Package ‘spray’

February 15, 2021

Type Package
Title Sparse Arrays and Multivariate Polynomials
Version 1.0-11
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
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 .................................................. 2
  arity ............................................................. 3
  as.array ......................................................... 3
  as.function.spray ............................................... 4
  asum ............................................................ 5
  constant ......................................................... 6
  deriv ............................................................ 7
  Extract.spray .................................................. 9
  homog .......................................................... 10
  knight .......................................................... 11
  nterms ......................................................... 12
  ooom ............................................................ 13
  Ops.spray ....................................................... 14
  pmax ............................................................ 15
  print.spray ..................................................... 17
  rspray .......................................................... 18
  spray ........................................................... 19
  spray-class ..................................................... 20
Description

Functionality for sparse arrays, with emphasis on sparse arrays interpreted 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.

One natural application for sparse arrays is multivariate polynomials and the package vignette presents an extended discussion.

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.

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:

```r
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**

```r
arity(S)
```

**Arguments**

- `S`  
  a spray object

**Value**

Returns an integer

**Author(s)**

Robin K. S. Hankin

**Examples**

```r
arity(spray(diag(1:6)))
```

---

### as.array

**Coerce spray objects to arrays**

**Description**

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

**Usage**

```r
data <- spray(vec)
dim(data)
```

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

```r
## 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 \( \text{dim}(S) \). The “meat” of the function is

\[
\text{out} <- \text{array}(0, \text{dS})
\]

\[
\text{out}[\text{ind}] <- \text{value}(S)
\]

Author(s)

Robin K. S. Hankin

Examples

```r
M <- matrix(sample(0:4, 28, replace=TRUE), ncol=4)
S <- spray(M, sample(7), addrepeats=TRUE)
A <- as.array(S, offbyone=TRUE)

S <- spray(matrix(sample(1:4, 28, replace=TRUE), ncol=4), sample(7))
A <- as.array(S)  # S has no zero indices
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

```r
## 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

```r
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

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

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

constant(S,drop=FALSE)
constant(S) <- value
Arguments

- **S**: 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 `drop` (sort of) matches that of the spray extractor method.

Author(s)

Robin K. S. Hankin

See Also

- Extract

Examples

```r
S <- spray(partitions::blockparts(rep(2,4),3,TRUE))
constant(S)
constant(S) <- 33
S
```

---

**deriv**

Partial differentiation of spray objects

Description

Partial differentiation of spray objects interpreted as multivariate polynomials

Usage

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

eexpr 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 $\frac{\partial^n S}{\partial x_i^n} = S^{(\cdot, \ldots, x_i)}$.

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

```r
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

```r
## 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[,p,1:3]` to work but this is problematic and a discussion is given in `inst/missing_accessor.txt`.

Examples

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

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

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

options(polyform = FALSE) # print in sparse array form
```
Various functions to create simple spray objects

Description

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

Usage

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

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*x + 3*z^2
lone(3)       # z
lone(2,3)     # y
one(3)        # 1
zero(3)       # 0
xyz(3)        # xyz

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

```r
## 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(value(((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

```r
nterms(S)
```

### Arguments

- **S**: Object of class `spray`

### Author(s)

Robin K. S. Hankin

### Examples

```r
nterms(spray(diag(seq_len(5))))
```
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

\texttt{ooom(S, n)}

Arguments

- \texttt{S} \quad object of class spray
- \texttt{n} \quad Order of the approximation

Details

Returns the Taylor expansion to order \(n\).

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

\begin{verbatim}
a <- homog(4,2)
jj <- (1-a)*ooom(a,3)

constant(jj)  # should be 1
rowSums(index(jj))  # a single 0 and lots of 8s.
\end{verbatim}
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

```r
## 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)
```

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 ("+", "-", "\times", "/", ":=", and "\^\small{\text{ }}") to the appropriate specialist function.

The most interesting operators are "\times" and "\^\small{\text{ }}" 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.

Note

Notes here

Author(s)

Robin K. S. Hankin

See Also

oom
pmax

Examples

```r
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 )
```

Description

Parallel (pairwise) maxima and minima for sprays.

Usage

```r
maxpair_spray(S1,S2)
minpair_spray(S1,S2)
```

## S3 method for class `spray`

```r
pmax(x, ...)
```

## S3 method for class `spray`

```r
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 \texttt{pmax.spray()} operates right-associatively:

\[
p\text{max}(S_1,S_2,S_3,S_4) = f(S_1,f(S_2,f(S_3,S_4)))
\]
where \(f()\) is short for \texttt{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 \(S_3 < p\text{max}(S,x)\) we have \(S_3[v] = \text{max}(S[v],x)\) 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 \(S_3[v] != 0\), 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 \texttt{minpair.spray()} and \texttt{pmin.spray()} are analogous. Note that \texttt{minpair.spray(S1,S2)} is algebraically equivalent to \(-p\text{max.spray}(-S_1,-S_2)\); see the examples.

The value of \(p\text{max}(S)\) is problematic. Suppose \(\text{all(value}(S)<0)\); the current implementation returns \(p\text{max}(S)=S\) but there is a case for returning the null polynomial.

**Value**

Returns a spray object

**Author(s)**

Robin K. S. Hankin

**Examples**

```r
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
```

## Not run:

pmax(S1,3) # not defined

## End(Not run)
## 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.

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.

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

options(polyform = FALSE)
a^3

options(polyform = TRUE)
```
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 Create sparse array 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)
value(S)
value(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 `value<-()`, a scalar so that value(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

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 `value()` are accessor methods.

Author(s)

Robin K. S. Hankin

Examples

```r
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)
```

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
Low-level functions that call C++ source code

See Also

`spray`

**Description**

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

**Usage**

```r
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_derivative(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_derivative()
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 \$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**

```r
subs(S, dims, x)
```

**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

**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**

```r
S <- spray(matrix(sample(0:3,60,replace=TRUE),nrow=12))
subs(S,c(2,5),1:2)
subs(homog(3,3),1,3)
```

---

**zap**

*Zap small values in a spray object*

**Description**

Generic version of `zapsmall()`

**Usage**

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

\begin{itemize}
  \item \texttt{x} spray object
  \item \texttt{digits} number of digits to retain
\end{itemize}

Details

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

Note, \texttt{zap()} actually changes the numeric value, it is not just a print method.

Author(s)

Robin K. S. Hankin

Examples

\begin{verbatim}
S <- spray(matrix(sample(1:50),ncol=2),10^-(1:25))
zap(S)
S-zap(S) # print method will probably print zeros...
value(S-zap(S)) # ...but they are nevertheless nonzero
\end{verbatim}

\begin{center}
zero
\end{center}

\textit{The zero polynomial}

Description

Test for the zero, or empty, polynomial

Usage

\begin{verbatim}
zero(d)
is.zero(L)
is.empty(L)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{L} A two-element list of indices and values, possibly a spray object
  \item \texttt{d} Integer specifying dimensionality of the spray (the arity)
\end{itemize}

Details

Functions \texttt{is.empty()} and \texttt{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).
Examples

```r
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

* classes
  spray-class, 20
* datasets
  zero, 24
* mathsymbol
  deriv, 7
* package
  spray-package, 2
* symbolmath
  arity, 3
  as.array, 3
  as.function.spray, 4
  asum, 5
  constant, 6
  Extract.spray, 9
  homog, 10
  ooom, 13
  Ops.spray, 14
  print.spray, 17
  rspray, 18
  spray, 19
  spray_cpp, 21
 _subs, 23
  [.spray(Extract.spray), 9
  [<-.spray(Extract.spray), 9

  aderiv (deriv), 7
  arity, 3
  as.array, 3
  as.function.spray, 4
  as.spray (spray), 19
  asum, 5, 8
  asum_inverted (asum), 5
  chess_knight (knight), 11
  coeff (spray), 19
  coeffs (spray), 19
  constant, 6, 11
  constant<-(constant), 6
  deriv, 7
  dim.spray (as.array), 3
  empty (zero), 24

  Extract, 7
  extract(Extract.spray), 9
  Extract.spray, 9

  homog, 10

  index (spray), 19
  is.empty (zero), 24
  is.spray (spray), 19
  is.zero (zero), 24
  is_valid_spray (spray), 19

  king (knight), 11
  knight, 11

  linear (homog), 10
  lone (homog), 10

  maxpair_spray (pmax), 15
  minpair_spray (pmax), 15

  nterms, 12

  one (homog), 10
  ooom, 13, 14
  Ops (Ops.spray), 14
  Ops.spray, 14

  pmax, 15
  pmin (pmax), 15
  print.spray, 17
  print_spray_matrixform (print.spray), 17
  print_spray_polyform (print.spray), 17
  process_dimensions, 23
  process_dimensions (asum), 5
  product (homog), 10

  replace (Extract.spray), 9
  rspray, 18

  spray, 18, 19, 21, 22
  spray-class, 18, 19, 21, 22
  spray-package, 2
  spray_accessor (spray_cpp), 21
  spray_add (spray_cpp), 21
INDEX

spray_asum_exclude (spray_cpp), 21
spray_asum_include (spray_cpp), 21
spray_cpp, 21
spray_crush (spray_cpp), 21
spray_deriv (spray_cpp), 21
spray_eq_spray (Ops.spray), 14
spray_equality (spray_cpp), 21
spray_maker (spray_cpp), 21
spray_missing_accessor, 22
spray_mult (spray_cpp), 21
spray_negative (Ops.spray), 14
spray_overwrite (spray_cpp), 21
spray_plus_scalar (Ops.spray), 14
spray_plus_spray (Ops.spray), 14
spray_pmax (spray_cpp), 21
spray_pmin (spray_cpp), 21
spray_power (spray_cpp), 21
spray_power_scalar (Ops.spray), 14
spray_rcpp (spray_cpp), 21
spray_setter (spray_cpp), 21
spray_spray_accessor (spray_cpp), 21
spray_spray_add (spray_cpp), 21
spray_spray_asum_exclude (spray_cpp), 21
spray_spray_asum_include (spray_cpp), 21
spray_spray_crush (spray_cpp), 21
spray_spray_deriv (spray_cpp), 21
spray_spray_equality (spray_cpp), 21
spray_spray_maker (spray_cpp), 21
spray_spray_mult (spray_cpp), 21
spray_spray_overwrite (spray_cpp), 21
spray_spray_pmax (spray_cpp), 21
spray_spray_pmin (spray_cpp), 21
spray_spray_power (spray_cpp), 21
spray_spray_setter (spray_cpp), 21
spray_spray_times_scalar (Ops.spray), 14
spray_times_spray (Ops.spray), 14
spraymaker, 22
spraymaker (spray), 19
sprayvars (print.spray), 17
subs, 23
substitute (subs), 23

value (spray), 19
value<- (spray), 19
values (spray), 19

xyz (homog), 10

zap, 23
zapsmall (zap), 23
zapsmall, ANY-method (zap), 23
zapsmall, spray-method (zap), 23
zapsmall.spray (zap), 23