

# Package: frab (via r-universe)

September 7, 2024

**Type** Package

**Title** How to Add Two R Tables

**Version** 0.0-6

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

**Description** Methods to ``add'' two R tables; also an alternative interpretation of named vectors as generalized R tables, so that  $c(a=1,b=2,c=3) + c(b=3,a=-1)$  will return  $c(b=5,c=3)$ . Uses 'disordR' discipline (Hankin, 2022, <doi:10.48550/arXiv.2210.03856>). Extraction and replacement methods are provided. The underlying mathematical structure is the Free Abelian group, hence the name. To cite in publications please use Hankin (2023) <doi:10.48550/arXiv.2307.13184>.

**License** GPL (>= 2)

**Depends** R (>= 3.5.0)

**Suggests** knitr, markdown, rmarkdown, testthat, mvtnorm, covr

**VignetteBuilder** knitr

**Imports** Rcpp (>= 1.0-7), disordR (>= 0.9-8-2), methods

**LinkingTo** Rcpp

**URL** <https://github.com/RobinHankin/frab>

**BugReports** <https://github.com/RobinHankin/frab>

**Repository** <https://robinhankin.r-universe.dev>

**RemoteUrl** <https://github.com/robinhankin/frab>

**RemoteRef** HEAD

**RemoteSha** 4332faadf61703aac36fe06613bc3b908544f1b8

## Contents

frab-package . . . . .	2
Arith . . . . .	3

Compare-methods . . . . .	4
dataframe . . . . .	6
Extract . . . . .	6
frab . . . . .	8
frab-class . . . . .	9
misc . . . . .	10
namedvector . . . . .	11
pmax . . . . .	12
print . . . . .	13
rfrab . . . . .	14
sparsetable . . . . .	15
Summary-methods . . . . .	17
table . . . . .	17
zero . . . . .	19

<b>Index</b>	<b>20</b>
--------------	-----------

---

frab-package	<i>How to Add Two R Tables</i>
--------------	--------------------------------

---

## Description

Methods to "add" two R tables; also an alternative interpretation of named vectors as generalized R tables, so that  $c(a=1,b=2,c=3) + c(b=3,a=-1)$  will return  $c(b=5,c=3)$ . Uses 'disordR' discipline (Hankin, 2022, <doi:10.48550/arXiv.2210.03856>). Extraction and replacement methods are provided. The underlying mathematical structure is the Free Abelian group, hence the name. To cite in publications please use Hankin (2023) <doi:10.48550/arXiv.2307.13184>.

## Details

The DESCRIPTION file: This package was not yet installed at build time.

Index: This package was not yet installed at build time.

## Author(s)

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

Maintainer: Robin K. S. Hankin <[hankin.robin@gmail.com](mailto:hankin.robin@gmail.com)>

## Examples

```
x <- frab(c(a=1, b=2, c=5))
```

```
y <- frab(c(b=-2, c=1, d=8))
```

```
x+y
```

## Description

The frab class provides basic arithmetic methods for frab objects. Arithmetic operations are generally dispatched to one of self-describing functions in the following list:

- frab\_negative()
- frab\_reciprocal()
- frab\_plus\_frab()
- frab\_multiply\_frab()
- frab\_plus\_numeric()
- frab\_multiply\_numeric()
- frab\_power\_numeric()
- numeric\_power\_frab()

The most important one is, of course, frab\_plus\_frab() which is the *sine qua non* for the whole package. But these functions are not intended for user and are somewhat unfriendly. Use the arithmetic operators, as in "a + 2\*b" instead.

Low-level helper functions c\_frab\_add() and c\_frab\_multiply() etc. are generated by compileAttributes(). They call the C routines in the src directory. Low-level helper function c\_frab\_pmax() is documented here for consistency; but technically c\_frab\_pmax() is an "Extremes" function. They are documented at [Compare](#) and [pmax](#) respectively.

## Usage

```
frab_negative(x)
frab_reciprocal(x)
frab_plus_frab(F1,F2)
frab_multiply_numeric(e1,e2)
frab_power_numeric(e1,e2)
numeric_power_frab(e1,e2)
frab_unary(e1,e2)
frab_arith_frab(e1,e2)
frab_plus_numeric(e1,e2)
frab_arith_numeric(e1,e2)
numeric_arith_frab(e1,e2)
```

## Arguments

e1, e2, x, F1, F2    Objects of class frab, coerced if needed

## Value

Return frab objects

**Methods**

**Arith** signature(e1="frab" , e2="missing"): blah blah blah

**Arith** signature(e1="frab" , e2="frab" ): ...

**Arith** signature(e1="frab" , e2="numeric"): ...

**Arith** signature(e1="numeric" , e2="frab" ): ...

**Arith** signature(e1="ANY" , e2="frab" ): ...

**Arith** signature(e1="frab" , e2="ANY" ): ...

**Note**

There are a few peculiarities in the methods. Function `frab_plus_numeric(e1, e2)` assumes `e1` is a frab and `e2` is numeric. But if `e2` is a named vector, it is coerced to a frab; if not, a [simulated] **disordR** violation is raised.

**Author(s)**

Robin K. S. Hankin

**See Also**

[Compare](#)

**Examples**

```
(x <- frab(c(a=1, b=2, c=3)))
(y <- frab(c(b=-2, d=8, x=1, y=7)))
(z <- frab(c(c=2, x=5, b=1, a=6)))
```

```
x+y
x+y+z
```

```
x*y
```

---

Compare-methods

*Comparison methods*

---

**Description**

Methods for comparison (greater than, etc) in the **frab** package.

Frabs and sparsatables may be compared with length-one numeric vectors. Functions `frab_gt_num()` etc follow a consistent naming convention; the mnemonic is the old Fortran `.GT.` scheme [for “greater than”]. This allows one to use idiom such as `f >= 3`. For sparsatables, comparison with scalars is possible: but the result is flattened to a `disord` object (this can be confusing for two

dimensional tables when the default matrix-like print method is used, because zero entries are not “real”. For example, if `s` is a `sparsetable`, then `s==0` will return all `FALSE`).

Comparing a `frab` with another `frab` is generally meaningless. Idiom like “`e1 >= e2`”, for example, returns an error. The only comparison that makes any sense is whether two `frabs` are identical: this is detected by “`e1 == e2`” and its negation “`e1 != e2`”. Internally, equality is tested in `C` using a routine written for speed (specifically, returning `FALSE` as soon as it spots a difference between its two arguments); this is modelled on its equivalent in the **spray** package. If any value is `NA`, equality checks will return `FALSE`. Functions `frab_eq()` and `c_frab_eq()` are just `R` wrappers for the `C` routine `equal()`.

### Usage

```
frab_eq(e1, e2)
frab_eq_num(e1, e2)
frab_ne_num(e1, e2)
frab_gt_num(e1, e2)
frab_ge_num(e1, e2)
frab_lt_num(e1, e2)
frab_le_num(e1, e2)
num_eq_frab(e1, e2)
num_ne_frab(e1, e2)
num_gt_frab(e1, e2)
num_ge_frab(e1, e2)
num_lt_frab(e1, e2)
num_le_frab(e1, e2)
numeric_compare_frab(e1, e2)
frab_compare_frab(e1, e2)
frab_compare_numeric(e1, e2)
```

### Arguments

`e1, e2`            Objects of class `frab`

### Value

Generally, return a `frab` or a logical

### Author(s)

Robin K. S. Hankin

### See Also

[Arith](#)

### Examples

```
rfrab()
a <- rfrab(26, sym=letters)
a[a<4] <- 100
```

---

dataframe	<i>Coerce a data frame to a frab</i>
-----------	--------------------------------------

---

**Description**

Coerce a data frame to a frab

**Usage**

```
df_to_frab(from)
## S4 method for signature 'data.frame'
as.frab(x)
```

**Arguments**

x, from            Frab objects

**Details**

Coerces a data frame, with columns key and value, to the appropriate frab object. Repeated keys are summed.

**Value**

Returns a frab object or a dataframe.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
as.frab(data.frame(key=letters[1:5], value=1:5))
```

---

Extract	<i>Extraction and replacement methods for class "frab"</i>
---------	--

---

**Description**

The frab class provides basic arithmetic and extract/replace methods for frab objects.

Class *index* is taken from the excellent **Matrix** package and is a `setClassUnion()` of classes `numeric`, `logical`, and `character`.

**Value**

Generally, return a frab object.

**Methods**

```
[ signature(x = "frab", i = "character", j = "missing"): x["a"] <- 33
[ signature(x = "frab", i = "disord", j = "missing"): x[x>3]
[ signature(x = "frab", i = "missing", j = "missing"): x[]
[<- signature(x = "frab", i = "character", j = "missing", value = "ANY"): x["a"] <- 3
[<- signature(x = "frab", i = "disord", j = "missing", value="frab"): x[x<0] <- -x[x<0];
  not implemented
[<- signature(x = "frab", i = "disord", j = "missing", value="logical"): x[x<0] <- NA
[<- signature(x = "frab", i = "ANY", j = "ANY", value = "ANY"): not implemented
[<- signature(x = "frab", i = "disindex", j = "missing", value = "numeric"): x[x>0] <- 3
[<- signature(x = "frab", i = "character", j = "missing", value = "logical"): x["c"] <-
  NA
```

Double square extraction, as in `x[[i]]` and `x[[i]] <- value`, is not currently defined. In replacement methods, if `value` is logical it is coerced to numeric (this includes NA).

Special dispensation is given for extraction of a frab with a length zero index, as in `x[NULL]`, which returns the empty frab object.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
frab(setNames(seq_len(0), letters[seq_len(0)]))

a <- rfrab(26, sym=letters)
a<4
a[a<4]
a[a<4] <- 100
a

x <- rfrab()
values(x) <- values(x) + 66

x <- rfrabb()
v <- values(x)
v[v<0] <- abs(v[v<0]) + 50
values(x) <- v

names(x) <- toupper(names(x))
x
```

---

frab

*Creating frab objects*

---

## Description

Package idiom for creating frab objects

## Usage

```
frab(x)
as.frab(x)
is.frab(x)
list_to_frab(L)
```

## Arguments

x	object coerced to, or tested for, frab
L	List of two elements, a numeric vector named <code>values</code> and a character vector named <code>names</code>

## Details

Function `frab()` is the creation method, taking a named numeric vector as its argument; it is the only function in the package that actually calls `new("frab", ...)`.

Function `as.frab()` tries a bit harder to be useful and can coerce different types of object to a frab. If given a list it dispatches to `list_to_frab()`. If given a table it dispatches to `table_to_frab()`, documented at `table.Rd`; and if given a data frame it dispatches to `df_to_frab()`, documented at `dataframe.Rd`.

## Value

Returns a frab, or a boolean

## Author(s)

Robin K. S. Hankin

## See Also

[frab-class](#)



**Examples**

```

frab(c(x=6,y=6,z=-4,u=0,x=3))

as.frab(c(a=2,b=1,c=77))

as.frab(list(names=letters[5:2],values=1:4))

x <- rfrab()
y <- rfrab()
x+y

```

---

frab-class	Class "frab"
------------	--------------

---

**Description**

The formal S4 class for frab objects

**Usage**

```

## S4 method for signature 'frab'
namedvector(x)

```

**Arguments**

x                    Object of class frab

**Objects from the Class**

Formal class *frab* has a single slot *x* which is a named numeric vector.

The class has three accessor methods: `names()`, `values()`, and `namedvector()`.

**Author(s)**

Robin K. S. Hankin

**Examples**

```

new("frab",x=c(a=6,b=4,c=1)) # formal creation method (discouraged)

frab(c(a=4,b=1,c=5)) # use frab() in day-to-day work

frab(c(a=4,b=0,c=5)) # zero entries are discarded

frab(c(a=4,b=3,b=5)) # repeted entries are summed

frab(c(apple=4,orange=3,cherry=5)) # any names are OK

```

```

x <- frab(c(d=1,y=3,a=2,b=5,rug=7,c=2))
(y <- rfrab())

x+y          # addition works as expected
x + 2*y     # arithmetic
x>2         # extraction
x[x>3] <- 99 # replacement

# sum(x)      # some summary methods implemented
# max(x)

```

---

misc

*Miscellaneous functions*


---

## Description

This page documents various functions that work for frabs, and I will add to these from time to time as I add new functions that make sense for frab objects. To use functions like `sin()` and `abs()` on frab object `x`, work with `values(x)` (which is a `disord` object). However, there are a few functions that are a little more involved:

- `length()` returns the length of the data component of the object.
- `which()` returns an error when called with a frab object, but is useful here because it returns a `disind` when given a Boolean `disord` object. This is useful for idiom such as `x[x>0]`
- Functions `is.na()` and `is.notna()` return a `disind` object

## Usage

```

## S4 method for signature 'frab'
length(x)

```

## Arguments

```

x          Object of class frab

```

## Value

Generally return frabs

## Note

Constructions such as `!is.na(x)` do not work if `x` is a frab object: this is because `is.na()` returns a `disind` object, not a logical. Use `is.notna()` to identify elements that are not NA.

**Author(s)**

Robin K. S. Hankin

**See Also**[extract](#)**Examples**

```
(a <- frab(c(a=1,b=NA,c=44,x=NA,h=4)))  
is.na(a)
```

```
(x <- frab(c(x=5,y=2,z=3,a=7,b=6)))  
which(x>3)  
x[which(x>3)]  
x[which(x>3)] <- 4  
x
```

```
is.na(x) <- x<3  
x  
x[is.na(x)] <- 100  
x
```

```
y <- frab(c(a=5,b=NA,c=3,d=NA))  
y[is.notna(y)] <- 199  
y
```

---

namedvector

*Named vectors and the frab package*

---

**Description**

Named vectors are closely related to frab objects, but are not the same. However, there is a natural coercion from one to the other.

**Usage**

```
as.namedvector(v)  
is.namedvector(v)  
is.namedlogical(v)  
is.unnamedlogical(v)  
is.unnamedvector(v)
```

**Arguments**

v                   Argument to be tested or coerced

**Details**

Coercion and testing for named vectors. Function `nv_to_frab()`, documented at `frab.Rd`, coerces a named vector to a `frab`.

**Value**

Function `is.namedvector()` returns a boolean, function `as.namedvector()` returns a named vector.

**Note**

The issue of named logical vectors in the ‘`frab`’ package is discussed briefly at ‘`inst/wittgenstein.Rmd`’.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
x <- c(a=5, b=3, c=-2, b=-3, x=33)
is.namedvector(x)
as.namedvector(frab(x))

x <- c(a=5, b=3, c=-2)
y <- c(p=1, c=2, d= 6)

x
y
x+y

frab(x) + frab(y)
```

---

pmax

*Parallel maxima and minima for frabs*

---

**Description**

Parallel (pairwise) maxima and minima for `frabs`.

**Usage**

```
pmax_pair(F1,F2)
pmin_pair(F1,F2)
pmax_dots(x, ...)
pmin_dots(x, ...)
## S4 method for signature 'frab'
pmax(...)
## S4 method for signature 'frab'
pmin(...)
```

**Arguments**

F1, F2, x, ... Frab objects

**Details**

Pairwise minima and maxima for frabs, using names as the primary key.

Functions `pmax_pair()` calls `c_frab_pmax()` and `pmin_pair()` use

Functions `pmax()` and `pmin()` use the same mechanism as `cbrob()` of the **Brobdingnag** package, originally due to John Chambers (pers. comm.)

**Value**

Returns a frab object

**Author(s)**

Robin K. S. Hankin

**Examples**

```
x <- rfrab()
y <- rfrab()
```

---

print

*Methods for printing frabs*

---

**Description**

Methods for printing frabs nicely

**Usage**

```
## S4 method for signature 'frab'
show(object)
frab_print(object)
```

**Arguments**

object An object of class frab

**Details**

The method is sensitive to option `frab_print_hash`. If TRUE, the hash code is printed; otherwise it is not.

Function `frab_print()` returns its argument, invisibly.

There is special dispensation for the empty frab object.

**Value**

Returns its argument, invisibly

**Author(s)**

Robin K. S. Hankin

**Examples**

```
print(rfrab()) # default

options(frab_print_hash = TRUE)
print(rfrab()) # prints hash code

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

---

rfrab

*Random frabs*


---

**Description**

Random frab objects, intended as quick “get you going” examples

**Usage**

```
rfrab(n = 9, v = seq_len(5), symb = letters[seq_len(9)])
rfrabb(n = 100, v = -5:5, symb = letters)
rfrabbb(n = 5000, v = -10:10, symb = letters, i=3)
```

**Arguments**

n	Length of object to return
v	Values to assign to symbols (see details)
symb	Symbols to use
i	Exponentiating index for rfrabbb(). Symbols in returned value will be i concatenated elements of symb

**Details**

What you see is what you get, basically. If a symbol is chosen more than once, as in, `c(a=1, b=2, a=3)`, then the value for a will be summed.

Use function `rfrab()` for a small, easily-managed object; `rfrabb()` and `rfrabbb()` give successively larger objects.

**Value**

Returns a frab object

**Author(s)**

Robin K. S. Hankin

**Examples**

```
rfrab()
```

---

sparsetable

*Generalized sparse tables: sparsetable objects*

---

**Description**

Package idiom for creating and manipulating sparsetable objects

**Usage**

```
sparsetable(i, v=1)
rspar(n=15, l=3, d=3)
rspar2(n=15, l=6)
rsparr(n=20, d=6, l=5, s=4)
sparsetable_to_array(x)
array_to_sparsetable(x)
sparsetable_to_frab(x)
## S4 method for signature 'sparsetable'
index(x)
## S4 method for signature 'sparsetable'
values(x)
## S4 method for signature 'sparsetable'
dimnames(x)
## S4 method for signature 'sparsetable'
dim(x)
```

**Arguments**

<code>x</code>	In functions like <code>index()</code> , an object of class <code>sparsetable</code>
<code>i, v</code>	In standard constructor function <code>sparsetable()</code> , argument <code>i</code> is the index matrix of strings, and <code>v</code> a numeric vector of values
<code>n, l, d, s</code>	In functions <code>rspar()</code> , <code>rspar2()</code> , and <code>rsparr()</code> , <code>n</code> is the number of terms, <code>l</code> the number of letters, <code>d</code> the dimensionality and <code>s</code> the number of distinct marginal values to return

## Details

Most functions here mirror their equivalent in the **spray** package [which the C code is largely copied from] or the frab functionality. So, for example, `num_eq_sparsetable()` is the equivalent of `num_eq_spray()`.

The print method treats arity-2 sparsetable objects differently from other arities. By default, arity-2 sparsetable objects are displayed as two-dimensional tables. Control this behaviour with option `print_2dsparsetables_as_matrices`:

```
options("print_2dsparsetables_as_matrices" = FALSE)
```

The default value for this option, non-`FALSE` (including its out-of-the-box status of “unset”), directs the print method to coerce arity-2 sparsetable objects to two-dimensional tables before printing. If this option is `FALSE`, arity-2 sparsetables are printed using matrix index form, just the same as any other arity.

Functions `rspar()`, `rspar2()`, and `rsparr()` create random sparsetable objects of increasing complexity. The defaults are chosen so that the returned frabs are of sensible sizes.

Function `drop()` takes a sparsetable object of arity one and coerces to a frab object.

Function `dim()` returns a named vector, with names being the `dimnames` of its argument.

Extraction and replacement methods are a subset of **spray** methods, but most should work. There is special dispensation so that standard idiom for arrays [e.g. `x[ 'a' , 'b' , 'a' ]` and `x[ 'a' , 'b' , 'a' ] <- 55`] should work as expected, although the general expectation is that access and replacement use (character) matrices and an index object. However, indexing by `disord` and `disindex` objects should also work [e.g. `x[x>7]`].

The **spray** source code and the sparsetable functionality have about 90% overlap; there were enough small differences between the codes to make it worth maintaining two sets of source code, IMO.

There is a discussion of package idiom in the vignette, `vignette("frab")`.

## Note

The pronunciation of “sparsetable” has the emphasis on the first syllable, so it rhymes with [British river-port town] “Barnstaple”.

## Author(s)

Robin K. S. Hankin

## See Also

[frab-class](#)

## Examples

```
sparsetable(matrix(sample(letters[1:4], 36, replace=TRUE), ncol=2), 1:18)
sparsetable(matrix(sample(letters[1:4], 39, replace=TRUE), ncol=3), 1:13)
```

```
(x <- rspar2(9))
```



```
(y <- rspar2(9))
x + y

x["KT", "FF"] <- 100
x

rsparr()

a <- rspar(d=4)
asum(a, "Feb")
```

---

Summary-methods

*Methods for Function Summary*

---

### Description

Methods for S4 function Summary in the **frab** package. Currently, only `max()`, `min()`, `range()` and `sum()` are defined, and these operate in the natural way on the elements of a frab. Note that these functions are not susceptible to **disordR** violations.

### Methods

`signature(x = "frab")` Dispatches to `max(values(x))` etc.

### Examples

```
a <- rfrab()
a
max(a)
min(a)
range(a)
```

---

table

*Tables and frab objects*

---

### Description

Various methods and functions to deal with tables in the **frab** package.

### Usage

```
## S4 method for signature 'frab'
as.table(x, ...)
table_to_frab(x)
```

**Arguments**

x                    Object of class frab or table  
 ...                  Further arguments, currently ignored

**Details**

If a frab object has non-negative entries it may be interpreted as a table. However, in base R, table objects do not have sensible addition methods which is why the **frab** package is needed.

Function `is.1dtable()` checks for its argument being a one-dimensional table. The idea is that a table like `table(sample(letters,30,TRUE))`, being a table of a single observation, is accepted but a table like `table(data.frame(rnorm(20)>0,rnorm(20)>0))` is not acceptable because it is a *two*-dimensional contingency table.

**Value**

Generally return a table or frab.

**Note**

The order of the entries may be changed during the coercion, as per **disordR** discipline. Function `as.frab()` takes a table, dispatching to `table_to_frab()`.

**Author(s)**

Robin K. S. Hankin

**Examples**

```
X <- table(letters[c(1,1,1,1,2,3,3)])
Y <- table(letters[c(1,1,1,1,3,4,4)])
Z <- table(letters[c(1,1,2,3,4,5,5)])

X+Y # defined but nonsense

# X+Z # returns an error

as.frab(X) + as.frab(Y) # correct answer

plot(as.table(rfrab()))
```

---

zero

*The zero frab object*

---

### Description

Test for a frab object's being zero (empty).

### Usage

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

### Arguments

x	Object of class frab
...	Further arguments (currently ignored)

### Details

Function `zero()` returns the empty frab object; this is the additive identity 0 with property  $x + 0 = 0 + x = x$ .

Function `is.zero()` returns TRUE if its argument is indeed the zero object.

Function `is.empty()` is a synonym for `is.zero()`. Sometimes one is thinking about the free Abelian group, in which case `is.zero()` makes more sense, and sometimes one is thinking about maps and tables, in which case `is.empty()` is more appropriate.

### Value

Function `zero()` returns the zero frab object, function `is.zero()` a Boolean

### Author(s)

Robin K. S. Hankin

### Examples

```
zero()  
zero() + zero()  
  
x <- rfrab()  
  
x+zero() == x  
  
is.zero(zero())
```

# Index

- !, frab-method (misc), 10
- \* **classes**
  - frab-class, 9
- \* **math**
  - Compare-methods, 4
  - print, 13
- \* **methods**
  - Compare-methods, 4
  - Summary-methods, 17
- \* **package**
  - frab-package, 2
- \* **symbolmath**
  - zero, 19
- [ (Extract), 6
- [, ANY, frab, ANY-method (Extract), 6
- [, frab, ANY, ANY-method (Extract), 6
- [, frab, ANY, missing, ANY-method (Extract), 6
- [, frab, character, missing, ANY-method (Extract), 6
- [, frab, character, missing-method (Extract), 6
- [, frab, disindex, missing, ANY-method (Extract), 6
- [, frab, disord, missing, ANY-method (Extract), 6
- [, frab, disord, missing-method (Extract), 6
- [, frab, frab, missing, ANY-method (Extract), 6
- [, frab, frab, missing-method (Extract), 6
- [, frab, missing, index-method (Extract), 6
- [, frab, missing, missing, ANY-method (Extract), 6
- [, frab, missing, missing-method (Extract), 6
- [, frab-method (Extract), 6
- [, sparsetable, ANY, ANY, ANY-method (sparsetable), 15
- [, sparsetable, disindex, missing, ANY-method (sparsetable), 15
- [, sparsetable, disord, missing, ANY-method (sparsetable), 15
- [, sparsetable, disord, missing-method (sparsetable), 15
- [, sparsetable-method (sparsetable), 15
- [. frab (Extract), 6
- [<- (Extract), 6
- [<-, frab, ANY, ANY, ANY-method (Extract), 6
- [<-, frab, character, missing, logical-method (Extract), 6
- [<-, frab, character, missing, numeric-method (Extract), 6
- [<-, frab, character, missing-method (Extract), 6
- [<-, frab, disindex, missing, numeric, ANY-method (Extract), 6
- [<-, frab, disindex, missing, numeric-method (Extract), 6
- [<-, frab, disord, missing, frab-method (Extract), 6
- [<-, frab, disord, missing, logical-method (Extract), 6
- [<-, frab, disord, missing, numeric-method (Extract), 6
- [<-, frab, disord, missing-method (Extract), 6
- [<-, frab, missing, missing, ANY-method (Extract), 6
- [<-, frab, missing, missing, frab-method (Extract), 6
- [<-, frab, missing, missing, numeric-method (Extract), 6
- [<-, sparsetable, ANY, ANY, ANY-method (sparsetable), 15
- [<-, sparsetable, disindex, missing, ANY-method (sparsetable), 15
- [<-, sparsetable, disord, missing, numeric-method

- (sparsetable), 15
- [<- , sparsetable-method (sparsetable), 15
- Arith, 3, 5
- arith (Arith), 3
- Arith, ANY, frab-method (Arith), 3
- Arith, frab, ANY-method (Arith), 3
- Arith, frab, frab-method (Arith), 3
- Arith, frab, missing-method (Arith), 3
- Arith, frab, numeric-method (Arith), 3
- arity (sparsetable), 15
- arity, sparsetable-method (sparsetable), 15
- array\_to\_sparsetable (sparsetable), 15
- as.array, sparsetable-method (sparsetable), 15
- as.data.frame (dataframe), 6
- as.data.frame, frab-method (dataframe), 6
- as.frab (frab), 8
- as.frab, data.frame-method (dataframe), 6
- as.namedvector (namedvector), 11
- as.sparsetable (sparsetable), 15
- as.table (table), 17
- as.table, frab-method (table), 17
- asum (sparsetable), 15
- asum, sparsetable-method (sparsetable), 15
- asum.sparsetable (sparsetable), 15
- asum\_exclude\_sparsetable (sparsetable), 15
- asum\_sparsetable (sparsetable), 15
- c\_frab\_add (Arith), 3
- c\_frab\_eq (Compare-methods), 4
- c\_frab\_identity (Arith), 3
- c\_frab\_multiply (Arith), 3
- c\_frab\_pmax (Arith), 3
- Compare, 3, 4
- Compare (Compare-methods), 4
- Compare, frab, frab-method (Compare-methods), 4
- Compare, frab, numeric-method (Compare-methods), 4
- Compare, numeric, frab-method (Compare-methods), 4
- Compare-methods, 4
- data.frame (dataframe), 6
- dataframe, 6
- df\_to\_frab (dataframe), 6
- dim (sparsetable), 15
- dim, sparsetable-method (sparsetable), 15
- dimnames, sparsetable-method (sparsetable), 15
- dimnames<- , sparsetable, ANY-method (sparsetable), 15
- dimnames<- , sparsetable-method (sparsetable), 15
- drop (sparsetable), 15
- drop, sparsetable-method (sparsetable), 15
- empty (zero), 19
- Extract, 6
- extract, 11
- extract (Extract), 6
- frab, 8
- frab-class, 9
- frab-package, 2
- frab\_arith\_frab (Arith), 3
- frab\_arith\_numeric (Arith), 3
- frab\_compare\_frab (Compare-methods), 4
- frab\_compare\_numeric (Compare-methods), 4
- frab\_eq (Compare-methods), 4
- frab\_eq\_num (Compare-methods), 4
- frab\_ge\_num (Compare-methods), 4
- frab\_gt\_num (Compare-methods), 4
- frab\_le\_num (Compare-methods), 4
- frab\_lt\_num (Compare-methods), 4
- frab\_multiply\_frab (Arith), 3
- frab\_multiply\_numeric (Arith), 3
- frab\_ne\_num (Compare-methods), 4
- frab\_negative (Arith), 3
- frab\_plus\_frab (Arith), 3
- frab\_plus\_numeric (Arith), 3
- frab\_power\_numeric (Arith), 3
- frab\_print (print), 13
- frab\_reciprocal (Arith), 3
- frab\_unary (Arith), 3
- index (sparsetable), 15
- index, sparsetable-method (sparsetable), 15
- is.1dtable (table), 17
- is.empty (zero), 19

- is.empty, sparsetable-method (sparsetable), 15
- is.frab (frab), 8
- is.na (misc), 10
- is.na, frab-method (misc), 10
- is.na.frab (misc), 10
- is.na<- (misc), 10
- is.na<-, frab, disord-method (misc), 10
- is.na<-.frab (misc), 10
- is.namedlogical (namedvector), 11
- is.namedvector (namedvector), 11
- is.notna (misc), 10
- is.notna, frab-method (misc), 10
- is.notna.frab (misc), 10
- is.sparsetable (sparsetable), 15
- is.unnamedlogical (namedvector), 11
- is.unnamedvector (namedvector), 11
- is.zero (zero), 19
  
- lapply (misc), 10
- lapply, disord-method (misc), 10
- lapply.disord (misc), 10
- length (misc), 10
- length, frab-method (misc), 10
- length.frab (misc), 10
- list\_to\_frab (frab), 8
  
- misc, 10
  
- namedvector, 11
- namedvector (frab-class), 9
- namedvector, frab-method (frab-class), 9
- names (Extract), 6
- names, frab-method (Extract), 6
- names, sparsetable-method (sparsetable), 15
- names<-, frab, character-method (Extract), 6
- names<-, frab, disord-method (Extract), 6
- nterms (sparsetable), 15
- nterms, sparsetable-method (sparsetable), 15
- num\_eq\_frab (Compare-methods), 4
- num\_eq\_sparsetable (sparsetable), 15
- num\_ge\_frab (Compare-methods), 4
- num\_ge\_sparsetable (sparsetable), 15
- num\_gt\_frab (Compare-methods), 4
- num\_gt\_sparsetable (sparsetable), 15
- num\_le\_frab (Compare-methods), 4
- num\_le\_sparsetable (sparsetable), 15
- num\_lt\_frab (Compare-methods), 4
- num\_lt\_sparsetable (sparsetable), 15
- num\_ne\_frab (Compare-methods), 4
- numeric\_arith\_frab (Arith), 3
- numeric\_arith\_sparsetable (sparsetable), 15
- numeric\_compare\_frab (Compare-methods), 4
- numeric\_compare\_sparsetable (sparsetable), 15
- numeric\_multiply\_frab (Arith), 3
- numeric\_multiply\_sparsetable (sparsetable), 15
- numeric\_power\_frab (Arith), 3
- numeric\_power\_sparsetable (sparsetable), 15
  
- pmax, 3, 12
- pmax, ANY-method (pmax), 12
- pmax, frab-method (pmax), 12
- pmax, sparsetable-method (sparsetable), 15
- pmax.frab (pmax), 12
- pmax.sparsetable (sparsetable), 15
- pmax\_dots (pmax), 12
- pmax\_dots\_sparsetable (sparsetable), 15
- pmax\_pair (pmax), 12
- pmax\_pair\_sparsetable (sparsetable), 15
- pmax\_sparsetable (sparsetable), 15
- pmin (pmax), 12
- pmin, ANY-method (pmax), 12
- pmin, frab-method (pmax), 12
- pmin, sparsetable-method (sparsetable), 15
- pmin.frab (pmax), 12
- pmin.sparsetable (sparsetable), 15
- pmin\_dots (pmax), 12
- pmin\_dots\_sparsetable (sparsetable), 15
- pmin\_pair (pmax), 12
- pmin\_pair\_sparsetable (sparsetable), 15
- pmin\_sparsetable (sparsetable), 15
- print, 13
- print\_sparsetable\_matrixform (sparsetable), 15
  
- rfrab, 14
- rfrabb (rfrab), 14
- rfrabbb (rfrab), 14

- rspar (sparsetable), 15
- rspar2 (sparsetable), 15
- rsparr (sparsetable), 15
- show, frab-method (print), 13
- show, sparsetable-method (sparsetable), 15
- sparsetable, 15
- sparsetable-class (sparsetable), 15
- sparsetable\_accessor (sparsetable), 15
- sparsetable\_add (sparsetable), 15
- sparsetable\_arith\_numeric (sparsetable), 15
- sparsetable\_arith\_sparsetable (sparsetable), 15
- sparsetable\_asum\_exclude (sparsetable), 15
- sparsetable\_asum\_include (sparsetable), 15
- sparsetable\_compare\_numeric (sparsetable), 15
- sparsetable\_compare\_sparsetable (sparsetable), 15
- sparsetable\_eq (sparsetable), 15
- sparsetable\_eq\_num (sparsetable), 15
- sparsetable\_eq\_sparsetable (sparsetable), 15
- sparsetable\_equality (sparsetable), 15
- sparsetable\_ge\_num (sparsetable), 15
- sparsetable\_gt\_num (sparsetable), 15
- sparsetable\_le\_num (sparsetable), 15
- sparsetable\_lt\_num (sparsetable), 15
- sparsetable\_maker (sparsetable), 15
- sparsetable\_multiply\_numeric (sparsetable), 15
- sparsetable\_multiply\_sparsetable (sparsetable), 15
- sparsetable\_negative (sparsetable), 15
- sparsetable\_overwrite (sparsetable), 15
- sparsetable\_plus\_sparsetable (sparsetable), 15
- sparsetable\_pmax (sparsetable), 15
- sparsetable\_pmin (sparsetable), 15
- sparsetable\_power\_numeric (sparsetable), 15
- sparsetable\_reciprocal (sparsetable), 15
- sparsetable\_setter (sparsetable), 15
- sparsetable\_times\_scalar (sparsetable), 15
- sparsetable\_to\_array (sparsetable), 15
- sparsetable\_to\_frab (sparsetable), 15
- sparsetable\_to\_table (sparsetable), 15
- sparsetable\_unary (sparsetable), 15
- Summary, frab-method (Summary-methods), 17
- Summary-methods, 17
- table, 17
- table\_to\_frab (table), 17
- table\_to\_sparsetable (sparsetable), 15
- values (Extract), 6
- values, frab-method (Extract), 6
- values, sparsetable-method (sparsetable), 15
- values<- (Extract), 6
- values<- , frab, disord-method (Extract), 6
- values<- , frab, numeric-method (Extract), 6
- which, disindex-method (misc), 10
- which, frab-method (misc), 10
- zero, 19