Package ‘clifford’

May 1, 2022

Type Package

Title Arbitrary Dimensional Clifford Algebras

Version 1.0-7

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)

Suggests knitr,rmarkdown,testthat,onion,lorentz

VignetteBuilder knitr

Imports Rcpp (>= 0.12.5),mathjaxr,disordR (>= 0.0-8), magrittr, methods, partitions (>= 1.10-4)

LinkingTo Rcpp,BH

SystemRequirements C++11

URL https://github.com/RobinHankin/clifford

BugReports https://github.com/RobinHankin/clifford/issues

RdMacros mathjaxr

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:	1.0-7
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@
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.
License:	GPL (>= 2)
Suggests:	knitr,rmarkdown,testthat,onion,lorentz
VignetteBuilder:	knitr
Imports:	Rcpp (>= 0.12.5),mathjaxr,disordR (>= 0.0-8), magrittr, methods, partitions (>= 1.10-4)
LinkingTo:	Rcpp,BH
SystemRequirements:	C++11
URL:	https://github.com/RobinHankin/clifford
BugReports:	https://github.com/RobinHankin/clifford/issues
RdMacros:	mathjaxr
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

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
cartan	Cartan map between clifford algebras
clifford	Create, coerce, and test for 'clifford' objects
clifford-package	Arbitrary Dimensional Clifford Algebras
const	The constant term of a Clifford object
drop	Drop redundant information
even	Even and odd clifford objects
grade	The grade of a clifford object
homog	Homogenous Clifford objects
horner	Horner's method
involution	Clifford involutions
lowlevel	Low-level helper functions for 'clifford' objects
magnitude	Magnitude of a clifford object
minus	Take the negative of a vector
numeric_to_clifford	Coercion from numeric to Clifford form
print.clifford	Print clifford objects
quaternion	Quaternions using Clifford algebras
rcliff	Random clifford objects
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,grade)
```

Arguments

`n`	Integer specifying dimension of underlying vector space
`grade`	Grade of multivector to be returned. If missing, multivector contains every term of every grade $\leq n$

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))
as.antivector(v)
is.antivector(C, include.pseudoscalar=FALSE)
```

Arguments

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

Details

An *antivector* is an n-dimensional Clifford object, all of whose terms are of grade $n-1$. An antivector has n degrees of freedom. Function `antivector(v,n)` interprets `v[i]` as the coefficient of $e_1e_2\ldots e_{i-1}e_{i+1}\ldots e_n$.

Function `as.antivector()` is a convenience wrapper, coercing its argument to an antivector of minimal dimension (zero entries are interpreted consistently).

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.

Compare `as.1vector()`, which returns a clifford object of grade 1.

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

See Also

as.1vector

Examples

```
antivector(1:5)

as.1vector(c(1,1,2)) %X% as.1vector(c(3,2,2))
c(1*2-2*2, 2*3-1*2, 1*2-1*3)  # note sign of e_13
```

---

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
```

`cartan` *Cartan map between clifford algebras*

Description

Cartan's map isomorphisms from $Cl(p,q)$ to $Cl(p-4,q+4)$ and $Cl(p+4,q-4)$

Usage

```
cartan(C, n = 1)
cartan_inverse(C, n = 1)
```

Arguments

C	Object of class `clifford`
n	Strictly positive integer

Value

Returns an object of class `clifford`. The default value n=1 maps $Cl(4,q)$ to $Cl(0,q+4)$ (`cartan()`) and $Cl(0,q)$ to $Cl(4,q-4)$.

Author(s)

Robin K. S. Hankin

References

E. Hitzer and S. Sangwine 2017. "Multivector and multivector matrix inverses in real Clifford algebras", *Applied Mathematics and Computation*. 311:3755-89

See Also

`clifford`

Examples

```
a <- rcliff(d=7)   # Cl(4,3)
b <- rcliff(d=7)   # Cl(4,3)
signature(4,3)     # e1^2 = e2^2 = e3^2 = e4^2 = +1; e5^2 = e6^2=e7^2 = -1
ab <- a*b          # multiplication in Cl(4,3)

signature(0,7)   # e1^2 = ... = e7^2 = -1
cartan(a)*cartan(b) == cartan(ab) # multiplication in Cl(0,7); should be TRUE

signature(Inf)  # restore default
```

clifford — *Create, coerce, and test for `clifford` objects*

Description

An object of class `clifford` 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)
## S3 method for class 'clifford'
dim(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). For a scalar, this is zero and for the zero clifford object it (currently) returns zero as well although a case could be made for `NULL`.
- 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
```

---

const *The constant term of a Clifford object*

Description

Get and set the constant term of a clifford object.

Usage

```
const(C,drop=TRUE)
is.real(C)
## S3 replacement method for class 'clifford'
const(x) <- value
```

Arguments

`C,x`	Clifford object
`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 R idiom in `const<-()` is slightly awkward:

```
> body(`const<-.clifford`)
{
  stopifnot(length(value) == 1)
  x <- x - const(x)
  return(x + value)
}
```

The reason that it is not simply `return(x-const(x)+value)` or `return(x+value-const(x))` is to ensure numerical accuracy; see examples.

Author(s)

Robin K. S. Hankin

See Also

grade, clifford, getcoeffs, is.zero

Examples

```
X <- clifford(list(1,1:2,1:3,3:5),6:9)
X
X <- X + 1e300
X

const(X) # should be 1e300

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

# compare naive approach:

X <- clifford(list(1,1:2,1:3,3:5),6:9)+1e300
X+0.6-const(X)  # constant gets lost in the numerics

X <- clifford(list(1,1:2,1:3,3:5),6:9)+1e-300
X-const(X)+0.6  # answer correct by virtue of left-associativity


x <- 2+rcliff(d=3,g=3)
jj <- x*cliffconj(x)
is.real(jj*rev(jj))   # should be TRUE
```

---

drop *Drop redundant information*

Description

Coerce constant Clifford objects to numeric

Usage

```
drop(x)
```

Arguments

x	Clifford object

Details

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

Note

Many functions in the package take `drop` as an argument which, if `TRUE`, means that the function returns a `drop`ped value.

Author(s)

Robin K. S. Hankin

See Also

`grade`,`getcoeffs`

Examples

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

const(rcliff())
const(rcliff(),drop=FALSE)
```

`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
list_modifier(B)
getcoeffs(C, B)
```

Arguments

`C,x`	A clifford object
`index`	elements to extract or replace
`value`	replacement value
`B`	A list of integer vectors, terms
`...`	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. If `index` is a list, its elements are interpreted as integer vectors indicating which terms to be extracted (even if it is a `disord` object). If `index` is a `disord` object, standard consistency rules are applied. The extraction methods are designed so that idiom such as `a[coeffs(a)>3]` works.

For replacement methods, the standard use-case is `a[i] <- b` in which argument `i` is a list of integer vectors and `b` a length-one numeric vector. Otherwise, to manipulate parts of a clifford object, use `coeffs(a) <- value`; this effectively leverages `disord` formalism. Idiom such as `a[coeffs(a)<2] <- 0` is not currently implemented (to do this, use `coeffs(a)[coeffs(a)<2] <- 0`). Replacement using a list-valued index, as in `A[i] <-value` uses an ugly hack if `value` is zero. Replacement methods are not yet finalised and not yet fully integrated with the `disordR` package.

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()` extract the terms and coefficients from a clifford object. These functions return `disord` objects 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`.

Extraction and replacement methods treat `0` specially, translating it (via `list_modifier()`) to `numeric(0)`.

Extracting or replacing a list with a repeated elements is usually a Bad Idea (tm). However, if option `warn_on_repeats` is set to `FALSE`, no warning will be given (and the coefficient will be the sum of the coefficients of the term; see the examples).

Function `getcoeffs()` is a lower-level helper function that lacks the succour offered by `[.clifford()`. It returns a numeric vector [not a `disord` object: the order of the elements is determined by the order of argument `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

# clifford(list(1,1:2,1:2),1:3)  # would give a warning

options("warn_on_repeats" = FALSE)
clifford(list(1,1:2,1:2),1:3)  # works; 1e1 + 5e_12

options("warn_on_repeats" = TRUE) # return to default behaviour.
```

`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)
grade(C,n) <- value
grades(x)
gradesplus(x)
gradesminus(x)
gradeszero(x)
```

Arguments

C,x	Clifford object
n	Integer vector specifying grades to extract
value	Replacement value, a numeric vector
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 $e_3 + e_5$ 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

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

Function `grades()` returns an (unordered) vector specifying the grades of the constituent terms. Function `grades<-()` allows idiom such as `grade(x,1:2) <- 7` to operate as expected [here to set all coefficients of terms with grades 1 or 2 to value 7].

Function `gradesplus()` returns the same but counting only basis vectors that square to $+1$, and `gradesminus()` counts only basis vectors that square to -1. Function `signature()` controls which basis vectors square to $+1$ and which to -1.

From Perwass, page 57, given a bilinear form

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

and a basis blade e_A with $A \subseteq \{1,\ldots,p+q\}$, then

$$\mathrm{gr}(e_A) = |\{a\in A\colon 1\le a\le p+q\}| \qquad \mathrm{gr}_+(e_A) = |\{a\in A\colon 1\le a\le p\}| \qquad \mathrm{gr}_-(e_A) = |\{a\in A\colon p< a\le p+q\}|$$

Function `gradeszero()` counts only the basis vectors squaring to zero (I have not seen this anywhere else, but it is a logical suggestion).

If the signature is zero, then the Clifford algebra reduces to a Grassman algebra and products match the wedge product of exterior calculus. In this case, functions `gradesplus()` and `gradesminus()` return `NA`.

Function `grade(C,n)` returns a clifford object with just the elements of grade g, where `g %in% n`.

The zero grade term, `grade(C,0)`, is given more naturally by `const(C)`.

Function `c_grade()` is a helper function that is documented at `Ops.clifford.Rd`.

Note

In the C code, "term" 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.

See Also

signature, const

Examples

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


signature(2,2)
x <- rcliff()
drop(gradesplus(x) + gradesminus(x) + gradeszero(x) - grades(x))

a <- rcliff()
a == Reduce(`+`,sapply(unique(grades(a)),function(g){grade(a,g)}))
```

---

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

Note

Nonzero homogenous clifford objects have a multiplicative inverse.

Author(s)

Robin K. S. Hankin

Examples

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

---

horner *Horner's method*

Description

Horner's method for Clifford objects

Usage

```
horner(P,v)
```

Arguments

`P`	Multivariate polynomial
`v`	Numeric vector of coefficients

Details

Given a polynomial

$$p(x) = a_0 + a_1 + a_2x^2 + \cdots + a_nx^n$$

it is possible to express $p(x)$ in the algebraically equivalent form

$$p(x) = a_0 + x\left(a_1 + x\left(a_2 + \cdots + x\left(a_{n-1} + xa_n\right)\cdots\right)\right)$$

which is much more efficient for evaluation, as it requires only n multiplications and n additions, and this is optimal. The output of `horner()` depends on the `signature()`.

Note

Horner's method is not as cool for Clifford objects as it is for (e.g.) multivariate polynomials or `freealg` objects. This is because powers of Clifford objects don't get more complicated as the power increases.

Author(s)

Robin K. S. Hankin

Examples

```
horner(1+e(1:3)+e(2:3) , 1:6)

rcliff() |> horner(1:4)
```

`involution` *Clifford involutions*

Description

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 Chisholm use a dagger, as in $A^\dagger$. This page uses 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_A with $A \subseteq \{1,\dots,p+q\}$, then we have $e_A^\dagger = (-1)^m e_A{}^\sim$, where $m = \mathrm{gr}_-(A)$. 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 *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 $\langle A\rangle_{\overline{\{1,2,5\}}} = A-2\left(\langle A\rangle_1 + \langle A\rangle_2 + \langle A\rangle_5\right)$ 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 $I = e_1e_2\dots e_n$ 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.

Usage

```
## S3 method for class 'clifford'
rev(x)
## S3 method for class 'clifford'
Conj(z)
cliffconj(z)
neg(C,n)
gradeinv(C)
```

Arguments

`C,x,z`	Clifford object
`n`	Integer vector specifying grades to be negated in `neg()`

Author(s)

Robin K. S. Hankin

See Also

`grade`

Examples

```
x <- rcliff()
x
rev(x)


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

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

`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)
c_cartan(L, c, m, n)
c_cartan_inverse(L, c, m, n)
```

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`	Two positive integers, p and q, representing the number of $+1$ and -1 terms on the 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 (the function body is `sqrt(abs(eucprod(z)))`; the `abs()` is needed to avoid numerical roundoff errors in `eucprod()` giving a negative value).

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
```

`minus`   *Take the negative of a vector*

Description

Very simple function that takes the negative of a vector, here so that idiom such as

`coeffs(z)[gradesminus(z)%%2 != 0] %<>% minus`

works as intended (this taken from `Conj.clifford()`).

Usage

```
minus(x)
```

Arguments

x	Any vector or disord object

Value

Returns a vector or disord

Author(s)

Robin K. S. Hankin

---

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 `is.scalar()` is a synonym for `is.real()` which is documented at `const.Rd`.

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 wedge product of basis vectors; function `e()` is a synonym. There is special dispensation for zero, so `e(0)` returns the Clifford scalar 1.

Function `antivector()` should arguably be described here but is actually documented at `antivector.Rd`.

Author(s)

Robin K. S. Hankin

See Also

`getcoeffs`,`antivector`,`const`

Examples

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

e(5:8)

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 %dot% 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` or coerced if needed
`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.

Binary operator `%dot%` is a synonym for `%.%`, which causes problems for rmarkdown.

Function `clifford_inverse()` returns an inverse for nonnull Clifford objects $Cl(p, q)$ for $p+q \leq 5$, and a few other special cases. The functionality 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.

Division, as in idiom `x/y`, is defined as `x*clifford_inverse(y)`. Function `clifford_inverse()` uses the method set out by Hitzer and Sangwine but is limited to $p + q \leq 5$.

Value

The high-level functions documented here return a `clifford` object. The low-level functions are not really intended for the end-user.

Note

In the **clifford** package the caret "`^`" is reserved for multiplicative powers, as in `A^3=A*A*A`. All the different Clifford products have binary operators for convenience including the wedge product `%^%`. Compare the **stokes** package, where multiplicative powers do not really make sense and `A^B` is interpreted as a wedge product of differential forms A and B. In **stokes**, the wedge product is the *sine qua non* for the whole package and needs a terse idiomatic representation (although there `A%^%B` returns the wedge product too).

Author(s)

Robin K. S. Hankin

References

E. Hitzer and S. Sangwine 2017. "Multivector and multivector matrix inverses in real Clifford algebras". *Applied Mathematics and Computation* 311:375-389

Examples

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

u
v
u*v

u+(v+w) == (u+v)+w           # should be TRUE by associativity of "+"
u*(v*w) == (u*v)*w           # should be TRUE by associativity of "*"
u*(v+w) == u*v + u*w         # should be TRUE by distributivity

# Now if x,y are _vectors_ we have:

x <- as.1vector(sample(5))
y <- as.1vector(sample(5))
x*y == x%.%y + x%^%y
x %^% y == x %^% (y + 3*x)
x %^% y == (x*y-x*y)/2       # should be TRUE

#  above are TRUE for x,y vectors (but not for multivectors, in general)


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

## Other products should work as expected:

x %|_% y   ## left contraction
x %_|% y   ## right contraction
x %o% y    ## fat dot product
```

---

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 is sensitive to the value of option separate. If FALSE (the default), the method prints in a compact form, as in e_134. 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 as it will print e_10,11,12 instead of e_101112. If separate is TRUE, the method prints the basis vectors separately, as in e1 e3 e4.

Function as.character.clifford() is also sensitive to these options. The print method has special dispensation for length-zero clifford objects. Function catterm() is a low-level helper function.

Author(s)

Robin K. S. Hankin

See Also

clifford

Examples

```
a <- rcliff(d=15,g=9)
a   # incomprehensible

options("separate" = TRUE)
a    # marginally better


options("separate" = FALSE)
options(basissep=",")
a    #  clearer; YMMV

options(basissep = NULL)  # restore default
```

quaternion *Quaternions using Clifford algebras*

Description

Converting quaternions to and from Clifford objects is not part of the package but functionality and a short discussion is included in `inst/quaternion_clifford.Rmd`.

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).

Note

A different mapping, from the quaternions to $Cl(0, 2)$ is given at `signature.Rd`.

Author(s)

Robin K. S. Hankin

See Also

`signature`

---

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=7, g=3)
```

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` (including a term of grade zero, that is, a scalar)

Details

Function `rcliff()` gives a quick nontrivial Clifford object, typically with terms having a range of grades (see 'grade.Rd'); argument `include.fewer=FALSE` ensures that all terms are of the same grade.

Function `rblade()` gives a Clifford object that is a *blade* (see 'term.Rd'). It returns the wedge product of a number of 1-vectors, for example $(e_1 + 2e_2) \wedge (e_1 + 3e_5)$.

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.

Note

If the grade exceeds the dimensionality, $g > d$, then the result is arguably zero; `rcliff()` returns an error.

Author(s)

Robin K. S. Hankin

See Also

term,grade

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)  # 1 (to numerical precision)
zap(ainv*a)  # 1 (to numerical precision)
```

```
# Perwass 2009, lemma 3.9:


A <- rblade(d=9,g=4)
B <- rblade(d=9,g=5)
C <- rblade(d=9,g=6)

grade(A*B*C,0)-grade(C*A*B,0)   # zero to numerical precision



# Intermediate step

x1 <- grade(A*B,3) %star% C
x2 <- C %star% grade(A*B,3)
x3 <- grade(C*A*B,0)

max(x1,x2,x3) - min(x1,x2,x3)   # zero to numerical precision
```

---

signature *The signature of the Clifford algebra*

Description

Getting and setting the signature of the Clifford algebra

Usage

```
signature(p,q=0)
is_ok_sig(s)
showsig(s)
## S3 method for class 'sigobj'
print(x,...)
```

Arguments

s,p,q	Integers, specifying number of positive elements on the diagonal of the quadratic form, with s=c(p,q)
x	Object of class sigobj
...	Further arguments, currently ignored

Details

The signature functionality 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,q) is the *signature* of the bilinear form $\langle\cdot,\cdot\rangle$. This gives rise to the Clifford algebra $C_{p,q}$.

If the signature is (p,q), then we have

$$e_ie_i = \begin{cases} +1 & \text{if } 1 \leq i \leq p \\ -1 & \text{if } p+1 \leq i \leq p+q \\ 0 & \text{if } i > p+q. \end{cases}$$

Note that $(p,0)$ corresponds to a positive-semidefinite quadratic form in which $e_ie_i = +1$ for all $i \leq p$ and $e_ie_i = 0$ for all $i > p$. Similarly, $(0,q)$ corresponds to a negative-semidefinite quadratic form in which $e_ie_i = -1$ for all $i \leq q$ and $e_ie_i = 0$ for all $i > q$.

Package idiom for a strictly positive-definite quadratic form would be to specify infinite p [in which case q is irrelevant] and for a strictly negative-definite quadratic form we would need $p = 0, q = \infty$.

If we specify $e_ie_i = 0$ for all i, then the operation reduces to the wedge product of a Grassman algebra. Package idiom for this is to set $p = q = 0$, but this is not recommended: use the **stokes** package for Grassman algebras, which is much more efficient and uses nicer idiom.

Function `signature(p,q)` returns the signature silently; but setting option `show_signature` to `TRUE` makes `signature()` have the side-effect of calling `showsig()`, which changes the default prompt to display the signature, much like `showSOL` in the `lorentz` package. There is special dispensation for "infinite" p or q.

Calling `signature()` [that is, with no arguments] returns an object of class `sigobj` with elements corresponding to p and q. The `sigobj` class ensures that a near-infinite integer such as `.Machine$integer.max` will be printed as "`Inf`" rather than, for example, "`2147483647`".

Function `is_ok_sig()` is a helper function that checks for a proper signature.

Author(s)

Robin K. S. Hankin

Examples

```
signature()

e(1)^2
e(2)^2

signature(1)
e(1)^2
e(2)^2   # note sign

signature(3,4)
sapply(1:10,function(i){drop(e(i)^2)})


signature(Inf)   # restore default




# Nice mapping from Cl(0,2) to the quaternions (loading clifford and
# onion simultaneously is discouraged):

# library("onion")
# signature(0,2)
# Q1 <- rquat(1)
```

```
# Q2 <- rquat(1)
# f <- function(H){Re(H)+i(H)*e(1)+j(H)*e(2)+k(H)*e(1:2)}
# f(Q1)*f(Q2) - f(Q1*Q2) # zero to numerical precision
# signature(Inf)
```

`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* is a wedge product of basis vectors (or a geometric product of linearly independent basis vectors), something like e_{12} or e_{12569}. Sometimes I use the word "term" to mean a wedge product of basis vectors together with its associated coefficient: so $7e_{12}$ would be described as a term.

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 `rblade()`, documented at '`rcliff.Rd`', returns a random blade.

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 a 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`,`rblade`

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