(g=5)  # s=5
C <- rblade(g=6)  # t=6

grade(A*B*C,0)-grade(C*A*B,0)   # geometric product uses '*'

# Intermediate step

x1 <- grade(A*B,7) %star% C
x2 <- C %star% grade(A*B,7)
x3 <- grade(C*A*B,0)

max(x1,x2,x3) - min(x1,x2,x3)   # 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 $\tilde{A}$.

$$\left(A^\dagger\right)^\dagger = A \qquad (AB)^\dagger = B^\dagger A^\dagger \qquad (A+B)^\dagger = A^\dagger + B^\dagger \qquad \left\langle A^\dagger \right\rangle = \langle A \rangle$$

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

$$\left\langle A^\dagger \right\rangle_r = \langle A \rangle_r^\dagger = (-1)^{r(r-1)/2} \langle A \rangle_r$$

We can also show that

$$\langle AB \rangle_r = (-1)^{r(r-1)/2} \left\langle B^\dagger A^\dagger \right\rangle_r$$

Usage

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

Arguments

x	Clifford object

Author(s)

Robin K. S. Hankin

See Also

grade,Conj

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

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

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

```
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

```
## 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 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

C. Perwass. "Geometric algebra with applications in engineering". Springer, 2009.

See Also

`clifford`

Examples

```
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

```
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

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

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

Arguments

`C`	Clifford object

Details

To create the zero object *ab initio*, use

`clifford(list(),numeric(0))`

although note that `scalar(0)` will work too.

Author(s)

Robin K. S. Hankin

See Also

`scalar`

Examples

```
is.zero(rcliff())
```

Index