Package ‘ffbase’

March 23, 2016

Maintainer  Edwin de Jonge <edwindjonge@gmail.com>
License GPL-3
Title  Basic Statistical Functions for Package ‘ff’
Type  Package
LazyLoad  yes
Author  Edwin de Jonge, Jan Wijffels, Jan van der Laan
Description  Extends the out of memory vectors of ‘ff’ with statistical functions and other utilities to ease their usage.
Version  0.12.3
URL  http://github.com/edwindj/ffbase
Date  2015-07-25
Depends  R (>= 2.12.0),ff(>= 2.2-11)
Imports  fastmatch, bit
Suggests  testthat, parallel, LaF, biglm
RoxygenNote  5.0.1
NeedsCompilation  yes
Repository  CRAN
Date/Publication  2016-03-23 00:29:24

R topics documented:

ffbase-package ............................................. 3
+.ff_vector ............................................. 5
>.ff_vector ............................................. 6
abs.ff_vector ......................................... 7
addfforder ........................................... 9
all.ff .................................................. 10
any.ff ............................................... 11
as.character.ff ..................................... 12
as.Date.ff_vector ....................................... 12
R topics documented:

as.ffdf.ffdf .................................................. 13
as.ram.ffdf .................................................. 13
bigglm.ffdf ................................................... 14
binned_sum .................................................... 15
binned_sumsq ................................................ 15
binned_tabulate ............................................. 16
byMean ......................................................... 17
bySum ........................................................ 18
c.ff .......................................................... 18
chunkify ....................................................... 19
compact ....................................................... 19
condMean ...................................................... 20
condSum ....................................................... 20
cumsum.ff ...................................................... 21
cut.ff ........................................................ 22
diff.ff ........................................................ 23
droplevels.ff ............................................... 23
droplevels.ffdf ............................................. 24
duplicated.ff ............................................... 24
expand.ffgrid ............................................... 26
ffappend ...................................................... 27
ffdfappend .................................................. 27
ffdfdply ...................................................... 28
ffdfbind.fill ............................................... 29
ffdfsave .................................................... 30
ffdfwith ..................................................... 31
ffelse ......................................................... 32
ffmatch ...................................................... 33
ffrandom ..................................................... 34
ffrep.int .................................................... 35
ffseq ........................................................ 36
ffseq_len .................................................... 37
ffwhich ....................................................... 38
format.ff_vector ........................................... 39
grouprunningcumsum ....................................... 39
hist.ff ....................................................... 40
ikey .......................................................... 40
is.na.ff ...................................................... 41
laf_to_ffdf .................................................. 42
load.ffdf ..................................................... 43
mean.ff ....................................................... 44
merge.ffdf ................................................... 44
min.ff ........................................................ 47
move.ffdf ..................................................... 48
pack.ffdf .................................................... 49
quantile.ff .................................................. 50
rle_ff ........................................................ 50
save.ffdf ..................................................... 51
ffbase-package

Description

Basic statistical functions for ff vectors and ffdf data.frames. The aim of ffbase is to make working with ff vectors and ffdf data.frame a bit easier.

Basic operations

cut.ff
c.ff
unique
duplicated
ffmatch
ffdfmatch
%in%
is.na.ff
+
-, *, /, ^, %, %/
==, !=, <, <=, >, >, &,
abs, sign, sqrt, ceiling, floor, trunc, round, signif
log, log10, log2, log1p, exp, expm1
acos, acosh, asin, asinh, atan, atanh
cos, cosh, sin, sinh, tan, tanh
gamma, lgamma, digamma, trigamma
cut a ff vector.
concatenate ff vectors.
unique for a ff vector and ffdf.
duplicated for a ff vector and ffdf.
match for a 2 ff vectors.
match for 2 ffdf objects.
%in% operator for a ff vector and ffdf.
is.na for a ff vector.
operators for arithmetic on ff vector.
compare & logic operators for working with ff vectors.
Math operators for working on ff vectors.
Math operators for working on ff vectors.
Math operators for working on ff vectors.
Math operators for working on ff vectors.
Math operators for working on ff vectors.
Math operators for working on ff vectors.

Selections

subset.ffdf subset a ffdf.
transform.ffdf create a new ffdf based on an existing ffdf
with.ffdf create a ff vector based on columns of an existing ffdf
within.ffdf create a ffdf data.frame based on columns of an existing ffdf
ffwhich create a ff integer vector based on a logical expression

Aggregations

hist.ff Calculate a histogram for ff vector.
quantile.ff Get quantiles for ff vector.
sum.ff sum for a ff vector.
mean.ff (trimmed) mean for a ff vector.
all.ff all for logical ff vector.
min.ff min for ff vector.
max.ff max for ff vector.
cumsum.ff cumsum for ff vector.
cumprod.ff cumprod for ff vector.
rang.ff range for ff vector.
table table for ff vectors.
tabulate.ff tabulate for ff vectors.
ffdfdply Split, group and aggregate for ffdf operations.

Miscellaneous

ffordered Add a sorted index to a ff vector.
save.ffdf Save a ffdf in a directory with its containing ff columns.
load.ffdf Loads a ffdf from a directory
pack.ffdf Packs ffdf data.frames into a zip or tar file
unpack.ffdf Unpacks data.frames from a zip or tar file
ffappend Append data to a ff vector.
ffdfdappend Append data to a ffdf.
merge.ffdf Merge two ffdf objects.
ffmatch match two ff vectors
ffdfmatch match two ffdf data.frames
laf_to_ffdf Import csv and fixed width files through package LaF.

Examples
+.ff_vector

ffdat <- as.ffdf(data.frame(x=1:10, y=1:10))

# add a new ff vector z to the ffdf data.frame
within(ffdat, z <- x+y)[]

# add a new ff vector z to the ffdf data.frame using transform
transform(ffdat, z=x+y)[]
cut(ffdat$x, breaks=3)[]
tabulate.ff(ffdat$x)

---

+.ff_vector

Arithmetic Operators for ff vectors

Description

These binary operators perform arithmetic on numeric ff vectors. Arith family:

- Arith: "+", "-", "*", "/", "^", "%%", "/%"

The operators require either x or y to be an ff_vector or both. In case either x or y is not an ff_vector, the other object needs to be of length 1. Recycling is not implemented.

Usage

## S3 method for class 'ff_vector'

x + y

## S3 method for class 'ff_vector'

x - y

## S3 method for class 'ff_vector'

x * y

## S3 method for class 'ff_vector'

x / y

## S3 method for class 'ff_vector'

x ^ y

## S3 method for class 'ff_vector'

x %% y

## S3 method for class 'ff_vector'

x %/% y
Arguments

- **x**: either a numeric `ff_vector` or a vector of length 1 in RAM in which case `y` should be an `ff_vector`.
- **y**: either a numeric `ff_vector` or a vector of length 1 in RAM in which case `x` should be an `ff_vector`.

Value

- An `ff_vector`. For the definition of the operators see the base package of R.

Description

These operators implement `ff_vector` specific operators and handle the following operators from the Ops family:

- **Compare**: `"==", "!=", "<", "<=", ">", ">="`
- **Logic**: `"&", "|", "!"`

The operators require either `x` or `y` to be an `ff_vector` or both. In case either `x` or `y` is not an `ff_vector`, the other object needs to be of length 1. Recycling is not implemented.

Usage

```r
## S3 method for class 'ff_vector'
x > y
```

```r
## S3 method for class 'ff_vector'
x < y
```

```r
## S3 method for class 'ff_vector'
x == y
```

```r
## S3 method for class 'ff_vector'
x != y
```

```r
## S3 method for class 'ff_vector'
x <= y
```

```r
## S3 method for class 'ff_vector'
x >= y
```

```r
## S3 method for class 'ff_vector'
x & y
```
abs.ff_vector

```r
## S3 method for class 'ff_vector'
x | y
## S3 method for class 'ff_vector'
!x
```

### Arguments

- **x**: either a numeric `ff_vector` or a vector of length 1 in RAM in which case `y` should be an `ff_vector`.
- **y**: either a numeric `ff_vector` or a vector of length 1 in RAM in which case `x` should be an `ff_vector`.

### Value

An `ff_vector`. For the definition of the operators see the base package of R.

### Description

These mathematical functions implement `ff_vector` specific math and handle the following functions from the Math family:

- Math2: "round", "signif"

The operators require `x` to be an `ff_vector`.

### Usage

```r
## S3 method for class 'ff_vector'
abs(x)

## S3 method for class 'ff_vector'
sign(x)

## S3 method for class 'ff_vector'
sqrt(x)

## S3 method for class 'ff_vector'
ceiling(x)
```

floor(x)
## S3 method for class 'ff_vector'
trunc(x, ...)
## S3 method for class 'ff_vector'
log10(x)
## S3 method for class 'ff_vector'
log2(x)
## S3 method for class 'ff_vector'
log1p(x)
## S3 method for class 'ff_vector'
acos(x)
## S3 method for class 'ff_vector'
acosh(x)
## S3 method for class 'ff_vector'
asin(x)
## S3 method for class 'ff_vector'
asinh(x)
## S3 method for class 'ff_vector'
atan(x)
## S3 method for class 'ff_vector'
atanh(x)
## S3 method for class 'ff_vector'
exp(x)
## S3 method for class 'ff_vector'
expm1(x)
## S3 method for class 'ff_vector'
cos(x)
## S3 method for class 'ff_vector'
cosh(x)
## S3 method for class 'ff_vector'
sin(x)
## S3 method for class 'ff_vector'


```
sinh(x)
## S3 method for class 'ff_vector'
tan(x)
## S3 method for class 'ff_vector'
tanh(x)
## S3 method for class 'ff_vector'
gamma(x)
## S3 method for class 'ff_vector'
lgamma(x)
## S3 method for class 'ff_vector'
digamma(x)
## S3 method for class 'ff_vector'
trigamma(x)
## S3 method for class 'ff_vector'
log(x, base)
## S3 method for class 'ff_vector'
round(x, digits)
## S3 method for class 'ff_vector'
signif(x, digits)
```

**Arguments**

- `x` a numeric `ff_vector`
- `...` for `trunc`, currently not used
- `base` base for `log`
- `digits` digits for `round` and `signif`

**Value**

an `ff_vector`. For the definition of the operators see the base package of R.

---

**addfforder**

*Add the order of a `ff` vector of the `ff` vector* `x`

**Description**

Add a `ff` vector that contains the order of the `ff` vector `x` as an attribute. The order can be retrieved using `ffordered`. Note that you have to assign the result to the original vector `x`. 
Usage

addfforder(x, addsorted = FALSE, ...)

ffordered(x)

ffsorted(x)

Arguments

x ff vector to be indexed
addsorted should the sorted values also be stored in ffsorted?
... parameters that will be passed on to fforder.

Value

The updated vector x

Examples

x <- ff(rnorm(10))

# adds an index to x (note the assignment)
x <- addfforder(x)

# retrieve ffindex
o <- ffordered(x)

# use it to sort the original vector
x[o]
**Arguments**

- `x` a `ff` object
- `...` optional other (ff) objects
- `na.rm` should NA be removed?
- `range` a `ri` or an integer vector of length==2 giving a range restriction for chunked processing

**Value**

TRUE, FALSE or NA

---

**any.ff**

*Summary methods for ff objects*

**Description**

Summary methods for `ff` objects

**Usage**

```r
## S3 method for class 'ff'
any(x, ..., na.rm = FALSE, range = NULL)
```

**Arguments**

- `x` a `ff` object
- `...` optional other (ff) objects
- `na.rm` should NA be removed?
- `range` a `ri` or an integer vector of length==2 giving a range restriction for chunked processing

**Value**

TRUE, FALSE or NA
as.character.ff  Character Vectors

Description
The generic function as.character converts ff vectors to characters.

Usage
## S3 method for class 'ff'
as.character(x, ...)

Arguments
- x  a ff vector
- ... other parameters passed on to chunk

Value
A factor ff vector of the same length of x.

See Also
as.character

Examples
as.character(ff(c(NA, 1:100)))
as.character(ff(seq.Date(Sys.Date(), Sys.Date()+100, by = "day")))
as.character(ff(c(Sys.time())))

as.Date.ff_vector  Date Conversion Functions for ff vector

Description
Date Conversion Functions for ff vector.

Usage
## S3 method for class 'ff_vector'
as.Date(x, ..., inplace = FALSE)
Arguments

- **x**: an object of class \texttt{ff_vector}
- **\ldots**: other parameters passed on to \texttt{as.Date}
- **inplace**: passed on to \texttt{chunkify}

Value

An \texttt{ff_vector} of length\( (x) \) containing the result of \texttt{as.Date} applied to the elements in chunks

---

\texttt{as.ffdf.ffdf} \quad \textit{Trivial implementation, but very handy}

---

Description

Coerce a \texttt{ffdf} object to an \texttt{ffdf} object.

Usage

```r
## S3 method for class 'ffdf'
as.ffdf(x, \ldots)
```

Arguments

- **x**: \texttt{ffdf} object
- **\ldots**: not used.

---

\texttt{as.ram.ffdf} \quad \textit{As ram for an \texttt{ffdf} to get your \texttt{ffdf} as a data frame in RAM}

---

Description

Load your \texttt{ffdf} object in RAM into a data.frame.

Usage

```r
## S3 method for class 'ffdf'
as.ram(x, \ldots)
```

Arguments

- **x**: an object of class \texttt{ffdf}
- **\ldots**: not used.

Value

a data.frame in RAM
bigglm.ffdf  

**Bounded memory linear regression**

**Description**

bigglm.ffdf creates a generalized linear model object that uses only \(p^2\) memory for \(p\) variables. It uses the biglm package and is a simple wrapper to allow to work with an ffdf as input data. Make sure that package is loaded.

**Usage**

```
bigglm.ffdf(formula, data, family = gaussian(), ..., chunksize = 5000)
```

**Arguments**

- `formula`: a model formula
- `data`: an object of class ffdf
- `family`: A glm family object
- `...`: other parameters passed on to bigglm. See the biglm package: `biglm`
- `chunksize`: Size of chunks for processing the ffdf

**Value**

An object of class bigglm. See the bigglm package for a description: `bigglm`

**See Also**

`bigglm`

**Examples**

```r
library(biglm)
library(ff)

data(trees)
x <- as.ffdf(trees)
a <- bigglm(log(Volume)~log(Girth)+log(Height),
data=x, chunksize=10, sandwich=TRUE)
summary(a)

b <- bigglm(log(Volume)~log(Girth)+log(Height)+offset(2*log(Girth)+log(Height)),
data=x, chunksize=10, sandwich=TRUE)
summary(b)
```
**binned_sum**

**Fast summing in different bins**

**Description**

*binned_sum* implements fast summing for given bins by calling c-code.

**Usage**

```r
binned_sum(x, bin, nbins = max(bin), ...)
```

**Arguments**

- `x` numeric vector with the data to be summed
- `bin` integer vector with the bin number for each data point
- `nbins` integer maximum bin number
- `...` used by `binned_sum.ff`

**Value**

numeric matrix where each row is a bin

---

**binned_sumsq**

**Fast squared summing in different bins**

**Description**

*binned_sum* implements fast squared summing for given bins by calling c-code, which can be used to calculate variance and standard deviation. Please note that incorrect use of this function may crash your R-session. The values of bins must be in between 1:nbins and bin may not contain NA.
Usage

\[
\text{binned sumsq}(x, \text{mean} = \text{rep}(0, \text{nbins}), \text{bin}, \text{nbins} = \max(\text{bin}), \ldots)
\]

## Default S3 method:
\[
\text{binned sumsq}(x, \text{mean} = \text{rep}(0, \text{nbins}), \text{bin}, \\
\text{nbins} = \max(\text{bin}), \ldots)
\]

## S3 method for class 'ff'
\[
\text{binned sumsq}(x, \text{mean} = \text{rep}(0, \text{nbins}), \text{bin}, \text{nbins} = \max(\text{bin}), \\
\ldots)
\]

Arguments

- \(x\) numeric vector with the data to be summed squared
- \(\text{mean}\) numeric vector with an optional mean to be subtracted from the data to be summed and squared
- \(\text{bin}\) integer vector with the bin number for each observation
- \(\text{nbins}\) integer maximum bin number
- \(...\) will be passed on to the implementation.

Value

- numeric matrix where each row is a bin
- numeric matrix where each row is a bin
- numeric matrix where each row is a bin

---

**binned_tabulate**

*Fast tabulating in different bins*

Description

\text{binned sum} implements fast tabulating for given bins by calling C-code. It also returns the number of NA's per bin. Please note that incorrect use of this function may crash your R-session. The values of bins must be between 1 and \text{nbins} and may not contain NA. The values of \(\text{x}\) must be between 1 and \text{nlevels}.

Usage

\[
\text{binned tabulate}(x, \text{bin}, \text{nbins} = \max(\text{bin}), \text{nlevels} = \text{nlevels}(x), \ldots)
\]

## Default S3 method:
\[
\text{binned tabulate}(x, \text{bin}, \text{nbins} = \max(\text{bin}), \\
\text{nlevels} = \text{nlevels}(x), \ldots)
\]

## S3 method for class 'ff'
\[
\text{binned tabulate}(x, \text{bin}, \text{nbins} = \max(\text{bin}), \text{nlevels} = \text{nlevels}(x), \\
\ldots)
\]
byMean

Arguments

- **x**: factor or integer vector with the data to be tabulated
- **bin**: integer vector with the bin number for each data point
- **nbins**: integer maximum bin number
- **nlevels**: integer number of levels used in x
- **...**: used by binned_tabulate.ff

Value

numeric matrix where each row is a bin and each column a level

---

**byMean**  
*Fast conditional mean*

Description

byMean works like a very fast version of tapply with (weighted) FUN=mean or FUN=weighted.mean.

Usage

byMean(x, by, na.rm = FALSE, weight = NULL, ...)

Arguments

- **x**: numeric vector to be averaged
- **by**: (list of) factor(s) for which the mean will be calculated
- **na.rm**: logical If TRUE NA values will be removed
- **weight**: numeric with of same length as x
- **...**: not used

Value

array with dimensions of by
bySum

*Fast conditional sum*

**Description**
bySum works like a very fast version of `tapply` with (weighted) `FUN=sum`.

**Usage**
bySum(x, by, na.rm = FALSE, weight = NULL, ...)

**Arguments**
- `x` numeric vector to be summed
- `by` (list of) factor(s) for which the sum will be calculated
- `na.rm` logical If TRUE NA values will be removed
- `weight` numeric with of same length as `x`
- `...` not used

**Value**
array with dimensions of `by`

**Examples**
bySum(warpbreaks$breaks, warpbreaks$wool)
bySum(warpbreaks$breaks, warpbreaks[, -1])

---

c.ff

*Concatenate ff vectors*

**Description**
Concatenate ff vectors

**Usage**
```r
## S3 method for class 'ff'
c(...)
```

**Arguments**
- `...` ff ff vectors to be concatenated
chunkify

Value

a new ff object, data is physically copied

See Also

ffappend

---

chunkify

Chunkify an element-wise function

Description

Chunkify creates a new function that operates on a ff vector. It creates chunks from the ff vector and calls the original function fun on each chunk.

Usage

chunkify(fun)

Arguments

fun function to be 'chunkified', the function must accept a vector and return a vector of the same length

Value

'chunkified' function that accepts a ff vector as its first argument.

---

compact

Compact a ff vector or ffdf data frame

Description

Compact takes a ff vector and tries to use the smallest binary data type for this vector.

Usage

## S3 method for class 'ff'
compact(x, use.na = TRUE, ...)

Arguments

x ff or ffdf object
use.na logical if TRUE the resulting ff vector can contain NA, otherwise this is not checked
... other parameters
condSum

**Value**
compact cloned ff vector, or original if no compacting can be done

**condMean**

*Fast conditional mean*

**Description**
condMean works like a very fast version of tapply with FUN=mean.

**Usage**
condMean(x, index, na.rm = FALSE, ...)

**Arguments**
- x: numeric vector to be averaged
- index: (list of) factor(s) for which the mean will be calculated
- na.rm: logical; If TRUE NA values will be removed
- ...: not used

**Value**
array with dimensions of index

**condSum**

*Fast conditional sum*

**Description**
condSum works like a very fast version of tapply with FUN=sum.

**Usage**
condSum(x, index, na.rm = FALSE, ...)

**Arguments**
- x: numeric vector to be summed
- index: (list of) factor(s) for which the sum will be calculated
- na.rm: logical; If TRUE NA values will be removed
- ...: not used

**Value**
array with dimensions of index
Description
Cumulative Sums, Products, and Extremes

Usage

```r
## S3 method for class 'ff'
cumsum(x, ...)

## S3 method for class 'ff'
cumprod(x, ...)

## S3 method for class 'ff'
cummax(x, ...)

## S3 method for class 'ff'
cummin(x, ...)
```

Arguments

- `x` numeric vector or an object that can be coerced to one a numeric vector
- `...` other parameters passed on to `chunk`

Value
An `ff` vector of the same length and type as `x` (after coercion), except that `cumprod` returns a numeric vector for integer input.
An NA value in `x` causes the corresponding and following elements of the return value to be NA, as does integer overflow in `cumsum` (with a warning).

See Also

cumsum, cumprod, cummax, cummin

Examples

```r
x <- 1:10000
tmp <- cumsum(ff(x))
class(tmp)
table(tmp[] == cumsum(x))

x <- rnorm(1000)
tmp <- cummax(ff(x))
table(tmp[] == cummax(x))
tmp <- cummin(ff(x))
```
cut.ff

Convert Numeric ff vector to factor ff

description
cut divides the range of x into intervals and codes the values in x according to which interval they fall. The leftmost interval corresponds to level one, the next leftmost to level two and so on.

Usage

## S3 method for class 'ff'
cut(x, breaks, ...)

Arguments

x
a (numeric) ff object that will be cut into pieces

breaks
specifies the breaks for cutting this

... other parameters that can be given to cut.default

Details

The cut method for ff with the behaviour of link{cut}

Value

ff a new ff object with the newly created factor

See Also

cut
diff.ff

Lagged Differences

Description

Returned suitably lagged and iterated differences

Usage

```r
## S3 method for class 'ff'
diff(x, lag = 1L, differences = 1L, ...)
```

Arguments

- `x`: a `ff` vector containing values to be differenced
- `lag`: an integer indicating which lag to use
- `differences`: an integer indicating the order of the difference
- `...`: other parameters will be passed on to `diff`

droplevels.ff

The function `droplevels` is used to drop unused levels from a `ff` factor or, more commonly, from factors in a `ffdf`

Description

The function `droplevels` is used to drop unused levels from a `ff` factor or, more commonly, from factors in a `ffdf`

Usage

```r
## S3 method for class 'ff'
droplevels(x, ..., inplace = FALSE)
```

Arguments

- `x`: `ff` object
- `...`: not used
- `inplace`: if TRUE the columns will be physically changed, otherwise (default) a new `ff` vector will be created

Value

`ff` object where levels of factors are dropped

See Also

droplevels droplevels.ffdf
The function droplevels is used to drop unused levels from factors in a ffdf

### Description

The function droplevels is used to drop unused levels from factors in a ffdf.

### Usage

```r
## S3 method for class 'ffdf'
droplevels(x, except = NULL, ..., inplace = FALSE)
```

### Arguments

- `x`: ffdf object
- `except`: specify which columns will be excluded from dropping levels
- `...`: further arguments passed to `droplevels.ff`
- `inplace`: if TRUE the columns will be physically changed, otherwise (default) new ff vectors will be created

### Value

ffdf object where levels of factors are dropped

### See Also

`droplevels`, `droplevels.ff`

### duplicated.ff

`Duplicated for ff and ffdf objects`

### Description

Duplicated for ff and ffdf objects similar as in duplicated. Remark that this duplicated function is slightly different from the duplicated method in the base package as it first orders the ffdf or ff_vector object and then applies duplicated. This means you need to order the ffdf or ff_vector in case you want to have the exact same result as the result of the base package. See the example.
**Usage**

```r
## S3 method for class 'ff'
duplicated(x, incomparables = FALSE, fromLast = FALSE,
    trace = FALSE, ...)

## S3 method for class 'ffdf'
duplicated(x, incomparables = FALSE, fromLast = FALSE,
    trace = FALSE, ...)
```

**Arguments**

- `x` : ff object or ffdf object
- `incomparables` : a vector of values that cannot be compared. FALSE is a special value, meaning that all values can be compared, and may be the only value accepted for methods other than the default. It will be coerced internally to the same type as `x`.
- `fromLast` : logical indicating if duplication should be considered from the last, i.e., the last (or rightmost) of identical elements will be kept
- `trace` : logical indicating to show on which chunk the function is computing
- `...` : other parameters passed on to chunk

**Value**

A logical ff vector of length `nrow(x)` or `length(x)` indicating if each row or element is duplicated.

**See Also**

`duplicated`, `ffdforder`, `fforder`

**Examples**

```r
## duplicated.ffdf - mark that you need to order according to the records you
## like in order to have similar results as the base unique method
data(iris)
irisdouble <- rbind(iris, iris)
irisdouble <- irisdouble[ sample(x=1:nrow(irisdouble), size=nrow(irisdouble)
    , replace = FALSE), ]
ffiris <- as.ffdf(irisdouble)
duplicated(ffiris, by=10, trace=TRUE)
duplicated(ffiris$Sepal.Length, by=10, trace=TRUE)
table(duplicated(irisdouble), duplicated(ffiris, by=10)[])
irisdouble <- irisdouble[order(apply( irisdouble
    , FUN=function(x) paste(x, collapse=".")
    , MARGIN=1
    ), TRUE), ]

ffiris <- as.ffdf(irisdouble)
table(duplicated(irisdouble), duplicated(ffiris, by=10)[])
table(duplicated(ffiris$Sepal.Width, by=10)[], duplicated(ffiris$Sepal.Width[]))

measures <- c("Sepal.Width","Species")
```
irisdouble <- irisdouble[order(apply(irisdouble[, measures]
    , FUN=function(x) paste(x, collapse="."),
    , MARGIN=1))]

ffiris <- as.ffdf(irisdouble)
table(duplicated(irisdouble[, measures]), duplicated(ffiris[measures], by=1)[]
  table(duplicated(ffiris$Sepal.Width, by=1)[]
  duplicated(ffiris$Sepal.Width[]))

expand.ffgrid

Create a ffdf from All Combinations of Factors

Description
Similar as expand.grid in the base package generates an ffdf. Code is almost copy-pasted from expand.grid.

Usage
expand.ffgrid(..., KEEP.OUT.ATTRS = TRUE, stringsAsFactors = TRUE)

Arguments
... ff vectors, ff factors or a list containing these.
KEEP.OUT.ATTRS currently ignored
stringsAsFactors logical specifying if character vectors are converted to factors. Irrelevant for ff as character vectors are factors in package ff.

Value
A ffdf containing one row for each combination of the supplied factors. The first factors vary fastest. The columns are labelled by the factors if these are supplied as named arguments or named components of a list.

See Also
expand.grid

Examples
comb <- expand.ffgrid(ff(1:1000), ff(factor(LETTERS)))
dim(comb)

x <- ff(factor(LETTERS))
y <- ff(1:1000)
z <- ff(seq.Date(Sys.Date(), Sys.Date()+10, by = "day"))
comb <- expand.ffgrid(x, y, z)
dim(comb)
comb[1:100, ]
**ffappend**

*Append a ff vector to another ff vector*

**Description**

Appends (ff) vector \( y \) to ff vector \( x \). Please note that the data of \( x \) will be coerced to the type of \( y \) if \( y \) has a higher vmode.

**Usage**

\[
\text{ffappend}(x, y, \text{adjustvmode} = \text{TRUE}, \ldots)
\]

**Arguments**

- \( x \): ff object where data will be appended to. If \( x = \text{NULL} \) a new ff object will be created.
- \( y \): ff object or vector object
- \( \text{adjustvmode} \): logical, indicating to coerce \( x \) to a higher vmode to make sure \( y \) is appended without loss of information.
- \( \ldots \): parameter that will be passed on to chunk internally

**Value**

ff object with same physical storage as \( x \) unless \( y \) has a higher vmode in which case the data will be cloned to the higher vmode

**See Also**

c.ff

---

**ffdfappend**

*Append a dataframe or an ffdf to another ffdf*

**Description**

Appends a dataframe or an ffdf called \( \text{dat} \) to an existing ffdf called \( x \). Please note that the data of \( x \) will be coerced to the type of \( y \) if the corresponding column of \( y \) has a higher vmode.

**Usage**

\[
\text{ffdfappend}(x, \text{dat}, \text{recode} = \text{TRUE}, \text{adjustvmode} = \text{TRUE}, \ldots)
\]
Arguments

- **x**: ffdf object where data will be appended to. If x=NULL a new ffdf object will be created.
- **dat**: ffdf object or data.frame object.
- **recode**: should factors be recoded (default), or not (faster).
- **adjustvmode**: logical, indicating to coerce the columns of x to a higher vmode to make sure y is appended without loss of information.

... Further arguments passed to as.ffdf, when x=NULL

Value

ffdf object with same physical storage as x unless the corresponding column of y has a higher vmode in which case the data will be cloned to the higher vmode.

See Also

c.ff

---

**ffdfdply**

*Performs a split-apply-combine on an ffdf*

---

**Description**

Performs a split-apply-combine on an ffdf. Splits the x ffdf according to split and applies FUN to the data, stores the result of the FUN in an ffdf.

Remark that this function does not actually split the data. In order to reduce the number of times data is put into RAM for situations with a lot of split levels, the function extracts groups of split elements which can be put into RAM according to BATCHBYTES. Please make sure your FUN covers the fact that several split elements can be in one chunk of data on which FUN is applied. Mark also that NA’s in the split are not considered as a split on which the FUN will be applied.

**Usage**

```
ffdfdply(x, split, FUN, BATCHBYTES =getOption("ffbatchbytes"),
         RECORDBYTES = sum(.rambytes[vmode(x)]), trace = TRUE, ...)
```

**Arguments**

- **x**: an ffdf
- **split**: an ff vector which is part of the ffdf x
- **FUN**: the function to apply to each split. This function needs to return a data.frame
- **BATCHBYTES**: integer scalar limiting the number of bytes to be processed in one chunk
- **RECORDBYTES**: optional integer scalar representing the bytes needed to process one row of x
- **trace**: logical indicating to show on which split the function is computing
- ... other parameters passed on to FUN
Value

an ffdf

See Also

grouprunningcumsum, table

Examples

data(iris)
ffiris <- as.ffdf(iris)

youraggregatorFUN <- function(x){
dup <- duplicated(x[c("Species", "Petal.Width")])
o <- order(x$Petal.Width)
lowest_pw <- x[rev(o)][!dup,]
highest_pw <- x[o][!dup,]
lowest_pw$group <- factor("lowest", levels=c("lowest", "highest"))
highest_pw$group <- factor("highest", levels=c("lowest", "highest"))
rbind(lowest_pw, highest_pw)
}
result <- ffdfdpym( x = ffiris, split = ffiris$Species,
  FUN = function(x) youraggregatorFUN(x),
  BATCHBYTES = 5000, trace=TRUE)
dim(result)
dim(iris)
result[1:10,]

ffiris$integerkey <- with(ffiris, as.integer(Sepal.Length))
result <- ffdfdpym( x = ffiris, split = ffiris$integerkey
  , FUN = function(x) youraggregatorFUN(x), BATCHBYTES = 5000
  , trace=TRUE
  )

ffiris$datekey <- ff( as.Date(ffiris$Sepal.Length[], origin = "1970-01-01"),
  vmode = "integer")
result <- ffdfdpym( x = ffiris, split = ffiris$datekey
  , FUN = function(x) youraggregatorFUN(x)
  , BATCHBYTES = 5000, trace=TRUE
  )

ffdfrbind.fill

bind for ffdf where missing columns are added if not available in one of the ffdf objects

Description

bind for ffdf where missing columns are added if not available in one of the ffdf objects.
Similarly as rbind.fill but for ffdf objects.
Usage

`ffdf$rbind.fill(..., clone = TRUE)`

Arguments

- `...` 2 or more `ffdf` objects
- `clone` logical, indicating to clone the first `ffdf` object in `...` or not before appending the other objects. Defaults to `TRUE`.

Value

an `ffdf` where the `ffdf` objects are `rbind`-ed together. Missing columns in either one of the passed `ffdf` objects are set to `NA` values.

Examples

```r
define x <- ffdf$rbind.fill( as.ffdf(iris),
    as.ffdf(iris[, c("Sepal.Length", "Sepal.Width"
    , "Petal.Length")])

class(x)
ncol(x)
sum(is.na(x$Petal.Width))
```

---

**ffdfsave** Save a `ffdf` data.frame in directory

Description

`ffdfsave` saves a `ffdf` data.frame in the given filename (.rdata) and stores all `ff` columns in a subdirectory with the name "<filename>_ff". Each column will be named "<columnname>.ff". A saved `ffdf` data.frame is a .rdata file and can be loaded with the `load` function. Deprecated, the preferred method is `save.ffdf`

Usage

`ffdfsave(dat, filename)`

Arguments

- `dat` `ffdf` data.frame, to be saved
- `filename` path where .rdata file will be save and `<filename>_ff` directory will be created
**ffdfwith**

Evaluate an expression in a ffdf data environment

**Description**

Evaluate an R expression in an environment constructed from a ffdata data frame. Faster than `with.ffdf`, but in contrast `ffdfwith` can change the original data. Please note that `ffdfwith` assumes that the result must be of the same length as `nrow(data)`. You should write your expression as if it is a normal data.frame. The resulting return value however will be a ffdf object.

**Usage**

`ffdfwith(data, expr, ...)`

**Arguments**

- `data` *(ffdf)*: data object used as an environment for evaluation.
- `expr` : expression to evaluate.
- `...` : arguments to be passed to future methods.

**Value**

if expression is a vector a newly created ff vector will be returned otherwise if the expression is a data.frame a newly created ffdf object will be returned.

**Examples**

```r
dat <- data.frame(x=1:10, y=1:10)
ffdat <- as.ffdf(dat)

ffdfwith(ffdat, {
  x <- x + 1
  x + y
})

#notice that x has been altered
ffdat$x
```
ffifelse

Conditional Element Selection for ff vectors.

Description

Similar as ifelse in the base package but only works with yes and no as ff vectors.

Usage

ffifelse(test, yes, no)

Arguments

test logical or boolean ff vector

yes an ff vector with return values for true elements of test. If too short, their elements are recycled.

no an ff vector with return values for false elements of test. If too short, their elements are recycled.

Value

An ff vector of the same length as test.

See Also

ifelse

Examples

data(iris)
ffiris <- as.ffdf(iris)
ffifelse(ffiris$Sepal.Length < 5, TRUE, NA)
ffifelse(ffiris$Sepal.Length < 5, factor(rep("abc", nrow(ffiris))), NA)
ffifelse(ffiris$Sepal.Length < 5, Sys.Date(), factor("abc"))
ffifelse( ffiris$Sepal.Length < 5, Sys.Date(), ff(seq.Date( Sys.Date()+1 , Sys.Date()+nrow(ffiris), by = "day")))
Description

ffmatch returns an ff vector of the positions of (first) matches of its first argument in its second. Similar as match.

ffdfmatch allows to match ffdf objects by paste-ing together the columns of the ffdf and matching on the pasted column and returns an ff vector of the positions of (first) matches of its first argument in its second.

%in% returns a logical ff vector indicating if there is a match or not for its left operand. ffdf objects are also allowed in the left and right operand of the %in% operator. See the examples.

Usage

```r
ffmatch(x, table, nomatch = NA_integer_, incomparables = NULL,
        trace = FALSE, ...)
ffdfmatch(x, table, nomatch = NA_integer_, incomparables = NULL,
        trace = FALSE, ...)

x %in% table
```

Arguments

- `x` a ff object for ffmatch or an ffdf object for ffdfmatch
- `table` a ff object for ffmatch or an ffdf object for ffdfmatch
- `nomatch` the value to be returned in the case when no match is found. Note that it is coerced to integer.
- `incomparables` a vector of values that cannot be matched. Any value in `x` matching a value in this vector is assigned the `nomatch` value. For historical reasons, FALSE is equivalent to NULL.
- `trace` logical indicating to show on which chunk the function is computing
- `...` other parameters passed on to chunk

Value

An ff vector of the same length as `x`. An integer vector giving the position in table of the first match if there is a match, otherwise `nomatch`.

See Also

match, paste
Examples

```r
## Basic example of match.ff
x.ff <- ffmatch( as.ff(as.factor(c(LETTERS, NA)))
    , as.ff(as.factor(c("C","B","Z","X","HMMM","Nothing",NA)))
    , trace=TRUE
    , BATCHBYTES=20)
class(x.ff)
x <- match(c(LETTERS, NA), c("C","B","Z","X","HMMM","Nothing",NA))
table(x.ff[]) == x, exclude=c())
## ffdfmatch also allows to input an ffdf
data(iris)
ffiris <- as.ffdf(iris)
ffirissubset <- as.ffdf(iris[1:10, nrow(iris)], 1)
ffdfmatch(ffiris, ffirissubset, trace=TRUE, BATCHBYTES=500)
## %in% is masked from the base package
letter <- factor(c(LETTERS, NA))
check <- factor(c("C","B","Z","X","HMMM","Nothing",NA))
letter %in% check
as.ff(letter) %in% as.ff(check)
ffiris %in% ffirissubset
```

---

### ffrandom

*Generate ff vector with draws from distribution*

**Description**

A convenience function to generate ff vectors with draws from random distributions using functions such as `runif`, `rnorm` and `rlnorm`.

**Usage**

```r
ffrandom(n, rfun = runif, ..., vmode = NULL)
```

**Arguments**

- `n` number of observations
- `rfun` a function generating the draws from the random distribution. This function should expect the number of draws as its first argument. Valid examples are the routines `runif`, `rnorm`, and `rlnorm`.
- `...` additional arguments are passed on to `rfun`.
- `vmode` the vmode of the resulting vector. See `ff`. If none given the vmode is determined from a single draw from `rfun.`
Details

Before generating the vector a single draw is taken from the distribution. This might be important if one tries to reproduce draws directly from \texttt{rfun}.

Value

An \texttt{ff} vector with draws from the distribution.

Examples

```r
n <- ffrandom(1E3, rnorm, mean = 10, sd = 5)
set.seed(123)
runif(1)
a <- runif(10)
set.seed(123)
b <- ffrandom(10, runif)
identical(a, b[1])
```

---

<table>
<thead>
<tr>
<th>ffrep.int</th>
<th>Replicate Elements of \texttt{ff} vectors.</th>
</tr>
</thead>
</table>

Description

Similar as \texttt{rep.int} in the base package but for \texttt{ff} vectors.

Usage

```r
ffrep.int(x, times)
```

Arguments

- \texttt{x} an integer \texttt{ff} vector
- \texttt{times} integer \texttt{ff} vector giving the (non-negative) number of times to repeat each element if of length length(\texttt{x}), or an integer of length 1 indicating how many times to to repeat the whole vector. Negative or NA values are an error.

Value

An \texttt{ff} vector of integers where \texttt{x} is recycled

See Also

- \texttt{rep.int}
Examples

```r
ffrep.int(ff(1:1000), times=20)
ffrep.int(ff(factor(LETTERS)), times=20)
ffrep.int(ff(Sys.time()), times=20)
ffrep.int(ff(seq.Date(Sys.Date(), Sys.Date()+10, by = "day")), times=20)

x <- ff(factor(LETTERS), length=26)
ffrep.int(x, times=ff(1:26))

# Or supply an ff vector of the same length as x
x <- seq.Date(Sys.Date(), Sys.Date()+10, by = "day")
x <- as.ff(x)
ffrep.int(x, times=ff(0:10))

x <- ff(factor(LETTERS), length=26)
ffrep.int(x, times=ff(1:26))
```

---

**ffseq**

*Sequence Generation of ff vectors.*

**Description**

Similar as `seq` in the base package, generating an ff vector.

**Usage**

```r
ffseq(from = 1, to = 1, by = ((to - from)/(length.out - 1)),
     length.out = NULL, along.with = NULL, ...)
```

**Arguments**

- `from` the starting value of the sequence
- `to` the end (maximal) value of the sequence
- `by` number, increment of the sequence
- `length.out` desired length of the sequence. Only non-negative numbers larger than 0 are allowed.
- `along.with` take the length from the length of this argument
- `...` arguments passed to or from methods

**Value**

An ff vector with the generated sequence, similar as what `seq` generates but as an ff vector. Mark: in case this would generate a sequence of length 0, will return `integer()`.
### ffseq_len

**Sequence Generation of ff vectors.**

**Description**

Similar as `seq_len` in the base package, generating an ff vector.

**Usage**

`ffseq_len(length.out)`

**Arguments**

- `length.out` desired length of the sequence. Only non-negative numbers larger than 0 are allowed.

**Value**

An ff vector of integers with range from 1 to length.out

**See Also**

`seq_len`
Examples

```r
## ffseq_len
ffseq_len(1000)
ffseq_len(1000000)

## ffseq
ffseq(from = 1, to = 4, by = 1)
ffseq(from = 1, to = 4, by = 0.5)
ffseq(from = 4, to = 1, by = -0.5)
ffseq(from = -100, to = 100, by = 0.3)
ffseq(from = 2, to = -100, length.out = 4)
ffseq(from = 2, to = -100, length.out = 4)
ffseq(from = 2, along.with=1000)
ffseq(to = 2, along.with=1000)
ffseq(align.with=1000)
ffseq(length.out=1000000)
```

### ffwhich

Create an index from a filter statement

**Description**

`ffwhich` creates an `ff` integer index vector from a filter expression. The resulting vector can be used to index or subset a `ffdf` or `ff` vector.

**Usage**

```r
ffwhich(x, expr, ...)
```

**Arguments**

- `x` : `ff` or `ffdf` object
- `expr` : R code that evaluates to a logical
- `...` : not used

**See Also**

`ffindexget`, `ffindexset`

**Examples**

```r
# create a ff vector
x <- ff(10:1)
# make an ff index vector
idx <- ffwhich(x, x < 5)
# use it to retrieve values from x
x[idx][]
```
# create a ffdf data.frame
dat <- ffdf(x1=x, y1=x)
# create an ff index vector from a filter statement
idx <- ffwhich(dat, x1 < 5 & y1 > 2)
# use it to select data from the data.frame
dat[idx][,]

---

**format.ff_vector**

*Date Conversion Functions for ff vector*

### Description

Date Conversion Functions for ff vector.

### Usage

```r
## S3 method for class 'ff_vector'
format(x, ..., inplace = FALSE)
```

### Arguments

- `x` an object of class `ff_vector`
- `...` other parameters passed on to `format`
- `inplace` passed on to `chunkify`

### Value

An `ff_vector` of length(x) containing the result of format applied to the elements in chunks

---

**grouprunningcumsum**

*Groups the input integer vector into several groups if the running cumulative sum increases a certain maximum number*

### Description

Groups the input integer vector into several groups if the running cumulative sum increases a certain maximum number.

### Usage

```r
grouprunningcumsum(x, max)
```

### Arguments

- `x` an integer vector
- `max` the maximum running cumulative size before an extra grouping is done
Value

An integer vector of the same length of x, indicating groups

hist.ff

hist for ff vectors

Description

Currently this is a simple version of hist functionality.

Usage

## S3 method for class 'ff'

```r
hist(x, breaks = min(100, length(x)), plot = TRUE, ...)
```

Arguments

- `x`: ff vector of values for which the histogram is desired
- `breaks`: a single numeric given the number of cells for the histogram
- `plot`: logical. If TRUE (default), a histogram is plotted, otherwise a list of breaks and counts is returned
- `...`: further arguments supplied to plot.

Value

histogram object

ikey

Creates a unique integer key for unique combinations of rows of an ffdf

Description

Creates a unique integer key for unique combinations of rows of an ffdf. In database terms this would correspond to a primary or foreign key.

Orders the ffdf decreasingly alongside the columns with NA's as last in the order and creates the integer key.

Usage

```r
ikey(x, ...)
```
Arguments

x an ff vector
... other parameters passed on to chunk

Value

An integer ff vector of the same length as the number of rows in x with unique values for each unique row

Examples

oldffmaxbytes <- getOption("ffmaxbytes")
options(ffmaxbytes = 20)
ifiris <- as.ffdf(iris)
ffiris$key1 <- ikey(ffiris)
ffiris$key2 <- ikey(ffiris[c("Petal.Width","Species")])
unique(ffiris[c("key2","Petal.Width","Species")][,])
options(ffmaxbytes = oldffmaxbytes)

Description

The generic function is.na indicates which elements are missing. The generic function is.na<- sets elements to NA.

Usage

## S3 method for class 'ff'
is.na(x, ...)

## S3 replacement method for class 'ff'
is.na(x, ...) <- value

Arguments

x a ff vector
... other parameters passed on to chunk
value a suitable ff index vector for use with x

Value

A logical ff vector of the same length of x indicating if the ff vector contains missing values.
laf_to_ffdf

Use LaF to import data into ffdf data.frame

Description

Use LaF to import data into a ffdf data.frame

Usage

laf_to_ffdf(laf, x = NULL, nrows = 1e+05, transFUN = NULL, ...)

Arguments

laf 
  laf object pointing to a csv or fwf file

x 
  optional, ffdf object where laf data should be appended to.

nrows, 
  number of rows per block, passed on to next_block

transFUN 
  NULL or a function that is called on each data.frame chunk which is read in using next_block. This can be used for filtering and data transformations.

... 
  passed on to next_block

See Also

is.na, ffvecapply

Examples

is.na.ff(ff(c(NA, 1:100)), BATCHBYTES=20, VERBOSE=TRUE)
## S3 generic
is.na(ff(c(NA, 1:100)))
## Assign a missing value
x <- ff(c(NA, 1:100))
is.na(x) <- ff(c(3,5))
x
is.na(x) <- 7:8
x
**load.ffdf**  
*Loads fdf data.frames from a directory*

**Description**
load.ffdf loads fdf data.frames from the given dir, that were stored using save.ffdf. Each column is stored as with filename `<ffdfname>$<colname>.ff`. All variables are stored in .RData in the same directory. The data can be loaded by starting a R session in the directory or by using load.ffdf.

**Usage**

```r
load.ffdf(dir, envir = parent.frame())
```

**Arguments**
- `dir` path from where the data should be loaded
- `envir` environment where the stored variables will be loaded into.

**See Also**
- `load.ffdf`

**Examples**

```r
iris.ffdf <- as.ffdf(iris)

td <- tempfile()

# save the fdf into the supplied directory
save.ffdf(iris.ffdf, dir=td)

# what in the directory?
dir(td)

#remove the fdf from memory
rm("iris.ffdf")

# and reload the stored fdf
load.ffdf(dir=td)

tf <- paste(tempfile(), ".zip", sep="")
packed <- pack.ffdf(file=tf, iris.ffdf)

#remove the fdf from memory
rm("iris.ffdf")

# restore the fdf from the packed fdf
unpack.ffdf(tf)
```
mean.ff

Mean of ff vector

Description

Mean of ff vector

Usage

```r
## S3 method for class 'ff'
mean(x, trim = 0, ..., range = NULL)
```

Arguments

- `x`: a ff vector
- `trim`: percentage of robustness, between 0 and 1
- `...`: other arguments passed to `mean`
- `range`: a ri or an integer vector of length==2 giving a range restriction for chunked processing

Value

mean value

Examples

```r
# create a vector of length 10 million
x <- ff(vmode="double", length=1e7)

mean(x)
```

merge.ffdf

Merge two ffdf by common columns, or do other versions of database join operations.

Description

Merge two ffdf by common columns, or do other versions of database join operations. This method is similar to `merge` in the base package but only allows inner and left outer joins. Note that joining is done based on `ffmatch` or `ffdfmatch`: only the first element in `y` will be added to `x`; and since `ffdfmatch` works by `paste-ing` together a key, this might not be suited if your key contains columns of vmode double.
Usage

```r
## S3 method for class 'ffdf'
merge(x, y, by = intersect(names(x), names(y)), by.x = by,
     by.y = by, all = FALSE, all.x = all, all.y = all, sort = FALSE,
     suffixes = c(".x", ",.y"), incomparables = NULL, trace = FALSE, ...)
```

Arguments

- `x`: an `ffdf`
- `y`: an `ffdf`
- `by`: specifications of the common columns. Columns can be specified by name, number or by a logical vector.
- `by.x`: specifications of the common columns of the `x` `ffdf`, overruling the `by` parameter
- `by.y`: specifications of the common columns of the `y` `ffdf`, overruling the `by` parameter
- `all`: see `merge` in R base
- `all.x`: if `TRUE`, then extra rows will be added to the output, one for each row in `x` that has no matching row in `y`. These rows will have NAs in those columns that are usually filled with values from `y`. The default is `FALSE`, so that only rows with data from both `x` and `y` are included in the output.
- `all.y`: similar as `all.x`
- `sort`: logical, currently not used yet, defaults to `FALSE`.
- `suffixes`: character(2) specifying the suffixes to be used for making non-by `names()` unique.
- `incomparables`: values which cannot be matched. See `match`. Currently not used.
- `trace`: logical indicating to show on which chunk the function is computing
- `...`: other options passed on to `ffdftable`.

Details

If a left outer join is performed and no matching record in `x` is found in `y`, columns with vmodes 'boolean', 'quad', 'nibble', 'ubyte', 'ushort' are coerced respectively to vmode 'logical', 'byte', 'byte', 'short', 'integer' to allow NA values.

Value

- an `ffdf`

See Also

- `merge`
Examples

```r
authors <- data.frame(
  surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4)))
books <- data.frame(
  name = c("Tukey", "Venables", "Tierney",
    "Ripley", "Ripley", "McNeil", "R Core"),
  title = c("Exploratory Data Analysis",
    "Modern Applied Statistics ...",
    "LISP-STAT",
    "Spatial Statistics", "Stochastic Simulation",
    "Interactive Data Analysis",
    "An Introduction to R"),
  other.author = c(NA, "Ripley", NA, NA, NA, NA,
    "Venables & Smith"))
books <- lapply(1:100, FUN=function(x, books){
  books$price <- rnorm(nrow(books))
books }, books=books)
books <- do.call(rbind, books)
authors <- as.ffdf(authors)
books <- as.ffdf(books)

dim(books)
dim(authors)
## Inner join
olddfbatchbytes <- getOption("ffbatchbytes")
options(ffbatchbytes = 100)
m1 <- merge( books, authors, by.x = "name", by.y = "surname"
  , all.x=FALSE, all.y=FALSE, trace = TRUE)
dim(m1)
unique(paste(m1$name[], m1$nationality[]))
unique(paste(m1$name[], m1$deceased[]))
m2 <- merge( books[,], authors[,], by.x = "name", by.y = "surname"
  , all.x=FALSE, all.y=FALSE, sort = FALSE)
dim(m2)
unique(paste(m2$name[], m2$nationality[]))
unique(paste(m2$name[], m2$deceased[]))
## Left outer join
m1 <- merge( books, authors, by.x = "name", by.y = "surname"
  , all.x=TRUE, all.y=FALSE, trace = TRUE)
class(m1)
dim(m1)
names(books)
names(m1)
unique(paste(m1$name[], m1$nationality[]))
unique(paste(m1$name[], m1$deceased[]))
## Show coercion to allow NA's
authors$test <- ff(TRUE, length=nrow(authors), vmode = "boolean")
vmode(authors$test)
m1 <- merge( books, authors, by.x = "name", by.y = "surname"
  , all.x=TRUE, all.y=FALSE, trace = TRUE)
```
\textbf{Description}

default behaviour of \texttt{min}, \texttt{max} and \texttt{range}

\textbf{Usage}

\begin{verbatim}
## S3 method for class 'ff'
min(x, ..., na.rm = FALSE, range = NULL)
\end{verbatim}

\textbf{Arguments}

- \texttt{x} \hspace{1cm} \texttt{ff} object
- \texttt{...} \hspace{1cm} optional other (\texttt{ff}) objects
- \texttt{na.rm} \hspace{1cm} should NA be removed?
- \texttt{range} \hspace{1cm} a \texttt{ri} or an integer vector of length\(\geq\)2 giving a range restriction for chunked processing

\textbf{Value}

minimum, maximum or range values

\textbf{Examples}

\begin{verbatim}
x <- ff(1:100)

min(x)
max(x)
range(x)

is.na(x) <- 10
min(x)
max(x)
range(x)

min(x, na.rm=TRUE)
max(x, na.rm=TRUE)
range(x, na.rm=TRUE)
\end{verbatim}
move.ffdf

Moves all the columns of ffdf data.frames into a directory

Description

move.ffdf saves all columns into the given dir. Each column is stored as with filename <ffdf-name>$<colname>.ff. If you want to store the data for an other session please use save.ffdf or pack.ffdf

Usage

move.ffdf(x, dir = ".", name = as.character(substitute(x)),
                   relativepath = FALSE)

Arguments

x  ffdf data.frame to be moved
dir  path were all of supplied ffdf’s, will be saved. It will be created if it doesn’t exist.
name  name to be used as data.frame name
relativepath  If TRUE the ffdf will contain relativepaths. Use with care...

See Also

load.ffdf save.ffdf

Examples

iris.ffdf <- as.ffdf(iris)

td <- tempfile()

# save the ffdf into the supplied directory
save.ffdf(iris.ffdf, dir=td)

# what in the directory?
dir(td)

#remove the ffdf from memory
m("iris.ffdf")

# and reload the stored ffdf
load.ffdf(dir=td)

tf <- paste(tempfile(), ".zip", sep="")
packed <- pack.ffdf(file=tf, iris.ffdf)

#remove the ffdf from memory
Description

pack.ffd stores .ffd data.frames into the given file for easy archiving and movement of data. The file can be restored using unpack.ffd. If file ends with ".zip", the package will be zipped otherwise it will be tar.gz-ed.

Usage

pack.ffd(file, ...)

Arguments

filepackaged file, zipped or tar.gz.

...ff objects to be packed

See Also

save.ffd unpack.ffd

Examples

iris.ffd <- as.ffd(iris)

td <- tempfile()

# save the .ffd into the supplied directory
save.ffd(iris.ffd[, dir=td])

# what in the directory?
dir(td)

#remove the .ffd from memory
rm("iris.ffd")

# and reload the stored .ffd
load.ffd(dir=td)

tf <- paste(tempfile(), ".zip", sep="")
packed <- pack.ffd(file=tf, iris.ffd)

#remove the .ffd from memory
rm("iris.ffdf")

# restore the ffdf from the packed ffdf
unpack.ffdf(tf)

quantile.ff quantiles

Description

The function quantile produces quantiles corresponding to the given probabilities. The smallest observation corresponds to a probability of 0 and the largest to a probability of 1. Current implementation doesn’t use the type parameter of `quantile`. For large `ff` vectors the difference between the types is (very) small. If `x` has been `ffordered`, quantile is fast, otherwise it is $n \log(n)$.

Usage

```r
## S3 method for class 'ff'
quantile(x, probs = seq(0, 1, 0.25), na.rm = FALSE,
         names = TRUE, ...)
```

Arguments

- **x**: `ff` vector
- **probs**: numeric vector of probabilities with values in [0,1].
- **na.rm**: logical; if true, any NA and NaN's are removed from `x` before the quantiles are computed.
- **names**: logical; if true, the result has a names attribute. Set to FALSE for speedup with many probs.
- **...**: currently not used

---

rle_ff

*Compute the lengths and values of runs of equal values in a vector*

Description

Similar `rle` in the base package but for `ff` vectors.

Usage

```r
rle_ff(x, ...)
```

Arguments

- **x**: an `ff` vector
- **...**: further arguments are passed on the `chunk`
save.ffdf

Value

An object of class rle which is a list with components

- **lengths**: an integer vector containing the length of each run.
- **values**: a vector of the same length as 'lengths' with the corresponding values.

Note

The resulting rle object is a memory object and must fit into memory.

See Also

rle for an implementation that runs on ordinary vectors.

save.ffdf  

Save ffdf data.frames in a directory

Description

save.ffdf saves all ffdf data.frames in the given dir. Each column is stored as with filename `<ffdfname>$<colname>.ff`. All variables given in "..." are stored in ".RData" in the same directory. The data can be reloaded by starting a R session in the directory or by using load.ffdf. Note that calling save.ffdf multiple times for the same directory will only store the ffdf's that were given in the last call.

Usage

```r
save.ffdf(..., dir = "./ffdf", clone = FALSE, relativepath = TRUE, overwrite = FALSE)
```

Arguments

- `...`: ffdf data.frames, ff vectors, or other variables to be saved in the directory
- `dir`: path where .RData file will be saved and all columns of supplied ffdf's. It will be created if it doesn't exist.
- `clone`: should the ff vectors be clone'd, creating a snapshot of the supplied ffdf or ff objects? This should only be necessary if you still need the ff vectors in their current storage location.
- `relativepath`: logical if TRUE the stored ff vectors will have relative paths, making moving the data to another storage a simple copy operation.
- `overwrite`: logical If TRUE save.ffdf will overwrite an previous stored ffdf, .Rdata file.
Details

Using `save_ffdf` automagically sets the `finalizers` of the `ff` vectors to "close". This means that the data will be preserved on disk when the object is removed or the R session is closed. Data can be deleted either using `delete` or by removing the directory where the object was saved (`dir`).

Note

When saving in the temporary directory pointed at by `getOption("fftempdir")`, `ff` assumes that the resulting files are to be deleted. Be sure to change the finalizers of the `ff` vectors when saving in the temporary directory.

See Also

`load_ffdf`

Examples

```r
iris_ffdf <- as_ffdf(iris)

td <- tempfile()

# save the ffdf into the supplied directory
save_ffdf(iris_ffdf, dir=td)

# what in the directory?
dir(td)

# remove the ffdf from memory
rm("iris_ffdf")

# and reload the stored ffdf
load_ffdf(dir=td)

tf <- paste(tempfile(), ".zip", sep="")
packed <- pack_ffdf(file=tf, iris_ffdf)

# remove the ffdf from memory
rm("iris_ffdf")

# restore the ffdf from the packed ffdf
unpack_ffdf(tf)
```

---

`set_ffbase_logging` sets the logging of `ffbase`

Description

sets the logging of `ffbase`
Usage

```r
set_ffbase_logging(level = c("info"), logger = if (interactive()) cat)
```

Arguments

- **level**: logging level: info/debug
- **logger**: function to be called for logging statements, by default this is `cat`

---

## subset.ff

**Subsetting a ff vector or ffdf data frame**

### Description

Subsetting a ff vector or ffdf data frame

### Usage

```r
## S3 method for class 'ff'
subset(x, subset, ...)
```

### Arguments

- **x**: ff vector or ffdf data.frame to be subset
- **subset**: an expression, `ri`, `bit` or logical ff vector that can be used to index `x`
- **...**: not used

### Value

A new ff vector containing the subset, data is physically copied

---

## sum.ff

**Sum of ff vector Elements**

### Description

sum returns the sum of all the values present in its arguments.

### Usage

```r
## S3 method for class 'ff'
sum(x, ..., na.rm = FALSE, range = NULL)
```
Arguments

\begin{itemize}
\item \textbf{x} \hspace{1cm} a \texttt{ff} object
\item \ldots \hspace{1cm} optional other (\texttt{ff}) objects
\item \texttt{na.rm} \hspace{1cm} should NA be removed?
\item \texttt{range} \hspace{1cm} a \texttt{ri} or an integer vector of length\(\leq 2\) giving a range restriction for chunked processing
\end{itemize}

Value

\begin{itemize}
\item \texttt{sum of elements}
\end{itemize}

\section*{Description}

Upgrades table to a generic function and implements a method for \texttt{ff} vectors which works for \texttt{ff} factors. For other arguments passed on to \texttt{table}, uses \texttt{table}

\section*{Usage}

\begin{verbatim}
table(..., exclude = if (useNA == "no") c(NA, NaN),
   useNA = c("no", "ifany", "always"), dnn = list.names(...),
   deparse.level = 1)
\end{verbatim}

\begin{verbatim}
table(..., exclude = if (useNA == "no") c(NA, NaN),
   useNA = c("no", "ifany", "always"), dnn = list.names(...),
   deparse.level = 1)
\end{verbatim}

Arguments

\begin{itemize}
\item \ldots \hspace{1cm} \texttt{ff} factors or \texttt{ff} integers
\item \texttt{exclude} \hspace{1cm} see \texttt{table}
\item \texttt{useNA} \hspace{1cm} see \texttt{table}
\item \texttt{dnn} \hspace{1cm} see \texttt{table}
\item \texttt{deparse.level} \hspace{1cm} see \texttt{table}
\end{itemize}

Details

\texttt{table.ff} uses the cross-classifying factors to build a contingency table of the counts at each combination of factor levels.

If \ldots\ does not contain factors, \texttt{unique.ff} will add a levels attribute to the non-factors.

Value

\begin{itemize}
\item \texttt{table object}
\end{itemize}
**tabulate.ff**

*Tabulation for ff vectors*

---

**Description**

`tabulate.ff` takes the integer-valued `ff` vector `bin` and counts the number of times each integer occurs in it.

**Usage**

`tabulate.ff(bin, nbins = max(bin, 1, na.rm = TRUE))`

**Arguments**

- `bin` factor to be binned.
- `nbins` number of bins

**Details**

Behaviour of `tabulate`

**Value**

integer vector or if FFRETURN is TRUE a `ff` vector

**Examples**

```r
# create a vector of 10 million
x <- ff(vmode="integer", length=1e7)

# fill first 200 with values
x[1:100] <- 1
x[101:200] <- 2

# lets count
tabulate.ff(x)
```
**transform.ffdf**  
*Transform a ffdf data.frame*

**Description**
Same functionality as `transform`, but on a ffdf object. Please note that you should write your expression as if it is a normal data.frame. The resulting data.frame however will be a ffdf data.frame.

**Usage**
```r
## S3 method for class 'ffdf'
transform(ffdf, ...)
```

**Arguments**
- `data` : ffdf data object to be transformed.
- `...` : named arguments that will be added to the ffdf data.frame

**Value**
a modified clone of `data`.

**Examples**
```r
transform(as.ffdf(airquality), Ozone = -Ozone)
transform(as.ffdf(airquality), new = -Ozone, Temp = (Temp-32)/1.8)
```

---

**unique.ff**  
*Unique values for ff and ffdf objects*

**Description**
Unique values for ff and ffdf objects

**Usage**
```r
## S3 method for class 'ff'
unique(x, incomparables = FALSE, fromLast = FALSE, 
       trace = FALSE, ...)

## S3 method for class 'ffdf'
unique(x, incomparables = FALSE, fromLast = FALSE, 
       trace = FALSE, ...)
```
Arguments

- **x**: ff object or ffdf object
- **incomparables**: a vector of values that cannot be compared. FALSE is a special value, meaning that all values can be compared, and may be the only value accepted for methods other than the default. It will be coerced internally to the same type as `x`.
- **fromLast**: logical indicating if duplication should be considered from the last, i.e., the last (or rightmost) of identical elements will be kept
- **trace**: logical indicating to show on which chunk the function is computing
- **...**: other parameters passed on to chunk

Value

An ffdf with unique values in `x` or an ff vector with unique values in `x` if `x` is a ff vector.

See Also

- `unique`

Examples

```r
data(iris)
irisdouble <- rbind(iris, iris)
ffiris <- as.ffdf(irisdouble)
## unique.ff
unique(ffiris$Sepal.Length)
unique(ffiris$Petal.Length)
ffiris$Species[1] <- NA
unique(ffiris$Species)
levels(unique(ffiris$Species))
## unique.ffdf
uiris <- unique(ffiris, trace=TRUE, by=10)[,]
test <- unique(irisdouble)
dim(iris)
dim(irisdouble)
dim(uiris)
dim(test)

apply(uiris, MARGIN=1, FUN=function(x) paste(x, collapse="",))
apply(test, MARGIN=1, FUN=function(x) paste(x, collapse="",))
apply(test, MARGIN=1, FUN=function(x) paste(x, collapse="",))
```
**unpack.ffdf**

Unpacks previously stored ffdf data.frame into a directory

---

**Description**

unpack.ffdf restores ffdf data.frames into the given dir, that were stored using pack.ffdf. If dir is NULL (the default) the data.frames will restored in a temporary directory.

**Usage**

```r
unpack.ffdf(file, dir = NULL, envir = parent.frame())
```

**Arguments**

- `file` : packaged file, zipped or tar.gz.
- `dir` : path where the data will be saved and all columns of supplied ffdf's. It will be created if it doesn't exist.
- `envir` : the environment where the stored variables should be loaded into.

**See Also**

load.ffdf pack.ffdf

**Examples**

```r
iris.ffdf <- as.ffdf(iris)

td <- tempfile()

# save the ffdf into the supplied directory
save.ffdf(iris.ffdf, dir=td)

# what in the directory?
dir(td)

#remove the ffdf from memory
rm("iris.ffdf")

# and reload the stored ffdf
load.ffdf(dir=td)

tf <- paste(tempfile(), ".zip", sep="")
packed <- pack.ffdf(file=tf, iris.ffdf)

#remove the ffdf from memory
rm("iris.ffdf")

# restore the ffdf from the packed ffdf
unpack.ffdf(tf)
```
with.ffdf

Evaluate an expression in a ffdf data environment

Description

Evaluate an R expression in an environment constructed from a ffdf data frame. (see with). Please note that you should write your expression as if it is a normal data.frame. The resulting return value however will be a ff object.

Usage

```r
## S3 method for class 'ffdf'
with(data, expr, ...)
```

Arguments

data ffdf data object used as an environment for evaluation.
expr expression to evaluate.
... arguments to be passed to chunk.

Value

if expression is a vector a newly created ff vector will be returned otherwise if the expression is a data.frame a newly created ffdf object will be returned.

Note

‘with.ffdf’ assumes that the returned object is of equal length as ‘nrow(data)’ and must be converted to a ‘ff’ object In case this is not true, the result won’t be correct.

See Also

ffdfwith

Examples

```r
dat <- data.frame(x=1:10, y=10:1)
ffdat <- as.ffdf(dat)
with(ffdat, {x+y})
```
within.ffdf  

Evaluate an expression in a ffdf data environment

Description

Same functionality as within. Please note that you should write your expression as if it is a normal data.frame. The resulting data.frame however will be a new ffdf data.frame.

Usage

```r
## S3 method for class 'ffdf'
within(data, expr, ...)
```

Arguments

- `data` *ffdf* data object used as an environment for evaluation.
- `expr` expression to evaluate.
- `...` arguments to be passed to chunk.

Value

a modified clone of data.

Examples

```r
ffdat <- as.ffdf(data.frame(x=1:10, y=1:1))
# add z to the ffdat
within(ffdat, {z <- x+y})
```

[.[ff]

Reading and writing vectors extended to handle logical ff vectors as indexes

Description

Package ff does not allow to extract and set values of ff vectors based on logical ff vectors. For this reason the extractor functions [.ff and [<-.ff defined in package ff are overloaded. If you supply a logical ff vector as an index to another ff vector, the overloaded function will convert it to an integer ff. index before using the [.ff and [<-.ff function from the ff package. This allows to do ff(1:10)[ff(c(FALSE, TRUE, NA, TRUE))]

Mark that all other functionality from the extractor functions [.ff and [<-.ff in package ff are retained. This is an extension to handle logical ff vectors as indexes.
Usage

## S3 method for class 'ff'

```r
x[i, pack = FALSE]
```

## S3 replacement method for class 'ff'

```r
x[i, add = FALSE, pack = FALSE] <- value
```

Arguments

- `x` an `ff` object
- `i` missing OR a single index expression OR a `hi` object
- `pack` FALSE to prevent rle-packing in hybrid index preprocessing, see `as.hi`
- `add` TRUE if the values should rather increment than overwrite at the target positions, see `readwrite.ff`
- `value` the values to be assigned, possibly recycled

Value

See `Extract.ff`. Mark that if a logical `ff` vector is used for `i`, and if only FALSE or NA values are present, NULL is returned in case of the extractor function `[..ff` while for the setter function `[<-.ff`, if the length value is zero, this is not allowed.

See Also

- `Extract.ff`

Examples

## extractors

```r
x <- ff(1:10)
y <- ff(11:20)
idx <- ff(c(FALSE, TRUE, NA, TRUE))
x[idx]
idx <- ff(c(FALSE, FALSE, TRUE))
x[idx]
idx <- ff(1:3)
x[idx]
```

## setters

```r
idx <- ff(c(FALSE, TRUE, NA, TRUE))
x[idx] <- y[idx]
x
idx <- ff(c(FALSE, FALSE, TRUE))
try(x[idx] <- y[idx], silent = T) ## not allowed
x
idx <- ff(1:3)
x[idx] <- y[idx]
x
```
Description

Package ff does not allow to extract and set values of ffdf objects based on logical ff vectors. For this reason the extractor functions [.ffdf and [<-.ffdf defined in package ff are overloaded. If you supply a logical ff vector as an index to subset an ffdf object, the overloaded function will convert the logical ff vector to an integer ff index before using the [.ffdf and [<-.ffdf functions from the ff package. This allows to do as.ffdf(iris)[as.ff(iris$Species ==", FALSE),]

This is an extension to handle logical ff vectors as indexes to ffdf objects.

Usage

```r
## S3 method for class 'ffdf'
x[i, j, drop = TRUE]

## S3 replacement method for class 'ffdf'
x[i, j] <- value
```

Arguments

- `x`: an ff object
- `i`: a row subscript
- `j`: a column subscript
- `drop`: logical. If TRUE the result is coerced to the lowest possible dimension.
- `value`: A suitable replacement value

Value

See `Extract.ffdf`. Mark that if a logical ff vector is used for i, and if only FALSE or NA values are present, this is not allowed as ffdf with zero rows do not exist.

See Also

`Extract.ffdf`
**Examples**

```r
## extractors for ffdf objects
data(iris)
x <- as.ffdf(iris)
x[x$Sepal.Length > 5, ]
x[x$Sepal.Length > 5, 1:3]
x[x$Sepal.Length > 5, 1, drop=TRUE]
x[x$Sepal.Length > 5, 1]
x[, 1]
x[, ]
x[c("Sepal.Length", "Sepal.Width")]
x[1:2]

## setters
data(iris)
x <- as.ffdf(iris)
testpositions <- x$Sepal.Length > 5
testpositions <- ffwhich(testpositions, testpositions == TRUE)
mynewdata <- x[testpositions, c("Sepal.Length", "Sepal.Width")]
mynewdata$Sepal.Length <- ff(1, length = nrow(mynewdata))
x[x$Sepal.Length > 5, c("Sepal.Length", "Sepal.Width")]
```

```r
x <- as.ffdf(iris)
testpositions <- x$Sepal.Length > 5
testpositions <- ffwhich(testpositions, testpositions == TRUE)
mynewdata <- x[testpositions, c("Sepal.Length", "Sepal.Width")]
mynewdata$Sepal.Length <- ff(1, length = nrow(mynewdata))
x[x[testpositions, c("Sepal.Length", "Sepal.Width")]
```

```r
x <- as.ffdf(iris)
testpositions <- x$Sepal.Length > 5
testpositions <- ffwhich(testpositions, testpositions == TRUE)
mynewdata <- x[testpositions, c("Sepal.Length", "Sepal.Width")]
mynewdata$Sepal.Length <- ff(1, length = nrow(mynewdata))
x[x[testpositions, c("Sepal.Length", "Sepal.Width")]
```
Index

bigglm, 14
bigglm.ffdf, 14
biglm, 14
binned_sum, 15
binned_sumsq, 15
binned_tabulate, 16
byMean, 17
bySum, 18
c.ff, 3, 18, 27, 28
ceiling.ff_vector(abs.ff_vector), 7
chunk, 50, 59, 60
chunkify, 13, 19, 39
cmclone, 51
compact, 19
condMean, 20
condSum, 20
cos.ff_vector(abs.ff_vector), 7
cosh.ff_vector(abs.ff_vector), 7
cummax, 21
cummax.ff(cumsum.ff), 21
cummin, 21
cummin.ff(cumsum.ff), 21
cumprod, 21
cumprod.ff, 4
cumprod.ff(cumsum.ff), 21
cumsum, 21
cumsum.ff, 4, 21
cut.default, 22
cut.ff, 3, 22
delete, 52
diff.ff, 23
digamma.ff_vector(abs.ff_vector), 7
droplevels, 23, 24
droplevels.ff, 23, 24
droplevels.ffdf, 23, 24
duplicated., 3, 24, 25
duplicated.ff, 24
duplicated.ffdf(duplicated.ff), 24
exp.ff_vector(abs.ff_vector), 7
expand.ffgrid, 26
expand.grid, 26
expml.ff_vector(abs.ff_vector), 7
Extract.ff, 61
Extract.ffdf, 62
ff, 3, 9, 22, 34, 38, 50
ffappend, 4, 19, 27
ffbase (ffbase-package), 3
ffbase-package, 3
ffdf, 3, 24, 31, 42, 56, 59, 60
ffdfappend, 4, 27
ffdfdpoly, 4, 28
ffdfindexget, 45
ffdfmatch, 3, 4, 44
ffdfmatch(ffmatch), 33
ffdforder, 25
ffdfrbind.fill, 29
ffdfsave, 30
ffdfwith, 31, 59
ffifelse, 32
ffmatch, 3, 4, 33, 44
fforder, 10, 25
ffordered, 4, 50
ffordered(ffdforder), 9
ffrandom, 34
ffrep.int, 35
ffseq, 36
ffseq_len, 37
ffsorted(ffdforder), 9
ffvecapply, 42
ffwhich, 4, 38
finalizer, 52
floor.ff_vector(abs.ff_vector), 7
format, 39
format.ff_vector, 39

gamma.ff_vector(abs.ff_vector), 7

grouprunningcumsum, 29, 39

hi, 61
hist, 40
hist.ff, 4, 40

ifelse, 32
ikey, 40
is.na, 42
is.na ff, 3, 41

is.na<-.ff(is.na ff), 41
laf_to_ffdf, 4, 42
lgamma.ff_vector(abs.ff_vector), 7
load.ffdf, 4, 43, 48, 51, 52, 58
log.ff_vector(abs.ff_vector), 7
log10.ff_vector(abs.ff_vector), 7
log1p.ff_vector(abs.ff_vector), 7
log2.ff_vector(abs.ff_vector), 7

match, 33, 45
Math(abs.ff_vector), 7
max, 47
max.ff, 4
max.ff(min.ff), 47
mean.ff, 4, 44
merge, 44, 45
merge.ffdf, 4, 44
min, 47
min.ff, 4, 47
move.ffdf, 48

Operators (> .ff_vector), 6
pack.ffdf, 4, 48, 49, 58
paste, 33, 44

quantile, 50
quantile.ff, 4, 50

range, 47
range.ff, 4
range.ff(min.ff), 47
readwrite.ff, 61
rep.int, 35
rle, 50, 51
rle_ff, 50
rlnorm, 34
rnorm, 34
round.ff_vector(abs.ff_vector), 7
runif, 34

save.ffdf, 4, 30, 43, 48, 49, 51
seq, 37
seq_len, 37
set_ffbase_logging, 52
sign.ff_vector(abs.ff_vector), 7
signif.ff_vector(abs.ff_vector), 7
sin.ff_vector(abs.ff_vector), 7
sinh.ff_vector(abs.ff_vector), 7
sqrt.ff_vector (abs.ff_vector), 7
subset.ff, 53
subset.ffdf, 7
subset.ffdf (subset.ff), 53
sum.ff, 4, 53
table, 4, 29, 54, 54, 55
tabulate, 55
tabulate.ff, 4, 55
tan.ff_vector (abs.ff_vector), 7
tanh.ff_vector (abs.ff_vector), 7
transform, 56
transform.ffdf, 4, 56
trigamma.ff_vector (abs.ff_vector), 7
trunc.ff_vector (abs.ff_vector), 7
unique, 3, 57
unique.ff, 56
unique.ffdf (unique.ff), 56
unpack.ffdf, 4, 49, 58
with, 59
with.ffdf, 4, 31, 59
within, 60
within.ffdf, 4, 60