

Package ‘spray’

May 25, 2017

Type Package

Title Sparse Arrays and Multivariate Polynomials

Version 1.0-3

Date 2017-05-18

Author Robin K. S. Hankin

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

Description Sparse arrays interpreted as multivariate polynomials.

License GPL (>= 2)

Depends methods

Suggests polynom

Imports Rcpp (>= 0.12.3),partitions,magic

LinkingTo Rcpp

SystemRequirements C++11

NeedsCompilation yes

Repository CRAN

Date/Publication 2017-05-25 21:45:34 UTC

R topics documented:

spray-package	2
arity	3
as.array	4
as.function.spray	5
asum	6
constant	7
deriv	8
Extract.spray	10
homog	11
oom	12
Ops.spray	13
pmax	15

print.spray	16
rspray	18
spray	19
spray_cpp	20
subs	22
zero	23

Index	24
--------------	-----------

spray-package	<i>Sparse arrays and multivariate polynomials</i>
---------------	---------------------------------------------------

Description

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

Details

Sparse arrays are a dictionary with keys being (unsigned) integer vectors, and values floats.

Author(s)

Robin K. S. Hankin

See Also

[multipol](#)

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

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

as.array	<i>Coerce spray objects to arrays</i>
----------	---------------------------------------

Description

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

Usage

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

Arguments

<code>x</code>	spray object
<code>offbyone</code>	Boolean with default <code>FALSE</code> meaning to interpret the index entries as positions in their dimension, and <code>TRUE</code> meaning to add one to index values so that zero entries appear in the first place
<code>compact</code>	Boolean with default <code>FALSE</code> meaning to translate the spray as is, and <code>TRUE</code> meaning to add constants to each column of the index matrix so that the resulting array is as small as possible
<code>...</code>	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] <- value(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)
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

```
## 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

```

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

```
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

```
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

```
## 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 $\partial^n S / \partial x_i^n = S^{(x_i, \dots, x_i)}$.

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.

Examples

```

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 is sparse array form
a

S1 <- spray(diag(5),1:5)
S2 <- spray(1-diag(5),1:5)
S1[] <- 3
S1[] <- S2

```

homog

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 `multicol` 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
```

oom

One-over-one-minus for spray objects

Description

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

Usage

```
oom(S, n)
```

Arguments

`S` object of class `spray`
`n` 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

```
a <- homog(4,2)
jj <- (1-a)*oom(a,3)

constant(jj) # should be 1
rowSums(index(jj)) # 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)
```

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.

Note

Notes here

Author(s)

Robin K. S. Hankin

See Also

[oom](#)

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(value(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

<code>x, S</code>	spray object
<code>...</code>	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 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.

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, vals = 1, arity = 3, powers = 0:2)
```

Arguments

<code>n</code>	Number of distinct rows (maximum); repeated rows are merged (argument <code>addrepeats</code> is TRUE)
<code>vals</code>	Values to use for coefficients
<code>arity</code>	Arity of the spray; the number of columns in the index matrix
<code>powers</code>	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(10)
rspray(4)*rspray(3,rnorm(3))
rspray(3,arity=7,powers=-2:2)^3
rspray(1000,vals=rnorm(1000))
```

spray	<i>Create sparse array objects</i>
-------	------------------------------------

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 <code>value<-()</code> , a scalar so that <code>value(S) <- x</code> works as expected
offbyone	In function <code>as.spray()</code> , when converting from an array. Argument <code>offbyone</code> 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 <code>spraymaker()</code> , integer specifying the arity (number of columns of the index matrix <code>L[[1]]</code>); 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.

Note

Idiom such as `spray(S,x)` is not defined if `length(x)>1` because the elements of a spray are not stored in any specific order.

Author(s)

Robin K. S. Hankin

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)
```

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_crush(M, d)
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_crush()
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 <code>spray_power()</code>

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

Author(s)

Robin K. S. Hankin

See Also

[spraymaker](#), [spray](#)

subs *Substitute values into a spray object*

Description

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

Usage

```
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

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

zero

The zero polynomial

Description

Test for the zero, or empty, polynomial

Usage

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

Arguments

L	A two-element list of indices and values, possibly a spray object
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).

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

- *Topic **datasets**
 - zero, [23](#)
- *Topic **mathsymbol**
 - deriv, [8](#)
- *Topic **package**
 - spray-package, [2](#)
- *Topic **symbolmath**
 - arity, [3](#)
 - as.array, [4](#)
 - as.function.spray, [5](#)
 - asum, [6](#)
 - constant, [7](#)
 - Extract.spray, [10](#)
 - homog, [11](#)
 - oom, [12](#)
 - Ops.spray, [13](#)
 - print.spray, [16](#)
 - rspray, [18](#)
 - spray, [19](#)
 - spray_cpp, [20](#)
 - subs, [22](#)
- [.spray (Extract.spray), [10](#)
- [<-.spray (Extract.spray), [10](#)

- aderiv (deriv), [8](#)
- arity, [3](#)
- as.array, [4](#)
- as.function.spray, [5](#)
- as.spray (spray), [19](#)
- asum, [6](#), [9](#)
- asum_inverted (asum), [6](#)

- constant, [7](#), [12](#)
- constant<- (constant), [7](#)

- deriv, [8](#)
- dim.spray (as.array), [4](#)

- empty (zero), [23](#)
- Extract, [8](#)

- Extract.spray, [10](#)

- homog, [11](#)

- index (spray), [19](#)
- is.empty (zero), [23](#)
- is.spray (spray), [19](#)
- is.zero (zero), [23](#)
- is_valid_spray (spray), [19](#)

- linear (homog), [11](#)
- lone (homog), [11](#)

- maxpair_spray (pmax), [15](#)
- minpair_spray (pmax), [15](#)
- multipol, [2](#)

- one (homog), [11](#)
- oom, [12](#), [14](#)
- Ops (Ops.spray), [13](#)
- Ops.spray, [13](#)

- pmax, [15](#)
- pmin (pmax), [15](#)
- print.spray, [16](#)
- print_spray_matrixform (print.spray), [16](#)
- print_spray_polyform (print.spray), [16](#)
- process_dimensions, [22](#)
- process_dimensions (asum), [6](#)
- product (homog), [11](#)

- rspray, [18](#)

- spray, [18](#), [19](#), [21](#)
- spray-package, [2](#)
- spray_accessor (spray_cpp), [20](#)
- spray_add (spray_cpp), [20](#)
- spray_asum_exclude (spray_cpp), [20](#)
- spray_asum_include (spray_cpp), [20](#)
- spray_cpp, [20](#)
- spray_crush (spray_cpp), [20](#)

spray_deriv (spray_cpp), 20
spray_eq_spray (Ops.spray), 13
spray_equality (spray_cpp), 20
spray_maker (spray_cpp), 20
spray_missing_accessor (spray_cpp), 20
spray_mult (spray_cpp), 20
spray_negative (Ops.spray), 13
spray_overwrite (spray_cpp), 20
spray_plus_scalar (Ops.spray), 13
spray_plus_spray (Ops.spray), 13
spray_pmax (spray_cpp), 20
spray_pmin (spray_cpp), 20
spray_power (spray_cpp), 20
spray_power_scalar (Ops.spray), 13
spray_rcpp (spray_cpp), 20
spray_setter (spray_cpp), 20
spray_spray_accessor (spray_cpp), 20
spray_spray_add (spray_cpp), 20
spray_spray_asum_exclude (spray_cpp), 20
spray_spray_asum_include (spray_cpp), 20
spray_spray_crush (spray_cpp), 20
spray_spray_deriv (spray_cpp), 20
spray_spray_equality (spray_cpp), 20
spray_spray_maker (spray_cpp), 20
spray_spray_mult (spray_cpp), 20
spray_spray_overwrite (spray_cpp), 20
spray_spray_pmax (spray_cpp), 20
spray_spray_pmin (spray_cpp), 20
spray_spray_power (spray_cpp), 20
spray_spray_setter (spray_cpp), 20
spray_times_scalar (Ops.spray), 13
spray_times_spray (Ops.spray), 13
spraymaker, 21
spraymaker (spray), 19
subs, 22
substitute (subs), 22

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

xyz (homog), 11

zero, 12, 23