Package ‘spray’

July 6, 2022

Type Package

Title Sparse Arrays and Multivariate Polynomials

Version 1.0-20

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

Description Sparse arrays interpreted as multivariate polynomials.

License GPL (>= 2)

Depends methods

Suggests polynom, testthat

Imports Rcpp (>= 0.12.3),partitions,magic,mathjaxr,disordR (>= 0.0-8)

LinkingTo Rcpp

SystemRequirements C++11

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

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

RdMacros mathjaxr

R topics documented:

`spray-package` *Sparse arrays and multivariate polynomials*

Description

Functionality for sparse arrays, with emphasis on their interpretation as multivariate polynomials.

Details

Base R has the capability of dealing with arbitrary dimensioned numerical arrays, with the `array` class.

A sparse array is a type of array in which nonzero elements are stored along with an index vector describing their coordinates—instead of arrays. This allows for efficient storage and manipulation as base arrays often require the storing of many zero elements which consume computational and memory resources.

In the package, sparse arrays are represented as objects of class `spray`. They use the `C++` standard template library (STL) `map` class, with keys being (unsigned) integer vectors, and values floats.

One natural application of sparse arrays, for which the package was written, is multivariate polynomials and the package vignette presents an extended discussion. Note that other interpretations exist: the **stokes** and **weyl** packages interpret `spray` objects as differential forms and elements of a Weyl algebra respectively.

Author(s)

Robin K. S. Hankin

Examples

```
# define a spray using a matrix of indices and a vector of values:
M <- matrix(sample(0:3,21,replace=TRUE),ncol=3)
a <- spray(M,sample(7))

# there are many pre-defined simple sprays:
b <- homog(3,4)

# arithmetic operators work:
a + 2*b
a - a*b^2/4
a+b

#  we can sum over particular dimensions:
asum(a+b,1)
```

```
# differentiation is supported:
deriv(a^6,2)

# extraction and replacement work as expected:

b[1,2,1]
b[1,2,1,drop=TRUE]

b[diag(3)] <- 3
```

---

arity *The arity of a spray object*

Description

The arity of a spray object: the number of indices needed to retrieve an entry, or the number of columns in the index matrix.

Usage

```
arity(S)
```

Arguments

S	a spray object

Value

Returns an integer

Author(s)

Robin K. S. Hankin

Examples

```
(a <- rspray())
arity(a)
```

`as.array` *Coerce spray objects to arrays*

Description

Coerces spray objects to arrays. Includes off-by-one functionality via option `offbyone`.

Usage

```
## S3 method for class 'spray'
as.array(x, offbyone=FALSE, compact=FALSE, ...)
## S3 method for class 'spray'
dim(x)
```

Arguments

`x`	spray object
`offbyone`	Boolean with default `FALSE` meaning to interpret the index entries as positions in their dimension, and `TRUE` meaning to add one to index values so that zero entries appear in the first place
`compact`	Boolean with default `FALSE` meaning to translate the spray as is, and `TRUE` meaning to add constants to each column of the index matrix so that the resulting array is as small as possible
`...`	Further arguments, currently ignored

Details

Argument `offbyone` defaults to `FALSE`; but if it is set to `TRUE`, it effectively adds one from the index matrix, so a zero entry in the index matrix means the first position in that dimension.

After the subtraction, if performed, the function will not operate if any index is less than 1.

Value

Returns an array of dimension `dim(S)`. The "meat" of the function is

```
out <- array(0, dS)
out[ind] <- coeffs(S)
```

Author(s)

Robin K. S. Hankin

Examples

```
(M <- matrix(sample(0:4,28,replace=TRUE),ncol=4))
(S <- spray(M,sample(7),addrepeats=TRUE))
as.array(S,offbyone=TRUE)      # a large object!  sprays are terse


S <- spray(matrix(sample(1:4,28,replace=TRUE),ncol=4),sample(7))
A <- as.array(S)   # S has no zero indices [if it did, we would need to use offbyone=TRUE]

stopifnot(all(S[index(S),drop=TRUE] == A[index(S)]))
```

as.function.spray *Coerce a spray object to a function*

Description

Coerce a spray object to a function

Usage

```
## S3 method for class 'spray'
as.function(x,...)
```

Arguments

x	spray object, interpreted as a multivariate polynomial
...	Further arguments, currently ignored

Value

Returns a function; this function returns a numeric vector.

Note

Coercion is possible even if some indices are zero or negative. The function is not vectorized in the arity of its argument.

Author(s)

Robin K. S. Hankin

Examples

```
(S <- spray(matrix(1:6,3,2),1:3))
(f <- as.function(S))
f(2:3) == 3*2^3*3^6 + 2*2^2*3^5 + 1*2^1*3^4  # should be TRUE


S1 <- spray(matrix(sample(-2:2,replace=TRUE,21),ncol=3),rnorm(7),addrepeats=TRUE)
S2 <- spray(matrix(sample(-2:2,replace=TRUE,15),ncol=3),rnorm(5),addrepeats=TRUE)

f1 <- as.function(S1)
f2 <- as.function(S2)

f3 <- as.function(S1*S2)


x <- 4:6

f1(x)*f2(x)-f3(x)  # should be zero



# coercion is vectorized:
```

```
f1(matrix(1:33,ncol=3))
```

---

asum *Sum over dimension margins*

Description

Sum over specified dimension margins.

Usage

```
## S3 method for class 'spray'
asum(S, dims, drop=TRUE, ...)
asum_inverted(S, dims)
process_dimensions(S,dims)
```

Arguments

S	spray object
dims	Vector of strictly positive integers corresponding to dimensions to be summed over
drop	Boolean, with default `TRUE` meaning to drop the summed dimensions, and `FALSE` meaning to retain them.
...	Further arguments, currently ignored

Details

Function `asum.spray()` is the method for `asum()`. This takes a spray, and a vector of integers corresponding to dimensions to be summed over.

Function `asum_inverted()` is the same, but takes a vector of integers corresponding to dimensions not to sum over. This function is here because there is nice `C++` idiom for it.

Function `process_dimensions()` ensures that the `dims` argument is consistent with the spray `S` and returns a cleaned version thereof.

Value

Returns a spray object.

Author(s)

Robin K. S. Hankin

Examples

```
S <- spray(matrix(sample(0:2,60,replace=TRUE),ncol=3),addrepeats=TRUE)
S

asum(S,1)
asum(S,1:2)

asum(S,1:2,drop=FALSE)


asum(S,c(1,3)) == asum_inverted(S,2)
```

`constant` *Get or set the constant term of a spray object*

Description

The constant term of a spray object is the coefficient corresponding to an index of all zeros. These functions get or set the constant of a spray object.

Usage

```
is.constant(x)
constant(x,drop=FALSE)
constant(x) <- value
drop(x)
```

Arguments

`x`	Object of class spray
`value`	Numeric value to set the constant coefficient to
`drop`	Boolean, with default `FALSE` meaning to return a spray object and `TRUE` meaning to return a numeric value

Value

In function `constant()`, return the coefficient, or a constant multivariate polynomial, depending on the value of `drop`.

Note

The behaviour of the `drop` argument (sort of) matches that of the spray extractor method. Function `drop()` returns the elements of the coefficients.

Function `constant()` ensures that zero spray objects retain the argument's arity.

It might have been better to call `is.constant()` `is.scalar()`, for consistency with the `stokes` and `clifford` packages. But this is not clear.

Author(s)

Robin K. S. Hankin

See Also

Extract

Examples

```
(S <- spray(partitions::blockparts(rep(2,3),3,TRUE)))

constant(S)
constant(S) <- 33

S

drop(constant(S,drop=FALSE))
```

---

deriv *Partial differentiation of spray objects*

Description

Partial differentiation of spray objects interpreted as multivariate polynomials

Usage

```
## S3 method for class 'spray'
deriv(expr, i , derivative = 1, ...)
aderiv(S,orders)
```

Arguments

expr	A spray object, interpreted as a multivariate polynomial
i	Dimension to differentiate with respect to
derivative	How many times to differentiate
...	Further arguments, currently ignored
S	spray object
orders	The orders of the differentials

Details

Function `deriv.spray()` is the method for generic `spray()`; if `S` is a spray object, then `spray(S,i,n)` returns $\partial^n S/\partial x_i^n = S^{(x_i,\ldots,x_i)}$.

Function `aderiv()` is the generalized derivative; if `S` is a spray of arity 3, then `aderiv(S,c(i,j,k))` returns $\frac{\partial^{i+j+k}S}{\partial x_1^i \partial x_2^j \partial x_3^k}$.

Value

Both functions return a spray object.

Author(s)

Robin K. S. Hankin

See Also

asum

Examples

```
(S <- spray(matrix(sample(-2:2,15,replace=TRUE),ncol=3),addrepeats=TRUE))

deriv(S,1)
deriv(S,2,2)

# differentiation is invariant under order:
aderiv(S,1:3) == deriv(deriv(deriv(S,1,1),2,2),3,3)

# Leibniz's rule:
S1 <- spray(matrix(sample(0:3,replace=TRUE,21),ncol=3),sample(7),addrepeats=TRUE)
S2 <- spray(matrix(sample(0:3,replace=TRUE,15),ncol=3),sample(5),addrepeats=TRUE)

S1*deriv(S2,1) + deriv(S1,1)*S2 == deriv(S1*S2,1)

# Generalized Leibniz:
aderiv(S1*S2,c(1,1,0)) == (
aderiv(S1,c(0,0,0))*aderiv(S2,c(1,1,0)) +
aderiv(S1,c(0,1,0))*aderiv(S2,c(1,0,0)) +
aderiv(S1,c(1,0,0))*aderiv(S2,c(0,1,0)) +
aderiv(S1,c(1,1,0))*aderiv(S2,c(0,0,0))
)
```

Extract.spray *Extract or Replace Parts of a spray*

Description

Extract or replace subsets of sprays.

Usage

```
## S3 method for class 'spray'
S[..., drop=FALSE]
## S3 replacement method for class 'spray'
S[index, ...] <- value
```

Arguments

S	A spray object
index	elements to extract or replace
value	replacement value
...	Further arguments
drop	Boolean, with default `FALSE` meaning to return a spray object and `TRUE` meaning to drop the spray structure and return a numeric vector

Details

These methods should work as expected, although the off-by-one issue might be a gotcha.

If `drop` is `TRUE`, a numeric vector is returned but the elements may be in any order.

If `a <- spray(diag(3))`, for example, then idiom such as `a[c(1,2,3)]` cannot work, because one would like `a[1,2,3]` and `a[1:3,2,3]` to work.

If `p <- 1:3`, then one might expect idiom such as `S[1,,p,1:3]` to work but this is problematic and a discussion is given in `inst/missing_accessor.txt`.

Examples

```
(a <- spray(diag(5)))
a[rbind(rep(1,5))] <- 5
a

a[3,4,5,3,1]  # the NULL polynomial

a[0,1,0,0,0]
a[0,1,0,0,0,drop=TRUE]

a[2,3:5,4,3,3] <- 9
a

options(polyform = TRUE)   # print as a multivariate polynomial
a

options(polyform = FALSE)  # print in sparse array form
a

(S1 <- spray(diag(5),1:5))
(S2 <- spray(1-diag(5),11:15))
(S3 <- spray(rbind(c(1,0,0,0,0),c(1,2,1,1,1))))

S1[] <- 3
S1[] <- S2

S1[S3] <- 99
S1
```

---

homog *Various functions to create simple spray objects*

Description

Various functions to create simple spray objects such as single-term, homogeneous, and constant multivariate polynomials.

Usage

```
product(power)
homog(d,power=1)
linear(x,power=1)
lone(n,d=n)
```

```
one(d)
as.id(S)
xyz(d)
```

Arguments

d	An integer; generally, the dimension or arity of the resulting spray object
power	Integer vector of powers
x	Numeric vector of coefficients
S	A spray object
n	In function `lone()`, the term to raise to power 1

Value

All functions documented here return a spray object

Note

The functions here are related to their equivalents in the multipol package, but are not exactly the same.

Function `zero()` is documented at `zero.Rd`, but is listed below for convenience.

Author(s)

Robin K. S. Hankin

See Also

`constant`, `zero`

Examples

```
product(1:3)            #     x * y^2 * z^3
homog(3)                #     x + y + z
homog(3,2)              #     x^2  + xy + xz + y^2 + yz + z^2
linear(1:3)             #     1*x + 2*y + 3*z
linear(1:3,2)           #     1*x^2 + 2*y^2 + 3*z^2
lone(3)                 #     z
lone(2,3)               #     y
one(3)                  #     1
zero(3)                 #     0
xyz(3)                  #     xyz
```

knight *Generating function for a chess knight and king*

Description

Generating function for a chess knight and king on an arbitrarily-dimensioned chessboard

Usage

```
knight(d=2)
king(d=2)
```

Arguments

d	Dimensionality of the board, defaulting to 2

Value

Returns the generating function of the piece in question.

Note

The pieces are forced to move; if they have the option of not moving, add 1 to the returned spray. The vignette contains a short discussion.

Author(s)

Robin K. S. Hankin

Examples

```
knight()  # default 2D chess board
king()    # ditto

knight()^2  # generating function for two knight's moves

## How many ways can a knight return to its starting square in 6 moves?
constant(knight()^6)

## How many in 6 or fewer?
constant((1+knight())^6)

## Where does a randomly-moving knight end up?
d <- xyz(2)
kt <- (1+knight())*d^2/9
persp(1:25,1:25,as.array(d*kt^6))


## what is the probability that a 4D king is a knight's move from
##   (0,0,0,0) after 6 moves?

sum(coeffs(((king(4)/80)^4)[knight(4)]))
```

`nterms` — *Number of nonzero terms in a* `spray` *object*

Description

Number of nonzero terms in a `spray` object

Usage

```
nterms(S)
```

Arguments

`S`	Object of class `spray`

Author(s)

Robin K. S. Hankin

Examples

```
(a <- rspray())
nterms(a)
```

`ooom` — *One-over-one-minus for spray objects*

Description

One-over-one-minus for spray objects; the nearest to ‘division’ that we can get.

Usage

```
ooom(S, n)
```

Arguments

`S`	object of class spray
`n`	Order of the approximation

Details

Returns the Taylor expansion to order n of $1/(1-S)$, that is, $1+S+S^2+S^3+\ldots+S^n$.

Value

Returns a spray object of the same arity as `S`.

Note

Uses Horner's method for efficiency

Author(s)

Robin K. S. Hankin

Examples

```
(x <- spray(matrix(1)))
ooom(x,5)  # 1 + x + x^2 + x^3 + x^4 + x^5


(a <- homog(4,2))
d <- (1-a)*ooom(a,3)

constant(d)    # should be 1
rowSums(index(d))   # a single 0 and lots of 8s.
```

Ops.spray *Arithmetic Ops Group Methods for sprays*

Description

Allows arithmetic operators to be used for spray calculations, such as addition, multiplication, division, integer powers, etc. Objects of class spray are interpreted as sparse multivariate polynomials.

Usage

```
## S3 method for class 'spray'
Ops(e1, e2 = NULL)
spray_negative(S)
spray_times_spray(S1,S2)
spray_times_scalar(S,x)
spray_plus_spray(S1,S2)
spray_plus_scalar(S,x)
spray_power_scalar(S,n)
spray_eq_spray(S1,S2)
spray_eq_numeric(S1,x)
```

Arguments

`e1,e2,S,S1,S2`	Objects of class spray, here interpreted as sparse multivariate polynomials
`x`	Real valued scalar
`n`	Non-negative integer

Details

The function `Ops.spray()` passes unary and binary arithmetic operators ("+", "-", "*", "/","==", and "^") to the appropriate specialist function.

The most interesting operators are "*" and "+" which execute multivariate polynomial multiplication and addition respectively.

Testing for equality uses `spray_eq_spray()`. Note that `spray_eq_spray(S1,S2)` is algebraically equivalent to `is.zero(S1-S2)`, but faster (`FALSE` is returned as soon as a mismatch is found).

Value

The functions all return spray objects except "==", which returns a logical.

Author(s)

Robin K. S. Hankin

See Also

ooom

Examples

```
M <- matrix(sample(0:3,21,replace=TRUE),ncol=3)
a <- spray(M,sample(7))
b <- homog(3,4)


# arithmetic operators mostly work as expected:
a + 2*b
a - a*b^2/4
a+b

S1 <- spray(partitions::compositions(4,3))
S2 <- spray(diag(3))  # S2 = x+y+z


stopifnot( (S1+S2)^3 == S1^3 + 3*S1^2*S2 + 3*S1*S2^2 + S2^3 )
```

---

pmax   *Parallel maxima and minima for sprays*

Description

Parallel (pairwise) maxima and minima for sprays.

Usage

```
maxpair_spray(S1,S2)
minpair_spray(S1,S2)
## S3 method for class 'spray'
pmax(x, ...)
## S3 method for class 'spray'
pmin(x, ...)
```

Arguments

x,S1,S2	Spray objects
...	spray objects to be compared

Details

Function `maxpair_spray()` finds the pairwise maximum for two sprays. Specifically, if `S3 <- maxpair_spray(S1,S2)`, then `S3[v] == max(S1[v],S2[v])` for every index vector `v`.

Function `pmax.spray()` is the method for the generic `pmax()`, which takes any number of arguments. If `S3 <-maxpair_spray(S1,S2,...)`, then `S3[v] == max(S1[v],S2[v],...)` for every index vector `v`.

Function `pmax.spray()` operates right-associatively:

`pmax(S1,S2,S3,S4) == f(S1,f(S2,f(S3,S4)))` where `f()` is short for `maxpair_spray()`. So if performance is important, put the smallest spray (in terms of number of nonzero entries) last.

In these functions, a scalar is interpreted as a sort of global maximum. Thus if `S3 <- pmax(S,x)` we have `S3[v] == max(S[v],x)` for every index `v`. Observe that this operation is not defined if `x>0`, for then there would be an infinity of `v` for which `S3[v] != 0`, an impossibility (or at least counter to the principles of a sparse array). Note also that `x` cannot have length `>1` as the elements of a spray object are stored in an arbitrary order.

Functions `minpair_spray()` and `pmin.spray()` are analogous. Note that `minpair_spray(S1,S2)` is algebraically equivalent to `-pmax_spray(-S1,-S2)`; see the examples.

The value of `pmax(S)` is problematic. Suppose `all(coeffs(S)<0)`; the current implementation returns `pmax(S)==S` but there is a case for returning the null polynomial.

Value

Returns a spray object

Author(s)

Robin K. S. Hankin

Examples

```
S1 <- rspray(100,vals=sample(100)-50)
S2 <- rspray(100,vals=sample(100)-50)
S3 <- rspray(100,vals=sample(100)-50)


# following comparisons should all be TRUE:

jj <- pmax(S1,S2,S3)
```

```
jj ==  maxpair_spray(S1,maxpair_spray(S2,S3))
jj ==  maxpair_spray(maxpair_spray(S1,S2),S3)

pmax(S1,S2,S3)  == -pmin(-S1,-S2,-S3)
pmin(S1,S2,S3)  == -pmax(-S1,-S2,-S3)

pmax(S1,-Inf) == S1
pmin(S1, Inf) == S2


pmax(S1,-3)

## Not run:
pmax(S1,3) # not defined

## End(Not run)
```

---

print.spray *Print methods for spray objects*

Description

Print methods for spray objects with options for printing in matrix form or multivariate polynomial form

Usage

```
## S3 method for class 'spray'
print(x, ...)
print_spray_matrixform(S)
print_spray_polyform(S)
```

Arguments

x,S	spray object
...	Further arguments (currently ignored)

Details

The print method, `print.spray()`, dispatches to helper functions `print_spray_matrixform()` and `print_spray_polyform()` depending on the value of option `polyform`; see the examples section.

Option `sprayvars` is a character vector with entries corresponding to the variable names for printing. The `sprayvars` option has no algebraic significance: all it does is affect the print method.

Note that printing a spray object (in either matrix form or polynomial form) generally takes much longer than calculating it.

Value

Returns its argument invisibly.

Note

There are a couple of hard-wired symbols for multiplication and equality which are defined near the top of the helper functions.

Author(s)

Robin K. S. Hankin

Examples

```
(a <- spray(diag(3)))


options(polyform = FALSE)
a^3

options(polyform = TRUE)
a^3


options(sprayvars=letters)
a <- diag(26)
spray(a)

## Following example from mpoly:
a[1 + cbind(0:25, 1:26) %% 26] <- 2
spray(a)
```

`rspray` *Random spray objects*

Description

Creates random spray objects as quick-and-dirty examples of multivariate polynomials

Usage

```
rspray(n=9, vals = seq_len(n), arity = 3, powers = 0:2)
```

Arguments

`n`	Number of distinct rows (maximum); repeated rows are merged (argument `addrepeats` is `TRUE`)
`vals`	Values to use for coefficients
`arity`	Arity of the spray; the number of columns in the index matrix
`powers`	Set from which to sample the entries of the index matrix

Value

Returns a spray object

Note

If the index matrix contains repeated rows, the returned spray object will contain fewer than `n` entries

Author(s)

Robin K. S. Hankin

See Also

`spray`

Examples

```
rspray()

rspray(4)*rspray(3,rnorm(3))

rspray(3,arity=7,powers=-2:2)^3

rspray(1000,vals=rnorm(1000))
```

---

spray — *Sparse arrays:* `spray` *objects*

Description

Create, coerce, and test for sparse array objects

Usage

```
spray(M, x, addrepeats=FALSE)
spraymaker(L, addrepeats=FALSE, arity=ncol(L[[1]]))
is.spray(S)
as.spray(arg1, arg2, addrepeats=FALSE, offbyone=FALSE)
index(S)
coeffs(S)
coeffs(S) <- value
is_valid_spray(L)
```

Arguments

`M`	Integer matrix with rows corresponding to index positions
`x`	Numeric value with elements corresponding to spray entries
`S`	Object to be tested for being a spray
`L`	A list, nominally of two elements (index matrix and value) which is to be tested for acceptability to be coerce to class spray
`arg1,arg2`	Various arguments to be coerced to a spray
`addrepeats`	Boolean, with default `FALSE` meaning to check for repeated index rows and, if any are found, return an error

`value`	In the assignment operator `coeffs<-()`, a `disord` object (or a length-one numeric vector), so that `coeffs(S) <- x` works as expected
`offbyone`	In function `as.spray()`, when converting from an array. Argument `offbyone` is Boolean with default `FALSE` meaning to insert array elements in positions corresponding to index elements, and `TRUE` meaning to add one
`arity`	In function `spraymaker()`, integer specifying the arity (number of columns of the index matrix `L[[1]]`); ignored if `L` is non-empty. See details

Details

Spray objects are sparse arrays interpreted as multivariate polynomials. They can be added and subtracted; "`*`" is interpreted as polynomial multiplication,

To create a spray object the user should use `spray()`, if a matrix of indices and vector of values is available, or `as.spray()` which tries hard to do the Right Thing (tm).

Function `spraymaker()` is the formal creator function, and it is written to take the output of the C++ routines and return a spray object. The reason this needs an `arity` argument is that C++ sometimes returns `NULL` (in lieu of a zero-row matrix, which it cannot deal with). In this case, we need some way to tell R the arity of the corresponding spray object.

Functions `index()` and `coeffs()` are accessor methods.

There is an extensive vignette available; type `vignette("spray")` at the command line.

Note

Function `coeffs()` was formerly known as `value()`; function `value()` will be deprecated

Author(s)

Robin K. S. Hankin

See Also

Ops,spray-package

Examples

```
S <- spray(diag(5))    # missing second argument interpreted as '1'.
as.array(S,offbyone=TRUE) # zero indices interpreted as ones.

M <- matrix(1:5,6,5) # note first row matches the sixth row

## Not run:  spray(M,1:6)  # will not work because addrepeats is not TRUE


spray(M,1:6,addrepeats=TRUE)  # 7=1:6


S <- spray(matrix(1:7,5,7))
a <- as.array(S)    # will not work if any(M<1)
S1 <- as.spray(a)
stopifnot(S==S1)

a <- rspray(20)
coeffs(a)[coeffs(a) %% 2 == 1] <- 99  # every odd coefficient -> 99
```

`spray-class` *Class "spray"*

Description

The formal S4 class for sprays.

Objects from the Class

Objects *can* be created by calls of the form `new("spray", ...)` but this is not encouraged. Use functions `spray()` or `as.spray()` instead.

Slots

`index`: Index matrix

`value`: Numeric vector holding coefficients

Author(s)

Robin K. S. Hankin

See Also

`spray`

`spraycross` *Cross product for spray objects*

Description

Provides a natural cross product for spray objects, useful for tensors and k-forms

Usage

```
spraycross(S, ...)
spraycross2(S1,S2)
```

Arguments

`S,S1,S2,...`	spray objects

Details

Cross products for sprays. This is *not* an algebraic product of sprays interpreted as multivariate polynomials. The function is used in the **stokes** package.

Function `spraycross2()` is a helper function that takes exactly two arguments. Function `spraycross()` is a more general function that takes any number of arguments.

Value

Returns a spray object

Author(s)

Robin K. S. Hankin

Examples

```
a <- spray(matrix(1:4,2,2),c(2,5))
b <- spray(matrix(c(10,11,12,13),2,2),c(7,11))
a
b
spraycross2(a,b)
spraycross2(b,a)

spraycross(a,b,b)
```

`spray_cpp` *Low-level functions that call C++ source code*

Description

Low-level functions that call C++ source code, as detailed in the automatically generated `RcppExports.R` file.

Usage

```
spray_maker(M, d)
spray_add(M1, d1, M2, d2)
spray_mult(M1, d1, M2, d2)
spray_overwrite(M1, d1, M2, d2)
spray_accessor(M, d, Mindex)
spray_setter(M1, d1, M2, d2)
spray_equality(M1, d1, M2, d2)
spray_asum_include(M,d,n)
spray_asum_exclude(M,d,n)
spray_deriv(M,d,n)
spray_pmax(M1,d1,M2,d2)
spray_pmin(M1,d1,M2,d2)
spray_power(M,d,pow)
spray_spray_accessor()
spray_spray_add()
spray_spray_asum_exclude()
spray_spray_asum_include()
spray_spray_deriv()
spray_spray_equality()
spray_spray_maker()
spray_spray_mult()
spray_spray_overwrite()
spray_spray_pmax()
```

```
spray_spray_pmin()
spray_spray_setter()
spray_spray_power()
```

Arguments

`M,M1,M2,Mindex`	Integer valued matrices with rows corresponding to array indices
`d,d1,d2`	Vector of values corresponding to nonzero array entries
`n`	Integer vector corresponding to dimensions to sum over for the sum functions
`pow`	Nonnegative integer for `spray_power()`

Value

These functions return a two-element list which is coerced to an object of class `spray` by function `spraymaker()`.

Note

These functions aren't really designed for the end-user.

Function `spray_equality()` cannot simply check for equality of `$value` because the order of the index rows is not specified in a spray object. Function `spray_crush()` has been removed as it is redundant.

Author(s)

Robin K. S. Hankin

See Also

`spraymaker`,`spray`

`spray_missing_accessor` *Discussion document*

Description

Discussion about the difficulties of implementing idiom like `S[1,,5,,]` in the package

Usage

```
spray_missing_accessor(S, dots)
```

Arguments

`S`	Object of class spray
`dots`	further arguments

Details

Look at the source which contains an extended discussion of the difficulties

Author(s)

Robin K. S. Hankin

---

subs *Substitute values into a spray object*

Description

Substitute values into a spray object, interpreted as a multivariate polynomial

Usage

```
subs(S, dims, x, drop=TRUE)
```

Arguments

`S`	spray object
`dims`	Integer or logical vector with entries corresponding to the dimensions to be substituted
`x`	Numeric vector of values to be substituted
`drop`	Boolean, with default `TRUE` meaning to return the `drop()` of the result, and `FALSE` meaning to return a `spray` object consistently

Note

It is much easier if argument `dims` is sorted into increasing order. If not, caveat emptor!

Author(s)

Robin K. S. Hankin

See Also

`process_dimensions`

Examples

```
(S <- spray(matrix(sample(0:3,60,replace=TRUE),nrow=12)))

subs(S,c(2,5),1:2)

P <- homog(3,3)
subs(P,1,2)
```

summary.spray *Summaries of spray objects*

Description

A summary method for spray objects, and a print method for summaries.

Usage

```
## S3 method for class 'spray'
summary(object, ...)
## S3 method for class 'summary.spray'
print(x, ...)
```

Arguments

`object,x`	Object of class `spray`
`...`	Further arguments, passed to `head()`

Details

A `summary.spray` object is summary of a `spray` object `x`: a list with first element being a `summary()` of the coefficients (which is a `disord` object), and the second being a `spray` object comprising a few selected index-coefficient pairs. The selection is done by `head()`.

Note

The “representative selection” is impementation-specific, as it uses `disordR::elements()` to extract rows of the index matrix and coefficients.

Author(s)

Robin K. S. Hankin

Examples

```
a <- rspray()^2
a
summary(a)
summary(a,2)

options(polyform=TRUE)
summary(a^4,3)
options(polyform=FALSE) # restore default
```

zap — *Zap small values in a spray object*

Description

Generic version of `zapsmall()`

Usage

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

Arguments

`x`	spray object
`digits`	number of digits to retain

Details

Given a spray object, coefficients close to zero are ‘zapped’, i.e., replaced by ‘0’, using `base::zapsmall()`. Function `zap()` is an easily-typed alias; `zapsmall()` is the S4 generic.

Note, `zap()` actually changes the numeric value, it is not just a print method.

Author(s)

Robin K. S. Hankin

Examples

```
(S <- spray(matrix(sample(1:50),ncol=2),10^-(1:25)))
zap(S)

S-zap(S)        # print method will probably print zeros...
coeffs(S-zap(S)) # ...but they are nevertheless nonzero
```

zero — *The zero polynomial*

Description

Test for the zero, or empty, polynomial

Usage

```
zero(d)
is.zero(x)
is.empty(L)
```

Arguments

`L,x`	A two-element list of indices and values, possibly a spray object or numeric vector
`d`	Integer specifying dimensionality of the spray (the arity)

Details

Functions `is.empty()` and `is.zero()` are synonyms. If spray objects are interpreted as multivariate polynomials, "`is.zero()`" is more intuitive, if sprays are interpreted as sparse arrays, "`is.empty()`" is better (for me).

Passing a zero-row index matrix can have unexpected effects:

```
   dput(spray(matrix(0,0,5),9))
structure(list(structure(numeric(0), .Dim = c(0L, 5L)), numeric(0)), class = "spray")
```

Above, the index matrix has zero rows (and no elements) but the fact that it has five columns is retained. The `spray` object has no coefficients as the index matrix has zero rows. Compare:

```
   spray(matrix(0,0,5),0)
empty sparse array with 5 columns
   spray(matrix(0,1,5),0)
empty sparse array with  columns

   dput(spray(matrix(0,1,5),0))
structure(list(index = NULL, value = NULL), class = "spray")
>


   spray(matrix(0,1,5),0)
empty sparse array with  columns

   dput(spray(matrix(0,1,5),0))
structure(list(index = NULL, value = NULL), class = "spray")
>
```

Above, the index matrix given to `spray()` has one row but the coefficient is zero. The resulting `spray` object has a `NULL` index matrix [because rows with zero coefficients are removed] and a `NULL` coefficient.

Arguably, the output should include the fact that we are dealing with a 5-dimensional array; but the index matrix is `NULL` so this information is lost (note that the value is `NULL` too). However, observe that the following works:

```
> a1 <- spray(matrix(0,1,4),0)
> a2 <- spray(t(1:5))
> a1+a2
               val
 1 2 3 4 5  =    1
 >
```

Examples

```
(a <- lone(1,3))

is.zero(a-a)  # should be TRUE

is.zero(zero(6))

x <- spray(t(0:1))
y <- spray(t(1:0))

is.zero((x+y)*(x-y)-(x^2-y^2)) # TRUE
```

Index