Package ‘matter’

December 1, 2023

Type Package

Title Out-of-memory dense and sparse signal arrays

Version 2.4.0

Date 2016-10-11

Author Kylie A. Bemis <k.bemis@northeastern.edu>

Maintainer Kylie A. Bemis <k.bemis@northeastern.edu>

Description Memory-efficient file-based data structures for dense and sparse vectors, matrices, arrays, and lists with applications to nonuniformly sampled signals and spectra.

License Artistic-2.0

Depends R (>= 4.0), BiocParallel, Matrix, methods

Imports BiocGenerics, ProtGenerics, digest, irlba, biglm, stats, stats4, graphics, grDevices, utils

Suggests BiocStyle, knitr, testthat

VignetteBuilder knitr

Collate matterGenerics.R utils.R drle.R atoms.R ops.R matter.R matter_arr.R matter_fct.R matter_list.R matter_str.R search.R sparse_arr.R stream_stat.R apply.R rowStats.R stats.R scale.R dist.R biglm.R pls.R prcomp.R fastmap.R nnmf.R signal.R signal2.R altrep.R plot.R vizi.R

biocViews Infrastructure, DataRepresentation, DataImport, DimensionReduction, Preprocessing

URL https://github.com/kuwisdelu/matter

BugReports https://github.com/kuwisdelu/matter/issues

git_url https://git.bioconductor.org/packages/matter

git_branch RELEASE_3_18

git_last_commit 0254cdd

git_last_commit_date 2023-10-24

Repository Bioconductor 3.18

Date/Publication 2023-12-01

R topics documented:

`approx1` *Resampling in 1D with Interpolation*

Description

Resample the given data at specified points. Interpolation can be performed within a tolerance using several interpolation methods.

Usage

```
approx1(x, y, xout, interp = "linear", n = length(x),
tol = NA_real_, tol.ref = "abs", extrap = NA_real_)
```

Arguments

`x, y`	The data to be interpolated.
`xout`	A vector of values where the resampling should take place.
`interp`	Interpolation method. One of 'none', 'mean', 'sum', 'max', 'min', 'area', 'linear', 'cubic', 'gaussian', or 'lanczos'.
`n`	If `xout` is not given, then interpolation is performed at `n` equally spaced data points along the range of `x`.
`tol`	The tolerance for the data points used for interpolation. Must be nonnegative. If `NA`, then the tolerance is estimated from the maximum differences in `x`.
`tol.ref`	If 'abs', then comparison is done by taking the absolute difference. If 'x', then relative differences are used.
`extrap`	The value to be returned when performing extrapolation, i.e., in the case when there is no data within `tol`.

Details

The algorithm is implemented in C and provides several fast interpolation methods. Note that interpolation is limited to using data within the given tolerance. This is also used to specify the width for kernel-based interpolation methods such as `interp = "gaussian"`. The use of a tolerance also means that interpolating within the range of the data but where no data points are within the tolerance window is considered extrapolation. This can be useful when resampling sparse signals with large empty regions, by setting `extrap = 0`, and setting an appropriate tolerance.

Value

A vector of the same length as `xout`, giving the resampled data.

Author(s)

Kylie A. Bemis

See Also

`asearch`, `approx` `approx2`

Examples

```
x <- c(1.11, 2.22, 3.33, 5.0, 5.1)
y <- x^1.11

approx1(x, y, 2.22) # 2.42359
approx1(x, y, 3.0) # NA
approx1(x, y, 3.0, tol=0.2, tol.ref="x") # 3.801133
```

---

approx2 — *Resampling in 2D with Interpolation*

Description

Resample the (potentially scattered) 2D data to a rectilinear grid at the specified coordinates. Interpolation can be performed within a tolerance using several interpolation methods.

Usage

```
approx2(x, y, z, xout, yout,
interp = "linear", nx = length(x), ny = length(y),
tol = NA_real_, tol.ref = "abs", extrap = NA_real_)
```

Arguments

`x, y, z`	The data to be interpolated. Alternatively, `x` can be a matrix, in which case the matrix elements are used for `z` and `x` and `y` are generated from the matrix's dimensions.
`xout, yout`	The coordinate (grid lines) where the resampling should take place. These are expanded into a rectilinear grid using `expand.grid()`.
`interp`	Interpolation method. One of 'none', 'mean', 'sum', 'max', 'min', 'area', 'linear', 'cubic', 'gaussian', or 'lanczos'.
`nx, ny`	If `xout` is not given, then interpolation is performed at `nx * ny` equally spaced data points along the range of `x` and `y`.
`tol`	The tolerance for the data points used for interpolation. Must be nonnegative. May be length-2 to have different tolerances for `x` and `y`. If `NA`, then the tolerance is estimated from the maximum differences in `x` and `y`.
`tol.ref`	If 'abs', then comparison is done by taking the absolute difference. If 'x', then relative differences are used.
`extrap`	The value to be returned when performing extrapolation, i.e., in the case when there is no data within `tol`.

Details

See `approx1` for details of the 1D implementation. The 2D implementation is mostly the same, except it uses a kd-tree to quickly find neighboring points.

Note that `interp = "linear"` and `interp = "cubic"` use a kernel-based approximation. Traditionally, bilinear and bicubic interpolation use 4 and 16 neighboring points, respectively. However, to support scattered data, `approx2` will use as many points as are found within the given tolerance, and scale the kernels accordingly. If the input data falls on a regular grid already, then the tolerance should be specified accordingly. Set `tol` equal to the sampling rate for `interp = "linear"` and twice the sampling rate for `interp = "cubic"`.

Value

A vector of the same length as `xout`, giving the resampled data.

Author(s)

Kylie A. Bemis

See Also

`expand.grid`, `asearch`, `approx`, `approx1`

Examples

```
x <- matrix(1:25, nrow=5, ncol=5)

approx2(x, nx=10, ny=10, interp="cubic") # upsampling
```

asearch *Approximate Key-Value Search*

Description

Search a set of values indexed by a sorted (non-decreasing) vector of keys. Finds the values corresponding to matches between the elements of the first argument and the keys. Approximate matching is allowed within a specified tolerance.

Usage

```
asearch(x, keys, values, tol = 0, tol.ref = "abs",
nomatch = NA_integer_)
```

Arguments

x	A vector of values to be matched. Only integer, numeric, and character vectors are supported.
keys	A sorted (non-decreasing) vector of keys to match against. Only integer, numeric, and character vectors are supported.
values	A vector of values corresponding to the keys. Only numeric types are supported.
tol	The tolerance for matching. Must be nonnegative.
tol.ref	If 'abs', then comparison is done by taking the absolute difference. If 'x', then relative differences are used.
nomatch	The value to be returned in the case when no match is found.

Details

The algorithm is implemented in C and relies on binary search when the keys are sorted. The keys are sorted internally if necessary. See details for bsearch for matching behavior.

Value

A vector of the same length as x, giving the values corresponding to matching keys.

Author(s)

Kylie A. Bemis

See Also

bsearch, approx1

Examples

```
keys <- c(1.11, 2.22, 3.33, 5.0, 5.1)
values <- keys^1.11

asearch(2.22, keys, values) # 2.42359
asearch(3.0, keys, values) # NA
asearch(3.0, keys, values, tol=0.2, tol.ref="x") # 3.801133
```

---

biglm — *Linear Regression for "matter" Matrices*

Description

This method allows bounded memory linear regression with `matter_mat` and `sparse_mat` matrices using the "biglm" package.

Usage

```
## S4 method for signature 'formula,matter_mat'
bigglm(formula, data, ..., nchunks = NA, verbose = NA)

## S4 method for signature 'formula,sparse_mat'
bigglm(formula, data, ..., nchunks = NA, verbose = NA)
```

Arguments

`formula`	A model `formula`.
`data`	A `matter` matrix with column names.
`nchunks`	The number of chunks to use. If `NA` (the default), this is taken from `getOption("matter.default.nchunks")`. For IO-bound operations, using fewer chunks will often be faster, but use more memory.
`verbose`	Should user messages be printed with the current chunk being processed? If `NA` (the default), this is taken from `getOption("matter.default.verbose")`.
`...`	Additional options passed to `bigglm`.

Value

An object of class `bigglm`.

Author(s)

Kylie A. Bemis

See Also

`bigglm`

Examples

```
set.seed(1)

x <- matter_mat(rnorm(1000), nrow=100, ncol=10)

colnames(x) <- c(paste0("x", 1:9), "y")

fm <- paste0("y ~ ", paste0(paste0("x", 1:9), collapse=" + "))
fm <- as.formula(fm)

fit <- bigglm(fm, x, nchunks=10)
coef(fit)
```

---

binpeaks *Peak Processing*

Description

Combine peaks from multiple signals.

Usage

```
# Bin a list of peaks
binpeaks(peaklist, domain = NULL, xlist = peaklist,
    tol = NA_real_, tol.ref = "abs", merge = FALSE,
    na.drop = TRUE)

# Merge peaks
mergepeaks(peaks, n = nobs(peaks), x = peaks,
    tol = NA_real_, tol.ref = "abs",
    na.drop = TRUE)
```

Arguments

`peaklist, xlist`
: A list of vectors of peak indices (or domain values), and the values to be binned according to the peak locations.

`peaks, x`
: The indices (or domain values) of peaks which should be merged, or for which the widths or areas should be calculated. If `n` is not provided, this should be a numeric `stream_stat` vector produced by `binpeaks()`.

`domain`
: The domain variable of the signal.

`tol, tol.ref`
: A tolerance specifying the maximum allowed distance between binned or merged peaks. See `bsearch` for details. If missing, `binpeaks` estimates it as one half the minimum gap between same-signal peaks, and `mergepeaks` estimates it as one hundredth of the average gap beteen peaks.

`merge`
: Should the binned peaks be merged?

`na.drop`	Should missing values be dropped from the result?
`n`	The number of times each peak was observed. This is used to weight the averaging.

Details

`binpeaks()` is used to bin a list of peaks from multiple signals to a set of common peaks. This is done by creating a dummy reference based on the given tolerance and the range of the domain. The peaks (or corresponding values) are then binned to the dummy reference and averaged within each bin.

`mergepeaks()` is used to merge any peaks with gaps smaller than the given tolerance. The merged peaks are averaged together.

Value

A numeric `stream_stat` vector, giving the average locations of each peak.

Author(s)

Kylie A. Bemis

Examples

```
x <- c(0, 1, 1, 2, 3, 2, 1, 4, 5, 1, 1, 0)
y <- c(0, 1, 1, 3, 2, 2, 1, 5, 4, 1, 1, 0)

p1 <- findpeaks(x)
p2 <- findpeaks(y)
binpeaks(list(p1, p2), merge=FALSE)
```

`binvec` *Binned Summaries of a Vector*

Description

Summarize a vector in the bins at the specified indices.

Usage

```
binvec(x, lower, upper, stat = "sum", prob = 0.5)
```

Arguments

`x`	A numeric vector.
`lower, upper`	The (inclusive) lower and upper indices of the bins
`stat`	The statistic used to summarize the values in each bin. Must be one of "sum", "mean", "max", "min", "sd", "var", "mad", or "quantile".
`prob`	The quantile for `stat = "quantile"`.

Value

An numeric vector of the summarized (binned) values.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)

x <- sort(runif(20))

binvec(x, c(1,6,11,16), c(5,10,15,20))

binvec(x, seq(from=1, to=16, by=5), stat="mean")
```

---

bsearch *Binary Search with Approximate Matching*

Description

Use a binary search to find approximate matches for the elements of its first argument among those in its second. This implementation allows for returning the index of the nearest match if there are no exact matches. It also allows specifying a tolerance for the comparison.

Usage

```
bsearch(x, table, tol = 0, tol.ref = "abs",
nomatch = NA_integer_, nearest = FALSE)

reldiff(x, y, ref = "y")
```

Arguments

`x`	A vector of values to be matched. Only integer, numeric, and character vectors are supported.
`y, table`	A sorted (non-decreasing) vector of values to be matched against. Only integer, numeric, and character vectors are supported.
`tol`	The tolerance for matching doubles. Must be >= 0.
`ref, tol.ref`	One of 'abs', 'x', or 'y'. If 'abs', then comparison is done by taking the absolute difference. If either 'x' or 'y', then relative differences are used, and this specifies which to use as the reference (target) value. For strings, this uses the Hamming distance (number of errors), normalized by the length of the reference string for relative differences.
`nomatch`	The value to be returned in the case when no match is found, coerced to an integer. (Ignored if `nearest = TRUE`.)
`nearest`	Should the index of the closest match be returned if no exact matches are found?

Details

The algorithm is implemented in C and currently only works for 'integer', 'numeric', and 'character' vectors. If there are multiple matches, then the first match that is found will be returned, with no guarantees. If a nonzero tolerance is provided, the closest match will be returned.

The "nearest" match for strings when there are no exact matches is decided by the match with the most initial matching characters. Tolerance is ignored for strings and integers. Behavior is undefined and results may be unexpected if `values` includes NAs.

Value

A vector of the same length as `x`, giving the indexes of the matches in `table`.

Author(s)

Kylie A. Bemis

See Also

`asearch`, `match`, `pmatch`, `findInterval`

Examples

```
a <- c(1.11, 2.22, 3.33, 5.0, 5.1)

bsearch(2.22, a) # 2
bsearch(3.0, a) # NA
bsearch(3.0, a, nearest=TRUE) # 3
bsearch(3.0, a, tol=0.1, tol.ref="values") # 3

b <- c("hello", "world!")
bsearch("world!", b) # 2
bsearch("worl", b) # NA
bsearch("worl", b, nearest=TRUE) # 2
```

`checksum` *Calculate Checksums and Cryptographic Hashes*

Description

This is a generic function for applying cryptographic hash functions and calculating checksums for externally-stored R objects.

Usage

```
checksum(x, ...)

## S4 method for signature 'character'
checksum(x, algo = "sha1", ...)

## S4 method for signature 'matter_'
checksum(x, algo = "sha1", ...)
```

Arguments

x	A file path or an object to be hashed.
algo	The hash function to use.
...	Additional arguments to be passed to the hash function.

Details

The method for `matter` objects calculates checksums of each of the files in the object's `paths`.

Value

A character vector giving the hash or hashes of the object.

Author(s)

Kylie A. Bemis

See Also

`digest`

Examples

```
x <- matter(1:10)
y <- matter(1:10)

checksum(x)
checksum(y) # should be the same
```

`chunkApply` *Apply Functions Over Chunks of a List, Vector, or Matrix*

Description

Perform equivalents of `apply`, `lapply`, and `mapply`, but over parallelized chunks of data. This is most useful if accessing the data is potentially time-consuming, such as for file-based `matter` objects. Operating on chunks reduces the number of I/O operations.

Usage

```
## Operate on elements/rows/columns
chunkApply(X, MARGIN, FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = NA, BPPARAM = bpparam())

chunkLapply(X, FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = NA, BPPARAM = bpparam())

chunkMapply(FUN, ...,
    simplify = FALSE, outpath = NULL,
    verbose = NA, BPPARAM = bpparam())


## Operate on complete chunks
chunk_rowapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = NA, BPPARAM = bpparam())

chunk_colapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = NA, BPPARAM = bpparam())

chunk_lapply(X, FUN, ...,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = NA, BPPARAM = bpparam())

chunk_mapply(FUN, ..., MoreArgs = NULL,
    simplify = "c", nchunks = NA, depends = NULL,
    verbose = NA, BPPARAM = bpparam())
```

Arguments

`X` A matrix for `chunkApply()`, a list or vector for `chunkLapply()`, or lists for `chunkMapply()`. These may be any class that implements suitable methods for `[`, `[[`, `dim`, and `length()`.

MARGIN
: If the object is matrix-like, which dimension to iterate over. Must be 1 or 2, where 1 indicates rows and 2 indicates columns. The dimension names can also be used if X has dimnames set.

FUN
: The function to be applied.

MoreArgs
: A list of other arguments to FUN.

...
: Additional arguments to be passed to FUN.

simplify
: Should the result be simplified into a vector, matrix, or higher dimensional array?

nchunks
: The number of chunks to use. If NA (the default), this is taken from getOption("matter.default.nchunk For IO-bound operations, using fewer chunks will often be faster, but use more memory.

depends
: A list with length equal to the extent of X. Each element of depends should give a vector of indices which correspond to other elements of X on which each computation depends. These elements are passed to FUN. For time efficiency, no attempt is made to verify these indices are valid.

outpath
: If non-NULL, a file path where the results should be written as they are processed. If specified, FUN must return a 'raw', 'logical', 'integer', or 'numeric' vector. The result will be returned as a matter object.

verbose
: Should user messages be printed with the current chunk being processed? If NA (the default), this is taken from getOption("matter.default.verbose").

BPPARAM
: An optional instance of BiocParallelParam. See documentation for bplapply.

Details

For chunkApply(), chunkLapply(), and chunkMapply():

For vectors and lists, the vector is broken into some number of chunks according to chunks. The individual elements of the chunk are then passed to FUN.

For matrices, the matrix is chunked along rows or columns, based on the number of chunks. The individual rows or columns of the chunk are then passed to FUN.

In this way, the first argument of FUN is analogous to using the base apply, lapply, and mapply functions.

For chunk_rowapply(), chunk_colapply(), chunk_lapply(), and chunk_mapply():

In this situation, the entire chunk is passed to FUN, and FUN is responsible for knowing how to handle a sub-vector or sub-matrix of the original object. This may be useful if FUN is already a function that could be applied to the whole object such as rowSums or colSums.

When this is the case, it may be useful to provide a custom simplify function.

For convenience to the programmer, several attributes are made available when operating on a chunk.

- "chunkid":The index of the chunk currently being processed by FUN.
- "index":The indices of the elements of the chunk, as elements/rows/columns in the original matrix/vector.

- "depends" (optional):If depends is given, then this is a list of indices within the chunk. The length of the list is equal to the number of elements/rows/columns in the chunk. Each list element either NULL or a vector of indices giving the elements/rows/columns of the chunk that should be processed for that index. The indices that should be processed will be non-NULL, and indices that should be ignored will be NULL.

The depends argument can be used to iterate over dependent elements of a vector, or dependent rows/columns of a matrix. This can be useful if the calculation for a particular row/column/element depends on the values of others.

When depends is provided, multiple rows/columns/elements will be passed to FUN. Each element of the depends list should be a vector giving the indices that should be passed to FUN.

For example, this can be used to implement a rolling apply function.

Value

Typically, a list if simplify=FALSE. Otherwise, the results may be coerced to a vector or array.

Author(s)

Kylie A. Bemis

See Also

apply, lapply, mapply,

Examples

```
register(SerialParam())

set.seed(1)
x <- matrix(rnorm(1000^2), nrow=1000, ncol=1000)

out <- chunkApply(x, 1L, mean, nchunks=10)
```

---

colscale *Scaling and Centering by Row or Column Based on Grouping*

Description

Apply the equivalent of scale to either columns or rows of a matrix, using a grouping variable.

Usage

```
## S4 method for signature 'ANY'
colscale(x, center=TRUE, scale=TRUE,
    group = NULL, ..., BPPARAM = bpparam())

## S4 method for signature 'ANY'
rowscale(x, center=TRUE, scale=TRUE,
    group = NULL, ..., BPPARAM = bpparam())
```

Arguments

- `x` — A matrix-like object.
- `center` — Either a logical value or a numeric vector of length equal to the number of columns of 'x' (for `colscale()`) or the number of the rows of 'x' (for `rowscale()`). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
- `scale` — Either a logical value or a numeric vector of length equal to the number of columns of 'x' (for `colscale()`) or the number of the rows of 'x' (for `rowscale()`). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
- `group` — A vector or factor giving the groupings with length equal to the number of rows of 'x' (for `colscale()`) or the number of the columns of 'x' (for `rowscale()`).
- `...` — Arguments passed to `rowStats()` or `colStats()` respectively, if `center` or `scale` must be calculated.
- `BPPARAM` — An optional instance of `BiocParallelParam`. See documentation for `bplapply`.

Details

See `scale` for details.

Value

A matrix-like object (usually of the same class as `x`) with either 'col-scaled:center' and 'col-scaled:scale' attributes or 'row-scaled:center' and 'row-scaled:scale' attributes.

Author(s)

Kylie A. Bemis

See Also

`scale`

Examples

```
x <- matter(1:100, nrow=10, ncol=10)

colscale(x)
```

colStats *Row and Column Summary Statistics Based on Grouping*

Description

These functions perform calculation of summary statistics over matrix rows and columns for each level of a grouping variable.

Usage

```
## S4 method for signature 'ANY'
rowStats(x, stat, ..., BPPARAM = bpparam())

## S4 method for signature 'ANY'
colStats(x, stat, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_mat'
rowStats(x, stat, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_mat'
colStats(x, stat, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat'
rowStats(x, stat, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat'
colStats(x, stat, ..., BPPARAM = bpparam())

.rowStats(x, stat, group = NULL,
    na.rm = FALSE, simplify = TRUE, drop = TRUE,
    iter.dim = 1L, BPPARAM = bpparam(), ...)

.colStats(x, stat, group = NULL,
    na.rm = FALSE, simplify = TRUE, drop = TRUE,
    iter.dim = 2L, BPPARAM = bpparam(), ...)
```

Arguments

x	A matrix on which to calculate summary statistics.
stat	The name of summary statistics to compute over the rows or columns of a matrix. Allowable values include: "min", "max", "prod", "sum", "mean", "var", "sd", "any", "all", and "nnzero".
group	A factor or vector giving the grouping. If not provided, no grouping will be used.
na.rm	If TRUE, remove NA values before summarizing.

simplify	Simplify the results from a list to a vector or array. This also drops any additional attributes (besides names).
drop	If only a single summary statistic is calculated, return the results as a vector (or matrix) rather than a list.
iter.dim	The dimension to iterate over. Must be 1 or 2, where 1 indicates rows and 2 indicates columns.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to chunk_rowapply() or chunk_colapply(), such as the number of chunks.

Details

The summary statistics methods are calculated over chunks of the matrix using s_colstats and s_rowstats. For matter objects, the iteration is performed over the major dimension for IO efficiency.

Value

A list for each stat requested, where each element is either a vector (if no grouping variable is provided) or a matrix where each column corresponds to a different level of group.

If drop=TRUE, and only a single statistic is requested, then the result will be unlisted and returned as a vector or matrix.

Author(s)

Kylie A. Bemis

See Also

colSums

Examples

```
register(SerialParam())

set.seed(1)

x <- matrix(runif(100^2), nrow=100, ncol=100)

g <- as.factor(rep(letters[1:5], each=20))

colStats(x, "mean", group=g)
```

colsweep *Sweep out Matrix Summaries Based on Grouping*

Description

Apply the equivalent of `sweep` to either columns or rows of a matrix, using a grouping variable.

Usage

```
## S4 method for signature 'ANY'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'matter_mat'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'sparse_mat'
colsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'ANY'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'matter_mat'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)

## S4 method for signature 'sparse_mat'
rowsweep(x, STATS, FUN = "-", group = NULL, ...)
```

Arguments

`x`	A matrix-like object.
`STATS`	The summary statistic to be swept out, with length equal to the number of columns of 'x' (for `colsweep()`) or the number of the rows of 'x' (for `rowsweep()`). If a grouping variable is given, then this must be a matrix with number of columns equal to the number of groups.
`FUN`	The function to be used to carry out the sweep.
`group`	A vector or factor giving the groupings with length equal to the number of rows of 'x' (for `colsweep()`) or the number of the columns of 'x' (for `rowsweep()`).
`...`	Ignored.

Details

See `sweep` for details.

Value

A matrix-like object (usually of the same class as `x`) with the statistics swept out.

Author(s)

Kylie A. Bemis

See Also

sweep

Examples

```
set.seed(1)

x <- matrix(1:100, nrow=10, ncol=10)

colsweep(x, colStats(x, "mean"))
```

`convolve_at` *Convolution at Arbitrary Indices*

Description

Convolve a signal with weights at arbitrary indices.

Usage

```
convolve_at(x, index, weights, ...)
```

Arguments

`x`	A numeric vector.
`index`	A list of numeric vectors giving the indices to convolve. Must be as long as `x`. Lengths must match `weights`.
`weights`	A list giving the weights of the kernels to convolve for each element of `x`. Lengths must match `index`.
`...`	Additional arguments passed to `sum()`. (E.g, `na.rm`.)

Details

This is essentially just a weighted sum defined by `x[i] = sum(weights[[i]] * x[index[[i]]])`.

Value

A numeric vector the same length as `x` with the smoothed result.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)
t <- seq(from=0, to=6 * pi, length.out=5000)
y <- sin(t) + 0.6 * sin(2.6 * t)
x <- y + runif(length(y))

i <- roll(seq_along(x), width=15)
wt <- dnorm((-7):7, sd=7/2)
wt <- wt / sum(wt)

xs <- convolve_at(x, i, wt)

plot(x, type="l")
lines(xs, col="red")
```

---

cpal   *Color Palettes*

Description

These functions provide simple color palettes.

Usage

```
## Continuous color palettes
cpal(palette = "Viridis")

## Discrete color palettes
dpal(palette = "Tableau 10")

# Add transparency to colors
add_alpha(colors, alpha = 1, exp = 2)
```

Arguments

`palette`	The name of a color palette. See `palette.pals` and `hcl.pals`.
`colors`	A character vector of colors to add transparency to.
`alpha`	A numeric vector giving the level of transparency in the range [0, 1] where 0 is fully transparent and 1 is fully opaque.
`exp`	The power scaling of the alpha channel. A linear alpha scale often results in poor interpretability for superposed images, so raising the alpha channel (already in range [0, 1]) to a power > 1 can improve interpretability in these cases.

Value

A character vector of colors or a function for generating `n` colors.

Author(s)

Kylie A. Bemis

See Also

vizi, image

Examples

```
f <- cpal("viridis")
cols <- f(10)
add_alpha(cols, 1:10/10)
```

---

deferred-ops *Deferred Operations on "matter" Objects*

Description

Some arithmetic, comparison, and logical operations are available as delayed operations on matter_arr and sparse_arr objects.

Details

Currently the following delayed operations are supported:

'Arith': '+', '-', '*', '/', '^', '

'Compare': '==', '>', '<', '!=', '<=', '>='

'Logic': '&', '|'

'Ops': 'Arith', 'Compare', 'Logic'

'Math': 'exp', 'log', 'log2', 'log10'

Arithmetic operations are applied in C++ layer immediately after the elements are read from virtual memory. This means that operations that are implemented in C and/or C++ for efficiency (such as summary statistics) will also reflect the execution of the deferred arithmetic operations.

Value

A new matter object with the registered deferred operation. Data in storage is not modified; only object metadata is changed.

Author(s)

Kylie A. Bemis

See Also

Arith, Compare, Logic, Ops, Math

Examples

```
x <- matter(1:100)
y <- 2 * x + 1

x[1:10]
y[1:10]

mean(x)
mean(y)
```

---

downsample *Downsample a Signal*

Description

Downsamples a signal for the purposes of visualization. A subset of the original samples are used to represent the signal. The downsampled signal is intended to resemble the original when visualized and should not typically be used for downstream processing.

Usage

```
downsample(x, n = length(x) / 10L, domain = NULL,
method = c("lttb", "ltob", "dynamic"))
```

Arguments

`x`	A numeric vector.
`n`	The length of the downsampled signal.
`domain`	The domain variable of the signal.
`method`	The downsampling method to be used. Must be one of `"lttb"`, `"ltob"`, or `"dynamic"`.

Details

This function implements the downsampling methods from Sveinn Steinarsson's 2013 MSc thesis *Downsampling Time Series for Visual Representation*, including largest-triangle-three-buckets (LTTB), largest-triangle-one-bucket (LTOB), and dynamic binning.

Value

A vector of length `n`, giving the downsampled signal.

Author(s)

Kylie A. Bemis

References

S. Steinarsson. "Downsampling Time Series for Visual Representation." MSc, School of Engineering and Natural Sciences, University of Iceland, Reykjavik, Iceland, June 2013.

See Also

approx1

Examples

```
set.seed(1)
t <- seq(from=0, to=6 * pi, length.out=2000)
x <- sin(t) + 0.6 * sin(2.6 * t)
x <- x + runif(length(x))
xs <- downsample(x, n=200)
s <- attr(xs, "sample")

plot(x, type="l")
points(s, xs, col="red", type="b")
```

---

drle-class *Delta Run Length Encoding*

Description

The `drle` class stores delta-run-length-encoded vectors. These differ from other run-length-encoded vectors provided by other packages in that they allow for runs of values that each differ by a common difference (delta).

Usage

```
## Instance creation
drle(x, cr_threshold = 0)

is.drle(x)
## Additional methods documented below
```

Arguments

`x`	An integer or numeric vector to convert to delta run length encoding for `drle()`; an object to test if it is of class `drle` for `is.drle()`.
`cr_threshold`	The compression ratio threshold to use when converting a vector to delta run length encoding. The default (0) always converts the object to `drle`. Values of `cr_threshold` < 1 correspond to compressing even when the output will be larger than the input (by a certain ratio). For values > 1, compression will only take place when the output is (approximately) at least `cr_threshold` times smaller.

Value

An object of class `drle`.

Slots

`values`: The values that begin each run.

`lengths`: The length of each run.

`deltas`: The difference between the values of each run.

Creating Objects

`drle` instances can be created through `drle()`.

Methods

Standard generic methods:

`x[i]`: Get the elements of the uncompressed vector.

`length(x)`: Get the length of the uncompressed vector.

`c(x, ...)`: Combine vectors.

Author(s)

Kylie A. Bemis

See Also

`rle`

Examples

```
## Create a drle vector
x <- c(1,1,1,1,1,6,7,8,9,10,21,32,33,34,15)
y <- drle(x)

# Check that their elements are equal
x == y[]
```

enhance *Contrast Enhancement*

Description

Enhance the contrast in a 2D signal.

Usage

```
# Histogram equalization
enhance_hist(x, nbins = 256L)

# Contrast-limited adaptive histogram equalization (CLAHE)
enhance_adapt(x, width = sqrt(length(x)) %/% 5L,
    clip = 0.1, nbins = 256L)
```

Arguments

`x`	A numeric matrix.
`nbins`	The number of gray levels in the output image.
`clip`	The normalized clip limit, expressed as a fraction of the neighborhood size. This is used to limit the maximum value of any bin in the adaptive histograms, in order avoid amplifying local noise.
`width`	The width of the sliding window used when calculating the local adaptive histograms.

Details

`enhance_heq()` performs histogram equalization. Histogram equalization transforms the pixel values so that the histogram of the image is approximately flat. This is done by replacing the original pixel values with their associated probability in the image's empirical cumulative distribution.

`enhance_aheq()` performs contrast-limited adaptive histogram equalization (CLAHE) from Zuiderveld (1994). While ordinary histogram equalization performs a global transformation on the image, adaptive histogram equalization calculates a histogram in a local neighborhood around each pixel to perform the transformation, thereby enhancing the local contrast across the image. However, this can amplify local noise, so to avoid this, the histogram is clipped to a maximum allowed bin value before transforming the pixel values. To speed up the computation, it is implemented here using a sliding window technique as described by Wang and Tao (2006).

These methods rescale the output image so that its median equals the median of the original image and it has equal interquartile range (IQR).

Value

A numeric matrix the same dimensions as `x` with the smoothed result.

Author(s)

Kylie A. Bemis

References

K. Zuiderveld. "Contrast Limited Adaptive Histogram Equalization." Graphics Gems IV, Academic Press, pp. 474-485, 1994.

Z. Wang and J. Tao. "A Fast Implementation of Adaptive Histogram Equalization." IEEE 8th international Conference on Signal Processing, Nov 2006.

Examples

```
set.seed(1)
x <- matrix(0, nrow=32, ncol=32)
x[9:24,9:24] <- 10
x <- x + runif(length(x))
y <- x + rlnorm(length(x))
z <- enhance_hist(y)

par(mfcol=c(1,3))
image(x, col=hcl.colors(256), main="original")
image(y, col=hcl.colors(256), main="multiplicative noise")
image(z, col=hcl.colors(256), main="histogram equalization")
```

---

estbase *Continuum Estimation*

Description

Estimate the continuum (baseline) of a signal.

Usage

```
# Continuum based on local extrema
estbase_loc(x,
    smooth = c("none", "loess", "spline"),
    span = 1/10, spar = NULL, upper = FALSE)

# Convex hull
estbase_hull(x, upper = FALSE)

# Sensitive nonlinear iterative peak clipping (SNIP)
estbase_snip(x, width = 100L, decreasing = TRUE)

# Running medians
estbase_med(x, width = 100L)
```

Arguments

`x`	A numeric vector.
`smooth`	A smoothing method to be applied after linearly interpolating the continuum.
`span, spar`	Smoothing parameters for loess and spline smoothing, respectively.
`upper`	Should the upper continuum be estimated instead of the lower continuum?
`width`	The width of the smoothing window in number of samples.
`decreasing`	Use a decreasing clipping window instead of an increasing window.

Details

`estbase_loc()` uses a simple method based on linearly interpolating from local extrema. It typically performs well enough for most situations. Signals with strong noise or wide peaks may require stronger smoothing after the interpolation step.

`estbase_hull()` estimates the continuum by finding the lower or upper convex hull using the monotonic chain algorithm of A. M. Andrew (1979).

`estbase_snip()` performs sensitive nonlinear iterative peak (SNIP) clipping using the adaptive clipping window from M. Morhac (2009).

`estbase_med()` estimates the continuum from running medians.

Value

A numeric vector the same length as `x` with the estimated continuum.

Author(s)

Kylie A. Bemis

References

A. M. Andrew. "Another efficient algorithm for convex hulls in two dimensions." Information Processing Letters, vol. 9, issue 5, pp. 216-219, Dec. 1979.

M. Morhac. "An algorithm for determination of peak regions and baseline elimination in spectroscopic data." Nuclear Instruments and Methods in Physics Research A, vol. 600, issue 2, pp. 478-487, Mar. 2009.

Examples

```
set.seed(1)
t <- seq(from=0, to=6 * pi, length.out=2000)
x <- sin(t) + 0.6 * sin(2.6 * t)
lo <- estbase_hull(x)
hi <- estbase_hull(x, upper=TRUE)

plot(x, type="l")
lines(lo, col="red")
lines(hi, col="blue")
```

estdim — *Estimate Raster Dimensions*

Description

Estimate the raster dimensions of a scattered 2D signal based on its pixel coordinates.

Usage

```
estdim(x, tol = 1e-6)
```

Arguments

`x`	A numeric matrix or data frame where each column gives the pixel coordinates for a different dimension. Only 2 or 3 dimensions are supported if the coordinates are irregular. Otherwise, any number of dimensions are supported.
`tol`	The tolerance allowed when estimating the resolution (i.e., pixel sizes) using `estres()`. If estimating the resolution this way fails, then it is estimated from the coordinate ranges instead.

Value

A numeric vector giving the estimated raster dimensions.

Author(s)

Kylie A. Bemis

Examples

```
co <- expand.grid(x=1:12, y=1:9)
co$x <- jitter(co$x)
co$y <- jitter(co$y)

estdim(co)
```

estnoise — *Local Noise Estimation*

Description

Estimate the noise across a signal.

Usage

```
# Quantile-based noise estimation
estnoise_quant(x, n = 25L, prob = 0.95, niter = 3L)

# Derivative-based noise estimation
estnoise_diff(x, nbins = 1L, dynamic = FALSE)

# Dynamic noise level filtering
estnoise_filt(x, nbins = 1L, msnr = 2,
    threshold = 0.5, peaks = FALSE)

# SD-based noise estimation
estnoise_sd(x, n = 25L, wavelet = ricker)

# MAD-based noise estimation
estnoise_mad(x, n = 25L, wavelet = ricker)
```

Arguments

`x`	A numeric vector.
`n`	The number of sample points in the rolling estimation of quantile, standard deviation, or median absolute deviation.
`prob`	The quantile used to estimate the noise.
`niter`	The number of iterations of nonlinear diffusion smoothing to be applied to the signal.
`nbins`	The number of bins to divide the signal into before estimating the noise. The noise is estimated locally in each bin.
`dynamic`	Should the bins be equally spaced (`FALSE`) or dynamically spaced (`TRUE`) based on the local signal?
`msnr`	The minimum signal-to-noise ratio for distinguishing signal peaks from noise peaks.
`threshold`	The required signal-to-noise difference for the first non-noise peak.
`peaks`	Does `x` represent a signal profile (`FALSE`) or peaks (`TRUE`)?
`wavelet`	The wavelet to be convolved with the signal to produce the noise estimate. This should be a function that takes as its first argument the width of the wavelet (in number of points). If this is `NULL`, then no convolution is performed, and the raw signal is used.

Details

`estnoise_quant()` estimates the local noise by first smoothing the signal with a nonlinear diffusion filter, and then subtracting the raw signal from the smoothed signal to isolate the noise component. A rolling quantile of this noise component is used to estimate the local noise in the signal.

`estnoise_diff()` estimates the local noise from the mean absolute deviation of the derivative of the signal in each bin. For noisy signals, the derivative is dominated by the noise, making it a useful estimator of the noise.

estnoise_filt() uses the dynamic noise level filtering algorithm of Xu and Freitas (2010) based on the local signal in an approach similar to Gallia et al. (2013). The peaks in the signal are sorted, and the smallest peak is assumed to be noise and is used to estimate the noise level. Each peak is then compared to the previous peak. A peak is labeled a signal peak only if it exceeds a minimum signal-to-noise ratio. Otherwise, the peak is labeled noise, and the noise level is re-estimated. This process continues until a signal peak is found, and the noise level is estimated from the noise peaks.

estnoise_sd() and estnoise_mad() estimate the local noise from the standard deviation (SD) or median absolute deviation (MAD), respectively, after (optionally) convolving the signal with a wavelet.

Value

A numeric vector the same length as x with the estimated local noise level.

Author(s)

Kylie A. Bemis

References

H. Xu and M. A. Freitas. “A Dynamic Noise Level Algorithm for Spectral Screening of Peptide MS/MS Spectra.” BMC Bioinformatics, vol. 11, no. 436, Aug. 2010.

J. Gallia, K. Lavrich, A. Tan-Wilson, and P. H. Madden. “Filtering of MS/MS data for peptide identification.” BMC Genomics, vol. 14, suppl. 7, Nov. 2013.

Examples

```
# simple signal
set.seed(1)
n <- 500
x <- rnorm(n)
x <- x + 90 * dnorm(seq_along(x), mean=n/4)
x <- x + 80 * dnorm(seq_along(x), mean=n/2)
x <- x + 70 * dnorm(seq_along(x), mean=3*n/4)

ns <- estnoise_quant(x)
plot(x, type="l")
lines(ns, col="blue")

# simulated spectrum
set.seed(1)
x <- simspec(size=5000)

ns <- estnoise_quant(x, n=101)
plot(x, type="l")
lines(ns, col="blue")
```

estres — *Estimate Signal Resolution*

Description

Estimate the resolution (approximate sampling rate) of a signal based on its domain values.

Usage

```
estres(x, tol = 1e-6, tol.ref = NA_character_)
```

Arguments

`x`	A numeric vector giving the domain values of the signal.
`tol`	The tolerance allowed when determining if the estimated resolution is valid (i.e., actually matches the given domain values). Noise in the sampling rate will be allowed up to this amount.
`tol.ref`	If 'abs', then comparison is done by taking the absolute difference. If 'x', then relative differences are used. If missing, then the funciton will try to determine which gives a better fit to the domain values.

Value

A single number named "absolute" or "relative" giving the approximate constant sampling rate matching the given domain values. NA if a sampling rate could not be determined.

Author(s)

Kylie A. Bemis

Examples

```
x <- seq_rel(501, 600, by=1e-3)

estres(x)
```

fastmap — *FastMap Projection*

Description

The FastMap algorithm performs approximate multidimensional scaling (MDS) based on any distance function. It is faster and more efficient than traditional MDS algorithms, scaling as O(n) rather than O(n^2). FastMap accomplishes this by finding two distant pivot objects on some hyperplane for each projected dimension, and then projecting all other objects onto the line between these pivots.

Usage

```
# FastMap projection
fastmap(x, k = 3L, distfun = NULL,
transpose = FALSE, niter = 3L, verbose = NA, ...)

## S3 method for class 'fastmap'
predict(object, newdata, ...)

# Distance functionals
rowDistFun(x, y, metric = "euclidean", p = 2,
BPPARAM = bpparam())
colDistFun(x, y, metric = "euclidean", p = 2,
BPPARAM = bpparam())
```

Arguments

`x, y`	A numeric matrix-like object.
`k`	The number of FastMap components to project.
`distfun`	The function of the form `function(x, y, ...)` used to generate a distance function of the form `function(i)` giving the distances between the `i`th object in `x` and all objects in `y`.
`transpose`	A logical value indicating whether `x` should be considered transposed or not. This only used internally to indicate whether the input matrix is (P x N) or (N x P), and therefore extract the number of objects and their names.
`niter`	The maximum number of iterations for finding the pivots.
`verbose`	Should progress be printed for each iteration?
`...`	Additional options passed to `distfun`.
`object`	An object inheriting from `fastmap`.
`newdata`	An optional data matrix to use for the prediction.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`.
`metric`	Distance metric to use when finding the nearest neighbors. Supported metrics include "euclidean", "maximum", "manhattan", and "minkowski".
`p`	The power for the Minkowski distance.

Details

The pivots are initialized randomly for each new dimension, so the selection of pivots (and therefore the resulting projection) can be sensitive to the random seed for some datasets.

A custom distance function can be passed via `distfun`. If not provided, then this defaults to `rowDistFun()` if `transpose=FALSE` or `colDistFun()` if `transpose=TRUE`.

If a custom function is passed, it should take the form `function(x, y, ...)`, and it must *return* a function of the form `function(i)`. The returned function should return the distances between the `i`th object in `x` and all objects in `y`. `rowDistFun()` and `colDistFun()` are examples of functions that satisfy these properties.

Value

An object of class `fastmap`, with the following components:

- `x`: The projected variable matrix.
- `sdev`: The standard deviations of each column of the projected matrix `x`.
- `pivots`: A matrix giving the indices of the pivots and the distances between them.
- `pivot.array`: A subset of the original data matrix containing only the pivots.
- `distfun`: The function used to generate the distance function.

Author(s)

Kylie A. Bemis

References

C. Faloutsos, and D. Lin. "FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets." Proceedings of the 1995 ACM SIGMOD international conference on Management of data, pp. 163 - 174, June 1995.

See Also

`cmdscale`, `prcomp`

Examples

```
register(SerialParam())
set.seed(1)

a <- matrix(sort(runif(500)), nrow=50, ncol=10)
b <- matrix(rev(sort(runif(500))), nrow=50, ncol=10)
x <- cbind(a, b)

fm <- fastmap(x, k=2)
```

---

filt1 *Smoothing Filters in 1D*

Description

Smooth a uniformly sampled 1D signal.

Usage

```
# Moving average filter
filt1_ma(x, width = 5L)

# Linear convolution filter
filt1_conv(x, weights)

# Gaussian filter
filt1_gauss(x, width = 5L, sd = (width %/% 2) / 2)

# Bilateral filter
filt1_bi(x, width = 5L, sddist = (width %/% 2) / 2,
    sdrange = mad(x, na.rm = TRUE))

# Bilateral filter with adaptive parameters
filt1_adapt(x, width = 5L, spar = 1)

# Nonlinear diffusion
filt1_diff(x, niter = 3L, kappa = 50,
    rate = 0.25, method = 1L)

# Guided filter
filt1_guide(x, width = 5L, guide = x,
    sdreg = mad(x, na.rm = TRUE))

# Peak-aware guided filter
filt1_pag(x, width = 5L, guide = NULL,
    sdreg = mad(x, na.rm = TRUE), ftol = 1/10)

# Savitzky-Golay filter
filt1_sg(x, width = 5L, order = min(3L, width - 2L),
    deriv = 0, delta = 1)
```

Arguments

`x`	A numeric vector.
`width`	The width of the smoothing window in number of samples. Must be positive. Must be odd.
`weights`	The weights of the linear convolution kernel. Length must be odd.
`sd, sddist`	The spatial parameter for kernel-based filters. This controls the strength of smoothing for samples farther from the center of the smoothing window.
`sdrange`	The range parameter for kernel-based filters. This controls the strength of the smoothing for samples with signal values very different from the center of the smoothing window.
`spar`	The strength of the smoothing when calculating the adaptive bilateral filtering parameters. The larger the number, the stronger the smoothing. Must be positive.

`kappa` The constant for the conduction coefficient for nonlinear diffusion. Must be positive.

`rate` The rate of diffusion. Must be between 0 and 0.25 for stability.

`method` The diffusivity method, where `1` and `2` correspond to the two diffusivity functions proposed by Perona and Malik (1990). For `1`, this is `exp(-(|grad x|/K)^2)`, and `2` is `1/(1+(|grad x|/K)^2)`. An additional method `3` implements the peak-aware weighting `1/(1+(|x|/K)^2)`, which does *not* use the gradient, but is used to create the guidance signal for peak-aware guided filtering.

`niter` The number of iterations for nonlinear diffusion. Must be positive.

`guide` The guide signal for guided filtering. This is the signal used to determine the degree of filtering for different regions of the sample. By default, it is the same as the signal to be smoothed.

`sdreg` The regularization parameter for guided filtering. This is analagous to the range parameter for kernel-based filters. Signal regions with variance much smaller than this value are smoothed, while signal regions with varaince much larger than this value are preserved.

`ftol` Specifies how large the signal value must be before it is considered a peak, expressed as a fraction of the maximum value in the signal.

`order` The polynomial order for the Savitzky-Golay filter coefficients.

`deriv` The order of the derivative for the Savitzky-Golay filter coefficients.

`delta` The sample spacing for the Savitzky-Golay filter. Only used if `deriv > 0`.

Details

`filt1_ma()` performs mean filtering in O(n) time. This is fast and especially useful for calculating other filters that can be constructed as a combination of mean filters.

`filt1_gauss()` performs Gaussian filtering.

`filt1_bi()` and `filt1_adapt()` perform edge-preserving bilateral filtering. The latter calculates the kernel parameters adapatively based on the local signal, using a strategy adapted from Joseph and Periyasamy (2018).

`filt1_diff()` performs the nonlinear diffusion filtering of Perona and Malik (1990). Rather than relying on a filter width, it progressively diffuses (smooths) the signal over multiple iterations. More iterations will result in a smoother image.

`filt1_guide()` performs edge-preserving guided filtering. Guided filtering uses a local linear model based on the structure of a so-called "guidance signal". By default, the guidance signal is often the same as the input signal. Guided filtering performs similarly to bilateral filtering, but is often faster (though with more memory use), as it is implemented as a combination of mean filters.

`filt1_pag()` performs peak-aware guided filtering using a regularization parameter that focuses on preserving peaks rather than edges, using a strategy adapted from Liu and He (2022). By default, the guidance signal is generated by smoothing the input signal with nonlinear diffusion.

`filt1_sg()` performs traditional Savitzky-Golay filtering, which uses a local least-squares polynomial approximation to perform the smoothing. It reduces noise while attempting to retain the peak shape and height.

Value

A numeric vector the same length as x with the smoothed result.

Author(s)

Kylie A. Bemis

References

J. Joseph and R. Perisamy. “An image driven bilateral filter with adaptive range and spatial parameters for denoising Magnetic Resonance Images.” Computers and Electrical Engineering, vol. 69, pp. 782-795, July 2018.

P. Perona and J. Malik. “Scale-space and edge detection using anisotropic diffusion.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, issue 7, pp. 629-639, July 1990.

K. He, J. Sun, and X. Tang. “Guided Image Filtering.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 6, pp. 1397-1409, June 2013.

D. Liu and C. He. “Peak-aware guided filtering for spectrum signal denoising.” Chemometrics and Intelligent Laboratory Systems, vol. 222, March 2022.

A. Savitzky and M. J. E. Golay. “Smoothing and Differentiation of Data by Simplified Least Squares Procedures.” Analytical Chemistry, vol. 36, no. 8, pp. 1627-1639, July 1964.

Examples

```
set.seed(1)
t <- seq(from=0, to=6 * pi, length.out=5000)
y <- sin(t) + 0.6 * sin(2.6 * t)
x <- y + runif(length(y))
xs <- filt1_gauss(x, width=25)

plot(x, type="l")
lines(xs, col="red")
```

---

filt2 *Smoothing Filters in 2D*

Description

Smooth a uniformly sampled 2D signal.

Usage

```
# Moving average filter
filt2_ma(x, width = 5L)

# Linear convolution filter
filt2_conv(x, weights)
```

```
# Gaussian filter
filt2_gauss(x, width = 5L, sd = (width %/% 2) / 2)

# Bilateral filter
filt2_bi(x, width = 5L, sddist = (width %/% 2) / 2,
    sdrange = mad(x, na.rm = TRUE))

# Bilateral filter with adaptive parameters
filt2_adapt(x, width = 5L, spar = 1)

# Nonlinear diffusion
filt2_diff(x, niter = 3L, kappa = 50,
    rate = 0.25, method = 1L)

# Guided filter
filt2_guide(x, width = 5L, guide = x,
    sdreg = mad(x, na.rm = TRUE))
```

Arguments

`x`	A numeric matrix.
`width`	The width of the smoothing window in number of samples. Must be positive. Must be odd.
`weights`	A matrix of weights for the linear convolution kernel. Dimensions must be odd.
`sd, sddist`	The spatial parameter for kernel-based filters. This controls the strength of smoothing for samples farther from the center of the smoothing window.
`sdrange`	The range parameter for kernel-based filters. This controls the strength of the smoothing for samples with signal values very different from the center of the smoothing window.
`spar`	The strength of the smoothing when calculating the adaptive bilateral filtering parameters. The larger the number, the stronger the smoothing. Must be positive.
`kappa`	The constant for the conduction coefficient for nonlinear diffusion. Must be positive.
`rate`	The rate of diffusion. Must be between 0 and 0.25 for stability.
`method`	The diffusivity method, where `1` and `2` correspond to the two diffusivity functions proposed by Perona and Malik (1990). For `1`, this is `exp(-(|grad x|/K)^2)`, and 2 is `1/(1+(|grad x|/K)^2)`.
`niter`	The number of iterations for nonlinear diffusion. Must be positive.
`guide`	The guide signal for guided filtering. This is the signal used to determine the degree of filtering for different regions of the sample. By default, it is the same as the signal to be smoothed.
`sdreg`	The regularization parameter for guided filtering. This is analagous to the range parameter for kernel-based filters. Signal regions with variance much smaller than this value are smoothed, while signal regions with varaince much larger than this value are preserved.

Details

`filt2_ma()` performs mean filtering in O(n) time. This is fast and especially useful for calculating other filters that can be constructed as a combination of mean filters.

`filt2_gauss()` performs Gaussian filtering.

`filt2_bi()` and `filt2_adapt()` perform edge-preserving bilateral filtering. The latter calculates the kernel parameters adapatively based on the local signal, using a strategy adapted from Joseph and Periyasamy (2018).

`filt2_diff()` performs the nonlinear diffusion filtering of Perona and Malik (1990). Rather than relying on a filter width, it progressively diffuses (smooths) the signal over multiple iterations. More iterations will result in a smoother image.

`filt2_guide()` performs edge-preserving guided filtering. Guided filtering uses a local linear model based on the structure of a so-called "guidance signal". By default, the guidance signal is often the same as the input signal. Guided filtering performs similarly to bilateral filtering, but is often faster (though with more memory use), as it is implemented as a combination of mean filters.

Value

A numeric matrix the same dimensions as `x` with the smoothed result.

Author(s)

Kylie A. Bemis

References

J. Joseph and R. Perisamy. “An image driven bilateral filter with adaptive range and spatial parameters for denoising Magnetic Resonance Images.” Computers and Electrical Engineering, vol. 69, pp. 782-795, July 2018.

P. Perona and J. Malik. “Scale-space and edge detection using anisotropic diffusion.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, issue 7, pp. 629-639, July 1990.

K. He, J. Sun, and X. Tang. “Guided Image Filtering.” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 35, no. 6, pp. 1397-1409, June 2013.

Examples

```
set.seed(1)
i <- seq(-4, 4, length.out=12)
j <- seq(1, 3, length.out=9)
co <- expand.grid(i=i, j=j)
x <- matrix(atan(co$i / co$j), nrow=12, ncol=9)
x <- 10 * (x - min(x)) / diff(range(x))
x <- x + 2.5 * runif(length(x))
xs <- filt2_gauss(x)

par(mfcol=c(1,2))
image(x, col=hcl.colors(256), main="original")
image(xs, col=hcl.colors(256), main="smoothed")
```

`findpeaks` *Peak Detection*

Description

Find peaks in a signal based on its local maxima, as determined by a sliding window.

Usage

```
# Find peaks
findpeaks(x, width = 5L, prominence = NULL,
    snr = NULL, noise = "quant", bounds = TRUE,
    relheight = 0.005, ...)

# Local maxima
locmax(x, width = 5)

# Local minima
locmin(x, width = 5)
```

Arguments

`x`	A numeric vector.
`width`	The number of signal elements to consider when determining if the center of the sliding window is a local extremum.
`prominence`	The minimum peak prominence used for filtering the peaks. The prominence of a peak is the height that the peak rises above the higher of its bases (i.e., its lowest contour line). A peak's bases are found as the local minima between the peak and the next higher peaks on either side.
`snr`	The minimum signal-to-noise ratio used for filtering the peaks.
`noise`	The method used to estimate the noise. See *Details*.
`bounds`	Whether the boundaries of each peak should be calculated and returned. A peak's boundaries are found as the nearest local minima on either side.
`relheight`	The minimum relative height (proportion of the maximum peak value) used for filtering the peaks.
`...`	Arguments passed to the noise estimation function.

Details

For `locmax()` and `locmin()`, a local extremum is defined as an element greater (or less) than all of the elements within `width / 2` elements to the left of it, and greater (or less) than or equal to all of the elements within `width / 2` elements to the right of it.

For `findpeaks()`, the peaks are simply the local maxima of the signal. The peak boundaries are found by descending a local maximum until a local minimum is found on either side, using the same criteria as above. The peaks are optionally filtered based on their prominences.

Optionally, the signal-to-noise ratio (SNR) can be estimated and used for filtering the peaks. These use the functions estnoise_quant, estnoise_diff, estnoise_filt, etc., to estimate the noise in the signal.

Value

For locmax() and locmin(), an logical vector indicating whether each element is a local maximum.

For findpeaks(), an integer vector giving the indices of the peaks, with attributes 'left_bounds' and 'right_bounds' giving the left and right boundaries of the peak as determined using the rule above.

Author(s)

Kylie A. Bemis

See Also

findpeaks_cwt, estnoise_diff, estnoise_quant, estnoise_filt, estnoise_sd, estnoise_mad, peakwidths, peakareas binpeaks, mergepeaks

Examples

```
# simple signal
x <- c(0, 1, 1, 2, 3, 2, 1, 4, 5, 1, 1, 0)
locmax(x)
findpeaks(x)

# simulated spectrum
set.seed(1)
x <- simspec(size=5000)

# find peaks with snr >= 3
p <- findpeaks(x, snr=3, noise="quant")
plot(x, type="l")
points(p, x[p], col="red")

# find peaks with derivative-based noise
p <- findpeaks(x, snr=3, noise="diff")
plot(x, type="l")
points(p, x[p], col="red")
```

---

findpeaks_cwt *CWT-based Peak Detection*

Description

Find peaks in a signal using continuous wavelet transform (CWT).

Usage

```
# Find peaks with CWT
findpeaks_cwt(x, snr = 2, wavelet = ricker, scales = NULL,
    maxdists = scales, ngaps = 3L, ridgelen = length(scales) %/% 4L,
    qnoise = 0.95, width = length(x) %/% 20L, bounds = TRUE)

# Find ridges lines in a matrix
findridges(x, maxdists, ngaps)

# Continuous Wavelet Transform
cwt(x, wavelet = ricker, scales = NULL)
```

Arguments

`x`	A numeric vector for `findpeaks_cwt()` and `cwt()`. A matrix of CWT coefficients for `findridges()`.
`snr`	The minimum signal-to-noise ratio used for filtering the peaks.
`wavelet`	The wavelet to be convolved with the signal. Must be a function that takes two arguments: the number of points in the wavelet `n` as the first argument and the scale `a` of the wavelet as the second argument. The default `ricker()` function satisfies this.
`scales`	The scales at which to perform CWT. A reasonable sequence is generated automatically if not provided.
`maxdists`	The maximum allowed shift distance between local maxima allowed when connecting maxima into ridge lines. Should be a vector the same length as `scales`.
`ngaps`	The number of gaps allowed in a ridge line before it is removed from the search space.
`ridgelen`	The minimum ridge length allowed when filtering peaks.
`qnoise`	The quantile of the CWT coefficients at the smallest scale used to estimate the noise.
`width`	The width of the rolling estimation of noise quantile.
`bounds`	Whether the boundaries of each peak should be calculated and returned. A peak's boundaries are found as the nearest local minima on either side.

Details

`findpeaks_cwt()` uses the peak detection method based on continuous wavelet transform (CWT) proposed by Du, Kibbe, and Lin (2006).

The raw signal is convolved with a wavelet (by default, a Ricker wavelet is used) at a range of different scales. This produces a matrix of CWT coefficients with a number of rows equal to the length of the original signal and each column representing a different scale of convolution.

The convolution at the smallest scales represent a good estimate of noise and peak location. The larger scales represent a smoother signal where larger peaks are prominent and smaller peaks are removed.

The method proceeds by identifying ridge lines in the CWT coefficient matrix using `findridges()`. Local maxima are identified at each scale and connected across each scale, forming the ridge lines.

Finally, the local noise is estimated from the CWT coefficients at the smallest scale. The peaks are filtered based on signal-to-noise ratio and the length of their ridge lines.

Value

For `findpeaks_cwt()`, an integer vector giving the indices of the peaks, with attributes 'left_bounds' and 'right_bounds' giving the left and right boundaries of the peak as determined using the rule above.

For `findridges()`, a list of matrices giving the row and column indices of the entries of each detected ridge line.

Author(s)

Kylie A. Bemis

See Also

`findpeaks`, `peakwidths`, `peakareas`, `binpeaks`, `mergepeaks`

Examples

```
# simple signal
x <- c(0, 1, 1, 2, 3, 2, 1, 4, 5, 1, 1, 0)
locmax(x)
findpeaks(x)

# simulated spectrum
set.seed(1)
x <- simspec(size=5000)

# find peaks with snr >= 3
p <- findpeaks_cwt(x, snr=3)
plot(x, type="l")
points(p, x[p], col="red")

# plot ridges
ridges <- attr(p, "ridges")
plot(c(0, length(x)), c(0, 25), type="n")
for ( ri in ridges )
    lines(ri, type="o", pch=20, cex=0.5)
```

`image` *Display a Color Image from a Formula*

Description

Plot a false-color image using a formula interface.

Usage

```
## S4 method for signature 'formula'
image(x, data = NULL, zlim, xlim, ylim, col = NULL,
        add = FALSE, key = FALSE, zlab, xlab, ylab, ...)
```

Arguments

`x`	A `formula` specifying the image to be plotted. The LHS should be the color variable, and the RHS should include the x and y variables.
`data`	A `data.frame`.
`xlim, ylim, zlim`	The plot limits. See `plot.window`
`col`	A vector giving the color map for encoding the image, or a function that returns a vector of `n` colors.
`add`	Should a new plot be created or should the image be added to an existing plot?
`key`	Should a color key be generated for the image?
`xlab, ylab, zlab`	Plotting labels.
`...`	Additional graphical parameters (as in `par`) or arguments to the `vizi` plotting method.

Value

An object of class `vizi_plot`.

Author(s)

Kylie A. Bemis

See Also

`vizi`, `vizi_pixels`

Examples

```
require(datasets)

coords <- expand.grid(x=1:nrow(volcano), y=1:ncol(volcano))

# volcano image
image(volcano ~ x + y, data=coords, zlab="elevation", key=TRUE)

volcano2 <- trans2d(volcano, rotate=15, translate=c(-5, 5))
df <- data.frame(
    x=c(coords$x, coords$x),
    y=c(coords$y, coords$y),
    z=c(as.vector(volcano), as.vector(volcano2)),
    g=rep(c("original", "transformed"), each=nrow(coords)))

# plot original and transformed images
image(z ~ x + y | g, data=df, col=cpal("plasma"))
```

---

inpoly *Point in polygon*

Description

Check if a series of x-y points are contained in a closed 2D polygon.

Usage

```
inpoly(points, poly)
```

Arguments

`points`	A 2-column numeric matrix with the points to check.
`poly`	A 2-column numeric matrix with the vertices of the polygon.

Details

This function works by extending a horizontal ray from each point and counting the number of times it crosses an edge of the polygon.

Value

A logical vector that is `TRUE` for points that are fully inside the polygon, a vertex, or on an edge, and `FALSE` for points fully outside the polygon.

Note

There are various public implementations of this function with no clear original source. The version implemented here is loosely based on code by W. Randolph Franklin with modifications so that vertices and points on edges are considered *inside* the polygon.

Author(s)

W. R. Franklin and Kylie A. Bemis

References

W. R. Franklin. “PNPOLY - Point Inclusion in Polygon Test.” https://wrfranklin.org/Research/Short_Notes/pnpoly.html, 1970.

M. Shimrat, "Algorithm 112, Position of Point Relative to Polygon", Comm. ACM 5(8), pp. 434, Aug 1962.

E. Haines. “Point in Polygon Strategies.” http://www.acm.org/pubs/tog/editors/erich/ptinpoly/, 1994.

See Also

kdsearch

Examples

```
poly <- data.frame(
        x=c(3,5,5,3),
        y=c(3,3,5,5))
xy <- data.frame(
    x=c(4,6,4,2,3,5,5,3,4,5,4,4,
        3.5,4.5,4.0,3.5),
    y=c(2,4,6,4,3,3,5,5,3,4,5,3,
        4.0,4.0,4.5,4.0),
    ref=c(
        rep("out", 4),
        rep("vertex", 4),
        rep("edge", 4),
        rep("in", 4)))

xy$test <- inpoly(xy[,1:2], poly)
xy
```

---

knnsearch *K-Dimensional Nearest Neighbor Search*

Description

Search a matrix of K-dimensional data points and return the indices of the nearest neighbors or of all data points that are within a specified tolerance in each dimension.

Usage

```
# Range search
kdsearch(x, data, tol = 0, tol.ref = "abs")

# Nearest neighbor search
knnsearch(x, data, k = 1L, metric = "euclidean", p = 2)

# Nearest neighbor pairs
nnpairs(x, y, metric = "euclidean", p = 2)

# K-D tree
kdtree(data)
```

Arguments

`x, y`	A numeric matrix of coordinates to be matched. Each column should be dimension. Each row should be a query point.
`data`	Either a `kdtree` object returned by `kdtree()`, or a numeric matrix of coordinates to search, where each column is a different dimension.
`k`	The number of nearest neighbors to find for each point (row) in `x`.
`metric`	Distance metric to use when finding the nearest neighbors. Supported metrics include "euclidean", "maximum", "manhattan", and "minkowski".
`p`	The power for the Minkowski distance.
`tol`	The tolerance for finding neighboring points in each dimension. May be a vector with the same length as the number of dimensions. Must be positive.
`tol.ref`	One of 'abs', 'x', or 'y'. If 'abs', then comparison is done by taking the absolute difference. If either 'x' or 'y', then relative differences are used, and this specifies which to use as the reference (target) value.

Details

`knnsearch()` performs k-nearest neighbor searches. `kdsearch()` performs range searches for points within a given tolerance of the query points. `nnpairs()` finds the nearest neighbor for each point between two datasets.

The algorithm is implemented in C and works by building a kd-tree to perform the search. If multiple calls to `kdsearch()` or `knnsearch()` are expected on the same data, it can be much faster to build the tree once with `kdtree()`.

A kd-tree is essentially a multidimensional generalization of a binary search tree. Building the search tree is O(n * log n) and searching for a single data point is O(log n).

Value

For `kdsearch()`, a list with length equal to the number of rows of `x`, where each list element is a vector of indexes of the matches in `data`.

For `knnsearch()`, a matrix with rows equal to the number of rows of `x` and columns equal to `k` giving the indices of the k-nearest neighbors.

For `nnpairs()`, a matrix where each column gives the indices of nearest neighbor pairs.

Author(s)

Kylie A. Bemis

See Also

asearch, bsearch, approx2,

Examples

```
d <- expand.grid(x=1:10, y=1:10)
x <- rbind(c(1.11, 2.22), c(3.33, 4.44))

knnsearch(x, d, k=3)
```

---

matter-class   *Vectors, Matrices, and Arrays Stored in Virtual Memory*

Description

The matter class and its subclasses are designed for easy on-demand read/write access to binary virtual memory data structures, and working with them as vectors, matrices, arrays, lists, and data frames.

Usage

```
## Instance creation
matter(...)

# Check if an object is a matter object
is.matter(x)

# Coerce an object to a matter object
as.matter(x)

## Additional methods documented below
```

Arguments

...	Arguments passed to subclasses.
x	An object to check if it is a matter object or coerce to a matter object.

Value

An object of class matter.

Slots

`data`**:** This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

`type`**:** The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

`dim`**:** Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

`names`**:** The names of the data elements for vectors.

`dimnames`**:** Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

Creating Objects

`matter` is a virtual class and cannot be instantiated directly, but instances of its subclasses can be created through `matter()`.

Methods

Class-specific methods:

`atomdata(x)`**:** Access the 'data' slot.

`adata(x)`**:** An alias for atomdata(x).

`type(x), type(x) <- value`**:** Get or set data 'type'.

Standard generic methods:

`length(x), length(x) <- value`**:** Get or set length.

`dim(x), dim(x) <- value`**:** Get or set 'dim'.

`names(x), names(x) <- value`**:** Get or set 'names'.

`dimnames(x), dimnames(x) <- value`**:** Get or set 'dimnames'.

Author(s)

Kylie A. Bemis

See Also

`matter_arr`, `matter_mat`, `matter_vec`, `matter_fct`, `matter_list`, `matter_str`

Examples

```
## Create a matter_vec vector
x <- matter(1:100, length=100)
x

## Create a matter_mat matrix
y <- matter(1:100, nrow=10, ncol=10)
y
```

matter-options *Options for "matter" Objects*

Description

The `matter` package provides the following options:

- `options(matter.compress.atoms=3)`:The compression ratio threshold to be used to determine when to compress atoms in a matter object. Setting to 0 or `FALSE` means that atoms are never compressed.
- `options(matter.default.nchunks=20L)`:The default number of chunks to use when iterating over matter objects.
- `options(matter.default.verbose=FALSE)`:The default verbosity for printing progress messages.
- `options(matter.matmul.bpparam=NULL)`:An optional `BiocParallelParam` passed to `bplapply` used when performing matrix multiplication with `matter_mat` and `sparse_mat` objects.
- `options(matter.show.head=TRUE)`:Should a preview of the beginning of the data be displayed when the object is printed?
- `options(matter.show.head.n=6)`:The number of elements, rows, and/or columns to be displayed by the object preview.
- `options(matter.coerce.altrep=FALSE)`:When coercing `matter` objects to native R objects (such as `matrix`), should a `matter`-backed ALTREP object be returned instead? The initial coercion will be cheap, and the result will look like a native R object. This does not guarantee that the full data is never read into memory. Not all functions are ALTREP-aware at the C-level, so some operations may still trigger the full data to be read into memory. This should only ever happen once, as long as the object is not duplicated, though.
- `options(matter.wrap.altrep=FALSE)`:When coercing to a `matter`-backed ALTREP object, should the object be wrapped in an ALTREP wrapper? (This is always done in cases where the coercion preserves existing attributes.) This allows setting of attributes without triggering a (potentially expensive) duplication of the object when safe to do so.
- `options(matter.dump.dir=tempdir())`:Temporary directory where `matter` object files should be dumped when created without user-specified file paths.

matter-types *Data Types for "matter" Objects*

Description

The `matter` package defines a number of data types for translating between data elements stored in virtual memory and data elements loaded into R. These are typically set and stored via the `datamode` argument and slot.

At the R level, `matter` objects may be any of the following data modes:

- raw:matter objects of this mode are typically vectors of raw bytes.
- logical:Any matter object that represents a logical vector or has had any Compare or Logic delayed operations applied to it will be of this type.
- integer:matter objects represented as integers in R.
- numeric:matter objects represented as doubles in R.
- character:matter objects representated as character vectors in R.

In virtual memory, matter objects may be composed of atomic units of the following data types:

- char:8-bit signed integer; defined as char.
- uchar:8-bit unsigned integer; used for 'Rbyte' or 'raw'; defined as unsigned char.
- int16:16-bit signed integer; defined as int16_t. May be aliased as 'short' and '16-bit integer'.
- uint16:16-bit unsigned integer; defined as uint16_t. May be aliased as 'ushort' and '16-bit unsigned integer'.
- int32:32-bit signed integer; defined as int32_t. May be aliased as 'int' and '32-bit integer'.
- uint32:32-bit unsigned integer; defined as uint32_t. May be aliased as 'uint' and '32-bit unsigned integer'.
- int64:64-bit signed integer; defined as int64_t. May be aliased as 'long' and '64-bit integer'.
- uint64:64-bit unsigned integer; defined as uint64_t. May be aliased as 'ulong' and '64-bit unsigned integer'.
- float32:32-bit float; defined as float. May be aliased as 'float' and '32-bit float'.
- float64:64-bit float; defined as double. May be aliased as 'double' and '64-bit float'.

While a substantial effort is made to coerce data elements properly between data types, sometimes this cannot be done losslessly. Loss of precision is silent, while values outside of the representable range will generate a warning (sometimes *many* such warnings) and will be set to NA if available or 0 otherwise.

Note that the unsigned data types do not support NA; coercion between signed integer types attempts to preserve missingness. The special values NaN, Inf, and -Inf are only supported by the floating-point types, and will be set to NA for signed integral types, and to 0 for unsigned integral types.

---

matter-utils *Internal Utilities for "matter" Package*

Description

Low-level utility functions, classes, and data defined in the **matter** package. They are not intended to be used directly.

matter_arr-class *Out-of-Memory Arrays*

Description

The matter_arr class implements out-of-memory arrays.

Usage

```
## Instance creation
matter_arr(data, type = "double", path = NULL,
    dim = NA_integer_, dimnames = NULL, offset = 0, extent = NA_real_,
    readonly = NA, append = FALSE, rowMaj = FALSE, ...)

matter_mat(data, type = "double", path = NULL,
    nrow = NA_integer_, ncol = NA_integer_, dimnames = NULL,
    offset = 0, extent = NA_real_, readonly = NA,
    append = FALSE, rowMaj = FALSE, ...)

matter_vec(data, type = "double", path = NULL,
    length = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, append = FALSE, rowMaj = FALSE, ...)

## Additional methods documented below
```

Arguments

data — An optional data vector which will be initially written to virtual memory if provided.

type — A 'character' vector giving the storage mode of the data in virtual memory such. See ?"matter-types" for possible values.

path — A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.

dim, nrow, ncol, length — The dimensions of the array, or the number of rows and columns, or the length.

dimnames, names — The names of the matrix dimensions or vector elements.

offset — A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.

extent — A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.

readonly — Whether the data and file(s) should be treated as read-only or read/write.

append — If TRUE, then all offsets will be adjusted to be from the *end-of-file* (for all files in path), and readonly will be set to FALSE.

rowMaj	Whether the data is stored in row-major or column-major order. The default is to use column-major order, which is the same as native R matrices.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_arr.

Slots

- **data:** This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).
- **type:** The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.
- **dim:** Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.
- **names:** The names of the data elements for vectors.
- **dimnames:** Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.
- **ops:** Deferred arithmetic operations.
- **transpose:** Indicates whether the data is stored in row-major order (TRUE) or column-major order (FALSE). For a matrix, switching the order that the data is read is equivalent to transposing the matrix (without changing any data).
- **indexed:** For matter_mat only. Indicates whether the pointers to rows or columns are indexed for quick access or not.

Extends

matter

Creating Objects

matter_arr instances can be created through matter_arr() or matter(). Matrices and vectors can also be created through matter_mat() and matter_vec().

Methods

Standard generic methods:

- **length(x), length(x) <- value:** Get or set length.
- **dim(x), dim(x) <- value:** Get or set 'dim'.
- **names(x), names(x) <- value:** Get or set 'names'.
- **dimnames(x), dimnames(x) <- value:** Get or set 'dimnames'.
- **x[...], x[...] <- value:** Get or set the elements of the array.
- **cbind(x, ...), rbind(x, ...):** Combine matrices by row or column.

t(x): Transpose a matrix. This is a quick operation which only changes metadata and does not touch the data representation.

rowMaj(x): Check the data orientation.

Author(s)

Kylie A. Bemis

See Also

matter

Examples

```
x <- matter_arr(1:1000, dim=c(10,10,10))
x
```

---

matter_fct-class *Out-of-Memory Factors*

Description

The matter_fct class implements out-of-memory factors.

Usage

```
## Instance creation
matter_fct(data, levels, path = NULL,
    length = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, append = FALSE, labels = as.character(levels), ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to the data in virtual memory if provided.
levels	The levels of the factor. These should be of the same type as the data. (Use labels for the string representation of the levels.)
path	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.
length	The length of the factor.
names	The names of the data elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.

extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
append	If TRUE, then all offsets will be adjusted to be from the *end-of-file* (for all files in path), and readonly will be set to FALSE.
labels	An optional character vector of labels for the factor levels.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_fct.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

levels: The levels of the factor.

labels: The labels for the levels.

Extends

matter, matter_vec

Creating Objects

matter_fct instances can be created through matter_fct() or matter().

Methods

Standard generic methods:

length(x), length(x) <- value: Get or set length.

names(x), names(x) <- value: Get or set 'names'.

x[i], x[i] <- value: Get or set the elements of the factor.

levels(x), levels(x) <- value: Get or set the levels of the factor.

Author(s)

Kylie A. Bemis

See Also

matter, matter_vec

Examples

```
x <- matter_fct(rep(c("a", "a", "b"), 5), levels=c("a", "b", "c"))
x
```

---

matter_list-class *Out-of-Memory Lists of Vectors*

Description

The matter_list class implements out-of-memory lists.

Usage

```
## Instance creation
matter_list(data, type = "double", path = NULL,
    lengths = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, append = FALSE, ...)

## Additional methods documented below
```

Arguments

data	An optional data vector which will be initially written to virtual memory if provided.
type	A 'character' vector giving the storage mode of the data in virtual memory. See ?"matter-types" for possible values.
path	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using tempfile.
lengths	The lengths of the list elements.
names	The names of the list elements.
offset	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
extent	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
readonly	Whether the data and file(s) should be treated as read-only or read/write.
append	If TRUE, then all offsets will be adjusted to be from the *end-of-file* (for all files in path), and readonly will be set to FALSE.
...	Additional arguments to be passed to constructor.

Value

An object of class matter_list.

Slots

data: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.

Extends

matter

Creating Objects

`matter_list` instances can be created through `matter_list()` or `matter()`.

Methods

Standard generic methods:

`x[[i]]`, `x[[i]] <- value`: Get or set a single element of the list.

`x[[i, j]]`: Get the `j`th sub-elements of the `i`th element of the list.

`x[i]`, `x[i] <- value`: Get or set the `i`th elements of the list.

`lengths(x)`: Get the lengths of all elements in the list.

Author(s)

Kylie A. Bemis

See Also

matter

Examples

```
x <- matter_list(list(c(TRUE,FALSE), 1:5, c(1.11, 2.22, 3.33)), lengths=c(2,5,3))
x[]
x[1]
x[[1]]

x[[3,1]]
x[[2,1:3]]
```

---

matter_str-class *Out-of-Memory Strings*

Description

The `matter_str` class implements out-of-memory strings.

Usage

```
## Instance creation
matter_str(data, encoding, path = NULL,
    nchar = NA_integer_, names = NULL, offset = 0, extent = NA_real_,
    readonly = NA, append = FALSE, ...)

## Additional methods documented below
```

Arguments

`data`	An optional data vector which will be initially written to virtual memory if provided.
`encoding`	The character encoding to use (if known).
`path`	A 'character' vector of the path(s) to the file(s) where the data are stored. If 'NULL', then a temporary file is created using `tempfile`.
`nchar`	A vector giving the length of each element of the character vector.
`names`	The names of the data elements.
`offset`	A vector giving the offsets in number of bytes from the beginning of each file in 'path', specifying the start of the data to be accessed for each file.
`extent`	A vector giving the length of the data for each file in 'path', specifying the number of elements of size 'type' to be accessed from each file.
`readonly`	Whether the data and file(s) should be treated as read-only or read/write.
`append`	If `TRUE`, then all offsets will be adjusted to be from the *end-of-file* (for all files in `path`), and `readonly` will be set to `FALSE`.
`...`	Additional arguments to be passed to constructor.

Value

An object of class `matter_str`.

Slots

- `data`: This slot stores any information necessary to access the data for the object (which may include the data itself and/or paths to file locations, etc.).
- `type`: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.
- `dim`: Either 'NULL' for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.
- `names`: The names of the data elements for vectors.
- `dimnames`: Either 'NULL' or the names for the dimensions. If not 'NULL', then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always 'NULL' for vectors.
- `encoding`: The string encoding used.

Extends

`matter_list`

Creating Objects

`matter_str` instances can be created through `matter_str()` or `matter()`.

Methods

Standard generic methods:

- `x[i], x[i] <- value`: Get or set the string elements of the vector.
- `lengths(x)`: Get the number of characters (in bytes) of all string elements in the vector.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matter_str(rep(c("hello", "world!"), 50))
x
```

memtime *Check Memory Use*

Description

These are utility functions for checking memory used by objects and by R during the execution of an expression.

Usage

```
mem(x, reset = FALSE)

memtime(expr)
```

Arguments

x	An object, to identify how much memory it is using.
reset	Should the maximum memory used by R be reset?
expr	An expression to be evaluated.

Details

These are wrappers around the built-in gc function. Note that they only count memory managed by R.

Value

For memtime, a vector giving [1] the amount of memory used at the start of execution, [2] the amount of memory used at the end of execution, [3] the maximum amount of memory used during execution, [4] the memory overhead as defined by the maximum memory used minus the starting memory use, and [5] the execution time in seconds.

For mem, either a single numeric value giving the memory used by an object, or a vector providing a more readable version of the information returned by gc (see its help page for details).

Author(s)

Kylie A. Bemis

See Also

gc,

Examples

```
x <- 1:100

mem(x)

memtime(mean(x + 1))
```

---

nnmf *Nonnegative Matrix Factorization*

Description

Nonnegative matrix factorization (NMF) decomposes a nonnegative data matrix into a matrix of basis variables and a matrix of activations (or coefficients). The factorization is approximate and may be less accurate than alternative methods such as PCA, but can greatly improve the interpretability of the reduced dimensions.

Usage

```
# Alternating least squares
nnmf_als(x, k = 3L, s = 1e-9, transpose = FALSE,
niter = 100L, tol = 1e-5, verbose = NA, ...)

# Multiplicative updates
nnmf_mult(x, k = 3L, s = 1e-9, cost = c("euclidean", "KL", "IS"),
transpose = FALSE, niter = 100L,
tol = 1e-5, verbose = NA, ...)

## S3 method for class 'nnmf'
predict(object, newdata, ...)

# Nonnegative double SVD
nndsvd(x, k = 3L, ...)
```

Arguments

`x`	A nonnegative matrix.
`k`	The number of NMF components to extract.
`s`	A regularization parameter to prevent singularities.
`transpose`	A logical value indicating whether `x` should be considered transposed or not. This can be useful if the input matrix is (P x N) instead of (N x P) and storing the transpose is expensive. This is not necessary for `matter_mat` and `sparse_mat` objects, but can be useful for large in-memory (P x N) matrices.
`niter`	The maximum number of iterations.
`tol`	The tolerance for convergence, as measured by the Frobenius norm of the differences between the W and H matrices in successive iterations.

verbose	Should progress be printed for each iteration?
cost	The cost function (i.e., error measure between the reconstructed matrix and original x) to optimize, where 'euclidean' is the Frobenius norm, 'KL' is the Kullback-Leibler divergence, and 'IS' is the Itakura-Saito divergence. See **Details**.
...	Additional options passed to irlba.
object	An object inheriting from nmf.
newdata	An optional data matrix to use for the prediction.

Details

These functions implement nonnegative matrix factorization (NMF) using either alternating least squares as described by Berry at al. (2007) or multiplicative updates from Lee and Seung (2000) and further described by Burred (2014). The algorithms are initialized using nonnegative double singular value decomposition (NNDSVD) from Boutsidis and Gallopoulos (2008).

The algorithm using multiplicative updates (`nnmf_mult()`) tends to be more stable but converges more slowly. The alternative least squares algorithm (`nnmf_als()`) tends to converge faster to more accurate results, but can be less numerically stable than the multiplicative updates algorithm.

Note for `nnmf_mult()` that `method = "euclidean"` is the only method that can handle out-of-memory `matter_mat` and `sparse_mat` matrices. `x` will be coerced to an in-memory matrix for other methods.

Value

An object of class `nnmf`, with the following components:

- `activation`: The (transposed) coefficient matrix (H).
- `x`: The basis variable matrix (W).
- `iter`: The number of iterations performed.

Author(s)

Kylie A. Bemis

References

M. W. Berry, M. Browne, A. N. Langville, V. P. Pauca, R. J. Plemmons. “Algorithms and applications for approximate nonnegative matrix factorization.” Computational Statistics and Data Analysis, vol. 52, issue 1, pp. 155-173, Sept. 2007.

D. D. Lee and H. S. Seung. “Algorithms for non-negative matrix factorization.” Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS), pp. 535-541, Jan. 2000.

C. Boutsidis and E. Gallopoulos. “SVD based initialization: A head start for nonnegative matrix factorization.” Pattern Recognition, vol. 41, issue 4, pp. 1350-1362, Apr. 2008.

J. J. Burred. “Detailed derivation of multiplicative update rules for NMF.” Techical report, Paris, March 2014.

See Also

svd, prcomp

Examples

```
set.seed(1)

a <- matrix(sort(runif(500)), nrow=50, ncol=10)
b <- matrix(rev(sort(runif(500))), nrow=50, ncol=10)
x <- cbind(a, b)

mf <- nnmf_als(x, k=3)
```

---

peakwidths *Peak Summarization*

Description

Summarize peaks based on their shapes and properties.

Usage

```
# Get peak widths
peakwidths(x, peaks, domain = NULL,
    fmax = 0.5, ref = c("height", "prominence"))

# Get peak areas
peakareas(x, peaks, domain = NULL)
```

Arguments

x	A numeric vector.
peaks	The indices (or domain values) of peaks for which the widths or areas should be calculated.
domain	The domain variable of the signal.
fmax	The fraction of the peak's height used for determining the peak's width.
ref	The reference value of the peak for determining the peak width: either the peak height of the peak prominence.

Value

A numeric vector giving the widths or areas of the peaks with attributes 'left_bounds' and 'right_bounds' giving the left and right boundaries of the peak.

Author(s)

Kylie A. Bemis

See Also

findpeaks, findpeaks_cwt, binpeaks, mergepeaks

Examples

```
x <- c(0, 1, 1, 2, 3, 2, 1, 4, 5, 1, 1, 0)

p <- findpeaks(x)

peakareas(x, p)
```

---

plot-vizi *Plotting Graphical Marks*

Description

These functions provide plotting methods for various graphical marks. They are not intended to be called directly.

Usage

```
## S3 method for class 'vizi_points'
plot(x, plot = NULL, add = FALSE, ...,
    n = Inf, downsampler = "lttb")

## S3 method for class 'vizi_lines'
plot(x, plot = NULL, add = FALSE, ...,
    n = Inf, downsampler = "lttb")

## S3 method for class 'vizi_peaks'
plot(x, plot = NULL, add = FALSE, ...,
    n = Inf, downsampler = "lttb")

## S3 method for class 'vizi_pixels'
plot(x, plot = NULL, add = FALSE, ...,
    enhance = FALSE, smooth = FALSE, scale = FALSE,
    useRaster = TRUE)

## S3 method for class 'vizi_voxels'
plot(x, plot = NULL, add = FALSE, ...,
    xslice = NULL, yslice = NULL, zslice = NULL)
```

Arguments

x	A graphical mark.
plot	A vizi_plot object.
add	Should the plot be added to the current panel or start a new panel?

`...`	Additional graphical parameters passed to the underlying base graphics plotting function.
`n`	Maximum number of points to plot. This is useful for downsampling series with far more data points than are useful to plot. See `downsample` for details.
`downsampler`	If `n` is less than the number of points, then this is the downsampling method to use. See `downsample` for details.
`enhance`	The name of a contrast enhancement method, such as `"hist"` or `"adapt"` for `enhance_hist()` and `enhance_adapt()`, etc. See `enhance` for details.
`smooth`	The name of a smoothing method, such as `"gauss"` or `"bi"` for `filt2_gauss()` and `filt2_bi()`, etc. See `filt2` for details.
`scale`	If `TRUE`, then all image values will be scaled to the range [0, 100]. This is useful for comparing images with differing intensity levels across facets or layers.
`useRaster`	Should a bitmap raster be used for plotting? This is typically faster on supported devices. A fallback to polygon-based plotting is used if raster plotting is not supported.
`xslice, yslice, zslice`	Numeric vectors giving the x, y, and/or z coordinates of the volumetric slices to plot. If none are provided, defaults to plotting all z-slices.

Author(s)

Kylie A. Bemis

See Also

`vizi`, `add_mark`

`pls` *Partial Least Squares*

Description

Partial least squares (PLS), also called projection to latent structures, performs multivariate regression between a data matrix and a response matrix by decomposing both matrixes in a way that explains the maximum amount of covariation between them. It is especially useful when the number of predictors is greater than the number of observations, or when the predictors are highly correlated. Orthogonal partial least squares (OPLS) is also provided.

Usage

```
# NIPALS algorithm
pls_nipals(x, y, k = 3L, center = TRUE, scale. = FALSE,
transpose = FALSE, niter = 100L, tol = 1e-5,
verbose = NA, ..., BPPARAM = bpparam())
```

```
# SIMPLS algorithm
pls_simpls(x, y, k = 3L, center = TRUE, scale. = FALSE,
transpose = FALSE, method = 1L, retscores = TRUE,
verbose = NA, ..., BPPARAM = bpparam())

# Kernel algorithm
pls_kernel(x, y, k = 3L, center = TRUE, scale. = FALSE,
transpose = FALSE, method = 1L, retscores = TRUE,
verbose = NA, ..., BPPARAM = bpparam())

## S3 method for class 'pls'
predict(object, newdata, k,
type = c("response", "class"), ..., BPPARAM = bpparam())

# O-PLS algorithm
opls_nipals(x, y, k = 3L, center = TRUE, scale. = FALSE,
transpose = FALSE, niter = 100L, tol = 1e-9,
verbose = NA, ..., BPPARAM = bpparam())

## S3 method for class 'opls'
predict(object, newdata, k, ..., BPPARAM = bpparam())

# Variable importance in projection
vip(object, type = c("projection", "weights"))
```

Arguments

`x`	The data matrix of predictors.
`y`	The response matrix. (Can also be a factor.)
`k`	The number of PLS components to use.
`center`	A logical value indicating whether the variables should be shifted to be zero-centered, or a centering vector of length equal to the number of columns of `x`. The centering is performed implicitly and does not change the out-of-memory data in `x`.
`scale.`	A logical value indicating whether the variables should be scaled to have unit variance, or a scaling vector of length equal to the number of columns of `x`. The scaling is performed implicitly and does not change the out-of-memory data in `x`.
`transpose`	A logical value indicating whether `x` should be considered transposed or not. This can be useful if the input matrix is (P x N) instead of (N x P) and storing the transpose is expensive. This is not necessary for `matter_mat` and `sparse_mat` objects, but can be useful for large in-memory (P x N) matrices.
`niter`	The maximum number of iterations (per component).
`tol`	The tolerance for convergence (per component).
`verbose`	Should progress be printed for each iteration?

`method`	The kernel algorithm to use, where `1` and `2` correspond to the two kernel algorithms described by Dayal and MacGregor (1997). For `1`, only of the covariance matrix `t(X) %*% Y` is computed. For `2`, the variance matrix `t(X) %*% X` is also computed. Typically `1` will be faster if the number of predictors is large. For a smaller number of predictors, `2` will be more efficient.
`retscores`	Should the scores be computed and returned? This also computes the amount of explained covariance for each component. This is done automatically for NIPALS, but requires additional computation for the kernel algorithms.
`...`	Not currently used.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`. Currently only used for centering and scaling. Use `options(matter.matmul.bpparam=TRUE)` to enable parallel matrix multiplication for `matter_mat` and `sparse_mat` matrices.
`object`	An object inheriting from `pls` or `opls`.
`newdata`	An optional data matrix to use for the prediction.
`type`	The type of prediction, where `"response"` means the fitted response matrix and `"class"` will be the vector of class predictions (only valid for discriminant analyses).

Details

These functions implement partial least squares (PLS) using the original NIPALS algorithm by Wold et al. (1983), the SIMPLS algorithm by de Jong (1993), or the kernel algorithms by Dayal and MacGregor (1997). A function for calculating orthogonal partial least squares (OPLS) processing using the NIPALS algorithm by Trygg and Wold (2002) is also provided.

Both regression and classification can be performed. If passed a `factor`, then partial least squares discriminant analysis (PLS-DA) will be performed as described by M. Barker and W. Rayens (2003).

The SIMPLS algorithm (`pls_simpls()`) is relatively fast as it does not require the deflation of the data matrix. However, the results will differ slightly from the NIPALS and kernel algorithms for multivariate responses. In these cases, only the first component will be identical. The differences are not meaningful in most cases, but it is worth noting.

The kernel algorithms (`pls_kernel()`) tend to be faster than NIPALS for larger data matrices. The original NIPALS algorithm (`pls_nipals()`) is the reference implementation. The results from these algorithms are proven to be equivalent for both univariate and multivariate responses.

Note that the NIPALS algorithms cannot handle out-of-memory `matter_mat` and `sparse_mat` matrices due to the need to deflate the data matrix for each component. `x` will be coerced to an in-memory matrix.

Variable importance in projection (VIP) scores proposed by Wold et al. (1993) measure of the influence each variable has on the PLS model. They can be calculated with `vip()`. Note that non-NIPALS models must have `retscores = TRUE` for VIP to be calculated. In practice, a VIP score greater than ~1 is a useful criterion for variable selection, although there is no statistical basis for this rule.

Value

An object of class `pls`, with the following components:

- `coefficients`: The regression coefficients.
- `projection`: The projection weights of the regression used to calculate the coefficients from the y-loadings or to project the data to the scores.
- `residuals`: The residuals from regression.
- `fitted.values`: The fitted y matrix.
- `weights`: (Optional) The x-weights of the regression.
- `loadings`: The x-loadings of the latent variables.
- `scores`: (Optional) The x-scores of the latent variables.
- `y.loadings`: The y-loadings of the latent variables.
- `y.scores`: (Optional) The y-scores of the latent variables.
- `cvar`: (Optional) The covariance explained by each component.

Or, an object of class `opls`, with the following components:

- `weights`: The orthogonal x-weights.
- `loadings`: The orthogonal x-loadings.
- `scores`: The orthogonal x-scores.
- `ratio`: The ratio of the orthogonal weights to the PLS loadings for each component. This provides a measure of how much orthogonal variation is being removed by each component and can be interpreted as a scree plot similar to PCA.
- `x`: The processed data matrix with orthogonal variation removed.

Author(s)

Kylie A. Bemis

References

S. Wold, H. Martens, and H. Wold. “The multivariate calibration method in chemistry solved by the PLS method.” Proceedings on the Conference on Matrix Pencils, Lecture Notes in Mathematics, Heidelberg, Springer-Verlag, pp. 286 - 293, 1983.

S. de Jong. “SIMPLS: An alternative approach to partial least squares regression.” Chemometrics and Intelligent Laboratory Systems, vol. 18, issue 3, pp. 251 - 263, 1993.

B. S. Dayal and J. F. MacGregor. “Improved PLS algorithms.” Journal of Chemometrics, vol. 11, pp. 73 - 85, 1997.

M. Barker and W. Rayens. “Partial least squares for discrimination.” Journal of Chemometrics, vol. 17, pp. 166-173, 2003.

J. Trygg and S. Wold. “Orthogonal projections to latent structures.” Journal of Chemometrics, vol. 16, issue 3, pp. 119 - 128, 2002.

S. Wold, A. Johansson, and M. Cocchi. “PLS: Partial least squares projections to latent structures.” 3D QSAR in Drug Design: Theory, Methods and Applications, ESCOM Science Publishers: Leiden, pp. 523 - 550, 1993.

See Also

prcomp

Examples

```
register(SerialParam())

x <- cbind(
c(-2.18, 1.84, -0.48, 0.83),
c(-2.18, -0.16, 1.52, 0.83))
y <- as.matrix(c(2, 2, 0, -4))

pls_nipals(x, y, k=2)
```

---

prcomp — *Principal Components Analysis for "matter" Matrices*

Description

This method allows computation of a truncated principal components analysis of `matter_mat` and `sparse_mat` matrices using the implicitly restarted Lanczos method from the "irlba" package.

Usage

```
## S4 method for signature 'matter_mat'
prcomp(x, k = 3L, retx = TRUE, center = TRUE, scale. = FALSE, ...)

## S4 method for signature 'sparse_mat'
prcomp(x, k = 3L, retx = TRUE, center = TRUE, scale. = FALSE, ...)

prcomp_lanczos(x, k = 3L, retx = TRUE,
    center = TRUE, scale. = FALSE, transpose = FALSE,
    verbose = NA, ..., BPPARAM = bpparam())
```

Arguments

`x`	A `matter` matrix, or any matrix-like object for `prcomp_lanczos`.
`k`	The number of principal components to return, must be less than `min(dim(x))`.
`retx`	A logical value indicating whether the rotated variables should be returned.
`center`	A logical value indicating whether the variables should be shifted to be zero-centered, or a centering vector of length equal to the number of columns of `x`. The centering is performed implicitly and does not change the out-of-memory data in `x`.
`scale.`	A logical value indicating whether the variables should be scaled to have unit variance, or a scaling vector of length equal to the number of columns of `x`. The scaling is performed implicitly and does not change the out-of-memory data in `x`.

transpose	A logical value indicating whether x should be considered transposed or not. This can be useful if the input matrix is (P x N) instead of (N x P) and storing the transpose is expensive. This is not necessary for matter_mat and sparse_mat objects, but can be useful for large in-memory (P x N) matrices.
verbose	Should progress messages be printed?
...	Additional options passed to irlba or prcomp_lanczos.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply. Currently only used for centering and scaling. Use options(matter.matmul.bpparam=TRUE) to enable parallel matrix multiplication for matter_mat and sparse_mat matrices.

Value

An object of class 'prcomp'. See ?prcomp for details.

Note

The built-in predict() method (from the stats package) is not compatible with the argument transpose=TRUE.

Author(s)

Kylie A. Bemis

See Also

irlba prcomp_irlba

Examples

```
register(SerialParam())
set.seed(1)

x <- matter_mat(rnorm(1000), nrow=100, ncol=10)

prcomp(x)
```

---

rollvec *Rolling Summaries of a Vector*

Description

Summarize a vector in rolling windows.

Usage

```
rollvec(x, width, stat = "sum", prob = 0.5)
```

Arguments

x	A numeric vector.
width	The width of the rolling window. Must be odd.
stat	The statistic used to summarize the values in each bin. Must be one of "sum", "mean", "max", "min", "sd", "var", "mad", or "quantile".
prob	The quantile for `stat = "quantile"`.

Value

An numeric vector with the same length as `x` with the summarized values from each rolling window.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)

x <- sort(runif(20))

rollvec(x, 5L, "mean")
```

---

rowDists *Compute Distances between Rows or Columns of a Matrix*

Description

Compute and return the distances between specific rows or columns of matrices according to a specific distance metric.

Usage

```
## S4 method for signature 'ANY,missing'
rowDists(x, at, ..., BPPARAM = bpparam())

## S4 method for signature 'ANY,missing'
colDists(x, at, ..., BPPARAM = bpparam())

## S4 method for signature 'matrix,matrix'
rowDists(x, y, ..., BPPARAM = bpparam())

## S4 method for signature 'matrix,matrix'
colDists(x, y, ..., BPPARAM = bpparam())

## S4 method for signature 'matter_mat,matrix'
```

```
rowDists(x, y, ..., BPPARAM = bpparam())

## S4 method for signature 'matrix,matter_mat'
colDists(x, y, ..., BPPARAM = bpparam())

## S4 method for signature 'sparse_mat,matrix'
rowDists(x, y, ..., BPPARAM = bpparam())

## S4 method for signature 'matrix,sparse_mat'
colDists(x, y, ..., BPPARAM = bpparam())

rowdist(x, y = x, metric = "euclidean", p = 2, weights = NULL)

coldist(x, y = x, metric = "euclidean", p = 2, weights = NULL)

rowdist_at(x, ix, y = x, iy = list(1L:nrow(y)),
metric = "euclidean", p = 2, weights = NULL)

coldist_at(x, ix, y = x, iy = list(1L:ncol(y)),
metric = "euclidean", p = 2, weights = NULL)
```

Arguments

x, y	Numeric matrices for which distances should be calculated according to rows or columns.
at	A list of specific row or column indices for which to calculate the distances. Each row or column of x will be compared to the rows or columns indicated by the corresponding element of at.
metric	Distance metric to use when finding the nearest neighbors. Supported metrics include "euclidean", "maximum", "manhattan", and "minkowski".
p	The power for the Minkowski distance.
weights	A numeric vector of weights for the distance components if calculating weighted distances. For example, the weighted Euclidean distance is sqrt(sum(w * (x - y)^2)).
ix, iy	A list of specific row or column indices for which to calculate the pairwise distances. Numeric vectors will be coerced to lists. Each list element should give a vector of indices to use for a distance computation. Elements of length 1 will be recycled to an appropriate length.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to rowdist() or coldist().

Details

rowdist() and coldist() calculate straightforward distances between each row or column, respectively in x and y. If y = x (the default), then the output of rowdist() will match the output of dist (except it will be an ordinary matrix).

rowdist_at() and coldist_at() allow passing a list of specific row or column indices for which to calculate the distances.

rowDists() and colDists() are S4 generics. The current methods provide (optionally parallelized) versions of rowdist() and coldist() for matter_mat and sparse_mat matrices.

Value

For rowdist() and coldist(), a matrix with rows equal to the number of observations in x and columns equal to the number of observations in y.

For rowdist_at() and coldist_at(), a list where each element gives the pairwise distances corresponding to the indices given by ix and iy.

rowDists() and colDists() have corresponding return values depending on whether at has been specified.

Author(s)

Kylie A. Bemis

See Also

dist

Examples

```
register(SerialParam())

set.seed(1)

x <- matrix(runif(25), nrow=5, ncol=5)
y <- matrix(runif(25), nrow=3, ncol=5)

rowDists(x) # same as as.matrix(dist(x))
rowDists(x, y)

# distances between:
# x[1,] vs x[,]
# x[5,] vs x[,]
rowdist_at(x, c(1,5))

# distances between:
# x[1,] vs x[1:3,]
# x[5,] vs x[3:5,]
rowdist_at(x, ix=c(1,5), iy=list(1:3, 3:5))

# distances between:
# x[i,] vs x[(i-1):(i+1),]
rowDists(x, at=roll(1:5, width=3))
```

simspec *Simulate Spectra*

Description

Simulate spectra from noise and a list of peaks.

Usage

```
simspec(n = 1L, npeaks = 50L,
x = rlnorm(npeaks, 7, 0.3), y = rlnorm(npeaks, 1, 0.9),
domain = c(0.9 * min(x), 1.1 * max(x)), size = 10000,
sdx = 1e-5, sdy = sdymult * log1p(y), sdymult = 0.2,
sdnoise = 0.1, resolution = 1000, fmax = 0.5,
baseline = 0, decay = 10, units = "relative")

simspec1(x, y, xout, peakwidths = NA_real_,
sdnoise = 0, resolution = 1000, fmax = 0.5)
```

Arguments

`n`	The number of spectra to simulate.
`npeaks`	The number of peaks to simulate. Not used if `x` and `y` are provided.
`x, y`	The locations and values of the spectral peaks.
`xout, domain`	The output domain variable or its range.
`size`	The number of samples in each spectrum.
`sdx`	The standard deviation of the error in the observed locations of peaks, in units indicated by `units`.
`sdy`	The standard deviation(s) for the distributions of observed peak values on the log scale.
`sdymult`	A multiplier used to calculate `sdy` based on the mean values of the peaks; used to simulate multiplicative variance. Not used if `sdy` is provided.
`sdnoise`	The standard deviation of the random noise in the spectra on the log scale.
`resolution`	The resolution as defined by `x / dx`, where `x` is the observed peak location and `dx` is the width of the peak at a proportion of its maximum height defined by `fmax` (defaults to full-width-at-half-maximum – FWHM – definition).
`fmax`	The fraction of the maximum peak height to use when defining the resolution.
`peakwidths`	The peak widths at `fmax`. Typically, these are calculated automatically from `resolution`.
`baseline`	The maximum intensity of the baseline. Note that `baseline=0` means there is no baseline.
`decay`	A constant used to calculate the exponential decay of the baseline. Larger values mean the baseline decays more sharply.
`units`	The units for `sdx`. Either `"absolute"` or `"relative"`.

Value

Either a numeric vector of the same length as size, giving the simulated spectrum, or a size x x matrix of simulated spectra.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)
y <- simspec(2)
x <- attr(y, "domain")

plot(x, y[,1], type="l", ylim=c(-max(y), max(y)))
lines(x, -y[,2], col="red")
```

---

sparse_arr-class *Sparse Vectors and Matrices*

Description

The sparse_mat class implements sparse matrices, potentially stored out-of-memory. Both compressed-sparse-column (CSC) and compressed-sparse-row (CSR) formats are supported. Sparse vectors are also supported through the sparse_vec class.

Usage

```
## Instance creation
sparse_mat(data, index, type = "double",
    nrow = NA_integer_, ncol = NA_integer_, dimnames = NULL,
    pointers = NULL, domain = NULL, offset = 0L, rowMaj = FALSE,
    tolerance = c(abs=0), sampler = "none", ...)

sparse_vec(data, index, type = "double",
    length = NA_integer_, names = NULL,
    domain = NULL, offset = 0L, rowMaj = FALSE,
    tolerance = c(abs=0), sampler = "none", ...)

# Check if an object is a sparse matrix
is.sparse(x)

# Coerce an object to a sparse matrix
as.sparse(x, ...)

## Additional methods documented below
```

Arguments

- `data` — Either the non-zero values of the sparse array, or (if `index` is missing) a numeric vector or matrix from which to create the sparse array. For a `sparse_vec`, these should be a numeric vector. For a `sparse_mat` these can be a numeric vector if `pointers` is supplied, or a list of numeric vectors if `pointers` is `NULL`.
- `index` — For `sparse_vec`, the indices of the non-zero items. For `sparse_mat`, either the row-indices or column-indices of the non-zero items, depending on the value of `rowMaj`.
- `type` — A 'character' vector giving the storage mode of the data in virtual memory such. See `?"matter-types"` for possible values.
- `nrow, ncol, length` — The number of rows and columns, or the length of the array.
- `dimnames` — The names of the sparse matrix dimensions.
- `names` — The names of the sparse vector elements.
- `pointers` — The (zero-indexed) pointers to the start of either the rows or columns (depending on the value of `rowMaj`) in `data` and `index` when they are numeric vectors rather than lists.
- `domain` — Either `NULL` or a vector with length equal to the number of rows (for CSC matrices) or the number of columns (for CSR matrices). If `NULL`, then `index` is assumed to be row or column indices. If a vector, then they define the how the non-zero elements are matched to rows or columns. The `index` value of each non-zero element is matched against this domain using binary search. Must be numeric.
- `offset` — If `domain` is `NULL` (i.e., `index` represents the actual row/column indices), then this is the index of the first row/column. The default of 0 means that `index` is indexed from 0.
- `rowMaj` — Whether the data should be stored using compressed-sparse-row (CSR) representation (as opposed to compressed-sparse-column (CSC) representation). Defaults to 'FALSE', for efficient access to columns. Set to 'TRUE' for more efficient access to rows instead.
- `tolerance` — For non-`NULL` domain, the tolerance used for floating-point equality when matching `index` to the `domain`. The vector should be named. Use 'absolute' to use absolute differences, and 'relative' to use relative differences.
- `sampler` — For non-zero tolerances, how the `data` values should be combined when there are multiple `index` values within the tolerance. Must be of 'none', 'mean', 'sum', 'max', 'min', 'area', 'linear', 'cubic', 'gaussian', or 'lanczos'. Note that 'none' means nearest-neighbor interpolation.
- `x` — An object to check if it is a sparse matrix or coerce to a sparse matrix.
- `...` — Additional arguments to be passed to constructor.

Value

An object of class `sparse_mat`.

Slots

data: The non-zero data values. Can be a numeric vector or a list of numeric vectors.

type: The storage mode of the *accessed* data when read into R. This is a 'factor' with levels 'raw', 'logical', 'integer', 'numeric', or 'character'.

dim: Either NULL for vectors, or an integer vector of length one of more giving the maximal indices in each dimension for matrices and arrays.

names: The names of the data elements for vectors.

dimnames: Either NULL or the names for the dimensions. If not NULL, then this should be a list of character vectors of the length given by 'dim' for each dimension. This is always NULL for vectors.

index: The indices of the non-zero items. Can be a numeric vector or a list of numeric vectors.

pointers: The pointers to the beginning of the rows or columns if index and data use vector storage rather than list storage.

domain: Either NULL or a vector with length equal to the number of rows (for CSC matrices) or the number of columns (for CSR matrices). If NULL, then index is assumed to be row or column indices. If a vector, then they define the how the non-zero elements are matched to rows or columns. The index value of each non-zero element is matched against this domain using binary search. Must be numeric.

offset: If domain is NULL (i.e., index represents the actual row/column indices), then this is the index of the first row/column. The default of 0 means that index is indexed from 0.

tolerance: For non-NULL domain, the tolerance used for floating-point equality when matching index to the domain. The vector should be named. Use 'absolute' to use absolute differences, and 'relative' to use relative differences.

sampler: The type of summarization or interpolation performed when there are multiple index values within the tolerance of the requested domain value(s).

ops: Deferred arithmetic operations.

transpose Indicates whether the data is stored in row-major order (TRUE) or column-major order (FALSE). For a matrix, switching the order that the data is read is equivalent to transposing the matrix (without changing any data).

Extends

matter

Creating Objects

sparse_mat and sparse_vec instances can be created through sparse_mat() and sparse_vec(), respectively.

Methods

Class-specific methods:

atomdata(x): Access the 'data' slot.

adata(x): An alias for atomdata(x).

`atomindex(x)`: Access the 'index' slot.

`aindex(x)`: An alias for atomindex(x).

`pointers(x)`: Access the 'pointers' slot.

`domain(x)`: Access the 'domain' slot.

`tolerance(x), tolerance(x) <- value`: Get or set resampling 'tolerance'.

`sampler(x), sampler(x) <- value`: Get or set the 'sampler' method.

Standard generic methods:

`dim(x)`: Get 'dim'.

`dimnames(x), dimnames(x) <- value`: Get or set 'dimnames'.

`x[i, j, ..., drop], x[i, j] <- value`: Get or set the elements of the sparse matrix. Use `drop = NULL` to return a subset of the same class as the object.

`cbind(x, ...), rbind(x, ...)`: Combine sparse matrices by row or column.

`t(x)`: Transpose a matrix. This is a quick operation which only changes metadata and does not touch the data representation.

`rowMaj(x)`: Check the data orientation.

Author(s)

Kylie A. Bemis

See Also

`matter`

Examples

```
x <- matrix(rbinom(100, 1, 0.2), nrow=10, ncol=10)

y <- sparse_mat(x)
y[]
```

`stream-stats` *Streaming Summary Statistics*

Description

These functions allow calculation of streaming statistics. They are useful, for example, for calculating summary statistics on small chunks of a larger dataset, and then combining them to calculate the summary statistics for the whole dataset.

This is not particularly interesting for simpler, commutative statistics like `sum()`, but it is useful for calculating non-commutative statistics like running `sd()` or `var()` on pieces of a larger dataset.

Usage

```
# calculate streaming univariate statistics
s_range(x, ..., na.rm = FALSE)

s_min(x, ..., na.rm = FALSE)

s_max(x, ..., na.rm = FALSE)

s_prod(x, ..., na.rm = FALSE)

s_sum(x, ..., na.rm = FALSE)

s_mean(x, ..., na.rm = FALSE)

s_var(x, ..., na.rm = FALSE)

s_sd(x, ..., na.rm = FALSE)

s_any(x, ..., na.rm = FALSE)

s_all(x, ..., na.rm = FALSE)

s_nnzero(x, ..., na.rm = FALSE)

# calculate streaming matrix statistics
s_rowstats(x, stat, group, na.rm = FALSE, ...)

s_colstats(x, stat, group, na.rm = FALSE, ...)

# calculate combined summary statistics
stat_c(x, y, ...)
```

Arguments

`x, y, ...`	Object(s) on which to calculate a summary statistic, or a summary statistic to combine.
`stat`	The name of a summary statistic to compute over the rows or columns of a matrix. Allowable values include: "range", "min", "max", "prod", "sum", "mean", "var", "sd", "any", "all", and "nnzero".
`group`	A factor or vector giving the grouping. If not provided, no grouping will be used.
`na.rm`	If `TRUE`, remove `NA` values before summarizing.

Details

These summary statistics methods are intended to be applied to chunks of a larger dataset. They can then be combined either with the individual summary statistic functions, or with `stat_c()`, to produce the combined summary statistic for the full dataset. This is most useful for calculating running variances and standard deviations iteratively, which would be difficult or impossible to calculate on the full dataset.

The variances and standard deviations are calculated using running sum of squares formulas which can be calculated iteratively and are accurate for large floating-point datasets (see reference).

Value

For all univariate functions except `s_range()`, a single number giving the summary statistic. For `s_range()`, two numbers giving the minimum and the maximum values.

For `s_rowstats()` and `s_colstats()`, a vector of summary statistics.

Author(s)

Kylie A. Bemis

References

B. P. Welford. "Note on a Method for Calculating Corrected Sums of Squares and Products." Technometrics, vol. 4, no. 3, pp. 1-3, Aug. 1962.

B. O'Neill. "Some Useful Moment Results in Sampling Problems." The American Statistician, vol. 68, no. 4, pp. 282-296, Sep. 2014.

See Also

Summary

Examples

```
set.seed(1)
x <- sample(1:100, size=10)
y <- sample(1:100, size=10)

sx <- s_var(x)
sy <- s_var(y)

var(c(x, y))
stat_c(sx, sy) # should be the same

sxy <- stat_c(sx, sy)

# calculate with 1 new observation
var(c(x, y, 99))
stat_c(sxy, 99)

# calculate over rows of a matrix
set.seed(2)
A <- matrix(rnorm(100), nrow=10)
B <- matrix(rnorm(100), nrow=10)
```

```
sx <- s_rowstats(A, "var")
sy <- s_rowstats(B, "var")

apply(cbind(A, B), 1, var)
stat_c(sx, sy) # should be the same
```

---

struct *C-Style Structs Stored in Virtual Memory*

Description

This is a convenience function for creating and reading C-style structs in a single file stored in virtual memory.

Usage

```
struct(..., path = NULL, readonly = FALSE, offset = 0, filename)
```

Arguments

`...`	Named integers giving the members of the `struct`. They should be of the form `name=c(type=length)`.
`path, filename`	A single string giving the name of the file.
`readonly`	Should the file be treated as read-only?
`offset`	A scalar integer giving the offset from the beginning of the file.

Details

This is simply a convenient wrapper around the wrapper around `matter_list` that allows easy specification of C-style structs in a file.

Value

A object of class `matter_list`.

Author(s)

Kylie A. Bemis

See Also

`matter_list`

Examples

```
x <- struct(first=c(int=1), second=c(double=1))

x$first <- 2L
x$second <- 3.33

x$first
x$second
```

---

summary-stats *Summary Statistics for "matter" Objects*

Description

These functions efficiently calculate summary statistics for matter_arr objects. For matrices, they operate efficiently on both rows and columns.

Usage

```
## S4 method for signature 'matter_arr'
range(x, ..., na.rm)
## S4 method for signature 'matter_arr'
min(x, ..., na.rm)
## S4 method for signature 'matter_arr'
max(x, ..., na.rm)
## S4 method for signature 'matter_arr'
prod(x, ..., na.rm)
## S4 method for signature 'matter_arr'
mean(x, ..., na.rm)
## S4 method for signature 'matter_arr'
sum(x, ..., na.rm)
## S4 method for signature 'matter_arr'
sd(x, na.rm)
## S4 method for signature 'matter_arr'
var(x, na.rm)
## S4 method for signature 'matter_arr'
any(x, ..., na.rm)
## S4 method for signature 'matter_arr'
all(x, ..., na.rm)
## S4 method for signature 'matter_mat'
colMeans(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
colSums(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
rowMeans(x, na.rm, dims = 1, ...)
## S4 method for signature 'matter_mat'
rowSums(x, na.rm, dims = 1, ...)
```

Arguments

x	A matter_arr object.
...	Arguments passed to chunk_lapply().
na.rm	If TRUE, remove NA values before summarizing.
dims	Not used.

Details

These summary statistics methods operate on chunks of data which are loaded into memory and then freed before reading the next chunk.

For row and column summaries on matrices, the iteration scheme is dependent on the layout of the data. Column-major matrices will always be iterated over by column, and row-major matrices will always be iterated over by row. Row statistics on column-major matrices and column statistics on row-major matrices are calculated iteratively.

Variance and standard deviation are calculated using a running sum of squares formula which can be calculated iteratively and is accurate for large floating-point datasets (see reference).

Value

For mean, sd, and var, a single number. For the column summaries, a vector of length equal to the number of columns of the matrix. For the row summaries, a vector of length equal to the number of rows of the matrix.

Author(s)

Kylie A. Bemis

References

B. P. Welford, "Note on a Method for Calculating Corrected Sums of Squares and Products," Technometrics, vol. 4, no. 3, pp. 1-3, Aug. 1962.

See Also

stream_stat

Examples

```
register(SerialParam())

x <- matter(1:100, nrow=10, ncol=10)

sum(x)
mean(x)
var(x)
sd(x)

colSums(x)
colMeans(x)
```

```
rowSums(x)
rowMeans(x)
```

---

to_raster *Rasterize a Scattered 2D or 3D Signal*

Description

Estimate the raster dimensions of a scattered 2D or 3D signal based on its pixel coordinates.

Usage

```
# Rasterize a 2D signal
to_raster(x, y, vals)

# Rasterize a 3D signal
to_raster3(x, y, z, vals)

# Check if coordinates are gridded
is_gridded(x, tol = 0.5)
```

Arguments

`x, y, z`	The coordinates of the data to be rasterized. For `is_gridded()`, a numeric matrix or data frame where each column gives the pixel coordinates for a different dimension..
`vals`	The data values to be rasterized.
`tol`	The tolerance allowed when estimating the resolution. Noise in the sampling rate will be allowed up to this amount when determining if the data is approximately gridded or not.

Details

This is meant to be a more efficient version of `approx2()` when the data is already (approximately) gridded. Otherwise, `approx2()` is used.

Value

A numeric vector giving the estimated raster dimensions.

Author(s)

Kylie A. Bemis

See Also

`approx2`

Examples

```
# create an image
set.seed(1)
i <- seq(-4, 4, length.out=12)
j <- seq(1, 3, length.out=9)
co <- expand.grid(x=i, y=j)
z <- matrix(atan(co$x / co$y), nrow=12, ncol=9)
vals <- 10 * (z - min(z)) / diff(range(z))

# scatter coordinates and flatten image
d <- expand.grid(x=jitter(1:12), y=jitter(1:9))
d$vals <- as.vector(z)

# rasterize
to_raster(d$x, d$y, d$vals)
```

---

trans2d *2D Spatial Transformation*

Description

Perform linear spatial transformations on a matrix, including rigid, similarity, and affine transformations.

Usage

```
trans2d(x, y, z, pmat,
rotate = 0, translate = c(0, 0), scale = c(1, 1),
interp = "linear", dimout = dim(z), ...)
```

Arguments

`x, y, z`	The data to be interpolated. Alternatively, `x` can be a matrix, in which case the matrix elements are used for `z` and `x` and `y` are generated from the matrix's dimensions.
`pmat`	A 3 x 2 transformation matrix for performing an affine transformation. Automatically generated from `rotate`, `translate`, and `scale` if not provided.
`rotate`	Rotation in degrees.
`translate`	Translation vector, in the same units as `x` and `y`, if given.
`scale`	Scaling factors.
`interp`	Interpolation method. See `approx2`.
`dimout`	The dimensions of the returned matrix.
`...`	Additional arguments passed to `approx2`.

Value

If `x` is a matrix or `z` is provided, returns a transformed matrix with the dimensions of `dimout`.

Otherwise, only the transformed coordinates are returned in a `data.frame`.

Author(s)

Kylie A. Bemis

See Also

`approx2`

Examples

```
set.seed(1)
x <- matrix(0, nrow=32, ncol=32)
x[9:24,9:24] <- 10
x <- x + runif(length(x))
xt <- trans2d(x, rotate=15, translate=c(-5, 5))

par(mfcol=c(1,2))
image(x, col=hcl.colors(256), main="original")
image(xt, col=hcl.colors(256), main="transformed")
```

---

uuid *Universally Unique Identifiers*

Description

Generate a UUID.

Usage

```
uuid(uppercase = FALSE)

hex2raw(x)

raw2hex(x, uppercase = FALSE)
```

Arguments

`x`	A vector of to convert between `raw` bytes and hexadecimal strings.
`uppercase`	Should the result be in uppercase?

Details

`uuid` generates a random universally unique identifier.

`hex2raw` converts a hexadecimal string to a `raw` vector.

`raw2hex` converts a `raw` vector to a hexadecimal string.

Value

For `uuid`, a list of length 2:

- `string`: A character vector giving the UUID.
- `bytes`: The `raw` bytes of the UUID.

For `hex2raw`, a `raw` vector.

For `raw2hex`, a character vector of length 1.

Author(s)

Kylie A. Bemis

Examples

```
id <- uuid()
id
hex2raw(id$string)
raw2hex(id$bytes)
```

---

vizi — *A Simple Grammar of Base Graphics*

Description

These functions provide a simple grammar of graphics approach to programming with R's base graphics system.

Usage

```
## Initialize a plot
vizi(data, ..., encoding = NULL, params = NULL)

## Add plot components
add_mark(plot, mark, ..., encoding = NULL, data = NULL,
    trans = NULL, params = NULL)

add_facets(plot, by = NULL, data = NULL,
    nrow = NA, ncol = NA, labels = NULL,
    drop = TRUE, free = "")
```

```
## Set plot attributes
set_title(plot, title)

set_channel(plot, channel, label = NULL,
    limits = NULL, scheme = NULL, key = TRUE)

set_coord(plot, xlim = NULL, ylim = NULL, zlim = NULL,
    log = "", asp = NA, grid = TRUE)

set_par(plot, ...)

## Combine plots
plot_facets(plotlist, nrow = NA, ncol = NA,
    labels = NULL, drop = TRUE, free = "")
```

Arguments

data	A data.frame.
...	For vizi and add_mark, these should be named arguments specifying the encoding for the plot. The argument names should specify channels, using either base R-style (e.g., pch, cex) *or* ggplot-style names (e.g., shape, size). One-sided formula arguments will be evaluated in the environment of data. Non-formula arguments will be used as-is. For set_par, these specify additional graphical parameters (as in par) or arguments to persp for 3D plots.
encoding	Encodings specified as a list rather than using ....
params	Additional graphical parameters that are not mapped to the data, using either base R-style (e.g., pch, cex) *or* ggplot-style names (e.g., shape, size)
plot, plotlist	A vizi_plot object, or a list of such objects, respectively.
title	The title of the plot.
channel	The channel to modify, using ggplot-style names (e.g., shape, size).
label, labels	Plotting labels.
limits	The limits for the channel, specified as c(min, max) for continuous variables or a character vector of possible levels for discrete variables. The data will be constrained to these limits *before* plotting.
scheme	A function or vector giving the scheme for encoding the channel. For example, a vector of colors, or a function that returns a vector of n colors.
key	Should a key be generated for the channel?
xlim, ylim, zlim	The plot limits. These only affect the plotting window, not the data. See plot.window
log, asp	See plot.window.
grid	Should a rectangular grid be included in the plot?
mark	The name of a supported mark, such as "points", "lines", etc.

trans	A list providing parameters for any transformations the mark supports.
by	A vector or formula giving the facet specification.
nrow, ncol	The number of rows and columns in the facet grid.
drop	Should empty facets be dropped?
free	A string specifying the free spatial dimensions during faceting. E.g., "", "x", "y", "xy", "yx".

Value

An object of class vizi_plot.

Author(s)

Kylie A. Bemis

See Also

vizi_points, vizi_lines, vizi_peaks, vizi_pixels, vizi_voxels

Examples

```
require(datasets)

mtcars <- transform(mtcars,
    am=factor(am, labels=c("auto", "manual")))

# faceted scatter plot
vizi(mtcars, x=~disp, y=~mpg) |>
    add_mark("points") |>
    add_facets(~mtcars$am)

# faceted scatter plot with color
vizi(mtcars, x=~disp, y=~mpg, color=~am) |>
    add_mark("points",
        params=list(shape=20, size=2, alpha=0.8)) |>
    add_facets(~mtcars$am)


coords <- expand.grid(x=1:nrow(volcano), y=1:ncol(volcano))

# volcano image
vizi(coords, x=~x, y=~y, color=volcano) |>
    add_mark("pixels") |>
    set_coord(grid=FALSE) |>
    set_par(xaxs="i", yaxs="i")
```

warp1 *Warping to Align 1D Signals*

Description

Two signals often need to be aligned in order to compare them. The signals may contain a similar pattern, but shifted or stretched by some amount. These functions warp the first signal to align it as closely as possible to the second signal.

Usage

```
# Signal warping based on local events
warp1_loc(x, y, tx = seq_along(x), ty = seq_along(y),
    events = c("maxmin", "max", "min"), lx = NULL, ly = NULL,
    interp = c("linear", "loess", "spline"), n = length(y),
    tol = NA_real_, tol.ref = "abs")

# Dynamic time warping
warp1_dtw(x, y, tx = seq_along(x), ty = seq_along(y),
    n = length(y), tol = NA_real_, tol.ref = "abs")

# Correlation optimized warping
warp1_cow(x, y, tx = seq_along(x), ty = seq_along(y),
    nbins = NA_integer_, n = length(y),
    tol = NA_real_, tol.ref = "abs")
```

Arguments

`x, y`	Signals to be aligned by warping `x` to match `y`.
`tx, ty`	The domain variable of the signals.
`events`	The type of events to use for calculating the alignment.
`lx, ly`	The domain locations of the events, if already known.
`n`	The number of samples in the warped `x` to be returned. By default, it matches the length of `y`.
`interp`	The interpolation method used when warping the signal.
`tol, tol.ref`	A tolerance specifying the maximum allowed distance between aligned samples. See `bsearch` for details. If missing, the tolerance is estimated as 5% of the signal's domain range.
`nbins`	The number of signal segments used for warping. The correlation is maximized for each segment.

Details

`warp1_loc()` uses a simple event-based alignment. Events are defined as local extrema. The events are matched between each signal based on proximity. The shift between the events in `x` and `y` are calculated and interpolated to find shifts for each sample. The warped signal is then calculated from these shifts. This is a simple heuristic method, but it is relatively fast and typically good enough for aligning peak locations.

`warp1_dtw()` performs dynamic time warping. In dynamic time warping, each sample in `x` is matched to a corresponding sample in `y` using dynamic programming to find the optimal matches. The version implemented here is constrained by the given tolerance. This both reduces the necessary memory, and in practice tends to give more realistic (and therefore accurate) results than an unconstrained alignment. An unconstrained alignment can still be obtained by setting a high tolerance, but this may use a lot of memory.

`warp1_cow()` performs correlation optimized. In correlation optimized warping, each signal is divided into some number of segments. Dynamic programming is then used to find the placement of the segment boundaries that maximizes the correlation of all the segments.

Value

A numeric vector the same length as `y` with the warped `x`.

Author(s)

Kylie A. Bemis

References

G. Tomasi, F. van den Berg, and C. Andersson. "Correlation optimized warping and dynamic time warping as preprocessing methods for chromatographic data." Journal of Chemometrics, vol. 18, pp. 231-241, July 2004.

N. V. Nielsen, J. M. Carstensen, and J. Smedsgaard. "Aligning of single and multiple wavelength chromatographic profiles for chemometric data analysis using correlation optimised warping." Journal of Chromatography A, vol. 805, pp. 17-35, Jan. 1998.

Examples

```
set.seed(1)
t <- seq(from=0, to=6 * pi, length.out=2000)
dt <- 0.3 * (sin(t) + 0.6 * sin(2.6 * t))
x <- sin(t + dt) + 0.6 * sin(2.6 * (t + dt))
y <- sin(t) + 0.6 * sin(2.6 * t)
xw <- warp1_dtw(x, y)

plot(y, type="l")
lines(x, col="blue")
lines(xw, col="red", lty=2)
```

warp2 *Warping to Align 2D Signals*

Description

Register two images by warping a "moving" image to align with the "fixed" image.

Usage

```
# Transformation-based registration
warp2_trans(x, y, control = list(),
    trans = c("rigid", "similarity", "affine"),
    metric = c("cor", "mse", "mi"), nbins = 64L,
    scale = TRUE, dimout = dim(y))

# Mutual information
mi(x, y, n = 64L)
```

Arguments

`x, y`	Images to be aligned by warping `x` (which is "moving") to match `y` (which is "fixed").
`control`	A list of optimization control parameters. Passed to `optim()`.
`trans`	The type of transformation allowed: "rigid" means translation and rotation, "similarity" means translation, rotation, and scaling, and "affine" means a general affine transformation.
`metric`	The metric to optimize.
`nbins, n`	The number of histogram bins to use when calculating mutual information.
`scale`	Should the images be normalized to have the same intensity range?
`dimout`	The dimensions of the returned matrix.

Details

`warp2_trans()` performs a simple transformation-based registration using `optim` for optimization.

Value

A numeric vector the same length as `y` with the warped `x`.

Author(s)

Kylie A. Bemis

See Also

`trans2d`, `optim`

Examples

```
set.seed(1)
x <- matrix(0, nrow=32, ncol=32)
x[9:24,9:24] <- 10
x <- x + runif(length(x))
xt <- trans2d(x, rotate=15, translate=c(-5, 5))
xw <- warp2_trans(xt, x)

par(mfcol=c(1,3))
image(x, col=hcl.colors(256), main="original")
image(xt, col=hcl.colors(256), main="transformed")
image(xw, col=hcl.colors(256), main="registered")
```

Index