Package 'onion'

March 29, 2024

Version 1.5-3

Title Octonions and Quaternions

LazyData TRUE

Description Quaternions and Octonions are four- and eight- dimensional extensions of the complex numbers. They are normed division algebras over the real numbers and find applications in spatial rotations (quaternions), and string theory and relativity (octonions). The quaternions are noncommutative and the octonions nonassociative. See the package vignette for more details.

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

License GPL-2

Depends methods, R (>= 3.5.0)

Suggests testthat,knitr,rmarkdown,covr

VignetteBuilder knitr

Imports emulator, Matrix, freealg (>= 1.0-4), mathjaxr

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

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

RdMacros mathjaxr

R topics documented:

`onion-package` *Octonions and Quaternions*

Description

Quaternions and Octonions are four- and eight- dimensional extensions of the complex numbers. They are normed division algebras over the real numbers and find applications in spatial rotations (quaternions), and string theory and relativity (octonions). The quaternions are noncommutative and the octonions nonassociative. See the package vignette for more details.

Details

Package:	onion
Version:	1.5-3
Title:	Octonions and Quaternions
LazyData:	TRUE
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gmai
Description:	Quaternions and Octonions are four- and eight- dimensional extensions of the complex numbers. They
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
License:	GPL-2
Depends:	methods, R (>= 3.5.0)
Suggests:	testthat,knitr,rmarkdown,covr
VignetteBuilder:	knitr
Imports:	emulator, Matrix, freealg (>= 1.0-4), mathjaxr
URL:	https://github.com/RobinHankin/onion
BugReports:	https://github.com/RobinHankin/onion/issues
RdMacros:	mathjaxr

Author: Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

There are precisely four normed division algebras over the reals: the reals themselves, the complex numbers, the quaternions, and the octonions. The R system is well equipped to deal with the first two: the **onion** package provides some functionality for the third and fourth.

Author(s)

Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

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

References

R. K. S. Hankin 2006. "Normed division algebras in R: introducing the onion package". *R News*, Volume 6, number 2

Examples

```
rquat(10)   # random quaternions

Ok + (Oi + Ojl)/(Oj-Oil)  # basic octonions

x <- roct(10)
y <- roct(10)
z <- roct(10)

x*(y*z) - (x*y)*z   # nonassociative!
```

`adjoint` *The adjoint map*

Description

The adjoint ad_X of X is a map from a Lie group G to the endomorphism group of G defined by

$$\mathrm{ad}_X(Y) = [X, Y]$$

Usage

```
ad(x)
```

Arguments

`x`	Object nominally of class `onion` but other classes accepted where they make sense

Details

Here for completeness really.

Author(s)

Robin K. S. Hankin

Examples

```
x <- rquat()
y <- rquat()

f <- ad(x)
f(y)

f(f(y)) # [x,[x,y]]
```

Arith *Methods for Function Arith in package Onion*

Description

Methods for Arithmetic functions for onions: +, -, *, /, ^

Usage

```
onion_negative(z)
onion_inverse(z)
onion_arith_onion(e1,e2)
onion_arith_numeric(e1,e2)
numeric_arith_onion(e1,e2)
harmonize_oo(a,b)
harmonize_on(a,b)
onion_plus_onion(a,b)
onion_plus_numeric(a,b)
onion_prod_onion(e1,e2)
octonion_prod_octonion(o1,o2)
quaternion_prod_quaternion(q1,q2)
onion_prod_numeric(a,b)
onion_power_singleinteger(o,n)
onion_power_numeric(o,p)
```

Arguments

z,e1,e2,a,b,o,o1,o2,n,q1,q2,p
: onions or numeric vectors

Details

The package implements the Arith group of S4 generics so that idiom like A + B*C works as expected with onions.

Functions like onion_inverse() and onion_plus_onion() are low-level helper functions. The only really interesting operation is multiplication; functions octonion_prod_octonion() and quaternion_prod_quate dispatch to C.

Names are implemented and the rules are inherited (via harmonize_oo() and harmonize_on()) from rbind().

Value

generally return an onion

Note

Previous versions of the package included the option to use native R rather than the faster compiled C code used here. But this was very slow and is now discontinued.

Author(s)

Robin K. S. Hankin

Examples

```
a <- rquat()
b <- rquat()
a
Re(a)
j(a) <- 0.2
a*b
b*a  # quaternions are noncommutative


x <- as.octonion(matrix(rnorm(40),nrow=8))
y <- roct()
z <- roct()


x*(y*z) - (x*y)*z  # octonions are nonassociative [use associator()]
```

`biggest` *Returns the biggest type of a set of onions*

Description

Returns the biggest type of a set of onions; useful for "promoting" a set of onions to the most general type.

Usage

```
biggest(...)
```

Arguments

`...`	Onionic vectors

Details

If any argument passed to `biggest()` is an octonion, then return the string "octonion". Failing that, if any argument is a quaternion, return the string "quaternion", and failing that, return "scalar".

Value

Character string representing the type

Author(s)

Robin K. S. Hankin

Examples

```
biggest(O1,rquat(100),1:4)
```

`bind` *Binding of onionmats*

Description

Methods for `rbind()` and `cbind()` of onionmats. These are implemented by specifying methods for `rbind2()` and `cbind2()`.

Usage

```
bind_onion(x,bind,...)
bind_onion_onion(x,y,bind,...)
bind_onion_onionmat(x,y,bind,...)
bind_onionmat_onion(x,y,bind,...)
```

Arguments

`x,y`	Onions or onionmats
`bind`	Either `rbind` or `cbind` as appropriate
`...`	Further arguments

Value

Return onionmats

Author(s)

Robin K. S. Hankin

Examples

```
rbind(rquat(3),rquat(3))

cbind(diag(5),roct(1))


cbind(matrix(Oil,4,2),matrix(roct(12),4,3))
```

bunny *The Stanford Bunny*

Description

A set of 3D points in the shape of a rabbit (the Stanford Bunny)

Usage

```
data(bunny)
```

Format

A three column matrix with 35947 rows. Each row is the Cartesian coordinates of a point on the surface of the bunny.

Value

as for format

Source

https://graphics.stanford.edu/data/3Dscanrep/

Examples

```
data(bunny)
p3d(rotate(bunny,Hk))
```

c *Concatenation*

Description

Combines its arguments to form a single onion.

Usage

```
c_onionpair(x,y)
## S4 method for signature 'onion'
c(x,...)
```

Arguments

x,y,...	onions

Details

Returns an onion of the same type as its arguments. Names are inherited from the behaviour of `cbind()`, not `c()`.

Value

An onion

Note

The method is not perfect; it will not, for example, coerce its arguments to the `biggest()` type, so `c(rquat(),roct())` will fail. You will have to coerce the arguments by hand.

Dispatch is based on the class of the first argument, so `c(1,rquat())` will return a list (not an onion), and `c(rquat(),1)` will fail.

Author(s)

Robin K. S. Hankin

Examples

```
a <- roct(3)
b <- seq_onion(from=Oil,to=Oj,len=6)
c(a,b)

c(rquat(3),H1,H0,Him)
```

Compare-methods *Methods for compare S4 group*

Description

Methods for comparison (equal to, greater than, etc) of onions. Only equality makes sense.

Value

Return a boolean

Examples

```
# roct() > 0 # meaningless and returns an error



x <- as.octonion(matrix(sample(0:1,800,TRUE,p=c(9,1)),nrow=8))
y <- as.octonion(matrix(sample(0:1,800,TRUE,p=c(9,1)),nrow=8))
x==y

matrix(as.quaternion(100+1:12),3,4) == 102
```

Complex *Complex functionality for onions*

Description

Functionality in the Complex group.

The *norm* `Norm(O)` of onion O is the product of O with its conjugate: $|O| = OO^*$ but a more efficient numerical method is used (see `dotprod()`).

The *Mod* `Mod(O)` of onion O is the square root of its norm.

The *sign* of onion O is the onion with the same direction as O but with unit Norm: `sign(O)=O/Mod(O)`.

Function `Im()` sets the real component of its argument to zero and returns that; `Conj()` flips the sign of its argument's non-real components. Function `Re()` returns the real component (first row) of its argument as a numeric vector. If `x` is an onion, then `x == Re(x) + Im(x)`.

Usage

```
## S4 method for signature 'onion'
Re(z)
## S4 method for signature 'onion'
Im(z)
Re(z) <- value
Im(x) <- value
## S4 method for signature 'onion'
Conj(z)
## S4 method for signature 'onion'
Mod(z)
onion_abs(x)
onion_conjugate(z)
## S4 method for signature 'onion'
sign(x)
```

Arguments

`x,z`	Object of class onion or glub
`value`	replacement value

Value

All functions documented here return a numeric vector or matrix of the same dimensions as their argument, apart from functions `Im()` and `Conj()`, which return an object of the same class as its argument.

Note

If `x` is a numeric vector and `y` an onion, one might expect typing `x[1] <- y` to result in `x` being a onion. This is impossible, according to John Chambers.

Extract and set methods for components such as `i,j,k` are documented at `Extract.Rd`

Compare `clifford::Conj()`, which is more complicated.

Author(s)

Robin K. S. Hankin

See Also

Extract

Examples

```
a <- rquat()
Re(a)
Re(a) <- j(a)

Im(a)

b <- romat()

A <- romat()
Im(A) <- Im(A)*10
```

---

condense *Condense an onionic vector into a short form*

Description

Condense an onion into a string vector showing whether the elements are positive, zero or negative.

Usage

```
condense(x,as.vector=FALSE)
```

Arguments

x	An onionic vector
as.vector	Boolean, indicating whether to return a vector or matrix

Value

If as.vector is TRUE, return a string vector of the same length as x whose elements are length 4 or 8 strings for quaternions or octonions respectively. If FALSE, return a matrix with these columns.

The characters are "+" for a positive, "-" for a negative, and "0" for a zero, element.

Author(s)

Robin K. S. Hankin

Examples

```
condense(roct(3))
condense(roct(3),as.vector=TRUE)
```

cumsum *Cumulative sums and products of onions*

Description

Cumulative sums and products of onions

Usage

```
onion_cumsum(x)
onion_cumprod(x)
```

Arguments

x	onion

Value

An onion

Note

The octonions are nonassociative but `cumprod()` operates left-associatively, as in `((a[1]*a[2])*a[3])*a[4]` etc.

Author(s)

Robin K. S. Hankin

Examples

```
cumsum(as.quaternion(matrix(runif(20),4,5)))
cumsum(roct(5))

cumprod(rquat(7))
```

dot-class *Class "dot"*

Description

The dot object is defined so that idiom like `.[x,y]` returns the commutator, that is, `xy-yx` or the Lie bracket $[x, y]$. It would have been nice to use `[x,y]` (that is, without the dot) but although this is syntactically consistent, it cannot be done in R.

The "meat" of the package is:

```
setClass("dot", slots = c(ignore='numeric'))
`.` <- new("dot")
setMethod("[",signature(x="dot",i="ANY",j="ANY"),function(x,i,j,drop){i*j-j*i})
```

The package code includes other bits and pieces such as informative error messages for idiom such as `.[]`. The package defines a matrix method for the dot object. This is because "`*`" returns (incorrectly, in my view) the elementwise product, not the matrix product.

The Jacobi identity, satisfied by any associative algebra, is

$$[x,[y,z]] + [y,[z,x]] + [z,[x,y]] = 0$$

Function `ad()` returns the adjoint operator. The `adjoint` vignette provides details and examples of the adjoint operator.

The dot object is generated by running script `inst/dot.Rmd`, which includes some further discussion and technical documentation, and creates file `dot.rda` which resides in the `data/` directory.

Value

Always returns an object of the same class as `xy`

Slots

`ignore`**:** Object of class `"numeric"`, just a formal placeholder

Methods

[`signature(x = "dot", i = "ANY", j = "ANY")`: ...

[`signature(x = "dot", i = "ANY", j = "missing")`: ...

[`signature(x = "dot", i = "function", j = "function")`: ...

[`signature(x = "dot", i = "matrix", j = "matrix")`: ...

[`signature(x = "dot", i = "missing", j = "ANY")`: ...

[`signature(x = "dot", i = "missing", j = "missing")`: ...

Author(s)

Robin K. S. Hankin

See Also

`adjoint`

Examples

```
x <- rquat()
y <- rquat()
z <- rquat()
.[x,y]


.[x,.[y,z]] + .[y,.[z,x]] + .[z,.[x,y]]  # Jacobi, expanded
```

drop *Drop zero imaginary parts of an onionic vector*

Description

If an onion has zero imaginary part, drop it

Usage

```
## S4 method for signature 'onion'
drop(x)
```

Arguments

x	onion

Details

Generally, "drop" means coercion of an object to a less general type without loss of information. In many contexts, function `drop()` means to lose redundant information. This is not done by default (doing so would result in unexpected coercions).

Methods are given for `onion` and `onionmat` objects.

Author(s)

Robin K. S. Hankin

Examples

```
a <- rsoct()
a
a-Im(a)
drop(a-Im(a))
```

Extract *Extract or Replace Parts of onions or glubs*

Description

Methods for `"["` and `"[<-"`, i.e., extraction or subsetting of onions.

Usage

```
## S4 method for signature 'onion'
i(z)
## S4 method for signature 'onion'
j(z)
## S4 method for signature 'onion'
k(z)
## S4 method for signature 'octonion'
```

```
l(z)
## S4 method for signature 'octonion'
il(z)
## S4 method for signature 'octonion'
jl(z)
## S4 method for signature 'octonion'
kl(z)
## S4 method for signature 'onionmat'
i(z)
## S4 method for signature 'onionmat'
j(z)
## S4 method for signature 'onionmat'
k(z)
## S4 method for signature 'onionmat'
il(z)
## S4 method for signature 'onionmat'
jl(z)
## S4 method for signature 'onionmat'
kl(z)
i(x) <- value
j(x) <- value
k(x) <- value
l(x) <- value
il(x) <- value
jl(x) <- value
kl(x) <- value
```

Arguments

x,z	Object of class onion
value	replacement value

Value

Extraction and methods return an onion or onionmat. Replacement methods return an object of the same class as x.

Note

If x is a numeric vector and y a onion, one might expect typing x[1] <- y to result in x being a onion. This is impossible, according to John Chambers.

Author(s)

Robin K. S. Hankin

Examples

```
a <- roct(9)
il(a)
Re(a) <- 1:9

j(a) <- l(a)
a
```

length *Length of an octonionic vector*

Description

Get or set the length of onions

Usage

```
## S4 method for signature 'onion'
length(x)
```

Arguments

x	An onion

Details

Operates on the columns of the matrix as expected.

Value

integer

Author(s)

Robin K. S. Hankin

Examples

```
a <- roct(5)
length(a)
```

Logic *Logical operations on onions*

Description

Logical operations on onions are not supported

Usage

```
onion_logic(e1,e2)
```

Arguments

e1,e2	onions

Value

none

Note

Carrying out logical operations in this group will report an error. Negation, "!", is not part of this group.

Author(s)

Robin K. S. Hankin

Examples

```
# roct() & roct()   # reports an error
```

`Math` *Various logarithmic and circular functions for onions*

Description

Various elementary functions for onions

Usage

```
onion_log(x,base=exp(1))
onion_exp(x)
onion_sign(x)
onion_sqrt(x)
onion_cosh(x)
onion_sinh(x)
onion_acos(x)
onion_acosh(x)
onion_asin(x)
onion_asinh(x)
onion_atan(x)
onion_atanh(x)
onion_cos(x)
onion_sin(x)
onion_tan(x)
onion_tanh(x)
onion_cos(x)
onion_sin(x)
onion_tan(x)
onion_tanh(x)
```

Arguments

`x`	Object of class onion
`base`	In function `log()`, the base of the logarithm

Details

Standard math stuff. I am not convinced that the trig functions (`sin()` etc) have any value.

Author(s)

Robin K. S. Hankin

Examples

```
x <- roct()
exp(x+x) - exp(x)*exp(x) # zero to numerical precision

jj <- exp(log(x)/2)      # use sqrt() here
jj*jj-x                  #  also small


y <- roct()
exp(x+y) - exp(x)*exp(y) # some rules do not operate for onions



max(Mod(c(sin(asin(x))-x,asin(sin(x))-x)))   # zero to numerical precision
```

`names` *Names of an onionic vector*

Description

Functions to get or set the names of an onion

Usage

```
## S4 method for signature 'onion'
names(x)
## S4 method for signature 'onionmat'
rownames(x)
## S4 method for signature 'onionmat'
colnames(x)
## S4 method for signature 'onionmat'
dimnames(x)
## S4 method for signature 'onionmat'
dim(x)
```

Arguments

`x`	onion

Details

Names attributes refers to colnames of the internal matrix, which are retrieved or set using `colnames()` or `colnames<-()`.

Author(s)

Robin K. S. Hankin

Examples

```
a <- roct(5)
names(a) <- letters[1:5]

b <- romat()
dimnames(b) <- list(month = month.abb[1:5], location=names(islands)[1:6])
```

O1 *Unit onions*

Description

Each of the eight unit quaternions and octonions

Usage

```
H1
Hi
Hj
Hk
H0
Him
Hall
O1
Oi
Oj
Ok
Ol
Oil
Ojl
O0
Oim
Oall
```

Format

Each one is an onionic vector of length one.

Details

Try `Hi` (`=quaternion(i=1)`) to get the pattern for the first four. The next ones are the zero quaternion, the pure imaginary quaternion with all components 1, and the quaternion with all components 1. The ones beginning with "O" follow a similar pattern.

These are just variables that may be overwritten and thus resemble `T` and `F` whose value may be changed.

Value

A length-one onion, either a quaternion or an octonion

Examples

```
Oall
seq_onion(from=O1,to=Oil,len=6)

stopifnot(Hj*Hk ==  Hi)
stopifnot(Okl*Oil == -Oj )  # See tests/test_aaa.R for the full set
```

`onion` *Basic onion functions*

Description

Construct, coerce to, test for, and print onions

Usage

```
octonion(length.out = NULL, Re = 0, i = 0, j = 0,
    k = 0, l = 0, il = 0, jl = 0, kl = 0)
as.octonion(x, single = FALSE)
is.octonion(x)
quaternion(length.out = NULL,Re = 0, i = 0, j = 0, k = 0)
as.quaternion(x, single = FALSE)
is.quaternion(x)
is.onion(x)
as.onion(x,type,single=FALSE)
quaternion_to_octonion(from)
octonion_to_quaternion(from)
## S4 method for signature 'onion'
as.matrix(x)
## S4 method for signature 'onion'
as.numeric(x)
```

Arguments

`length.out`	In functions `quaternion()` and `octonion()`, the length of the onionic vector returned
`Re`	The real part of the onionic vector returned
`i,j,k`	In functions `quaternion()` and `octonion()`, component i, j, k respectively of the returned onion
`l,il,jl,kl`	In function `octonion()`, component l, il, jl, kl respectively of the returned octonion
`x,from`	Onion to be tested or printed
`single`	In functions `as.octonion()` and `as.quaternion()`, Boolean with default `FALSE` meaning to interpret `x` as a vector of reals to be coerced into an onion with zero imaginary part; and `TRUE` meaning to interpret `x` as a length 4 (or length 8) vector and return the corresponding single onion.

type In function as.onion() a string either "quaternion" or "octonion" denoting the algebra to be forced into

Details

Functions quaternion() and octonion() use standard recycling where possible; rbind() is used.

Functions as.quaternion() and as.octonion() coerce to quaternions and octonions respectively. If given a complex vector, the real and imaginary components are interpreted as Re and i respectively.

The output of type() is accepted as the type argument of function as.onion(); thus as.onion(out,type=type(x)) works as expected.

Value

Generally return onions

Note

An *onion* is any algebra (over the reals) created by an iterated Cayley-Dickson process. Examples include quaternions, octonions, and sedenions. There does not appear to be a standard generic term for such objects (I have seen n-ion, anion and others. But "onion" is pronouncable and a bona fide English word).

Creating further onions—such as the sedenions—is intended to be straightforward.

There is a nice example of the onion package in use in the permutations package, under cayley.Rd. This also shows the quaternion group Q8, but from a different perspective.

Author(s)

Robin K. S. Hankin

Examples

```
x <- octonion(Re=1,il=1:3)
x
kl(x) <- 100
x

as.quaternion(diag(4))


# Cayley table for the quaternion group Q8:
a <- c(H1,-H1,Hi,-Hi,Hj,-Hj,Hk,-Hk)
names(a) <- c("+1","-1","+i","-i","+j","-j","+k","-k")

f <- Vectorize(function(x,y){names(a)[a==a[x]*a[y]]})
X <- noquote(outer(1:8,1:8, f))
rownames(X) <- names(a)
colnames(X) <- names(a)
X
```

onion-class *Class "onion"*

Description

The formal S4 class for onion and onionmat objects

Objects from the Class

Class *onion* is a virtual S4 class extending classes *quaternion* and *octonion*. In package documentation, "*onion*" means an R object that behaves as a vector of quaternions or octonions, stored as a four- or eight- row numeric matrix.

Class *onionmat* is the S4 class for matrices whose elements are quaternions or octonions. An onionmat is stored as a two-element list, the first being an onion and the second an integer matrix which holds structural matrix attributes such as dimensions and dimnames. Most standard arithmetic R idiom for matrices should work for onionmats.

Class *index* is taken from the excellent **Matrix** package and is a `setClassUnion()` of classes `numeric`, `logical`, and `character`, which mean that it is an arity-one matrix index.

Author(s)

Robin K. S. Hankin

Examples

```
as.octonion(1:8,single=TRUE)
as.quaternion(matrix(runif(20),nrow=4))

H <- matrix(rquat(21),3,7)
dimnames(H) <- list(foo=letters[1:3],bar=state.abb[1:7])

i(H) <- 0.1

I <- matrix(rquat(14),7,2)
dimnames(I) <- list(foo=state.abb[1:7],baz=LETTERS[1:2])
H %*% I
```

onionmat *Onionic matrices*

Description

Simple functionality for quaternionic and octonionic matrices, intended for use in the `jordan` package. Use idiom like `matrix(Him,4,5)` or `matrix(roct(6),2,3)` to create an `onionmat` object, a matrix of onions.

The package is intended to match base R's matrix functionality in the sense that standard R idiom just goes through for onionic matrices. Determinants are not well-defined for quaternionic or octonionic matrices, and matrix inverses are not implemented.

Usage

```
newonionmat(d, M)
onionmat(data = NA, nrow = 1, ncol = 1, byrow = FALSE, dimnames = NULL)
as.onionmat(x)
is.onionmat(x)
onionmat_negative(e1)
onionmat_inverse(e1)
onionmat_prod_onionmat(e1,e2)
onionmat_power_onionmat(...)
onionmat_prod_single(x,y)
onionmat_power_single(e1,e2)
onionmat_plus_onionmat(e1,e2)
matrix_arith_onion(e1,e2)
onion_arith_matrix(e1,e2)
matrix_plus_onion(e1,e2)
matrix_prod_onion(e1,e2)
## S4 method for signature 'onionmat,onionmat'
cprod(x,y)
## S4 method for signature 'onionmat,missing'
cprod(x,y)
## S4 method for signature 'onionmat,ANY'
cprod(x,y)
## S4 method for signature 'ANY,ANY'
cprod(x,y)
## S4 method for signature 'onion,missing'
cprod(x,y)
## S4 method for signature 'onion,onion'
cprod(x,y)
## S4 method for signature 'onion,onionmat'
cprod(x,y)
## S4 method for signature 'onionmat,onion'
cprod(x,y)
## S4 method for signature 'onionmat,onionmat'
tcprod(x,y)
## S4 method for signature 'onionmat,missing'
tcprod(x,y)
## S4 method for signature 'onionmat,ANY'
tcprod(x,y)
## S4 method for signature 'ANY,ANY'
tcprod(x,y)
## S4 method for signature 'onion,missing'
cprod(x,y)
## S4 method for signature 'onion,onion'
cprod(x,y)
## S4 method for signature 'onion,onionmat'
cprod(x,y)
## S4 method for signature 'onionmat,onion'
cprod(x,y)
## S4 method for signature 'onionmat'
t(x)
## S4 method for signature 'onion'
t(x)
```

```
## S4 method for signature 'onionmat'
ht(x)
## S4 method for signature 'onion'
ht(x)
nrow(x)
ncol(x)
herm_onion_mat(real_diagonal, onions)
onionmat_complex(z)
onionmat_conjugate(z)
onionmat_imag(z)
onionmat_re(z)
onionmat_mod(z)
onionmat_matrixprod_onionmat(x,y)
onion_matrixprod_onionmat(x,y)
onionmat_matrixprod_numeric(x,y)
onionmat_matrixprod_onion(x,y)
```

Arguments

d,M	data and matrix index
x,y,z,e1,e2	Objects of class onionmat
data,nrow,ncol,byrow,dimnames	In function onionmat(), as for matrix()
...	Further arguments (currently ignored)
real_diagonal, onions	In function herm_onion_mat(), on- and off- diagonal elements of an Hermitian matrix

Details

An object of class onionmat is a two-element list, the first of which is an onion, and the second an index matrix of integers used for tracking attributes such as dimensions and dimnames. This device makes the extraction and replacement methods easy. Use getM() to access the index matrix and getd() to access the onionic vector.

The S4 method for matrix() simply dispatches to onionmat(), which is a drop-in replacement for matrix().

Function drop() has a method for onionmat objects.

Function newonionmat() is lower-level: it also creates onionmat objects, but takes two arguments: an onion and a matrix; the matrix argument can be used to specify additional attributes via attr(), but this ability is not currently used in the package.

Functions such as onionmat_plus_onionmat() are low-level helper functions, not really designed for the end-user.

Vignette onionmat shows some use-cases.

The print method for onionmat objects is sensitive to option show_onionmats_in_place. If TRUE, it prints the matrix elements in-place, using onion_to_string(). It works best when option show_onions_compactly is effective.

Author(s)

Robin K. S. Hankin

Examples

```
matrix(rquat(28),4,7)

M <- onionmat(rquat(10),2,5)
cprod(M)

Re(M)
Re(M) <- 0.3

romat() %*% rquat(6)

a <- rsomat()
a            # default
options("show_onionmats_in_place" = TRUE)
a
options("show_onionmats_in_place" = FALSE) # restore default
```

`orthogonal` *Orthogonal matrix equivalents*

Description

Convert a quaternion to and from an equivalent orthogonal matrix

Usage

```
matrix2quaternion(M)
as.orthogonal(Q)
```

Arguments

`M`	A three-by-three orthogonal matrix
`Q`	A vector of quaternions

Value

Function `matrix2quaternion()` returns a quaternion.

Function `as.orthogonal()` returns either a 3×3 matrix or a $3 \times 3 \times n$ array of orthogonal matrices

Note

Function `matrix2quaternion()` is low-level; use `as.quaternion()` to convert arrays.

Author(s)

Robin K. S. Hankin

See Also

`rotate`

Examples

```
as.orthogonal(rquat(1))

o <- function(w){diag(3)-2*outer(w,w)/sum(w^2)}  # Householder
matrix2quaternion(o(1:3))   # Booorrrriiinnnggg
matrix2quaternion(o(1:3) %*% o(3:1))

Q <- rquat(7)
Q <- Q/abs(Q)
as.quaternion(as.orthogonal(Q))   # +/- Q


A <- replicate(7,o(rnorm(3)) %*% o(rnorm(3)))
max(abs(as.orthogonal(as.quaternion(A))-A))
```

---

p3d *Three dimensional plotting*

Description

Three dimensional plotting of points. Produces a nice-looking 3D scatterplot with greying out of further points giving a visual depth cue

Usage

```
p3d(x, y, z, xlim = NULL, ylim = NULL, zlim = NULL, d0 = 0.2, h = 1, ...)
```

Arguments

x,y,z	vector of x, y, z coordinates to be plotted. If x is a matrix, interpret the rows as 3D Cartesian coordinates
xlim,ylim,zlim	Limits of plot in the x, y, z directions, with default NULL meaning to use range()
d0	E-folding distance for graying out (depths are standardized to be between 0 and 1)
h	The hue for the points, with default value of 1 corresponding to red. If NULL, produce black points greying to white
...	Further arguments passed to persp() and points()

Value

Value returned is that given by function trans3d().

Author(s)

Robin K. S. Hankin

See Also

bunny

Examples

```
data(bunny)
p3d(bunny,theta=3,phi=104,box=FALSE)
```

`plot` *Plot onions*

Description

Plotting method for onionic vectors

Usage

```
## S4 method for signature 'onion'
plot(x,y, ...)
```

Arguments

`x,y`	Onions
`...`	Further arguments passed to `plot.default()`

Details

The function is `plot(Re(x), Mod(Im(x)), ...)`, and thus behaves similarly to `plot()` when called with a complex vector.

Value

Called for its side-effect of plotting a diagram

Author(s)

Robin K. S. Hankin

Examples

```
plot(roct(30))
```

`prods` *Various products of two onions*

Description

Returns various inner and outer products of two onionic vectors.

Usage

```
x %<*>% y
x %>*<% y
x %<.>% y
x %>.<% y
x %.% y
onion_g_even(x,y)
onion_g_odd (x,y)
onion_e_even(x,y)
onion_e_odd (x,y)
dotprod(x,y)
```

Arguments

`x,y`	onions

Details

This page documents an attempt at a consistent notation for onionic products. The default product for onions (viz "`*`") is sometimes known as the "Grassman product". There is another product known as the Euclidean product defined by $E(p,q) = p'q$ where x' is the conjugate of x.

Each of these products separates into an "even" and an "odd" part, here denoted by functions `g_even()` and `g_odd()` for the Grassman product, and `e_even()` and `e_odd()` for the Euclidean product. These are defined as follows:

- `g_even(x,y)=(xy+yx)/2`
- `g_odd(x,y)=(xy-yx)/2`
- `e_even(x,y)=(x'y+y'x)/2`
- `e_odd(x,y)=(x'y-y'x)/2`

These functions have an equivalent binary operator.

The Grassman operators have a "`*`"; they are "`%<*>%`" for the even Grassman product and "`%>*<%`" for the odd product.

The Euclidean operators have a "`.`"; they are "`%<.>%`" for the even Euclidean product and "`%>.<%`" for the odd product.

Function `dotprod()` returns the Euclidean even product of two onionic vectors. That is, if `x` and `y` are eight-element vectors of the components of two onions, return `sum(x*y)`.

Note that the returned value is a numeric vector (compare `%<.>%`, `e.even()`, which return onionic vectors with zero imaginary part).

There is no binary operator for the ordinary Euclidean product (it seems to be rarely needed in practice). For `Conj(x)*x`, `Norm(x)` is much more efficient and accurate.

Function `prod()` is documented at `Summary.Rd`.

Note

Frankly if you find yourself using these operators you might be better off using the **clifford** package, which has an extensive and consistent suite of product operators.

Author(s)

Robin K. S. Hankin

Examples

```
Oj %<.>% Oall
```

---

rep *Replicate elements of onionic vectors*

Description

Replicate elements of onionic vectors

Usage

```
## S4 method for signature 'onion'
rep(x, ...)
```

Arguments

x	Onionic vector
...	Further arguments passed to `seq.default()`

Author(s)

Robin K. S. Hankin

Examples

```
a <- roct(3)
rep(a,2) + a[1]
rep(a,each=2)
rep(a,length.out=5)
```

`roct` *Random onionic vectors*

Description

Random quaternion or octonion vectors and matrices

Usage

```
rquat(n=5)
roct(n=5)
rsquat(n=11,s=12)
rsoct(n=11,s=12)
romat(type="quaternion", nrow=5, ncol=6, ...)
rsomat(type="quaternion", nrow=5, ncol=6, ...)
```

Arguments

`n`	Length of random vector returned
`nrow,ncol,...`	Further arguments specifying properties of the returned matrix
`s`	In the sparse functions `rsquat()` and `rsoct()`, an integer specifying the level of sparsity, with higher values meaning to return sparser onions
`type`	string specifying type of elements

Details

Function `rquat()` returns a quaternionic vector, `roct()` returns an octonionic vector, and `romat()` a quaternionic matrix.

Functions `rquat()` and `roct()` give a quick "get you going" random onion to play with. Function `romat()` gives a simple onionmat, although arguably `matrix(roct(4),2,2)` is as convenient.

The "sparse" functions `rsquat()` and `rsoct()` and `rsomat()` return onions that have many zero entries; non-zero entries are small integers. They showcase the print method for the case when `show_onions_compactly` is set.

Author(s)

Robin K. S. Hankin

References

K. Shoemake 1992. "Uniform random rotations". In D. Kirk, editor, *Graphics Gems III* pages 129-130. Academic, New York.

Examples

```
rquat(3)
roct(3)
plot(roct(30))

romat()
```

```
rsquat()
rsoct()
```

`rotate` *Rotates 3D vectors using quaternions*

Description

Rotates a three-column matrix whose rows are vectors in 3D space, using quaternions

Usage

```
rotate(x, H)
```

Arguments

`x`	A matrix of three columns whose rows are points in 3D space
`H`	A quaternion. Does not need to have unit modulus

Value

Returns a matrix of the same size as `x`

Author(s)

Robin K. S. Hankin

See Also

`orthogonal`

Examples

```
data(bunny)
par(mfrow=c(2,2))
par(mai=rep(0,4))
p3d(rotate(bunny,Hi),box=FALSE)
p3d(rotate(bunny,H1-Hi+Hj),box=FALSE)
p3d(rotate(bunny,Hk),box=FALSE)
p3d(rotate(bunny,Hall),box=FALSE)

o <- function(w){diag(3)-2*outer(w,w)/sum(w^2)}  # Householder
O <- o(1:3) %*% o(3:1)

rotate(bunny,as.quaternion(O))
bunny %*% t(O)    # should be the same; note transpose
```

round *Rounding of onions*

Description

Round elements of an onion

Usage

```
   ## S4 method for signature 'onion'
round(x,digits=0)
   ## S4 method for signature 'onionmat'
round(x,digits=0)
```

Arguments

x	Object of class onion
digits	number of digits to round to

Details

For onions, coerce to a matrix, round, then coerce back to an onion. For onionmats, coerce to an onion, round, then coerce back to an onionmat.

Value

Return an onion

Author(s)

Robin K. S. Hankin

Examples

```
round(rquat()*100)
round(rquat()*100,3)
```

seq *seq method for onions*

Description

Rough equivalent of seq() for onions.

Usage

```
seq_onion(from=1,to=1,by=((to-from)/(length.out-1)),length.out=NULL,slerp=FALSE, ...)
```

Arguments

from	Onion for start of sequence
to	Onion for end of sequence
by	Onion for interval
length.out	Length of vector returned
slerp	Boolean, with default FALSE meaning to use linear interpolation and TRUE meaning to use spherical linear interpolation (useful for animating 3D rotation)
...	Further arguments (currently ignored)

Author(s)

Robin K. S. Hankin

Examples

```
seq(from=O1,to=Oil,length.out=6)
seq(from=H1,to=(Hi+Hj)/2,len=10,slerp=TRUE)
```

---

show *Print method for onions*

Description

Show methods for onions

Usage

```
## S4 method for signature 'onion'
show(object)
onion_show(x,
     comp = getOption("show_onions_compactly"),
     h    = getOption("show_onions_horizontally")
)
comp_names(x)
```

Arguments

x,object	Onions
comp	Boolean, with TRUE meaning to print onions compactly and any other value to print in matrix form
h	Boolean, with TRUE meaning to print by columns and any other value meaning to print horizontally

Details

Default behaviour is to print by rows. To print by columns, set option `show_onions_horizontally` to `TRUE`:

```
options("show_onions_horizontally" = TRUE)
```

Any non-`TRUE` value (including `NULL` and its being unset) will restore the default.

Similarly, to show onions compactly, set option `show_onions_compactly` to `TRUE`:

```
options("show_onions_compactly" = TRUE)
```

This option works best for simple onions with integer entries (or at least values with few decimal places), and especially if there are many zero entries.

Function `onion_show()` is a helper function, not really intended for the end-user.

The "names" of the components of an onion (viz `Re`, `i`, `j`, `k` for quaternions and `Re`, `i`, `j`, `k`, `l`,`il`,`jl`,`kl` for octonions) are given by function `comp_names()` which takes either a character string or an onion.

Note

The print method for `onionmat` objects is also sensitive to these options.

Author(s)

Robin K. S. Hankin

Examples

```
x <- roct(15)
x  #default

options("show_onions_horizontally" = TRUE)
roct(4)

options("show_onions_horizontally" = FALSE)  # restore default

options("show_onions_compactly" = TRUE)
x <- as.quaternion(matrix(sample(c(0,0,0,-1,1),80,replace=TRUE),4,20))
options("show_onions_compactly" = FALSE) # restore default
```

`sum` *Various summary statistics for onions*

Description

Various summary statistics for onions

Usage

```
onion_allsum(x)
## S4 method for signature 'onion'
sum(x)
## S4 method for signature 'quaternion'
prod(x)
## S4 method for signature 'octonion'
sum(x)
## S4 method for signature 'onionmat'
sum(x)
## S4 method for signature 'octonion'
prod(x)
## S4 method for signature 'onion'
str(object, ...)
str_onion(object, vec.len = 4, ...)
onion_allsum(x)
onionmat_allsum(x)
quaternion_allprod(x)
```

Arguments

`x,object,...`	Objects of class `onion`
`vec.len`	number of elements to display

Details

For a `onion` object, return the sum or product accordingly

Value

Return an onion

Note

Function `str()` uses functionality from `condense()`.

Author(s)

Robin K. S. Hankin

Examples

```
sum(roct())
str(roct())
```

`threeform` *Various non-field diagnostics*

Description

Diagnostics of non-field behaviour: threeform, associator, commutator

Usage

```
threeform(x1, x2, x3)
associator(x1, x2, x3)
commutator(x1, x2)
```

Arguments

`x1,x2,x3`	onionic vectors

Details

The threeform is defined as `Re(x1 * (Conj(x2) * x3) - x3 * (Conj(x2) * x1))/2`;

the associator is `(x1 * x2) * x3 - x1 * (x2 * x3)`;

the commutator is `x1 * x2 - x2 * x1`.

Value

Returns an octonionic vector

Author(s)

Robin K. S. Hankin

See Also

`dot`

Examples

```
x <- roct(7) ; y <- roct(7) ; z <- roct(7)
associator(x,y,z)
```

`zapsmall` *Concatenation*

Description

Zapping small components to zero

Usage

```
## S4 method for signature 'onion'
zapsmall(x,digits=getOption("digits"))
## S4 method for signature 'onionmat'
zapsmall(x,digits=getOption("digits"))
```

Arguments

`x`	An onion or onionmat
`digits`	integer indicating the precision to be used as in `base::zapsmall()`

Details

Uses `base::zapsmall()` to zap small elements to zero.

Value

An onion

Author(s)

Robin K. S. Hankin

Examples

```
zapsmall(as.octonion(0.01^(1:8),single=TRUE))


a <- roct(7)
x <- a^1/a
x
zapsmall(x)
```

Index