Package ‘frab’

August 16, 2023

Type Package

Title How to Add Two Tables

Version 0.0-3

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

Description Methods to ``add'' two tables; also an alternative
interpretation of named vectors as generalized 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, <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)
<arxiv:2307:13184>.

License GPL (>= 2)

Depends R (>= 3.5.0)

Suggests knitr, markdown, rmarkdown, testthat, mvtnorm

VignetteBuilder knitr

Imports Rcpp (>= 1.0-7), mathjaxr, disordR (>= 0.9-8-1), methods

LinkingTo Rcpp

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

BugReports https://github.com/RobinHankin/frab

RdMacros mathjaxr

R topics documented:

`frab-package` *How to Add Two Tables*

Description

Methods to "add" two tables; also an alternative interpretation of named vectors as generalized 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, <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) <arxiv:2307:13184>.

Details

The DESCRIPTION file:

Package:	frab
Type:	Package
Title:	How to Add Two Tables
Version:	0.0-3
Authors@R:	person(given=c("Robin", "K. S."), family="Hankin", role = c("aut","cre"), email="hankin.robin@gmai
Maintainer:	Robin K. S. Hankin <hankin.robin@gmail.com>
Description:	Methods to "add" two tables; also an alternative interpretation of named vectors as generalized tables,
License:	GPL (>= 2)
Depends:	R (>= 3.5.0)
Suggests:	knitr, markdown, rmarkdown, testthat, mvtnorm
VignetteBuilder:	knitr
Imports:	Rcpp (>= 1.0-7), mathjaxr, disordR (>= 0.9-8-1), methods
LinkingTo:	Rcpp
URL:	https://github.com/RobinHankin/frab
BugReports:	https://github.com/RobinHankin/frab
RdMacros:	mathjaxr
Author:	Robin K. S. Hankin [aut, cre] (<https://orcid.org/0000-0001-5982-0415>)

Index of help topics:

```
Compare-methods         Comparision methods
arith                   Arithmetic methods for class '"frab"'
extract                 Extraction and replacement methods for class
                        '"frab"'
frab                    Creating 'frab' objects
frab-class              Class "frab"
frab-package            How to Add Two Tables
is.namedvector          Named vectors and the frab package
misc                    Miscellaneous functions
pmax                    Parallel maxima and minima for frabs
```

```
print                   Methods for printing frabs
rfrab                   Random frabs
sparsetable             Generalized sparse tables: 'sparsetable'
                        objects
table                   Tables and frab objects
zero                    The zero frab object
```

Author(s)

NA

Maintainer: Robin K. S. Hankin <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
```

Arith — *Arithmetic methods for class* "frab"

Description

The frab class provides basic arithmetic methods for frab objects. Low-level helper functions `c_frab_eq()` amd `c_frab_pmax()` are documented here for consistency; but technically `c_frab_eq()` is a Comparison operator, and `c_frab_pmax()` is an “Extremes” function. They are documented at `Compare.Rd` and `pmax.Rd` 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" )`: ...

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` *Comparision methods*

Description

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

Functions `frab_gt_num()` etc follow a consistent naming convention; the mnemonic is the old Fortran `.GT.` scheme [for “greater than”].

Function `frab_eq()` is an odd-ball, formally documented at `Arith.Rd`. It is slightly different from the other comparisons: it calls low-level helper function `c_frab_eq()`, which calls its C namesake which is written for speed (specifically, returning `FALSE` as soon as it spots a difference between its two arguments). Note that if any value is `NA`, `frab_eq()` will return `FALSE`.

Usage

```
frab_eq(e1,e2)
frab_compare_frab(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)
frab_compare_numeric(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)
```

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
```

`Extract` *Extraction and replacement methods for class* `"frab"`

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`).

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 values and a character vector named names

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.

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

```
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.

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()

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

x	In functions like index(), an object of class sparsetable
i,v	In standard constructor function sparsetable(), argument i is the index matrix of strings, and v a numeric vector of values
n,l,d,s	In functions rspar(), rspar2(), and rsparr(), n is the number of terms, l the number of letters, d the dimensionality and s 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 to make the values 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] 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 sparstable functionality hve 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 "Barnable" or "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")
```

---

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