Package 'Cardinal'

April 15, 2024

Type Package

Title A mass spectrometry imaging toolbox for statistical analysis

Version 3.4.3

Date 2021-11-21

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

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

Description Implements statistical & computational tools for analyzing mass spectrometry imaging datasets, including methods for efficient pre-processing, spatial segmentation, and classification.

License Artistic-2.0

Depends ProtGenerics, BiocGenerics, BiocParallel, EBImage, graphics, methods, S4Vectors (>= 0.27.3), stats

Imports CardinalIO, Biobase, irlba, Matrix, matter, magrittr, mclust, nlme, parallel, signal, stats4, utils, viridisLite

Suggests BiocStyle, testthat, knitr, rmarkdown

VignetteBuilder knitr

biocViews Software, Infrastructure, Proteomics, Lipidomics, MassSpectrometry, ImagingMassSpectrometry, ImmunoOncology, Normalization, Clustering, Classification, Regression

URL http://www.cardinalmsi.org

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

git_branch RELEASE_3_18

git_last_commit eacb164

git_last_commit_date 2023-11-22

Repository Bioconductor 3.18

Date/Publication 2024-04-15

R topics documented:

`Cardinal-package` *Mass spectrometry imaging tools*

Description

Implements statistical & computational tools for analyzing mass spectrometry imaging datasets, including methods for efficient pre-processing, spatial segmentation, and classification.

Details

Cardinal provides an abstracted interface to manipulating mass spectrometry imaging datasets, simplifying most of the basic programmatic tasks encountered during the statistical analysis of imaging data. These include image manipulation and processing of both images and mass spectra, and dynamic plotting of both.

While pre-processing steps including normalization, baseline correction, and peak-picking are provided, the core functionality of the package is statistical analysis. The package includes classification and clustering methods based on nearest shrunken centroids, as well as traditional tools like PCA and PLS.

Type `browseVignettes("Cardinal")` to view a user's guide and vignettes of common workflows.

Options

The following options can be set via `options()`.

- `getCardinalgetCardinalBPPARAM()`, `setCardinalBPPARAM(BPPARAM=SerialParam())`: The default backend to use for parallel processing. By default, this is initially set to a serial backend (no parallelization).
- `getCardinalVerbose()`, `setCardinalVerbose(verbose=interactive())`: Should detailed messages be printed?
- `getCardinalNumBlocks()`, `setCardinalNumBlocks(n=20L)`: The default number of data chunks used by `pixelApply()`, `featureApply()`, and `spatialApply()` when `.blocks=TRUE`. Used by many methods internally.
- `getCardinalDelayProc()`, `setCardinalDelayProc(delay=TRUE)`: Should pre-processing functions like `normalize()` and `peakPeak()` be delayed (until `process()` is called)?

Logging

For support or debugging help, please provide the output of a call to `CardinalLog()`. By default, this saves a log to the file "Cardinal.log" in the current working directory.

Author(s)

Kylie A. Bemis

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

---

aggregate-methods *Calculating summary statistics*

Description

These methods calculate summary statistics over subsets of an imaging experiment.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
aggregate(x, by = c("feature", "pixel"), FUN,
        groups = NULL, tform = identity, as = "ImagingExperiment",
        BPPARAM = getCardinalBPPARAM(), ...)

summarizeFeatures(x, FUN = "mean", ...)

summarizePixels(x, FUN = "mean", ...)
```

Arguments

x	An imaging dataset.
by	Should the summarization be performed over pixels or features?
FUN	A function or list of functions that return statistical summaries. Known statistics can be provided to be computed more efficiently than providing the equivalent function. Known statistics include "min", "max", "mean", "sum", "sd", and "var".
groups	A grouping variable for summarization. The summary functions will be applied within each group.
tform	A unary transformation that should each row or column be transformed before summarization is applied.
as	What class of object should be returned (`ImagingExperiment` or `DataFrame`)?
BPPARAM	An optional `BiocParallelParam` instance to be passed to `bplapply()`.
...	Additional arguments to be passed to `FUN`.

Value

An `ImagingExperiment` subclass if `as=="ImagingExperiment"` or a `DataFrame` subclass otherwise.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# calculate median spectrum
aggregate(mse, by="feature", FUN=median, as="DataFrame")

# summarize mean spectrum
summarizeFeatures(mse, FUN="mean", as="DataFrame")

# summarize image by TIC
summarizePixels(mse, FUN=c(tic="sum"), as="DataFrame")

# summarize mean spectrum grouped by pixels in/out of circle
summarizeFeatures(mse, FUN="mean", groups=mse$circle, as="DataFrame")
```

---

colocalized-methods *Colocalized features*

Description

Find colocalized features in an imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment,missing'
colocalized(object, mz, ...)

## S4 method for signature 'SparseImagingExperiment,ANY'
colocalized(object, ref, n = 10,
sort.by = c("correlation", "M1", "M2"),
threshold = median,
BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialDGMM,ANY'
colocalized(object, ref, n = 10,
sort.by = c("Mscore", "M1", "M2"),
threshold = median,
BPPARAM = getCardinalBPPARAM(), ...)
```

Arguments

object	An imaging experiment.
mz	An m/z value giving the image to use as a reference.
ref	Either a numeric vector or logical mask of a region-of-interest, or the feature to use as a reference.
n	The number of top-ranked colocalized features to return.
sort.by	The colocalization measure used to rank colocalized features. Possible options include Pearson's correlation ("correlation"), match score ("Mscore"), and Manders' colocalization coefficients ("M1" and "M2").
threshold	A function that returns the cutoff to use for creating logical masks of numeric references.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	ignored.

Value

A data frame with the colocalized features.

Author(s)

Kylie A. Bemis

See Also

topFeatures

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
data <- simulateImage(preset=2, npeaks=10, representation="centroid")

# find features colocalized with first feature
colocalized(data, ref=1)
```

---

cvApply-methods *Apply cross-validation to imaging analyses*

Description

Apply cross-validation with an existing or a user-specified modeling function over an imaging datasets.

Usage

```
## S4 method for signature 'MSImagingExperiment'
crossValidate(.x, .y, .fun,
        .fold = run(.x),
        .predict = predict,
        .process = FALSE,
        .processControl = list(),
        .peaks = NULL,
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SparseImagingExperiment'
crossValidate(.x, .y, .fun, .fold = run(.x),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SparseImagingExperiment'
cvApply(.x, .y, .fun,
        .fold = run(.x),
        .predict = predict,
        .fitted = fitted,
        .simplify = FALSE,
        BPREDO = list(),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'CrossValidated2'
summary(object, ...)
```

Arguments

`.x`	An imaging dataset.
`.y`	The response variable for prediction.
`.fun`	A function for training a model where the first two arguments are the dataset and the response.
`.fold`	A variable determining the cross-validation folds. When specifying custom folds, it is important to make sure that data points from the same experimental run are not split among different folds. I.e., all data points from a run should belong to the same CV fold.
`.predict`	A function for predicting from a trained model. The first two arguments are the model and a new dataset.
`.fitted`	A function for extracting the predicted values from the result of a call to `.predict`.
`.simplify`	If FALSE (the default), the output of `.predict` is returned. If TRUE, then `.fitted` is applied to the results to extract fitted values. Only the fitted values, observed values, and basic model information is returned.
`.process`	Should pre-processing be applied before each training and test step? This includes peak alignment and peak filtering. Peak binning is also performed if `.peaks` is given.

.processControl	A list of arguments to be passed to the pre-processing steps.
.peaks	A peak-picked version of the full dataset .x, for use with pre-processing between training and test steps.
BPREDO	See documentation for bplapply.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
...	Additional arguments passed to .fun.
object	A fitted model object to summarize.

Details

This method is designed to be used with the provided classification methods, but can also be used with user-provided functions and methods as long as they fulfill certain expectations.

The function or method passed to '.fun' must take at least two arguments: the first argument must be a object derived from SparseImagingExperiment or SImageSet, and the second argument must be the response variable. The function should return an object of a class derived from ImagingResult or ResultSet, which should have a predict method that takes arguments 'newx' and 'newy' in addition to the fitted model.

For MSImagingExperiment objects, pre-processing can be performed. This is particularly useful if there is no reference to which to align peaks, except the mean spectrum (which is calculated from the whole dataset, and may invalidate cross-validation results).

If .process=TRUE and .peaks=NULL, then either the data should be profile spectra with no peak picking, or it should be a peak-picked dataset *before peak alignment*. If the data has no peak picking, then the pre-processing will consist of peak picking on the mean spectrum of the training sets, followed by peak alignment and peak filtering. If the data has been peak-picked but not aligned, then the pre-processing will consist of peak alignment to the mean spectrum of the training sets, and peak filtering.

If .process=TRUE and .peaks is given, then the data should be a dataset consisting of profile spectra, and .peaks should be a peak-picked version of the same dataset *before peak alignment*. The pre-processing will consist of peak alignment to the mean spectrum of the training sets, peak filtering, and peak binning the full data to the aligned peaks.

The crossValidate function calls cvApply internally and then post-processes the result to be more easily-interpretable and space-efficient. Accuracy metrics are reported for each set of modeling parameters.

Value

An object of class 'CrossValidated', which is derived from ResultSet, or an object of class 'CrossValidated2', which is derived from ImagingResult.

Author(s)

Kylie A. Bemis

See Also

spatialShrunkenCentroids, PLS, OPLS

defunct *Defunct functions and methods in Cardinal*

Description

These functions are defunct and are no longer available.

deprecated *Deprecated functions and methods in Cardinal*

Description

These functions are provided for compatibility with older versions of Cardinal, and will be defunct at the next release.

findNeighbors-methods *Find spatial neighbors and spatial weightst*

Description

Methods for calculating the spatial neighbors (pixels within a certain distance) or spatial weights for all pixels in a dataset.

Usage

```
#### Methods for Cardinal >= 2.x classes ####

## S4 method for signature 'ImagingExperiment'
findNeighbors(x, r, groups = run(x), ...)

## S4 method for signature 'PositionDataFrame'
findNeighbors(x, r, groups = run(x), dist = "chebyshev",
    offsets = FALSE, matrix = FALSE, ...)

## S4 method for signature 'ImagingExperiment'
spatialWeights(x, r, method = c("gaussian", "adaptive"),
    dist = "chebyshev", matrix = FALSE, BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'PositionDataFrame'
spatialWeights(x, r, matrix = FALSE, ...)

#### Methods for Cardinal 1.x classes ####
```

```
## S4 method for signature 'iSet'
findNeighbors(x, r, groups = x$sample, ...)

## S4 method for signature 'IAnnotatedDataFrame'
findNeighbors(x, r, groups = x$sample, dist = "chebyshev",
    offsets = FALSE, matrix = FALSE, ...)

## S4 method for signature 'iSet'
spatialWeights(x, r, method = c("gaussian", "adaptive"),
    matrix = FALSE, ...)

## S4 method for signature 'IAnnotatedDataFrame'
spatialWeights(x, r, matrix = FALSE, ...)
```

Arguments

x	An imaging dataset or data frame with spatial dimensions.
r	The spatial radius or distance.
groups	A factor giving which pixels should be treated as spatially-independent. Pixels in the same group are assumed to have a spatial relationship.
dist	The type of distance metric to use. The options are 'radial', 'manhattan', 'minkowski', and 'chebyshev' (the default).
offsets	Should the coordinate offsets from the center of each neighborhood be returned?
matrix	Should the result be returned as a sparse matrix instead of a list?
method	The method to use to calculate the spatial weights. The 'gaussian' method refers to Gaussian-distributed distance-based weights (alpha weights), and 'adaptive' refers to structurally-adaptive weights for bilaterla filtering (beta weights).
...	Addtional arguments to be passed to next method.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Value

Either a list of neighbors/weights or a sparse matrix (sparse_mat) giving the neighbors and weights for each pixel.

For spatialWeights, two types of weights are calculated and returned as a list:

The alpha weights are distance-based, following a Gaussian distributed that produces smaller weights for larger distances. The beta weights are adaptive weights used for bilateral filtering, which are based on the difference in the feature-vectors between pixels.

If method="gaussian" only the alpha weights are calcualated and the beta weights are all set to 1. If matrix=TRUE, the alpha and beta weights are multiplied together to produce the weights for the matrix; otherwise, both are returned separately.

Author(s)

Kylie A. Bemis

See Also

image

Examples

```
coord <- expand.grid(x=1:9, y=1:9)
values <- rnorm(nrow(coord))
pdata <- PositionDataFrame(coord=coord, values=values)

# find spatial neighbors
findNeighbors(pdata, r=1)

# calculate distance-based weights
spatialWeights(pdata, r=1)

# visualize weight matrix
W <- spatialWeights(pdata, r=1, matrix=TRUE)
image(as.matrix(W), col=bw.colors(100))
```

---

image-methods *Plot an image of the pixel data of an imaging dataset*

Description

Create and display images for the pixel data of an imaging dataset using a formula interface.

Usage

```
## S4 method for signature 'formula'
image(x, data = environment(x), ...,
    xlab, ylab, zlab, subset)

## S4 method for signature 'PositionDataFrame'
image(x, formula,
        groups = NULL,
        superpose = FALSE,
        strip = TRUE,
        key = superpose || !is.null(groups),
        normalize.image = c("none", "linear"),
        contrast.enhance = c("none", "suppression", "histogram"),
        smooth.image = c("none", "gaussian", "adaptive"),
        ...,
        xlab, xlim,
        ylab, ylim,
        zlab, zlim,
        asp = 1,
        layout,
        col = discrete.colors,
```

```
        colorscale = viridis,
        colorkey = !key,
        alpha.power = 1,
        subset = TRUE,
        add = FALSE)

## S4 method for signature 'SparseImagingExperiment'
image(x, formula,
        feature,
        feature.groups,
        groups = NULL,
        superpose = FALSE,
        strip = TRUE,
        key = superpose || !is.null(groups),
        fun = mean,
        normalize.image = c("none", "linear"),
        contrast.enhance = c("none", "suppression", "histogram"),
        smooth.image = c("none", "gaussian", "adaptive"),
        ...,
        xlab, xlim,
        ylab, ylim,
        zlab, zlim,
        asp = 1,
        layout,
        col = discrete.colors,
        colorscale = viridis,
        colorkey = !key,
        alpha.power = 1,
        subset = TRUE,
        add = FALSE)

## S4 method for signature 'SparseImagingExperiment'
image3D(x, formula, ..., alpha.power = 2)

## S4 method for signature 'MSImagingExperiment'
image(x, formula,
        feature = features(x, mz=mz),
        feature.groups,
        mz,
        plusminus,
        ...)

## S4 method for signature 'SparseImagingResult'
image(x, formula,
        model = modelData(x),
        superpose = is_matrix,
        ...,
        column,
```

```
        colorscale = cividis,
        colorkey = !superpose,
        alpha.power = 2)

## S4 method for signature 'PCA2'
image(x, formula,
    values = "scores", ...)

## S4 method for signature 'PLS2'
image(x, formula,
    values = c("fitted", "scores"), ...)

## S4 method for signature 'SpatialFastmap2'
image(x, formula,
    values = "scores", ...)

## S4 method for signature 'SpatialKMeans2'
image(x, formula,
    values = "cluster", ...)

## S4 method for signature 'SpatialShrunkenCentroids2'
image(x, formula,
    values = c("probability", "class", "scores"), ...)

## S4 method for signature 'SpatialDGMM'
image(x, formula,
    values = c("probability", "class", "mean"), ...)

## S4 method for signature 'MeansTest'
image(x, formula,
    values = "mean", jitter = TRUE, ...)

## S4 method for signature 'SegmentationTest'
image(x, formula,
    values = c("mean", "mapping"), jitter = TRUE, ...)

## S4 method for signature 'AnnotatedImage'
image(x, frame = 1, offset = coord(x),
        height, width,
        layout = !add,
        native = TRUE,
        interpolate = TRUE,
        add = FALSE, ...)

## S4 method for signature 'AnnotatedImageList'
image(x, i, frame = 1,
        strip = TRUE,
        layout = !add,
```

```
        native = TRUE,
        interpolate = TRUE,
        add = FALSE, ...)

## S4 method for signature 'AnnotatedImagingExperiment'
image(x, i, frame = 1, ...)
```

Arguments

x	An imaging dataset.
formula	A formula of the form 'z ~ x * y | g1 * g2 * ...' (or equivalently, 'z ~ x + y | g1 + g2 + ...'), indicating a LHS 'y' (on the y-axis) versus a RHS 'x' (on the x-axis) and conditioning variables 'g1, g2, ...'. Usually, the LHS is not supplied, and the formula is of the form '~ x * y | g1 * g2 * ...', and the y-axis is implicityl assumed to be the feature vectors corresponding to each pixel in the imaging dataset specified by the object 'x'. However, a variable evaluating to a vector of pixel values, or a sequence of such variables, can also be supplied. The RHS is evaluated in `pData(x)` and should provide values for the xy-axes. These must be spatial coordinates. The conditioning variables are evaluated in `fData(x)`. These can be specified in the formula as 'g1 * g2 * ...'. The argument 'feature.groups' allows an alternate way to specify a single conditioning variable. Conditioning variables specified using the formula interface will always appear on separate plots. This can be combined with 'superpose = TRUE' to both overlay plots based on a conditioning variable and use conditioning variables to create separate plots.
data	A `list` or `data.frame`-like object from which variables in `formula` should be taken.
mz	The m/z value(s) for which to plot the ion image(s).
plusminus	If specified, a window of m/z values surrounding the one given by `coord` will be included in the plot with `fun` applied over them, and this indicates the range of the window on either side.
feature	The feature or vector of features for which to plot the image. This is an expression that evaluates to a logical or integer indexing vector.
feature.groups	An alternative way to express a single conditioning variable. This is a variable or expression to be evaluated in `fData(x)`, expected to act as a grouping variable for the features specified by 'feature', typically used to distinguish different groups or ranges of features. Pixel vectors of images from features in the same feature group will have 'fun' applied over them; 'fun' will be applied to each feature group separately, usually for averaging. If 'superpose = FALSE' then these appear on separate plots.
groups	A variable or expression to be evaluated in `pData(x)`, expected to act as a grouping variable for the pixel regions in the image(s) to be plotted, typically used to distinguish different image regions by varying graphical parameters like color and line type. By default, if 'superpose = FALSE', these appear overlaid on the same plot.

`superpose` Should feature vectors from different feature groups specified by 'feature.groups' be superposed on the same plot? If 'TRUE' then the 'groups' argument is ignored.

`strip` Should strip labels indicating the plotting group be plotting along with the each panel? Passed to 'strip' in `levelplot` is 'lattice = TRUE'.

`key` A logical, or `list` containing components to be used as a key for the plot. This is passed to 'key' in `levelplot` if 'lattice = TRUE'.

`fun` A function to apply over pixel vectors of images grouped together by 'feature.groups'. By default, this is used for averaging over features.

`normalize.image` Normalization function to be applied to each image. The function can be user-supplied, of one of 'none' or 'linear'. The 'linear' normalization method normalized each image to the same intensity range using a linear transformation.

`contrast.enhance` Contrast enhancement function to be applied to each image. The function can be user-supplied, or one of 'none', 'histogram', or 'suppression'. The 'histogram' equalization method flatterns the distribution of intensities. The hotspot 'suppression' method uses thresholding to reduce the intensities of hotspots.

`smooth.image` Image smoothing function to be applied to each image. The function can be user-supplied, or one of 'none', 'gaussian', or 'adaptive'. The 'gaussian' smoothing method smooths images with a simple gaussian kernel. The 'adaptive' method uses bilateral filtering to preserve edges.

`xlab` Character or expression giving the label for the x-axis.

`ylab` Character or expression giving the label for the y-axis.

`zlab` Character or expression giving the label for the z-axis. (Only used for plotting 3D images.)

`xlim` A numeric vector of length 2 giving the left and right limits for the x-axis.

`ylim` A numeric vector of length 2 giving the top and bottom limits for the y-axis.

`zlim` A numeric vector of length 2 giving the lower and upper limits for the z-axis (i.e., the range of colors to be plotted).

`layout` The layout of the plots, given by a length 2 numeric as `c(ncol, nrow)`. This is passed to `levelplot` if 'lattice = TRUE'. For base graphics, this defaults to one plot per page.

`asp` The aspect ratio of the plot.

`col` A specification for the default plotting color(s) for groups.

`colorscale` The color scale to use for the z-axis of image intensities. This may be either a vector of colors or a function which takes a single numeric argument `n` and generates a vector of colors of length `n`.

`colorkey` Should a coloykey describing the z-axis be drawn with the plot?

`alpha.power` Opacity scaling factor (1 is linear).

`jitter` Should a small amount of noise be added to the image values before plotting them?

subset	An expression that evaluates to a logical or integer indexing vector to be evaluated in pData(x).
...	Additional arguments passed to the underlying plot functions.
i	Which data element should be plotted.
frame	Which frame of an image should be plotted.
offset	Absolute offset in x/y coordinates of the top-left corner of the image (from the origin).
height	The height of the plotted image.
width	The width of the plotted image.
native	Should a native raster (using integer color codes) be produced, or an rgb raster (using character color codes)?
interpolate	Should any linear interpolation be done when plotting the image?
model	A vector or list specifying which fitted model to plot. If this is a vector, it should give a subset of the rows of modelData(x) to use for plotting. Otherwise, it should be a list giving the values of parameters in modelData(x).
values	What kind of results should be plotted. This is the name of the object to plot in the ImagingResult object. Renamed from mode to avoid ambiguity.
column	What columns of the results should be plotted. If the results are a matrix, this corresponds to the columns to be plotted, which can be indicated either by numeric index or by name.
add	Should the method call plot.new() or be added to the current plot?

Note

In most cases, calling image3D(obj) is equivalent to image(obj, ~ x * y * z).

Author(s)

Kylie A. Bemis

See Also

plot, selectROI

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=2, npeaks=10, dim=c(10,10))
m <- mz(metadata(x)$design$featureData)

image(x, mz=m[1], plusminus=0.5)
image(x, mz=m[1], smooth.image="gaussian", contrast.enhance="histogram")
image(x, mz=m[1], colorscale=col.map("grayscale"))
image(x, mz=m[4:7], colorscale=col.map("cividis"))
image(x, mz=m[c(1,8)], normalize.image="linear", superpose=TRUE)
```

```
sm <- summarizePixels(x, FUN=c(tic="sum"), as="DataFrame")
pData(x)$tic <- sm$tic

image(x, tic ~ x * y, colorscale=magma)
```

ImageList-class *ImageList: Abstract image data list*

Description

The ImageList virtual class provides an formal abstraction for the imageData slot of ImagingExperiment objects. It is analogous to the Assays classes from the SummarizedExperiment package.

The ImageArrayList virtual class specializes the ImageList abstraction by assuming the array-like data elements all have conformable dimensions.

The SimpleImageList and SimpleImageArrayList subclasses are the default implementations.

The MSContinuousImagingSpectraList and MSProcessedImagingSpectraList classes are subclasses of SimpleImageArrayList that make certain assumptions about how the underlying data elements are stored (i.e., either dense or sparse). They are intended to be used with mass spectrometry imaging data.

Usage

```
# Create a SimpleImageList
ImageList(data)

# Create a SimpleImageArrayList
ImageArrayList(data)

# ImageArrayList class for 'continuous' (dense) MS imaging data
MSContinuousImagingSpectraList(data)

# ImageArrayList class for 'processed' (sparse) MS imaging data
MSProcessedImagingSpectraList(data)
```

Arguments

data	A SimpleList or list of array-like data elements, or an array-like object.

Details

ImageList and ImageArrayList objects have list-like semantics where the elements are array-like (i.e., have dim), where ImageArrayList makes the additional assumption that the array-like elements have identical dims for at least the first two dimensions.

The ImageList class includes:

- (1) The ImageList() and ImageArrayList() constructor functions.

- (2) Lossless back-and-forth coercion from/to SimpleList. The coercion method need not and should not check the validity of the returned object.
- (3) length, names, names<-, and `[[`, `[[<-` methods for ImageList, as well as `[`, `[<-`, rbind, and cbind methods for ImageArrayList.

See the documentation for the Assays class in the SummarizedExperiment package for additional details, as the implementation is quite similar, with the main difference being that all assumptions about the dimensions of the array-like data elements is contained in the ImageArrayList subclass. This is intended to allow subclasses of the ImageList class to handle images stored as arrays with non-conformable dimensions.

These classes are intended to eventually replace the ImageData classes.

Author(s)

Kylie A. Bemis

See Also

SimpleList

Examples

```
## create an ImageList object
data0 <- matrix(1:9, nrow=3)
data1 <- matrix(10:18, nrow=3)
data2 <- matrix(19:27, nrow=3)
idata <- ImageArrayList(list(d0=data0, d1=data1, d2=data2))

# subset all arrays at once
idataS <- idata[1:2,1:2]
all.equal(idataS[["d0"]], data0[1:2,1:2])

# combine over "column" dimension
idataB <- cbind(idata, idata)
all.equal(idataB[["d0"]], cbind(data0, data0))
```

ImagingExperiment-class

ImagingExperiment: Abstract class for imaging experiments

Description

The ImagingExperiment class is a virtual class for biological imaging experiments. It includes slots for sample/pixel metadata and for feature metadata. The class makes very few assumptions about the structure of the underlying imaging data, including the dimensions.

For a concrete subclass, see the SparseImagingExperiment class, which assumes that the image data can be represented as a matrix where columns represent pixels and rows represent features. The MSImagingExperiment subclass is further specialized for analysis of mass spectrometry imaging experiments.

Slots

`imageData`: An object inheriting from `ImageList`, storing one or more array-like data elements. No assumption is made about the shape of the arrays.

`featureData`: Contains feature information in a `DataFrame`. Each row includes the metadata for a single feature (e.g., a color channel, a molecular analyte, or a mass-to-charge ratio).

`elementMetadata`: Contains sample or pixel information in a `DataFrame`. Each row includes the metadata for a single observation (e.g., a sample or a pixel).

`metadata`: A `list` containing experiment-level metadata.

Methods

`imageData(object)`, `imageData(object) <- value`: Get and set the `imageData` slot.

`iData(object, i)`, `iData(object, i, ...) <- value`: Get or set the element `i` from the `imageData`. If `i` is missing, the first data element is returned.

`phenoData(object)`, `phenoData(object) <- value`: Get and set the `elementMetadata` slot.

`sampleNames(object)`, `sampleNames(object) <- value`: Get and set the row names of the `elementMetadata` slot.

`pData(object)`, `pData(object) <- value`: A shortcut for `phenoData(object)` and `phenoData(object)<-`.

`pixelData(object)`, `pixelData(object) <- value`: In subclasses where columns represent pixels, get and set the `elementMetadata` slot.

`pixelNames(object)`, `pixelNames(object) <- value`: In subclasses where columns represent pixels, get and set the row names of the `elementMetadata` slot.

`featureData(object)`, `featureData(object) <- value`: Get and set the `featureData` slot.

`featureNames(object)`, `featureNames(object) <- value`: Get and set the row names of the `featureData` slot.

`fData(object)`, `fData(object) <- value`: A shortcut for `featureData(object)` and `featureData(object)<-`.

`dim`: The dimensions of the object, as determined by the number of features (rows in `featureData`) and the number of samples/pixels (rows in `elementMetadata`).

`object$name`, `object$name <- value`: Get and set the `name` column in `pixelData`.

`object[[i]]`, `object[[i]] <- value`: Get and set the column `i` (a string or integer) in `pixelData`.

`object[i, j, ..., drop]`: Subset based on the rows (`fData`) and the columns (`pData`). The result is the same class as the original object.

`rbind(...)`, `cbind(...)`: Combine `ImagingExperiment` objects by row or column.

Author(s)

Kylie A. Bemis

See Also

`SparseImagingExperiment`, `MSImagingExperiment`

Examples

```
## cannot create an ImagingExperiment object
try(new("ImagingExperiment"))

## create an ImagingExperiment derived class
MyImagingExperiment <- setClass("MyImagingExperiment", contains="ImagingExperiment")
MyImagingExperiment()

removeClass("MyImagingExperiment")
```

ImagingResult-class *ImagingResult: Results of statistical analysis of imaging experiments*

Description

The `ImagingResult` class is a virtual class for containing the results of statistical analyses applied to imaging experiments. It includes the pixel and feature metadata of the original imaging experiment, but the image data may be missing. The results are stored as a list, where each element contains the results of a single model or parameter set. Results from multiple models or parameter sets may be stored together.

The `SparseImagingResult` subclass inherits from both `SparseImagingExperiment` and `ImagingResult`.

Slots

`imageData`**:** An object inheriting from `ImageArrayList`, storing one or more array-like data elements with conformable dimensions. This may be empty.

`featureData`**:** Contains feature information in a `XDataFrame`. Each row includes the metadata for a single feature (e.g., a color channel, a molecular analyte, or a mass-to-charge ratio).

`elementMetadata`**:** Contains pixel information in a `PositionDataFrame`. Each row includes the metadata for a single observation (e.g., a pixel), including specialized slot-columns for tracking pixel coordinates and experimental runs.

`resultData`**:** A `List` containing the results of statistical analysis. Each element contains the results of a single model or parameter set.

`modelData`**:** A `DataFrame` providing details about the models or parameters used in the analysis. Must have the same number of rows as the length of `resultData`.

`metadata`**:** A `list` containing experiment-level metadata.

Methods

All methods for `ImagingExperiment` also work on `ImagingResult` objects. Additional methods are documented below:

`modelData(object)`, `modelData(object) <- value`**:** Get or set the `modelData`.

`resultData(object, i, j)`, `resultData(object, i) <- value`**:** Get or set the corresponding element of `resultData`.

`resultNames(object)`**:** Get the names of the components of `resultData`.

Author(s)

Kylie A. Bemis

See Also

ImagingExperiment, SparseImagingExperiment

`intensity.colors` *Color palettes for imaging*

Description

Create a vector of n continuous or discrete colors.

Usage

```
color.map(map = c("redblack", "greenblack", "blueblack",
    "viridis", "cividis", "magma", "inferno", "plasma",
    "rainbow", "darkrainbow", "grayscale",
    "jet", "hot", "cool"), n = 100)

col.map(...)

intensity.colors(n = 100, alpha = 1)

jet.colors(n = 100, alpha = 1)

divergent.colors(n = 100, start = "#00AAEE",
    middle = "#FFFFFF", end = "#EE2200", alpha = 1)

risk.colors(n = 100, alpha = 1)

gradient.colors(n = 100, start = "#000000",
    end = "#00AAFF", alpha = 1)

bw.colors(n = 100, alpha = 1)

discrete.colors(n = 2, chroma = 150, luminance = 65, alpha = 1)

alpha.colors(col, n = 100,
    alpha = (seq_len(n)/n)^alpha.power,
    alpha.power = 2)

darkmode(default = TRUE)

lightmode(default = TRUE)
```

Arguments

map	the name of the colormap
n	the number of colors
...	arguments passed to color.map()
alpha	a vector of alpha values between 0 and 1
start	the start color value
middle	the middle color value
end	the end color value
chroma	the chroma of the color
luminance	the luminance of the color
col	the color(s) to expand with transparency
alpha.power	how the alpha should ramp as it increases
default	Should this be set as the default plotting mode?

Details

Most of these functions return a vector of colors.

Several of the options made available by color.map are borrowed from the viridisLite package, including 'viridis', 'cividis', 'magma', 'inferno', and 'plasma'. The original functions for these color palettes are also re-exported for use by users. See the documention for them in that package.

The darkmode and lightmode functions change the graphical parameters for the current graphics device accordingly. The new themes will be used for any subsequent plots.

Value

A palette of colors.

Author(s)

Kylie A. Bemis

See Also

viridis, cividis, magma, inferno, plasma

Examples

```
col <- gradient.colors(100^2)
if ( interactive() )
image(matrix(1:(100^2), nrow=100), col=col)
```

internal *Internal utilities for Cardinal*

Description

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

Author(s)

Kylie A. Bemis

legacy *Legacy classes and methods in Cardinal*

Description

These classes and methods are deprecated and should no longer be used.

The class definitions will remain for compatibility with `CardinalWorkflows` datasets and other older serialized datasets.

Objects of these classes should be updated to a supported version of the class.

For example, for an object of class `MSImageSet`:

```
object <- as(object, "MSImagingExperiment")
```

MassDataFrame-class *MassDataFrame: data frame with mass-to-charge ratio metadata*

Description

An `MassDataFrame` is an extension of the `XDataFrame` class with a special slot-column for observed mass-to-charge ratios.

Usage

```
MassDataFrame(mz, ..., row.names = NULL, check.names = TRUE)
```

Arguments

`mz`	A `numeric` vector of mass-to-charge ratios.
`...`	Named arguments that will become columns of the object.
`row.names`	Row names to be assigned to the object; no row names are assigned if this is `NULL`.
`check.names`	Should the column names be checked for syntactic validity?

Details

MassDataFrame is designed for mass spectrometry data. It includes a slot-column for the mass-to-charge ratio. It is intended to annotate either a single mass spectrum or an experiment where each mass spectrum share the same mass-to-charge ratios. The m/z values can be get and set by the mz(object) accessor, and are assumed to be unique and sorted in increasing order.

Methods

mz(object), mz(object) <- value: Get or set the mass-to-charge ratio slot-column.

resolution(object), resolution(object) <- value: Get or set the estimated mass resolution of the mass-to-charge ratios. Typically, this should not be set manually.

isCentroided(object): Guess whether the data are centroided or not, based on the m/z values.

as.list(x, ..., slots = TRUE): Coerce the object to a list, where the slot-columns are included by default. Use slots=FALSE to exclude the slot-columns.

Author(s)

Kylie A. Bemis

See Also

XDataFrame

Examples

```
## create an MassDataFrame object
mz <- mz(from=200, to=220, by=200)
values <- runif(length(mz))
fdata <- MassDataFrame(mz=mz, values=values)

## check the mass-to-charge ratio properties
head(mz(fdata))
resolution(fdata)
```

---

meansTest-methods *Linear model-based testing for summarized imaging experiments*

Description

Performs hypothesis testing for imaging experiments by fitting linear mixed models to summarizations or segmentations.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
meansTest(x, fixed, random, groups = run(x),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SparseImagingExperiment'
segmentationTest(x, fixed, random, groups = run(x),
        classControl = c("Ymax", "Mscore"),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialDGMM'
segmentationTest(x, fixed, random, model = modelData(x),
        classControl = c("Ymax", "Mscore"),
        BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'MeansTest'
summary(object, ..., BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SegmentationTest'
summary(object, ..., BPPARAM = getCardinalBPPARAM())
```

Arguments

`x`	An imaging dataset or segmented/summarized imaging dataset.
`fixed`	A one-sided formula giving the fixed effects of the model on the RHS. The response will added to the LHS, and the formula will be passed to the underlying modeling function.
`random`	A one-sided formula giving the random effects of the model on the RHS. See `lme` for the allowed specifications.
`groups`	The summarization units. Pixels from different groups will be segmented/summarized separately. *Each distinct observational unit (e.g., tissue sample) should be assigned to a unique group*.
`model`	An integer vector or `list` specifying which fitted model to plot. If this is an integer vector, it should give the rows indices of `modelData(x)` to use for plotting. Otherwise, it should be a list giving the values of parameters in `modelData(x)`.
`classControl`	Either the method used to match segmented classes to the fixed effects, or a list where each element is a vector of name-value pairs giving the mapping between groups and classes (e.g., c(group1=class1, group2=class2, ...)). For automated matching methods, 'Ymax' means to use the classes with the highest mean response for each group, and 'Mscore' means to select classses based on a match score quantifying the overlap between classes and fixed effects.
`...`	Passed to internal linear modeling methods.
`object`	A fitted model object to summarize.
`BPPARAM`	An optional instance of `BiocParallelParam`. See documentation for `bplapply`.

Value

An object of class `MeansTest` or `SegmentationTest`, which is a `ImagingResult`, where each element of the `resultData` slot contains at least the following components:

`model`: A linear model fitted using either `lm` or `lme`.

`data`: The summarized data used to fit the model.

Author(s)

Dan Guo and Kylie A. Bemis

See Also

`lm`, `lme`, `spatialDGMM`

Examples

```
set.seed(1)
x <- simulateImage(preset=4, nruns=3, npeaks=10,
    dim=c(10,10), peakheight=5, peakdiff=2,
    representation="centroid")

groups <- replace(run(x), !(x$circleA | x$circleB), NA)

fit <- meansTest(x, ~ condition, groups=groups)

summary(fit)
```

MSContinuousImagingExperiment-class

MSContinuousImagingExperiment: "Continuous" mass spectrometry imaging experiments

---

Description

The `MSContinuousImagingExperiment` class is a simple extension of `MSImagingExperiment` for dense spectra. All methods for that class apply. In addition, each data element must be stored as an ordinary R `matrix` or a column-major `matter_mat`.

Author(s)

Kylie A. Bemis

See Also

`MSImagingExperiment`, `MSProcessedImagingExperiment`

MSImagingExperiment-class
MSImagingExperiment: Mass spectrometry imaging experiments

Description

The MSImagingExperiment class is designed for mass spectrometry imaging experimental data and metadata. It is designed to contain full MSI experiments, including multiple runs and replicates, potentially across multiple files. Both 2D and 3D imaging experiments are supported, as well as any type of experimental metadata such as diagnosis, subject, time point, etc.

Usage

```
## Instance creation
MSImagingExperiment(
    imageData = matrix(nrow=0, ncol=0),
    featureData = MassDataFrame(),
    pixelData = PositionDataFrame(),
    metadata = list(),
    processing = SimpleList(),
    centroided = FALSE)

## Additional methods documented below
```

Arguments

imageData	Either a matrix-like object with number of rows equal to the number of features and number of columns equal to the number of pixels, or an ImageArrayList.
featureData	A MassDataFrame with feature metadata, with a row for each m/z value.
pixelData	A PositionDataFrame with pixel metadata, with a row for each pixel.
metadata	A list with experimental-level metadata.
processing	A SimpleList with processing steps. This should typically be empty for new objects.
centroided	FALSE if the object contains profile spectra and TRUE if the spectra have been peak-picked and centroided.

Details

The MSImagingExperiment class is designed as a replacement for the MSImageSet class, using a simplified, robust implementation that should be more future-proof and enable better support for large, high-resolution experiments, multimodal experiments, and experiments with specialized needs such as non-gridded pixel coordinates.

Subclasses MSContinuousImagingExperiment and MSProcessedImagingExperiment exist to allow downstream methods to make assumptions about the underlying data storage (dense matrices for 'continous' format and sparse matrices for 'processed' format), which can sometimes allow more efficient computations.

Slots

`imageData`: An object inheriting from `ImageArrayList`, storing one or more array-like data elements with conformable dimensions.

`featureData`: Contains feature information in a `MassDataFrame`. Each row includes the metadata associated with an m/z value.

`elementMetadata`: Contains pixel information in a `PositionDataFrame`. Each row includes the metadata for a single observation (e.g., a pixel), including specialized slot-columns for tracking pixel coordinates and experimental runs.

`metadata`: A `list` containing experiment-level metadata.

`processing`: A `SimpleList` containing processing steps (including both queued and previously executed processing steps).

`centroided`: `FALSE` if the object contains profile spectra and `TRUE` if the spectra have been peak-picked and centroided.

Methods

All methods for `ImagingExperiment` and `SparseImagingExperiment` also work on `MSImagingExperiment` objects. Additional methods are documented below:

`spectraData(object)`, `spectraData(object) <- value`: Get or set the spectra list (alias for `imageData(object)`).

`spectra(object)`, `spectra(object) <- value`: Get or set the spectra (alias for `iData(object)`).

`mz(object)`, `mz(object) <- value`: Get or set the m/z values from `pixelData`.

`resolution(object)`, `resolution(object) <- value`: Get or set the m/z resolution of the dataset. Typically, this should not be set manually.

`centroided(object)`, `centroided(object) <- value`: Get or set the spatial position slot-columns from `pixelData`.

`pixels(object, ..., coord, run)`: Returns the row indices of `pixelData` corresponding to conditions passed via `...`.

`features(object, ..., mz)`: Returns the row indices of `featureData` corresponding to conditions passed via `...`.

`pull(x, ...)`: Pull all data elements of `imageData` into memory as matrices.

`peaks(object)`, `peaks(object) <- value`: Attempt to get or set the matrix of peaks. Alias for `spectra()` if `centroided()` is TRUE; replacement version also sets `centroided` to TRUE.

`peakData(object)`, `peakData(object) <- value`: Attempt to get or set the underlying m/z and intensity arrays of the peak data in processed experiments. (Currently only implemented for `MSProcessedImagingExperiment`).

`isCentroided(object)`: Attempts to infer if the mass spectra are centroided or not (without referencing the `centroided` slot.

`msiInfo(object, ...)`: Returns metadata for writing the object to imzML.

`rbind(...)`, `cbind(...)`: Combine `MSImagingExperiment` objects by row or column.

Author(s)

Kylie A. Bemis

See Also

ImagingExperiment, SparseImagingExperiment, MSContinuousImagingExperiment, MSProcessedImagingExperiment

Examples

```
mz <- mz(from=200, to=220, by=400)
coord <- expand.grid(x=1:3, y=1:3)
data <- matrix(runif(length(mz) * nrow(coord)),
    nrow=length(mz), ncol=nrow(coord))

idata <- ImageArrayList(data)
fdata <- MassDataFrame(mz=mz)
pdata <- PositionDataFrame(coord=coord)

x <- MSImagingExperiment(
    imageData=idata,
    featureData=fdata,
    pixelData=pdata)

print(x)
```

MSImagingInfo-class *MSImagingInfo: Mass spectrometry imaging metadata for imzML conversion*

Description

The MSImagingInfo class is designed to contain metadata for reading/writing Cardinal objects from/to imzML files.

Methods

`length(object)`: The number of scans (i.e., the number of mass spectra).

`scans(object)`: Access the scan list metadata for writing to imzML.

`mzData(object)`: Access the m/z array list metadata for writing to imzML.

`intensityData(object)`: Access the intensity array list metadata for writing to imzML.

`isCentroided(object)`: Check whether the mass spectra are centroided.

`normalization(object)`, `normalization(object) <- value`: Accessor and setter function for the `normalization`.

`smoothing(object)`, `smoothing(object) <- value`: Accessor and setter function for the `smoothing`.

`baselineReduction(object)`, `baselineReduction(object) <- value`: Accessor and setter function for the `baselineReduction`.

`peakPicking(object)`, `peakPicking(object) <- value`: Accessor and setter function for the `peakPicking`.

`spectrumRepresentation(object)`, `spectrumRepresentation(object) <- value`: Accessor and setter function for the `spectrumRepresentation`.

`matrixApplication(object)`: Accessor function for `matrixApplication`.

`pixelSize(object)`: Accessor function for `pixelSize`.

`instrumentModel(object)`: Accessor function for `instrumentModel`.

`instrumentVendor(object)`: Accessor function for `instrumentVendor`.

`massAnalyzerType(object)`: Accessor function for `massAnalyzerType`.

`ionizationType(object)`: Accessor function for `ionizationType`.

`scanPolarity(object)`: Accessor function for `scanPolarity`.

`scanType(object)`: Accessor function for `scanType`.

`scanPattern(object)`: Accessor function for `scanPattern`.

`scanDirection(object)`: Accessor function for `scanDirection`.

`lineScanDirection(object)`: Accessor function for `lineScanDirection`.

Author(s)

Kylie A. Bemis

References

Schramm T, Hester A, Klinkert I, Both J-P, Heeren RMA, Brunelle A, Laprevote O, Desbenoit N, Robbe M-F, Stoeckli M, Spengler B, Rompp A (2012) imzML - A common data format for the flexible exchange and processing of mass spectrometry imaging data. Journal of Proteomics 75 (16):5106-5110. doi:10.1016/j.jprot.2012.07.026

See Also

MIAxE, MIAPE-Imaging

Examples

```
mz <- mz(from=200, to=220, by=400)
coord <- expand.grid(x=1:3, y=1:3)
data <- matrix(runif(length(mz) * nrow(coord)),
    nrow=length(mz), ncol=nrow(coord))

x <- MSImagingExperiment(
    imageData=ImageArrayList(data),
    featureData=MassDataFrame(mz=mz),
    pixelData=PositionDataFrame(coord=coord))

msiInfo(x)
```

MSProcessedImagingExperiment-class *MSProcessedImagingExperiment: "Processed" mass spectrometry imaging experiments*

Description

The MSProcessedImagingExperiment class is a simple extension of MSImagingExperiment for sparse spectra. All methods for that class apply. In addition, each data element must be stored as a column-major sparse_mat.

Methods

All methods for MSImagingExperiment also work on MSProcessedImagingExperiment objects. Additional methods are documented below:

`intensityData(object)`, `intensityData(object) <- value`: Get or set the underlying (pre-binned) intensity values associated with the sparse mass spectra.

`mzData(object)`, `mzData(object) <- value`: Get or set the underlying (pre-binned) m/z values associated with the sparse mass spectra.

`mz(object) <- value`: Setting the m/z values changes the m/z binning scheme for the entire dataset (without modifying the underlying data).

`resolution(object) <- value`: Setting the m/z resolution changes the m/z binning scheme for the entire dataset (without modifying the underlying data).

`tolerance(object)`, `tolerance(object) <- value`: Get or set the binning tolerance for sparse spectra or peaks.

`combiner(object)`, `combiner(object) <- value`: Get or set the binning function for sparse spectra or peaks.

`pull(x, ..., as.matrix=FALSE)`: Pull all data elements of `imageData` into memory as sparse matrices.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, MSContinuousImagingExperiment

`mz-methods` *Manipulate mass-to-charge-ratio values*

Description

This is a generic function for getting or setting 'mz' for an object with associated m/z values, or for generating a sequence of appropriate m/z values for such an object.

Usage

```
## S4 method for signature 'missing'
mz(from, to, by, resolution = 200, units = c("ppm", "mz"), ...)

mz(object, ...)

mz(object, ...) <- value
```

Arguments

`object`	An object with m/z values.
`value`	The value to set the m/z values.
`from, to`	The starting amd (maximal) end values of the sequence of m/z values.
`by`	The (approximate) interval between m/z values. For `units="ppm"`, rather than an exact step size, this actually corresponds to a binwidth, where each element of the sequence is considered the center of a bin.
`resolution`	Another way to specify the interval between m/z values. For `units="mz"`, this is the same as `by`. For `units="ppm"`, this is the half-binwdith.
`units`	The units for `by` and `resolution`. Either parts-per-million or absolute m/z increments.
`...`	Additional arguments (ignored).

Author(s)

Kylie A. Bemis

See Also

`MassDataFrame`

Examples

```
mz(from=200, to=220, by=300, units="ppm")
```

mzAlign-methods *Mass align an imaging dataset*

Description

Apply spectral alignment to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment,numeric'
mzAlign(object, ref, tolerance = NA, units = c("ppm", "mz"),
        span = 0.75, control = loess.control(), ...)

## S4 method for signature 'MSImagingExperiment,missing'
mzAlign(object, tolerance = NA, units = c("ppm", "mz"),
        span = 0.75, control = loess.control(), quantile = 0.2, ...)
```

Arguments

object	An imaging dataset.
ref	A reference to which to align the spectra.
tolerance	The tolerance to be used when matching the peaks in the unaligned spectra to the reference spectrum. If this is NA, then automatically guess a tolerance from the data.
units	The units to use for the tolerance.
span	The smoothing parameter for the local polynomial regression used to determine the warping function.
control	Additional control parameters for the local polynomial regression used to determine the warping function. See loess.control.
quantile	The top quantile of reference points (peaks detected via local maxima) to use from the reference spectrum.
...	Ignored.

Details

Mass alignment is performed against a vector of reference m/z values of expected peaks. The nearest local maxima to the reference peaks are detected in each unaligned spectrum (within tolerance), and then the unaligned spectra are warped to maximize correlation with the reference spectrum.

If no reference peaks are provided, then the mean spectrum is calculated instead, and reference peaks are selected by detecting local maxima. Some number of these reference points with the highest intensities (determined by quantile) are then used as the reference for alignment.

Internally, pixelApply is used to perform the alignment. See its documentation page for more details.

Value

An object of the same class with the aligned spectra.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, mzBin, peakAlign, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3), sdmz=500)
data <- data[,pData(data)$circle]

# queue spectral alignment
data <- mzAlign(data, tolerance=1, units="mz")

# apply spectral alignment
data_aligned <- process(data, plot=interactive())
```

---

mzBin-methods *Mass bin an imaging dataset*

Description

Apply mass binning to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment,numeric'
mzBin(object, ref, tolerance = NA, units = c("ppm", "mz"), fun=sum, ...)

## S4 method for signature 'MSImagingExperiment,missing'
mzBin(object, from=min(mz(object)), to=max(mz(object)), by,
        resolution = NA, units = c("ppm", "mz"), fun="sum", ...)
```

Arguments

object	An imaging dataset.
ref	A reference to which the m/z values are binned.
tolerance	The half-width(s) of the bins. If this is NA, then automatically guess a resolution from the data.

from, to	The starting amd (maximal) end values of the sequence of m/z values.
by	The (approximate) interval between m/z values. For units="ppm", rather than an exact step size, this actually corresponds to a binwidth, where each element of the sequence is considered the center of a bin.
resolution	Another way to specify the interval between m/z values. For units="mz", this is the same as by. For units="ppm", this is the half-binwdith. If this is NA, then automatically guess a resolution from the data.
units	The units for by and resolution. Either parts-per-million or absolute m/z increments.
fun	The function used to summarize each mass bin.
...	Ignored.

Details

The reference masses are considered to be the center of each bin. The bin is then expanded on either side according to half the value of width, and the intensities in each bin are summarized by applying fun.

Internally, pixelApply is used to apply the binning. See its documentation page for more details.

Value

An object of the same class with the binned spectra.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, mzAlign, peakBin, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]

# queue m/z binning
data <- mzBin(data, resolution=10, units="mz", fun="max")

# apply m/z binning
data_binned <- process(data, plot=interactive())
```

mzFilter-methods *Filter the features of an imaging dataset by intensity*

Description

Apply filtering to a mass spectrometry imaging dataset based on the intensities of each peak or mass feature.

Usage

```
## S4 method for signature 'MSImagingExperiment'
mzFilter(object, ..., freq.min = NA, rm.zero = TRUE)

## S4 method for signature 'MSImagingExperiment'
peakFilter(object, ..., freq.min = 0.01, rm.zero = TRUE)
```

Arguments

object	An object of class MSImagingExperiment.
freq.min	Minimum frequency; peaks that occur in the dataset in lesser proportion than this will be dropped.
rm.zero	Remove features with mean intensities of zero.
...	Additional arguments passed to the peak filtering method, or conditions evaluating to logical vectors where only those conditions that are TRUE are retained.

Details

When applied to a MSImagingExperiment object, mzFilter and peakFilter uses the summarize() to generate useful summary statistics about the mass features or detected peaks. These include the 'min', 'max', 'mean', 'sum', 'sd', and 'var' of the intensities for each mass feature or peak. These can be used in logical expressions to filter the features of the dataset.

Note that peakFilter is an alias for mzFilter, with different default parameters that are more appropriate for peak-picked data rather than profile spectra.

Unlike most other processing methods, peakFilter operates on the feature space (ion images) of the dataset.

Peak filtering is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: The vector of ion image intensities to filter.
- ...: Additional arguments.

A user-created function should return a logical: TRUE means keep the peak, and FALSE means remove the peak.

Internally, featureApply is used to apply the filtering. See its documentation page for more details on additional objects available to the environment installed to the peak filtering function.

Value

An object of the same class with the filtered peaks.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, peakPick, peakAlign, peakBin, featureApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]

# filter m/z features
process(mzFilter(data))

# queue peak picking, alignment, and filtering
data <- peakPick(data, method="simple", SNR=6)
data <- peakAlign(data, tolerance=200, units="ppm")
data <- peakFilter(data, freq.min=0.5)

# apply peak picking, alignment, and filtering
data_peaks <- process(data, plot=interactive())
```

---

normalize-methods *Normalize an imaging dataset*

Description

Apply normalization to the feature vectors of an imaging dataset.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
normalize(object, method = c("tic", "rms", "reference"), ...)

## Totial-ion-current normalization
normalize.tic(x, tic=length(x), ...)

## Root-mean-square normalization
normalize.rms(x, rms=1, ...)

## Reference normalization
normalize.reference(x, feature, scale=1, ...)
```

Arguments

object	An imaging dataset.
method	The normalization method to use.
...	Additional arguments passed to the normalization method.
x	The signal to be normalized.
tic	The value to which to normalize the total ion current.
rms	The value to which to normalize the root-mean-square.
feature	The feature to use as a reference for normalization.
scale	The value to which to normalize the reference.

Details

Normalization is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

A user-created function should return a numeric vector of the same length.

Internally, pixelApply is used to apply the normalization. See its documentation page for more details on additional objects available to the environment installed to the normalization function.

Value

An object of the same class with the normalized spectra.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]

# queue normalization
data <- normalize(data, method="tic")

# apply normalization
data_normalized <- process(data)
```

PCA-methods *Principal components analysis*

Description

Performs principal components analysis efficiently on large datasets using implicitly restarted Lanczos bi-diagonalization (IRLBA) algorithm for approximate singular value decomposition of the data matrix.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
PCA(x, ncomp = 3, center = TRUE, scale = FALSE, ...)

## S4 method for signature 'PCA2'
predict(object, newx, ncomp, ...)

## S4 method for signature 'PCA2'
summary(object, ...)
```

Arguments

x	The imaging dataset for which to calculate the principal components.
ncomp	The number of principal components to calculate.
center	Should the data be centered first? This is passed to scale.
scale	Shoud the data be scaled first? This is passed to scale.
...	Ignored.
object	The result of a previous call to PCA.
newx	An imaging dataset for which to calculate the principal components scores based on the aleady-calculated principal components loadings.

Value

An object of class PCA2, which is a ImagingResult, or an object of class PCA, which is a ResultSet. Each elemnt of resultData slot contains at least the following components:

loadings: A matrix with the principal component loadings.

scores: A matrix with the principal component scores.

sdev: The standard deviations of the principal components.

Author(s)

Kylie A. Bemis

See Also

OPLS, PLS, irlba, svd

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
data <- simulateImage(preset=2, npeaks=20, dim=c(6,6),
    representation="centroid")

# project to FastMap components
pca <- PCA(data, ncomp=2)

# visualize first 2 components
image(pca, superpose=FALSE)
```

---

peakAlign-methods *Peak align an imaging dataset*

Description

Apply peak alignment to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment,missing'
peakAlign(object, tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'MSImagingExperiment,character'
peakAlign(object, ref, ...)

## S4 method for signature 'MSImagingExperiment,numeric'
peakAlign(object, ref, ...)
```

Arguments

object	An imaging dataset.
ref	A reference to which to align the peaks.
tolerance	The tolerance to be used when aligning detected peaks to the reference. If this is NA, then automatically guess a tolerance from the data.
units	The units to use for the tolerance. Either parts-per-million or the raw m/z values.
...	Ignored.

Details

When applied to a MSImagingExperiment object with no other reference, peakAlign uses summarize() to calculate the mean spectrum, and then uses the local maxima of the mean spectrum as the reference. Alternatively, a vector of m/z values or a column name in the featureData that should be used as the reference may be provided. Finally, if the featureData has an numeric vector element named "reference peaks" among its metadata(), this vector is used as the reference.

Peak alignment is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: The vector of m/z values to be aligned.
- y: The vector of reference m/z values.
- ...: Additional arguments.

A user-created function should return a vector of the same length as x and y where NA values indicate no match, and non-missing values give the index of the matched peak in the reference set.

Internally, pixelApply is used to apply the peak alignment. See its documentation page for more details on additional objects available to the environment installed to the peak alignment function.

Value

An object of class MSImagingExperiment with the peak aligned spectra.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, peakPick, peakFilter, peakBin, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]

# queue peak picking and alignment
data <- peakPick(data, method="simple", SNR=6)
data <- peakAlign(data, tolerance=200, units="ppm")

# apply peak picking and alignment
data_peaks <- process(data, plot=interactive())
```

peakBin-methods *Peak bin an imaging dataset*

Description

Apply peak binning to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment,numeric'
peakBin(object, ref, type=c("area", "height"),
        tolerance = NA, units = c("ppm", "mz"), ...)

## S4 method for signature 'MSImagingExperiment,missing'
peakBin(object, type=c("area", "height"),
        tolerance = NA, units = c("ppm", "mz"), ...)
```

Arguments

object	An imaging dataset.
ref	A reference to which the peaks are binned.
type	Should the summarized intensity of the peak by the maximum height of the peak or the area under the curve?
tolerance	The tolerance to be used when matching the m/z features in the dataset to the reference. If this is NA, then automatically guess a resolution from the data.
units	The units to use for the tolerance.
...	Ignored.

Details

Peak binning is performed by first matching the m/z-values in the dataset to those in the reference, and then finding the boundaries of the peak by detecting the nearest local minima. Then either the maximum height or the area under the curve of the peak are returned.

Internally, pixelApply is used to apply the binning. See its documentation page for more details.

Value

An object of the same class with the binned peaks.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, peakPick, peakAlign, peakFilter, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]
ref <- mz(metadata(data)$design$featureData)

# queue peak binning
data <- peakBin(data, ref=ref, type="height")

# apply peak binning
data_peaks <- process(data, plot=interactive())
```

---

peakPick-methods *Peak pick an imaging dataset*

Description

Apply peak picking to a mass spectrometry imaging dataset.

Usage

```
## S4 method for signature 'MSImagingExperiment'
peakPick(object, method = c("mad", "simple", "adaptive"), ...)

## Local maxima and SNR with noise based on local MAD
peakPick.mad(x, SNR=6, window=5, blocks=1, fun=mean, tform=diff, ...)

## Local maxima and SNR with constant noise based on SD
peakPick.simple(x, SNR=6, window=5, blocks=100, ...)

## Local maxima and SNR with adaptive noise based on SD
peakPick.adaptive(x, SNR=6, window=5, blocks=100, spar=1, ...)

## LIMPIC peak detection
peakPick.limpic(x, SNR=6, window=5, blocks=100, thresh=0.75, ...)
```

Arguments

object	An imaging dataset.
method	The peak picking method to use.
...	Additional arguments passed to the peak picking method.
x	The mass spectrum to be peak picked.
SNR	The minimum signal-to-noise ratio to be considered a peak.
window	The window width for seeking local maxima.

blocks	The number of blocks in which to divide the mass spectrum in order to calculate the noise.
fun	The function used to estimate centrality and average absolute deviation.
tform	A transformation to be applied to the mass spectrum before estimating noise.
spar	Smoothing parameter for the spline smoothing applied to the spectrum in order to decide the cutoffs for throwing away false noise spikes that might occur inside peaks.
thresh	The thresholding quantile to use when comparing slopes in order to throw away peaks that are too flat.

Details

Peak picking is usually performed using the provided functions, but a user-created function can also be passed to method. In this case it should take the following arguments:

- x: A numeric vector of intensities.
- ...: Additional arguments.

When applied to an MSImagingExperiment object, a user-created function should return a integer vector giving the indices of the detected peaks.

Internally, pixelApply is used to apply the peak picking. See its documentation page for more details on additional objects available to the environment installed to the peak picking function.

Value

An object of the same class with the peak picked spectra. Note that the full mass range is retained and the peaks are unaligned, so peakAlign should be called before applying further methods.

Author(s)

Kylie A. Bemis

References

Mantini, D., Petrucci, F., Pieragostino, D., Del Boccio, P., Di Nicola, M., Di Ilio, C., et al. (2007). LIMPIC: a computational method for the separation of protein MALDI-TOF-MS signals from noise. BMC Bioinformatics, 8(101), 101. doi:10.1186/1471-2105-8-101

See Also

MSImagingExperiment, peakAlign, peakFilter, peakBin, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3))
data <- data[,pData(data)$circle]
```

```
# queue peak picking
data <- peakPick(data, method="simple", SNR=6)

# apply peak picking
data_peaks <- process(data, plot=interactive())
```

`pixelApply-methods` *Apply functions over imaging datasets*

Description

Apply an existing or a user-specified function over either all of the features or all of the pixels of a `SparseImagingExperiment` or `SImageSet`. These are provided by analogy to the 'apply' family of functions, but allowing greater efficiency and convenience when applying functions over an imaging dataset.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
pixelApply(.object, .fun, ...,
    .simplify = TRUE, .outpath = NULL,
    .blocks = getCardinalNumBlocks(), .verbose = getCardinalVerbose(),
    BPPARAM = getCardinalBPPARAM())

## S4 method for signature 'SparseImagingExperiment'
featureApply(.object, .fun, ...,
    .simplify = TRUE, .outpath = NULL,
    .blocks = getCardinalNumBlocks(), .verbose = getCardinalVerbose(),
    BPPARAM = getCardinalBPPARAM())
```

Arguments

`.object`	An imaging dataset.
`.fun`	The function to be applied.
`...`	Additional arguments passed to `.fun`.
`.blocks`	If FALSE (the default), each feature-vector or image-vector will be loaded and processed individually. If TRUE, or a positive integer, the data will be split into that many blocks, and the function (specified by `.fun`) will be applied to each block. The number of blocks can be specified as a number, or `getCardinalNumBlocks()` will be used.
`.simplify`	If applying over blocks, then a function to be used to simplify the list of results. Otherwise, a logical value giving whether the results should be simplified into a matrix or array rather than a list.
`.outpath`	The path to a file where the output data will be written. Results will be kept in-memory if this is NULL. Results will be coerced to a numeric vector before being written to file.

.verbose	Should progress messages be printed?
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Details

For SparseImagingExperiment-derived classes

For pixelApply, the function is applied to the feature vector(s) belonging to pixel(s).

For featureApply, the function is applied to the vector(s) of intensity values (i.e., the flattened image) corresponding to the feature(s).

For spatialApply, the function is applied to neighborhoods of feature-vectors corresponding to neighboring pixels. The maximum distance in each dimension for a pixel to be considered a neighbor is given by .r. The first argument to .fun is a matrix of column-vectors.

If .blocks is provided (either TRUE or a positive integer), then the data is split into blocks beforehand, and entire blocks are loaded and passed to the function as a matrix of column-vectors. Otherwise, single vectors are passed to the function individually. If blocks are used, then .simplify should be a function that simplifies a list of results.

Note that for spatialApply (only), if blocks are used, the result is NOT guaranteed to be in the correct order; instead the result will have a attr(ans, "idx") attribute giving the proper order (pixel IDs) of the results, and the .simplify function should likely re-order the results.

The following attributes are assigned to the object passed to .fun, accessible via attr():

- idx: The indices of the current pixel(s) or feature(s).
- mcols: Either featureData(.object) for pixelApply or pixelData(.object) for featureApply.

Additionally, the following attributes are made available during a call to spatialyApply():

- centers: A vector indicating which column(s) should be considered the center(s) of the neighborhood(s).
- neighbors: A list of vectors indicating which column(s) should be considered the neighborhoods. Only relevant if using .blocks.
- offsets: A matrix where the rows are the spatial offsets of the pixels in the neighborhood(s) from the center pixel(s).

Additionally, any named components of .params will also be provided as attributes, subsetted to the current element.

For SImageSet-derived classes

The use of .pixel and .feature can be used to apply the function over only a subset of pixels or features (or both), allowing faster computation when calculation on only a subset of data is needed.

For pixelApply, the function is applied to the feature vector belonging to each pixel. The use of .feature.groups allows codetapply-like functionality on the feature vectors, applied separately to each pixel.

For featureApply, the function is applied to the vector of intensity values (i.e., the flattened image) corresponding to each feature. The use of .feature.groups allows codetapply-like functionality on the flattened image intensity vectors, applied separately to each feature.

The fData from .object is installed into the environment of .fun for pixelApply, and the pData from .object is installed into the environment of .fun for featureApply. This allows access to the symbols from fData or pData during the execution of .fun. If .fun already has an environment, it is retained as the parent of the installed environment.

Additionally, the following objects are made available by installing them into the .fun environment:

- .Object: The passed .object. (Note the case.)
- .Index: The index of the current iteration.

Value

If .simplify = FALSE, a list. Otherwise, a vector or matrix, or a higher-dimensional array if grouping is specified, or the output of the provided .simplify function.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, MSImageSet

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# calculate TIC for each pixel
tic <- pixelApply(data, sum)

# calculate mean spectrum
ms <- featureApply(data, mean)
```

---

plot-methods *Plot a signal from the feature data of an imaging dataset*

Description

Create and display plots for the feature data of an imaging dataset using a formula interface.

Usage

```
## S4 method for signature 'DataFrame,ANY'
plot(x, y, ...)

## S4 method for signature 'XDataFrame,missing'
plot(x, formula,
        groups = NULL,
        superpose = FALSE,
        strip = TRUE,
        key = superpose || !is.null(groups),
        ...,
        xlab, xlim,
        ylab, ylim,
        layout,
        col = discrete.colors,
        breaks = "Sturges",
        grid = FALSE,
        jitter = FALSE,
        subset = TRUE,
        add = FALSE)

## S4 method for signature 'MassDataFrame,missing'
plot(x, ..., type = if (isCentroided(x)) "h" else "l")

## S4 method for signature 'SparseImagingExperiment,missing'
plot(x, formula,
        pixel,
        pixel.groups,
        groups = NULL,
        superpose = FALSE,
        strip = TRUE,
        key = superpose || !is.null(groups),
        fun = mean,
        hline = 0,
        ...,
        xlab, xlim,
        ylab, ylim,
        layout,
        col = discrete.colors,
        grid = FALSE,
        subset = TRUE,
        add = FALSE)

## S4 method for signature 'MSImagingExperiment,missing'
plot(x, formula,
        pixel = pixels(x, coord=coord, run=run),
        pixel.groups,
        coord,
```

```
        run,
        plusminus,
        ...,
        xlab, ylab,
        type = if ( is_centroided ) 'h' else 'l')

## S4 method for signature 'SparseImagingResult,missing'
plot(x, formula,
        model = modelData(x),
        superpose = is_matrix,
        ...,
        column,
        xlab, ylab,
        type = 'h')

## S4 method for signature 'PCA2,missing'
plot(x, formula,
    values = "loadings", ...)

## S4 method for signature 'PLS2,missing'
plot(x, formula,
    values = c("coefficients", "loadings", "weights"), ...)

## S4 method for signature 'SpatialFastmap2,missing'
plot(x, formula,
    values = "correlation", ...)

## S4 method for signature 'SpatialKMeans2,missing'
plot(x, formula,
    values = c("centers", "correlation"), ...)

## S4 method for signature 'SpatialShrunkenCentroids2,missing'
plot(x, formula,
    values = c("centers", "statistic", "sd"), ...)

## S4 method for signature 'SpatialDGMM,missing'
plot(x, model = modelData(x),
    values = "density", type = 'l', ...)

## S4 method for signature 'MeansTest,missing'
plot(x, model = modelData(x),
    values = "fixed", ...)

## S4 method for signature 'SegmentationTest,missing'
plot(x, model = modelData(x),
    values = "fixed", ...)

## S4 method for signature 'AnnotatedImage,ANY'
```

```
plot(x, breaks = "Sturges",
        key = TRUE, col,
        add = FALSE, ...)

## S4 method for signature 'AnnotatedImageList,ANY'
plot(x, i, breaks = "Sturges",
        strip = TRUE,
        key = TRUE, col,
        layout = !add,
        add = FALSE, ...)

## S4 method for signature 'AnnotatedImagingExperiment,ANY'
plot(x, i, ...)
```

Arguments

x	An imaging dataset.
formula, y	A formula of the form 'y ~ x | g1 * g2 * ...' (or equivalently, 'y ~ x | g1 + g2 + ...'), indicating a LHS 'y' (on the y-axis) versus a RHS 'x' (on the x-axis) and conditioning variables 'g1, g2, ...'. Usually, the LHS is not supplied, and the formula is of the form '~ x | g1 * g2 * ...', and the y-axis is implicityl assumed to be the feature vectors corresponding to each pixel in the imaging dataset specified by the object 'x'. However, a variable evaluating to a feature vector, or a sequence of such variables, can also be supplied. The RHS is evaluated in `fData(x)` and should provide values for the x-axis. The conditioning variables are evaluated in `pData(x)`. These can be specified in the formula as 'g1 * g2 * ...'. The argument 'pixel.groups' allows an alternate way to specify a single conditioning variable. Conditioning variables specified using the formula interface will always appear on separate plots. This can be combined with 'superpose = TRUE' to both overlay plots based on a conditioning variable and use conditioning variables to create separate plots.
coord	A named vector or list giving the coordinate(s) of the pixel(s) to plot.
run	A character, factor, or integer vector giving the run(s) of the pixel(s) to plot.
plusminus	If specified, a window of pixels surrounding the one given by `coord` will be included in the plot with `fun` applied over them, and this indicates the number of pixels to include on either side.
pixel	The pixel or vector of pixels for which to plot the feature vectors. This is an expression that evaluates to a logical or integer indexing vector.
pixel.groups	An alternative way to express a single conditioning variable. This is a variable or expression to be evaluated in `pData(x)`, expected to act as a grouping variable for the pixels specified by 'pixel', typically used to distinguish different regions of the imaging data for comparison. Feature vectors from pixels in the same pixel group will have 'fun' applied over them; 'fun' will be applied to each pixel group separately, usually for averaging. If 'superpose = FALSE' then these appear on separate plots.

groups	A variable or expression to be evaluated in fData(x), expected to act as a grouping variable for the features in the feature vector(s) to be plotted, typically used to distinguish different groups of features by varying graphical parameters like color and line type. By default, if 'superpose = FALSE', these appear overlaid on the same plot.
superpose	Should feature vectors from different pixel groups specified by 'pixel.groups' be superposed on the same plot?
strip	Should strip labels indicating the plotting group be plotting along with the each panel? Passed to 'strip' in xyplot.
key	A logical, or list containing components to be used as a key for the plot. This is passed to 'key' in levelplot if 'lattice = TRUE'.
fun	A function to apply over feature vectors grouped together by 'pixel.groups'. By default, this is used for averaging over pixels.
hline	The y-value(s) for a horizontal reference line(s).
xlab	Character or expression giving the label for the x-axis.
ylab	Character or expression giving the label for the x-axis.
xlim	A numeric vector of length 2 giving the left and right limits for the x-axis.
ylim	A numeric vector of length 2 giving the lower and upper limits for the y-axis.
layout	The layout of the plots, given by a length 2 numeric as c(ncol, nrow). This is passed to levelplot if 'lattice = TRUE'. For base graphics, this defaults to one plot per page.
col	A specification for the default plotting color(s).
type	A character indicating the type of plotting.
grid	Should a grid be added to the plot?
jitter	Should a small amount of noise be added to numeric variables before plotting them?
breaks	The number of breaks when plotting a histogram.
subset	An expression that evaluates to a logical or integer indexing vector to be evaluated in fData(x).
...	Additional arguments passed to the underlying plot or xyplot functions.
i	Which data element should be plotted.
model	A vector or list specifying which fitted model to plot. If this is a vector, it should give a subset of the rows of modelData(x) to use for plotting. Otherwise, it should be a list giving the values of parameters in modelData(x).
values	What kind of results should be plotted. This is the name of the object to plot in the ImagingResult object. Renamed from mode to avoid ambiguity.
column	What columns of the results should be plotted. If the results are a matrix, this corresponds to the columns to be plotted, which can be indicated either by numeric index or by name.
add	Should the method call plot.new() or be added to the current plot?

Author(s)

Kylie A. Bemis

See Also

image

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=2, npeaks=10, dim=c(10,10))
m <- mz(metadata(x)$design$featureData)

plot(x, pixel=23)
plot(x, coord=c(x=3, y=3), plusminus=1)
plot(x, coord=c(x=3, y=3), groups=mz > 1000)
plot(x, coord=c(x=7, y=7), superpose=TRUE)

sm <- summarizeFeatures(x, FUN=c("mean", "sd"), as="DataFrame")

featureData(x)$mean <- sm$mean
featureData(x)$sd <- sm$sd

plot(x, mean + I(-sd) ~ mz, superpose=TRUE)
```

PLS-methods *Partial least squares*

Description

Performs partial least squares (also called projection to latent structures or PLS) on an imaging dataset. This will also perform discriminant analysis (PLS-DA) if the response is a `factor`. Orthogonal partial least squares options (O-PLS and O-PLS-DA) are also available.

Usage

```
## S4 method for signature 'SparseImagingExperiment,ANY'
PLS(x, y, ncomp = 3, method = c("pls", "opls"),
        center = TRUE, scale = FALSE,
        iter.max = 100, ...)

## S4 method for signature 'SparseImagingExperiment,ANY'
OPLS(x, y, ncomp = 3, ...)

## S4 method for signature 'PLS2'
predict(object, newx, newy, ncomp, ...)
```

```
## S4 method for signature 'PLS2'
fitted(object, ...)

## S4 method for signature 'PLS2'
summary(object, ...)
```

Arguments

x	The imaging dataset on which to perform partial least squares.
y	The response variable, which can be a matrix or a vector for ordinary PLS, or a factor or a character for PLS-DA.
ncomp	The number of PLS components to calculate.
method	The function used to calculate the projection.
center	Should the data be centered first? This is passed to scale.
scale	Shoud the data be scaled first? This is passed to scale.
iter.max	The number of iterations to perform for the NIPALS algorithm.
...	Passed to the next PLS method.
object	The result of a previous call to PLS.
newx	An imaging dataset for which to calculate their PLS projection and predict a response from an already-calculated PLS object.
newy	Optionally, a new response from which residuals should be calcualted.

Value

An object of class PLS2, which is a ImagingResult, or an object of class PLS, which is a ResultSet. Each elemnt of resultData slot contains at least the following components:

fitted: The fitted response.

loadings: A matrix with the explanatory variable loadings.

weights: A matrix with the explanatory variable weights.

scores: A matrix with the component scores for the explanatary variable.

Yscores: A matrix objects with the component scores for the response variable.

Yweights: A matrix objects with the response variable weights.

coefficients: The matrix of the regression coefficients.

The following components may also be available for classes OPLS and OPLS2.

Oloadings: A matrix objects with the orthogonal explanatory variable loadings.

Oweights: A matrix with the orthgonal explanatory variable weights.

If y is a categorical variable, then a categorical class prediction will also be available in addition to the fitted numeric response.

Author(s)

Kylie A. Bemis

References

Trygg, J., & Wold, S. (2002). Orthogonal projections to latent structures (O-PLS). Journal of Chemometrics, 16(3), 119-128. doi:10.1002/cem.695

See Also

PCA, spatialShrunkenCentroids,

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=2, npeaks=10, dim=c(10,10),
    snoise=1, sdpeaks=1, representation="centroid")

y <- makeFactor(circle=pData(x)$circle, square=pData(x)$square)

pls <- PLS(x, y, ncomp=1:3)

summary(pls)

opls <- OPLS(x, y, ncomp=1:3)

summary(pls)
```

PositionDataFrame-class

PositionDataFrame: data frame with spatial position metadata

Description

An PositionDataFrame is an extension of the XDataFrame class with special slot-columns for spatial coordinates. It is designed specifically with biological imaging experiments in mind, so it also has an additional slot-column for tracking the experimental run.

Usage

```
PositionDataFrame(coord, run, ..., row.names = NULL, check.names = TRUE)
```

Arguments

coord	A data.frame-like object containing columns which are spatial coordinates. This will be coerced to a DataFrame.
run	A factor with levels for each experimental run.
...	Named arguments that will become columns of the object.
row.names	Row names to be assigned to the object; no row names are assigned if this is NULL.
check.names	Should the column names be checked for syntactic validity?

Details

PositionDataFrame is designed for spatial data, specifically for biological imaging data. It includes a slot-column for the experimental run. In most 2D imaging experiments, each distinct image is considered a distinct run. No additional assumptions are made about the spatial structure of the data, and non-gridded spatial coordinates are allowed.

This class is intended to eventually replace the IAnnotatedDataFrame class, and implements similar concepts but with a more robust and modern infrastructure.

Methods

run(object), run(object) <- value: Get or set the experimental run slot-column.

runNames(object), runNames(object) <- value: Get or set the experimental run levels.

coord(object), coord(object) <- value: Get or set the spatial position slot-columns.

coordLabels(object), coordLabels(object) <- value: Get or set the names of the spatial position slot-columns.

gridded(object), gridded(object) <- value: Get or set whether the spatial positions are gridded or not. Typically, this should not be set manually.

resolution(object), resolution(object) <- value: Get or set the spatial resolution of the spatial positions. Typically, this should not be set manually.

dims(object): Get the gridded dimensions of the spatial positions (i.e., as if projected to an image raster).

is3D(object): Check if the data is 3D or not.

as.list(x, ..., slots = TRUE): Coerce the object to a list, where the slot-columns are included by default. Use slots=FALSE to exclude the slot-columns.

Author(s)

Kylie A. Bemis

See Also

XDataFrame

Examples

```
## Create an PositionDataFrame object
coord <- expand.grid(x=1:3, y=1:3)
values <- seq_len(nrow(coord))
pdata <- PositionDataFrame(coord=coord, values=values)

## Check the spatial properties
gridded(pdata)
resolution(pdata)
dims(pdata)
```

---

process-methods *Delayed Processing of Imaging Datasets*

Description

Queue pre-processing steps on an imaging dataset and apply them, possibly writing out the processed data to a file.

Usage

```
## S4 method for signature 'MSImagingExperiment'
process(object, ..., delay = FALSE,
    outpath = NULL, imzML = FALSE)

## S4 method for signature 'SparseImagingExperiment'
process(object, fun, ...,
    kind = c("pixel", "feature", "global"),
    moreargs = NULL,
    prefun, preargs,
    postfun, postargs,
    plotfun,
    label = "",
    delay = FALSE,
    plot = FALSE,
    par = NULL,
    outpath = NULL,
    BPPARAM = getCardinalBPPARAM())
```

Arguments

object	An imaging dataset.
fun	A function to apply to each feature-vector or image-vector.
...	Additional arguments to fun.
delay	Should the function fun be applied now, or queued and delayed until process() is called again?

outpath	The path to a file where the results will be written by pixelApply or featureApply. If NULL, then the results are returned in-memory.
imzML	Should the output file be an imzML file? Note: some processing methods are not be supported with this option.
kind	What kind of processing to perform? Over pixels, over features, or global processing of the dataset as a single unit.
moreargs	Additional arguments to be passed to fun. This is primarily useful if some of the arguments to fun conflict with arguments to process.
prefun	A pre-processing function to be applied to the entire dataset, taking the dataset as its first argument. This should return another object of the same class.
preargs	Additional arguments to prefun, as a list.
postfun	A post-processing function to be applied to the output, taking the result as its first argument, and the original dataset as its second argument. This should return another object of the same class as the original dataset.
postargs	Additional arguments to postfun, as a list.
plotfun	A function to be used to plot the output of fun, taking at least two arguments: (1) the resulting vector and (2) the input vector.
label	The label of the processing step. This is used to identify it in the queue, and is printed as it is being processed.
plot	Plot the function for each pixel or feature while it is being processed? Only possible if BPPARAM=SerialParam().
par	Plotting parameters to be passed to plotfun.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Details

This method allows queueing of delayed processing to an imaging dataset. All of the registered processing steps will be applied in sequence whenever process() is called next with delay=FALSE. The processing can be over feature-vectors (e.g., mass spectra), over image-vectors, or over the entire dataset as a unit. The processing is performed in parallel using the current registered parallel backend.

The method for MSImagingExperiment allows writing the output directly to an imzML file, with certain restrictions. Some pre-processing methods are not supported with this option, and the experiment must not contain multiple runs.

Value

An object of the same class (or subclass) as the original imaging dataset, with the data processing queued or applied.

Author(s)

Kylie A. Bemis

See Also

SparseImagingExperiment, MSImagingExperiment, pixelApply, featureApply, normalize, smoothSignal, reduceBaseline, peakPick, peakAlign, peakFilter, peakBin

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, dim=c(10,10), baseline=1)
data_c <- data[,pData(data)$circle]

tmp <- process(data, function(s) log2(abs(s)))

tmp1 <- process(data, abs, delay=TRUE)

tmp2 <- process(tmp1, log2, delay=TRUE)

process(tmp2)
```

---

readMSIData *Read mass spectrometry imaging data files*

Description

Read supported mass spectrometry imaging data files. Supported formats include imzML and Analyze 7.5.

Usage

```
## Read any supported MS imaging file
readMSIData(file, ...)

## Read imzML files
readImzML(name, folder = getwd(), attach.only = TRUE,
mass.range = NULL, resolution = NA, units = c("ppm", "mz"),
guess.max = 1000L, as = "MSImagingExperiment", parse.only = FALSE,
BPPARAM = getCardinalBPPARAM(), ...)

## Read Analyze 7.5 files
readAnalyze(name, folder = getwd(), attach.only = TRUE,
as = "MSImagingExperiment", ...)
```

Arguments

file	A description of the data file to be read. This may be either an absolute or relative path. The file extension must be included.

name	The common (base) file name for the '.imzML' and '.ibd' files for imzML or for the '.hdr', '.t2m', and '.img' files for Analyze 7.5.
folder	The path to the folder containing the data files.
attach.only	Attach the file as a matter on-disk matrix for reading on-demand, rather than loading the data into memory.
mass.range	For 'processed' imzML files, the mass range to use for the imported data. If known, providing this can improve the loading time dramatically, as otherwise it is calculated from reading the dataset directly.
resolution	For 'processed' imzML files, the accuracy to which the m/z values will be binned after reading. For units="ppm", this is the half-binwidth, and should be set to the native accuracy of the mass spectrometer, if known. For units="mz", this is simply the step size between m/z bins. If this is NA, then automatically guess a resolution from the data.
units	The units for resolution. Either parts-per-million or absolute m/z units.
guess.max	The number of spectra used when guessing the mass range and resolution.
as	After reading in the data, what class of object should be returned? As of Cardinal version >= 2.6, only MSImagingExperiment is supported.
parse.only	If TRUE, return only the parsed imzML metadata without creating a new imaging experiment object. (May be useful for diagnosing import problems.)
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply. This is only used when mass.range=NULL and attach.only=TRUE, when reading the mass range from the m/z data of a "processed" imzML file.
...	Additional arguments passed to read functions.

Details

In the current implementation, the file extensions must match exactly: '.imzML' and '.ibd' for imzML and '.hdr', '.t2m', and '.img' for Analyze 7.5.

The readImzML function supports reading and returning both the 'continuous' and 'processed' formats.

When attach.only=TRUE, the data is not loaded into memory; only the experimental metadata is read, and the intensity data will only be accessed on-demand. For large datasets, this is memory-efficient. For smaller datasets, this may be slower than simply reading the entire dataset into memory.

If the mass range is known, setting mass.range will make reading data much faster for very large datasets.

If problems are encountered while trying to import imzML files, the files should be verified and fixed with the Java-based imzMLValidator application: https://gitlab.com/imzML/imzMLValidator/.

Value

A MSImagingExperiment object.

Author(s)

Kylie A. Bemis

References

Schramm T, Hester A, Klinkert I, Both J-P, Heeren RMA, Brunelle A, Laprevote O, Desbenoit N, Robbe M-F, Stoeckli M, Spengler B, Rompp A (2012) imzML - A common data format for the flexible exchange and processing of mass spectrometry imaging data. Journal of Proteomics 75 (16):5106-5110. doi:10.1016/j.jprot.2012.07.026

See Also

writeMSIData

`reduceBaseline-methods`

Reduce the baseline for an imaging dataset

Description

Apply baseline reduction to the feature vectors of an imaging dataset.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
reduceBaseline(object, method = c("locmin", "median"), ...)

## Local minima baseline reduction
reduceBaseline.locmin(x, window=5, ...)

## Interpolated median baseline reduction
reduceBaseline.median(x, blocks=500, fun=median, spar=1, ...)
```

Arguments

`object`	An imaging dataset.
`method`	The baseline reduction method to use.
`...`	Additional arguments passed to the baseline reduction method.
`x`	The signal to be baseline-corrected.
`blocks`	The number of intervals to break the mass spectrum into in order to choose minima or medians from which to interpolate the baseline.
`fun`	Function used to determine the points from which the baseline will be interpolated.
`spar`	Smoothing parameter for the spline smoothing applied to the spectrum in order to decide the cutoffs for throwing away baseline references that might occur inside peaks.
`window`	The sliding window (number of data points) to consider when determining the local minima.

Details

Baseline reduction is usually performed using the provided functions, but a user-created function can also be passed to `method`. In this case it should take the following arguments:

- `x`: A `numeric` vector of intensities.
- `...`: Additional arguments.

A user-created function should return a `numeric` vector of the same length. with the baseline-subtracted intensities.

Internally, `pixelApply` is used to apply the baseline reduction. See its documentation page for more details on additional objects available to the environment installed to the baseline reduction function.

Value

An object of the same class with the baseline-subtracted spectra.

Author(s)

Kylie A. Bemis

See Also

`MSImagingExperiment`, `pixelApply`, `process`

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3), baseline=1)
data <- data[,pData(data)$circle]

# queue baseline reduction
data <- reduceBaseline(data, method="median", blocks=100)

# apply baseline reduction
data_nobaseline <- process(data, plot=interactive())
```

`reexports` *Objects exported from other packages*

Description

These objects are imported from other packages and have been re-exported by Cardinal for user convenience.

`maggritr`: `%>%`

`viridisLite`: `viridis`, `cividis`, `magma`, `inferno`, `plasma`

`selectROI-methods` *Select regions-of-interest of an imaging dataset*

Description

Manually select regions-of-interest or pixels on an imaging dataset. The `selectROI` method uses the built-in `locator` function. The method has the same form as the `image` method for plotting imaging datasets.

The results are returned as logical vectors indicating which pixels have been selected. These logical vectors can be combined into factors using the `makeFactor` function.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
selectROI(object, ..., mode = c("region", "pixels"))

makeFactor(..., ordered = FALSE)
```

Arguments

`object`	An imaging dataset.
`mode`	What kind of selection to perform: 'region' to select a region-of-interest, or 'pixels' to select individual pixels.
`...`	Addtional arguments to be passed to `image` for `selectROI`, or name-value pairs of logical vectors to be combined by `makeFactor`.
`ordered`	Should the resulting factor be ordered or not?

Value

A `logical` vector of length equal to the number of pixels for `selectROI`.

A `factor` of the same length as the passed `logical` vectors for `makeFactor`.

Author(s)

Kylie A. Bemis

See Also

`image`

simulateSpectrum *Simulate a mass spectrum or MS imaging experiment*

Description

Simulate mass spectra or complete MS imaging experiments, including a possible baseline, spatial and spectral noise, mass drift, mass resolution, and multiplicative variation, etc.

A number of preset imaging designs are available for quick-and-dirty simulation of images.

These functions are designed for small proof-of-concept examples and testing, and may not scale well to simulating larger datasets.

Usage

```
simulateSpectrum(n = 1L, peaks = 50L,
    mz = rlnorm(peaks, 7, 0.3), intensity = rlnorm(peaks, 1, 0.9),
    from = 0.9 * min(mz), to = 1.1 * max(mz), by = 400,
    sdpeaks = sdpeakmult * log1p(intensity), sdpeakmult = 0.2,
    sdnoise = 0.1, sdmz = 10, resolution = 1000, fmax = 0.5,
    baseline = 0, decay = 10, units=c("ppm", "mz"),
    representation = c("profile", "centroid"), ...)

simulateImage(pixelData, featureData, preset,
    from = 0.9 * min(mz), to = 1.1 * max(mz), by = 400,
    sdrun = 1, sdpixel = 1, spcorr = 0.3, sptype = "SAR",
    representation = c("profile", "centroid"), units=c("ppm", "mz"),
    as = c("MSImagingExperiment", "SparseImagingExperiment"),
    BPPARAM = getCardinalBPPARAM(), ...)

addShape(pixelData, center, size, shape=c("circle", "square"), name=shape)

presetImageDef(preset = 1L, nruns = 1, npeaks = 30L,
    dim = c(20L, 20L), peakheight = 1, peakdiff = 1,
    sdsample = 0.2, jitter = TRUE, ...)
```

Arguments

n	The number of spectra to simulate.
peaks, npeaks	The number of peaks to simulate. Not used if mz and intensity are provided.
mz	The theoretical m/z values of the simulated peaks.
intensity	The mean intensities of the simulated peaks.
from	The minimum m/z value used for the mass range.
to	The maximum m/z value used for the mass range.
by	The step-size used for the observed m/z-values of the profile spectrum.
sdpeaks	The standard deviation(s) for the distributions of observed peak intensities on the log scale.

sdpeakmult	A multiplier used to calculate sdpeaks based on the mean intensities of peaks; used to simulate multiplicative variance. Not used if sdpeaks is provided.
sdnoise	The standard deviation of the random noise in the spectrum on the log scale.
sdmz	The standard deviation of the mass error in the observed m/z values of peaks, in units indicated by units.
resolution	The mass resolution as defined by m / dm, where m is the observed mass and dm 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). Note that this is NOT the same as the definition of resolution in the readImzML function.
fmax	The fraction of the maximum peak height to use when defining the mass 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 at the lower mass range of the spectrum.
units	The units for by and sdmz. Either parts-per-million or absolute m/z units.
representation	Should a profile spectrum be returned or only the centroided peaks?
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
pixelData	A PositionDataFrame giving the pixel design of the experiment. The names of the columns should match the names of columns in featureData. Each column should be a logical vector corresponding to a morphological substructure, indicate which pixels belong to that substructure.
featureData	A MassDataFrame giving the feature design of the experiment. Each row should correspond to an expected peak. The names of the columns should match the names of columns in pixelData. Each column should be a numeric vector corresponding to a morphological substructure, giving the mean intensity of that peak for that substructure.
preset	A number indicating a preset image definition to use.
nruns	The number of runs to simulate for each condition.
sdrun	A standard deviation giving the run-to-run variance.
sdpixel	A standard deviation giving the pixel-to-pixel variance.
spcorr	The spatial autocorrelation. Must be between 0 and 1, where spcorr=0 indicates no spatial autocorrelation.
sptype	The type of spatial autocorrelation.
as	The class of object to be returned.
...	Additional arguments to pass to simulateSpectrum or presetImageDef.
dim	The dimensions of the preset image.
peakheight	Reference intensities used for peak heights by the preset.
peakdiff	A reference intensity difference used for the mean peak height difference between conditions, for presets that simulate multiple conditions.
sdsample	A standard deviation giving the amount of variation from the true peak heights for this simulated sample.

jitter	Should random noise be added to the location and size of the shapes?
center	The center of the shape.
size	The size of the shape (from the center).
shape	What type of shape to add.
name	The name of the added column.

Details

The simulateSpectrum() and simulateImage() functions are used to simulate mass spectra and MS imaging experiments. They provide a great deal of control over the parameters of the simulation, including all sources of variation.

For simulateImage(), the user should provide the design of the simulated experiment via matching columns in pixelData and featureData, where each column corresponds to different morphological substructures or differing conditions. These design data frames are returned in the metadata() of the returned object for later reference.

A number of presets are defined by presetImageDef(), which returns only the pixelData and featureData necessary to define the experiment for simulateImage(). These can be referenced for help in understanding how to define experiments for simulateImage().

The preset images are:

- 1: a centered circle
- 2: a topleft circle and a bottomright square
- 3: two corner squares and a centered circle
- 4: a centered circle with conditions A and B in different runs
- 5: a topleft circle and a bottomright square with conditions A and B in different runs
- 6: two corner squares and a centered circle; the circle has conditions A and B in different runs
- 7: matched pairs of circles with conditions A and B within the same runs; includes reference peaks
- 8: matched pairs of circles inside squares with conditions A and B within the same runs; includes reference peaks
- 9: a small sphere inside a larger sphere (3D)

The addShape() function is provided for convenience when generating the pixelData for simulateImage(), as a simple way of adding morphological substructures using basic shapes such as squares and circles.

Value

For simulateSpectrum, a list with elements:

- mz: a numeric vector of the observed m/z values
- intensity: a numeric vector or matrix of the intensities

For simulateImage, a MSImagingExperiment or a SparseImagingExperiment.

For addShape, a new PositionDataFrame with a logical column added for the corresponding shape.

For presetImageDef, a list with two elements: the pixelData and featureData to be used as input to simulateImage().

Author(s)

Kylie A. Bemis

See Also

simulateSpectrum, simulateImage

Examples

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

# generate a spectrum
s <- simulateSpectrum(1)
plot(intensity ~ mz, data=s, type="l")

# generate a noisy low-resolution spectrum with a baseline
s <- simulateSpectrum(1, baseline=2, sdnoise=0.3, resolution=100)
plot(intensity ~ mz, data=s, type="l")

# generate a high-resolution spectrum
s <- simulateSpectrum(1, peaks=100, resolution=10000)
plot(intensity ~ mz, data=s, type="l")

# generate an image
x <- simulateImage(preset=1, npeaks=10, dim=c(10,10))
m <- mz(metadata(x)$design$featureData)

image(x, mz=m[5])

plot(x, coord=c(x=3, y=3))
```

---

slice-methods *Slice an image*

Description

Slice an imaging dataset as a "data cube".

Usage

```
## S4 method for signature 'SparseImagingExperiment'
slice(x, ..., drop=TRUE)
```

Arguments

x	An imaging dataset.
...	Conditions describing features to slice, passed to `features()`.
drop	Should redundant array dimensions be dropped? If TRUE, dimensions with only one level are dropped using `drop`.

Details

Because `SparseImagingExperiment` objects may be pixel-sparse, their data is always internally represented as a matrix rather than an array, where each column is a feature-vector. Only columns for non-missing pixels are retained. This is simpler and more space-efficient if the image is non-rectangular, non-gridded, or has many missing values.

However, it is often necessary to index into the data as if it were an actual "data cube", with explicit array dimensions for each spatial dimension. `slice()` allows this by slicing the object as a "data cube", and returning an image array from the object.

For non-rectangular data, this may result in missing values. For non-gridded data, images must be projected to an array (with a regular grid), and the result may not represent the underlying values exactly.

Value

An array representing the sliced image(s).

Author(s)

Kylie A. Bemis

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=1, npeaks=10, dim=c(10,10), representation="centroid")
m <- mz(metadata(x)$design$featureData)

# slice image for first feature
slice(x, 1)

# slice by m/z-value
slice(x, mz=m[1])

# slice multiple
slice(x, mz=m[1:3])
```

`smoothSignal-methods` *Smooth the signals of a imaging dataset*

Description

Apply smoothing to the feature vectors of an imaging dataset.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
smooth(x, ...)

## S4 method for signature 'SparseImagingExperiment'
smoothSignal(object, method = c("gaussian", "sgolay", "ma"), ...)

## Gaussian smoothing
smoothSignal.gaussian(x, sd=window/4, window=5, ...)

## Savitsky-Golay smoothing
smoothSignal.sgolay(x, order=3, window=order + 3 - order %% 2, ...)

## Moving average smoothing
smoothSignal.ma(x, coef=rep(1, window + 1 - window %% 2), window=5, ...)
```

Arguments

`object`	An imaging dataset.
`method`	The smoothing method to use.
`...`	Additional arguments passed to the smoothing method.
`x`	The signal or dataset to be smoothed.
`sd`	The standard deviation for the Gaussian kernel.
`window`	The smoothing window.
`order`	The order of the smoothing filter.
`coef`	The coefficients for the moving average filter.

Details

Smoothing is usually performed using the provided functions, but a user-created function can also be passed to `method`. In this case it should take the following arguments:

- `x`: A `numeric` vector of intensities.
- `...`: Additional arguments.

A user-created function should return a `numeric` vector of the same length.

Internally, `pixelApply` is used to apply the smooothing. See its documentation page for more details on additional objects available to the environment installed to the smoothing function.

Value

An object of the same class with the smoothed spectra.

Author(s)

Kylie A. Bemis

See Also

MSImagingExperiment, pixelApply, process

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
data <- simulateImage(preset=1, npeaks=10, dim=c(3,3), baseline=1)
data <- data[,pData(data)$circle]

# queue smoothing
data <- smoothSignal(data, method="ma", window=9)

# apply smoothing
data_smooth <- process(data, plot=interactive())
```

SparseImagingExperiment-class

SparseImagingExperiment: Pixel-sparse imaging experiments

Description

The SparseImagingExperiment class specializes the virtual ImagingExperiment class by assuming that each pixel may be a high-dimensional feature vector (e.g., a spectrum), but the pixels themselves may be sparse. Therefore, the data may be more efficiently stored as a matrix where rows are features and columns are pixels, rather than storing the full, dense datacube.

Both 2D and 3D data are supported. Non-gridded pixel coordinates are allowed.

The MSImagingExperiment subclass adds design features for mass spectrometry imaging experiments.

Usage

```
## Instance creation
SparseImagingExperiment(
    imageData = matrix(nrow=0, ncol=0),
    featureData = DataFrame(),
    pixelData = PositionDataFrame(),
    metadata = list(),
```

```
    processing = SimpleList())

## Additional methods documented below
```

Arguments

imageData	Either a matrix-like object with number of rows equal to the number of features and number of columns equal to the number of pixels, or an ImageArrayList.
featureData	A DataFrame with feature metadata, with a row for each feature.
pixelData	A PositionDataFrame with pixel metadata, with a row for each pixel.
metadata	A list with experimental-level metadata.
processing	A SimpleList with processing steps. This should typically be empty for new objects.

Slots

imageData: An object inheriting from ImageArrayList, storing one or more array-like data elements with conformable dimensions.

featureData: Contains feature information in a DataFrame. Each row includes the metadata for a single feature (e.g., a color channel, a molecular analyte, or a mass-to-charge ratio).

elementMetadata: Contains pixel information in a PositionDataFrame. Each row includes the metadata for a single observation (e.g., a pixel), including specialized slot-columns for tracking pixel coordinates and experimental runs.

metadata: A list containing experiment-level metadata.

processing: A SimpleList containing processing steps (including both queued and previously executed processing steps).

Methods

All methods for ImagingExperiment also work on SparseImagingExperiment objects. Additional methods are documented below:

pixels(object, ...): Returns the row indices of pixelData corresponding to conditions passed via ....

features(object, ...): Returns the row indices of featureData corresponding to conditions passed via ....

run(object), run(object) <- value: Get or set the experimental run slot-column from pixelData.

runNames(object), runNames(object) <- value: Get or set the experimental run levels from pixelData.

coord(object), coord(object) <- value: Get or set the spatial position slot-columns from pixelData.

coordLabels(object), coordLabels(object) <- value: Get or set the names of the spatial position slot-columns from pixelData.

gridded(object), gridded(object) <- value: Get or set whether the spatial positions are gridded or not. Typically, this should not be set manually.

`resolution(object)`, `resolution(object) <- value`: Get or set the spatial resolution of the spatial positions. Typically, this should not be set manually.

`dims(object)`: Get the gridded dimensions of the spatial positions (i.e., as if projected to an image raster).

`is3D(object)`: Check if the data is 3D or not.

`slice(object, ...)`: Slice the data as a data cube (i.e., as if projected to an multidimensional image raster).

`processingData(object)`, `processingData(object) <- value`: Get or set the `processing` slot.

`preproc(object)`: List the preprocessing steps queued and applied to the dataset.

`pull(x, ...)`: Pull all data elements of `imageData` into memory as matrices.

`object[i, j, ..., drop]`: Subset based on the rows (`featureData`) and the columns (`pixelData`). The result is the same class as the original object.

`rbind(...)`, `cbind(...)`: Combine `SparseImagingExperiment` objects by row or column.

Author(s)

Kylie A. Bemis

See Also

`ImagingExperiment`, `MSImagingExperiment`

Examples

```
data <- matrix(1:9^2, nrow=9, ncol=9)
t <- seq_len(9)
a <- seq_len(9)
coord <- expand.grid(x=1:3, y=1:3)

idata <- ImageArrayList(data)
fdata <- XDataFrame(t=t)
pdata <- PositionDataFrame(coord=coord, a=a)

x <- SparseImagingExperiment(
        imageData=idata,
        featureData=fdata,
        pixelData=pdata)

print(x)
```

spatialDGMM-methods *Spatially-aware Dirichlet Gaussian mixture model*

Description

Fits spatially-aware Dirichlet Gaussian mixture models to each feature and each run in an imaging experiment. Each image is segmented and the means and variances of all Gaussian components are estimated. A linear filter with a spatial kernel is applied to the component probabilities to achieve spatial smoothing. Simulated annealing is used in the EM-algorithm to avoid local optimia and achieve more accurate parameter estimates.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
spatialDGMM(x, r = 1, k = 3, groups = run(x),
    method = c("gaussian", "adaptive"),
    dist = "chebyshev", annealing = TRUE,
    init = c("kmeans", "gmm"), p0 = 0.05,
    iter.max = 100, tol = 1e-9,
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialDGMM'
summary(object, ...)
```

Arguments

x	The imaging dataset to segment or classify.
r	The spatial neighborhood radius of nearby pixels to consider. This can be a vector of multiple radii values.
k	The maximum number of segments (clusters). This can be a vector to try initializing the clustering with different numbers of maximum segments. The final number of segments may differ.
groups	Pixels from different groups will be segmented separately. For the validity of downstream statistical analysis, it is important that *each distinct observational unit (e.g., tissue sample) is assigned to a unique group*.
method	The method to use to calculate the spatial smoothing weights. The 'gaussian' method refers to spatially-aware (SA) weights, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) weights.
dist	The type of distance metric to use when calculating neighboring pixels based on r. The options are 'radial', 'manhattan', 'minkowski', and 'chebyshev' (the default).
annealing	Should simulated annealing be used during the optimization process to improve parameter estimates?
init	Should the parameter estimates be initialized using k-means ('kmeans') or Gaussian mixture model ('gmm')?

p0	A regularization parameter applied to estimated posterior class probabilities to avoid singularities. Must be positive for successful gradient descent optimization. Changing this value (within reason) should have only minimal impact on values of parameter estimates, but may greatly affect the algorithm's speed and stability.
iter.max	The maximum number of EM-algorithm iterations.
tol	The tolerance convergence criterion for the EM-algorithm. Corresponds to the change in log-likelihood.
...	Passed to internal methods.
object	A fitted model object to summarize.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Value

An object of class SpatialDGMM, which is a ImagingResult, where each element of the resultData slot contains at least the following components:

estimates: A list giving the parameter estimates for the means and variances for each Gaussian component.

class: The predicted Gaussian component.

probability: The probability of class membership for each Gaussian component.

Author(s)

Dan Guo and Kylie A. Bemis

References

Guo, D., Bemis, K., Rawlins, C., Agar, J., and Vitek, O. (2019.) Unsupervised segmentation of mass spectrometric ion images characterizes morphology of tissues. Proceedings of ISMB/ECCB, Basel, Switzerland, 2019.

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(2)
x <- simulateImage(preset=3, dim=c(10,10), npeaks=6,
    peakheight=c(4,6,8), representation="centroid")

res <- spatialDGMM(x, r=1, k=5, method="adaptive")

summary(res)

image(res, model=list(feature=3))
```

spatialFastmap-methods

Spatially-aware FastMap projection

Description

Performs spatially-aware FastMap projection.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
spatialFastmap(x, r = 1, ncomp = 3,
    method = c("gaussian", "adaptive"),
    metric = c("average", "correlation", "neighborhood"),
    dist = "chebyshev", tol.dist = 1e-9,
    iter.max = 1, BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialFastmap2'
summary(object, ...)
```

Arguments

x	The imaging dataset for which to calculate the FastMap components.
r	The neighborhood spatial smoothing radius.
ncomp	The number of FastMap components to calculate.
method	The method to use to calculate the spatial smoothing kernels for the embedding. The 'gaussian' method refers to spatially-aware (SA) weights, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) weights.
metric	The dissimilarity metric to use when comparing spectra, where 'average' (the default) means to use the differences of spatially-smoothed spectra, 'correlation' means to use the correlations of spatially-smoothed spectra, and 'neighborhood' means to use pairwise differences of each spectrum in the neighborhoods. Previous versions used 'neighborhood', which is the algorithm of Alexandrov & Kobarg; 'average' is the current default, because it handles non-gridded pixels better than 'neighborhood'.
dist	The type of distance metric to use when calculating neighboring pixels based on r. The options are 'radial', 'manhattan', 'minkowski', and 'chebyshev' (the default).
tol.dist	The distance tolerance used for matching pixels when calculating pairwise distances between neighborhoods. This parameter should only matter when the data is not gridded. (Only considers 'radial' distance.)
iter.max	The number of iterations to perform when choosing the pivot vectors for each dimension.
...	Ignored.
object	A fitted model object to summarize.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Value

An object of class `SpatialFastmap2`, which is a `ImagingResult`, or an object of class `SpatialFastmap`, which is a `ResultSet`. Each element of the `resultData` slot contains at least the following components:

`scores`**:** A matrix with the FastMap component scores.

`correlation`**:** A matrix with the feature correlations with each FastMap component.

`sdev`**:** The standard deviations of the FastMap scores.

Author(s)

Kylie A. Bemis

References

Alexandrov, T., & Kobarg, J. H. (2011). Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering. Bioinformatics, 27(13), i230-i238. doi:10.1093/bioinformatics/btr246

Faloutsos, C., & Lin, D. (1995). FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. Presented at the Proceedings of the 1995 ACM SIGMOD international conference on Management of data.

See Also

`PCA`, `spatialKMeans`, `spatialShrunkenCentroids`

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
data <- simulateImage(preset=2, npeaks=20, dim=c(6,6),
    representation="centroid")

# project to FastMap components
fm <- spatialFastmap(data, r=1, ncomp=2, method="adaptive")

# visualize first 2 components
image(fm, superpose=FALSE)
```

---

spatialKMeans-methods *Spatially-aware k-means clustering*

Description

Performs spatially-aware (SA) or spatially-aware structurally-adaptive (SASA) clustering of imaging data. The data are first projected into an embedded feature space where spatial structure is maintained using the Fastmap algorithm, and then ordinary k-means clustering is performed on the projected dataset.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
spatialKMeans(x, r = 1, k = 3,
    method = c("gaussian", "adaptive"),
    dist = "chebyshev", tol.dist = 1e-9,
    iter.max = 10, nstart = 10,
    algorithm = c("Hartigan-Wong", "Lloyd", "Forgy", "MacQueen"),
    ncomp = 10, BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialKMeans2'
summary(object, ...)
```

Arguments

x	The imaging dataset to cluster.
r	The spatial neighborhood radius of nearby pixels to consider. This can be a vector of multiple radii values.
k	The number of clusters. This can be a vector to try different numbers of clusters.
method	The method to use to calculate the spatial smoothing kernels for the embedding. The 'gaussian' method refers to spatially-aware (SA) clustering, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) clustering.
dist	The type of distance metric to use when calculating neighboring pixels based on r. The options are 'radial', 'manhattan', 'minkowski', and 'chebyshev' (the default).
tol.dist	The distance tolerance used for matching pixels when calculating pairwise distances between neighborhoods. This parameter should only matter when the data is not gridded. (Only considers 'radial' distance.)
iter.max	The maximum number of k-means iterations.
nstart	The number of restarts for the k-means algorithm.
algorithm	The k-means algorithm to use. See kmeans for details.
ncomp	The number of fastmap components to calculate.
...	Ignored.
object	A fitted model object to summarize.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.

Value

An object of class SpatialKMeans2, which is a ImagingResult, or an object of class SpatialKMeans, which is a ResultSet. Each element of the resultData slot contains at least the following components:

cluster: A vector of integers indicating the cluster for each pixel in the dataset.

centers: A matrix of cluster centers.

correlation: A matrix with the feature correlations with each cluster.

Author(s)

Kylie A. Bemis

References

Alexandrov, T., & Kobarg, J. H. (2011). Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering. Bioinformatics, 27(13), i230-i238. doi:10.1093/bioinformatics/btr246

Faloutsos, C., & Lin, D. (1995). FastMap: A Fast Algorithm for Indexing, Data-Mining and Visualization of Traditional and Multimedia Datasets. Presented at the Proceedings of the 1995 ACM SIGMOD international conference on Management of data.

See Also

spatialShrunkenCentroids

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=3, dim=c(10,10), npeaks=10,
    peakheight=c(4,6,8), representation="centroid")

res <- spatialKMeans(x, r=1, k=4, method="adaptive")

summary(res)

image(res, model=1)
```

---

spatialShrunkenCentroids-methods

Spatially-aware shrunken centroid clustering and classification

---

Description

Performs spatially-aware nearest shrunken centroid clustering or classification on an imaging dataset. These methods use statistical regularization to shrink the t-statistics of the features toward 0 so that unimportant features are removed from the analysis. A Gaussian spatial kernel or an adaptive kernel based on bilateral filtering are used for spatial smoothing.

Usage

```
## S4 method for signature 'SparseImagingExperiment,missing'
spatialShrunkenCentroids(x, r = 1, k = 3, s = 0,
    method = c("gaussian", "adaptive"),
    dist = "chebyshev", init = NULL,
    iter.max = 10, BPPARAM = getCardinalBPPARAM(), ...)
```

```
## S4 method for signature 'SparseImagingExperiment,ANY'
spatialShrunkenCentroids(x, y, r = 1, s = 0,
    method = c("gaussian", "adaptive"),
    dist = "chebyshev", priors = table(y),
    BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialShrunkenCentroids2'
predict(object, newx, newy, BPPARAM = getCardinalBPPARAM(), ...)

## S4 method for signature 'SpatialShrunkenCentroids2'
fitted(object, ...)

## S4 method for signature 'SpatialShrunkenCentroids2'
summary(object, ...)
```

Arguments

x	The imaging dataset to segment or classify.
y	A factor or character response.
r	The spatial neighborhood radius of nearby pixels to consider. This can be a vector of multiple radii values.
k	The maximum number of segments (clusters). This can be a vector to try initializing the clustering with different numbers of maximum segments. The final number of segments may differ.
s	The sparsity thresholding parameter by which to shrink the t-statistics.
method	The method to use to calculate the spatial smoothing weights. The 'gaussian' method refers to spatially-aware (SA) weights, and 'adaptive' refers to spatially-aware structurally-adaptive (SASA) weights.
dist	The type of distance metric to use when calculating neighboring pixels based on r. The options are 'radial', 'manhattan', 'minkowski', and 'chebyshev' (the default).
init	Initial cluster configuration. This may either be the result of a call to spatialKMeans, or a list of factors giving the initial cluster configurations.
iter.max	The maximum number of clustering iterations.
priors	Prior probabilities on the classes for classification. Improper priors will be normalized automatically.
...	Passed to internal methods.
BPPARAM	An optional instance of BiocParallelParam. See documentation for bplapply.
object	The result of a previous call to spatialShrunkenCentroids.
newx	An imaging dataset for which to calculate the predicted response from shrunken centroids.
newy	Optionally, a new response from which residuals should be calculated.

Value

An object of class `SpatialShrunkenCentroids2`, which is a `ImagingResult`, or an object of class `SpatialShrunkenCentroids`, which is a `ResultSet`. Each element of the `resultData` slot contains at least the following components:

`class, classes`: A factor indicating the predicted class for each pixel in the dataset.

`probability, probabilities`: A matrix of class probabilities.

`centers`: A matrix of shrunken class centers.

`statistic, tstatistics`: A matrix of shrunken t-statistics of the features.

`scores`: A matrix of discriminant scores.

`sd`: The pooled within-class standard deviations for each feature.

Author(s)

Kylie A. Bemis

References

Bemis, K., Harry, A., Eberlin, L. S., Ferreira, C., van de Ven, S. M., Mallick, P., Stolowitz, M., and Vitek, O. (2016.) Probabilistic segmentation of mass spectrometry images helps select important ions and characterize confidence in the resulting segments. Molecular & Cellular Proteomics. doi:10.1074/mcp.O115.053918

Tibshirani, R., Hastie, T., Narasimhan, B., & Chu, G. (2003). Class Prediction by Nearest Shrunken Centroids, with Applications to DNA Microarrays. Statistical Science, 18, 104-117.

Alexandrov, T., & Kobarg, J. H. (2011). Efficient spatial segmentation of large imaging mass spectrometry datasets with spatially aware clustering. Bioinformatics, 27(13), i230-i238. doi:10.1093/bioinformatics/btr246

See Also

`spatialKMeans`

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=2, dim=c(10,10), npeaks=10,
    peakheight=c(4,6,8), representation="centroid")

res <- spatialShrunkenCentroids(x, r=1, k=5, s=c(0,3,6), method="adaptive")

summary(res)

image(res, model=list(s=6))
```

subset-methods *Subsetting*

Description

Returns a subset of the dataset that meets the conditions.

Usage

```
## S4 method for signature 'SparseImagingExperiment'
subset(x, subset, select, ...)

subsetFeatures(x, ...)

subsetPixels(x, ...)
```

Arguments

x	An imaging dataset.
subset	Logical expression to be evaluated in the object's `featureData()` indicating which rows (features) to keep.
select	Logical expression to be evaluated in the object's `pixelData()` indicating which columns (pixels) to keep.
...	Conditions describing rows (features) or columns (pixels) to be retained. Passed to `features()` and `pixels()` methods to obtain the subset indices.

Value

An object of the same class as `x` with the appropriate subsetting applied to it.

Author(s)

Kylie A. Bemis

Examples

```
set.seed(1)
mse <- simulateImage(preset=1, npeaks=10, dim=c(10,10))

# subset features to mass range 1000 - 1500
subsetFeatures(mse, 1000 < mz, mz < 1500)

# select pixels to coordinates x = 1..3, y = 1..3
subsetPixels(mse, x <= 3, y <= 3)

# subset both features + pixels
subset(mse, 1000 < mz & mz < 1500, x <= 3 & y <= 3)
```

topFeatures-methods *Top-ranked features from imaging analysis results*

Description

Extract the top-ranked features from the results of imaging analysis, based on post-hoc correlation, test statistics, p-values, or adjusted p-values. The result is sorted data frame that can be further manipulated for downstream postprocessing.

Usage

```
#### Methods for Cardinal >= 2.x classes ####

## S4 method for signature 'SpatialShrunkenCentroids2'
topFeatures(object, ..., n = 10, model = modelData(object))

## S4 method for signature 'SpatialKMeans2'
topFeatures(object, ..., n = 10, model = modelData(object))

## S4 method for signature 'MeansTest'
topFeatures(object, ..., n = 10, p.adjust = "BH")

## S4 method for signature 'SegmentationTest'
topFeatures(object, ..., n = 10, model = modelData(object), p.adjust = "BH")

#### Methods for Cardinal 1.x classes ####

## S4 method for signature 'ResultSet'
topFeatures(object, n = 6,
    model = pData(modelData(object)),
    type = c('+', '-', 'b'),
    sort.by = fvarLabels(object),
    filter = list(),
    ...)

## S4 method for signature 'PCA'
topFeatures(object, n = 6,
    sort.by = "loadings",
    ...)

## S4 method for signature 'PLS'
topFeatures(object, n = 6,
    sort.by = c("coefficients", "loadings", "weights"),
    ...)

## S4 method for signature 'OPLS'
topFeatures(object, n = 6,
```

```
    sort.by = c("coefficients",
            "loadings", "Oloadings",
            "weights", "Oweights"),
    ...)

## S4 method for signature 'SpatialKMeans'
topFeatures(object, n = 6,
    sort.by = c("betweenss", "withinss"),
    ...)

## S4 method for signature 'SpatialShrunkenCentroids'
topFeatures(object, n = 6,
    sort.by = c("tstatistics", "p.values"),
    ...)

## S4 method for signature 'CrossValidated'
topFeatures(object, ...)
```

Arguments

`object`	The results of an imaging experiment analysis.
`n`	The number of top-ranked records to return.
`model`	If more than one model was fitted, results from which should be shown? Defaults to all models in the object This can name the models explicitly or specify a list of parameter values.
`p.adjust`	The `p.adjust` method used adjust p-values to account for multiple testing. Defaults to Benjamini and Hochberg ("BH") to control the false discovery rate (FDR).
`...`	For newer classes, additional arguments to be passed to `filter`, to further filter the results.
`type`	How should the records be ranked? '+' shows greatest values first (decreasing order), '-' shows least values first (increasing order), and 'b' uses decreasing order based on absolute values.
`sort.by`	What variable should be used for sorting?
`filter`	A list of named variables with values to use to filter the results. For example, for testing or classification, this can be used to only show rankings for a particular condition.

Value

A data frame with the top-ranked features.

Author(s)

Kylie A. Bemis

See Also

meansTest, segmentationTest, spatialShrunkenCentroids

Examples

```
setCardinalBPPARAM(SerialParam())

set.seed(1)
x <- simulateImage(preset=2, npeaks=10, dim=c(10,10),
    snoise=1, sdpeaks=1, representation="centroid")

y <- makeFactor(circle=pData(x)$circle, square=pData(x)$square)

res <- spatialShrunkenCentroids(x, y, r=1, s=c(0,3,6))

topFeatures(res, model=list(s=6))
```

---

writeMSIData *Write mass spectrometry imaging data files*

Description

Write supported mass spectrometry imaging data files. Supported formats include imzML and Analyze 7.5.

Usage

```
## S4 method for signature 'MSImageSet,character'
writeMSIData(object, file, outformat=c("imzML", "Analyze"), ...)

## S4 method for signature 'MSImageSet'
writeImzML(object, name, folder=getwd(), merge=FALSE,
mz.type="32-bit float", intensity.type="32-bit float", ...)

## S4 method for signature 'MSImageSet'
writeAnalyze(object, name, folder=getwd(),
intensity.type="16-bit integer", ...)
```

Arguments

object	An imaging dataset to be written to file.
file	A description of the data file to be write. This may be either an absolute or relative path. Any file extension will be ignored and replaced with an appropriate one.
name	The common file name for the '.imzML' and '.ibd' files for imzML or for the '.hdr', '.t2m', and '.img' files for Analyze 7.5.
folder	The path to the folder containing the data files.

outformat	The file format to write. Currently, the supported formats are "imzML" or "Analyze".
merge	Whether the samples/runs should be written to the same file (TRUE) or split into multiple files (FALSE). Currently, only FALSE is supported.
mz.type	The data type for the m/z values. Acceptable values are "32-bit float" and "64-bit float".
intensity.type	The data type for the intensity values. Acceptable values are "16-bit integer", "32-bit integer", "64-bit integer", "32-bit float" and "64-bit float".
...	Additional arguments passed to write functions.

Details

The writeImzML function supports writing both the 'continuous' and 'processed' formats.

Exporting the metadata is lossy, and not all metadata will be preserved. If exporting an object that was originally imported from an imzML file, any metadata that appears in metadata() of the object will be preserved when writing.

Different experimental runs are written to separate files.

The imzML files can be modified after writing (such as to add additional experimental metadata) using the Java-based imzMLValidator application: https://gitlab.com/imzML/imzMLValidator/.

Value

TRUE if the file was written successfully.

Author(s)

Kylie A. Bemis

References

Schramm T, Hester A, Klinkert I, Both J-P, Heeren RMA, Brunelle A, Laprevote O, Desbenoit N, Robbe M-F, Stoeckli M, Spengler B, Rompp A (2012) imzML - A common data format for the flexible exchange and processing of mass spectrometry imaging data. Journal of Proteomics 75 (16):5106-5110. doi:10.1016/j.jprot.2012.07.026

See Also

readMSIData

XDataFrame-class *XDataFrame: DataFrame with eXtra metadata columns*

Description

An XDataFrame is an (indirect) extension of the DataFrame class as defined in the 'S4Vectors' package, modified to support eXtra "slot-columns" that behave differently from other columns. It is intended to facilitate data.frame-like classes that require specialized column access and behavior. The specialized slot-columns are stored as distinct slots, unlike regular columns.

Usage

```
XDataFrame(...)
```

Arguments

...	Arguments passed to the DataFrame().

Details

For the most part, XDataFrame behaves identically to DataFrame, with the exception of certain methods being overwritten to account for the additional eXtra "slot-columns" not counted among those returned by ncol(x). These additional columns should typically have their own getter and setter methods.

To implement a subclass of XDataFrame, one needs to implement two methods to allow the slot-columns to be printed by show and retained during coercion: (1) the subclass should implement an as.list() method that includes the slot columns in the resulting list by default and (2) the subclass should implement a showNames() method that returns the names of all the printable columns (including slot-columns) in the same order as they are returned by as.list().

Methods

names(object): Return the column names, not including any slot-columns.

length(object): Return the number of columns, not including any slot-columns.

lapply(X, FUN, ..., slots = FALSE): Returns a list of the same length as X, where each element is the result of applying FUN to the corresponding element of X. This version includes an additional argument for whether the slot-columns should be included or not. This method should be overwritten by subclasses to ensure correct behavior.

as.env(x, ...): Create an environment from x with a symbol for each column, including the slot-columns. This method should be overwritten by subclasses to ensure correct behavior.

Author(s)

Kylie A. Bemis

See Also

DataFrame, MassDataFrame, PositionDataFrame

Examples

```
## Create an XDataFrame object
XDataFrame(x=1:10, y=letters[1:10])
```

Index