Package 'MRIaggr'

December 23, 2015

Type Package

Title Management, Display, and Processing of Medical Imaging Data

Version 1.1.5

Date 2015-12-22

Author Brice Ozenne

Maintainer Brice Ozenne <brice.ozenne@orange.fr>

Description Provide a compact storage for medical imaging data with access and display possibilities. Basic tools for processing brain imaging data are proposed like extraction of brain voxels, morphological image segmentation, and filtering / normalization of contrast parameters. Specific tools are also provided for blood perfusion imaging to calculate hypoperfusion and reperfusion volumes.

License GPL-3

Depends R (>= 2.10), Rcpp

Imports graphics, grDevices, Matrix, methods, oro.dicom, oro.nifti, RANN, ROCR, spam, stats, stats4, utils

Suggests coda, fields, mritc, nnet, parallel, pracma, rgl, snowfall

LazyLoad yes

LinkingTo Rcpp, RcppArmadillo, RcppProgress

Collate 'Generic_Functions.R' 'RcppExports.R' 'Functions_Valid.R' 'Functions_Miscellaneous.R' 'Functions_Options.R' 'Functions_GR.R' 'Functions_Potts.R' 'Functions_AssociatedClass.R' 'Class_MRIaggr.R' 'Class_Carto3D.R'

NeedsCompilation yes

Repository CRAN

Date/Publication 2015-12-23 23:44:19

R topics documented:

`MRIaggr-package` *Management, display and processing of cerebral imaging data.*

Description

MRIaggr provides a compact storage for medical imaging data with access and display possibilities. Basic tools for processing brain imaging data are proposed like extraction of brain voxels, morphological image segmentation, and filtering / normalization of contrast parameters. Specific tools are also provided for blood perfusion imaging to calculate hypoperfusion and reperfusion volumes.

Details

Package:	MRIaggr
Type:	Package
Version:	1.0
Date:	2014-06-25
License:	GPL (>=2.0)

Author(s)

brice ozenne

Maintainer: brice ozenne <brice.ozenne@orange.fr>

allocClinic *Allocate clinical data*

Description

Allocate clinical data to a MRIaggr object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocClinic(object, add = FALSE,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

object	an object of class MRIaggr. REQUIRED.
value	the clinical data. A one row *data.frame*. REQUIRED.
add	should value be added to the existing clinical slot ? *logical*.
overwrite	if clinical parameters with the same names are already stored in object@clinic, can they be overwritten ? *logical*.
verbose	should the execution of the function be traced ? *logical*.

Value

None.

See Also

selectClinic to extract the clinical data.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## allocate clinical data
allocClinic(MRIaggr.Pat1_red) <- data.frame(Age = 32, Sex = "Male", NIHSS_H0 = "5")
selectClinic(MRIaggr.Pat1_red, param = "Age")

## add a new parameter
allocClinic(MRIaggr.Pat1_red, add = TRUE, overwrite = TRUE) <- data.frame(City = "Lyon")
selectClinic(MRIaggr.Pat1_red)
```

`allocContrast` *Allocate new contrast parameters*

Description

Allocate one or several contrast parameters to a `MRIaggr` object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocContrast(object, param = NULL, default_value = NULL,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`value`	the value of each contrast parameter (in columns) at each voxel (in rows). *data.frame*. REQUIRED.
`param`	the names of the contrast parameters. *character vector* or `NULL` leading to use the name of the `value` argument.
`default_value`	the reference values of the contrast parameters (e.g background values). A one row *data.frame* where the column length must match the length of the `param` argument.
`overwrite`	if a contrast parameters with the same names are already stored in `object@data`, can they be overwritten ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.

Details

FUNCTION:
If the `param` argument is not specified then the names of the `value` argument will be used to define the parameter names.

If the `default_value` argument is `NULL` then default values `"undefined"` are attributed to each parameter.

If a parameter named `"mask"` is intended to be allocated it must be done alone and it must be of type logical.

Parameter names `"index"`, `"i"`, `"j"` and `"k"` are reserved and cannot be modified. Nevertheless if `value` contains parameters `"i"`, `"j"` and `"k"`, the correspondance between these coordinates and the object coordinates is tested.

Value

None.

See Also

calcContralateral, calcRegionalContrast, calcFilter and calcTissueType to compute, modify and allocate cartography.
selectContrast to select contrast parameters in the MRIaggr object.

Examples

```
## load NIFTI files and convert them to MRIaggr
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
ls.array <- list()
ls.array[[1]] <- readMRI(file.path(path.Pat1, "DWI_t0"),format = "nifti")
ls.array[[2]] <- readMRI(file.path(path.Pat1, "MASK_DWI_t0"),format = "nifti")
MRIaggr.Pat1 <- constMRIaggr(ls.array, identifier = "Pat1", param = c("DWI_t0", "MASK_DWI_t0"))

## allocate a new contrast parameters
allocContrast(MRIaggr.Pat1, param = "noise", overwrite = TRUE) <- rnorm(selectN(MRIaggr.Pat1))

## perform operations on a contrast parameters and store the results
myCarto <- selectContrast(MRIaggr.Pat1 , param = "DWI_t0")
myCarto <- myCarto * 2 + 1
allocContrast(MRIaggr.Pat1, param = "myCarto", overwrite = TRUE) <- myCarto

## import a contrast parameters in an already existing MRIaggr object
nifti.MTT_t0 <- readMRI(file.path(path.Pat1, "MTT_t0"), format = "nifti")
df.MTT_t0 <- array2df(nifti.MTT_t0, name_newparam = "MTT_t0")$MTT_t0
allocContrast(MRIaggr.Pat1, param = "MTT_t0", overwrite = TRUE) <- df.MTT_t0

## some calc methods automatically save results in the @data slot
calcFilter(MRIaggr.Pat1, param = "MTT_t0", filter = "2D_G3",
           update.object = TRUE, overwrite = TRUE)
res <- selectContrast(MRIaggr.Pat1, param = "MTT_t0_2D_G3")
```

---

allocDescStats *Allocate non standard elements*

Description

Allocate non standard elements to a MRIaggr object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocDescStats(object, name,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

object	an object of class MRIaggr. REQUIRED.
value	any R object. REQUIRED.
name	the name of the element storing value. *character*. REQUIRED.
overwrite	if an element with the same name is already stored in object@ls_descStats, can it be overwritten ? *logical*.
verbose	should the execution of the function be traced ? *logical*.

Details

FUNCTION:
Contrary to all other alloc. methods that impose restrictions on the objects that can be allocated, this function enable to allocate freely a R element. However these elements will have no interaction with the methods of this package.

Value

None.

See Also

selectDescStats to extract non standard elements.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## allocate a vector
allocDescStats(MRIaggr.Pat1_red,name = "spatial_res") <- c(1.875,1.875,6)

## select the corresponding element
selectDescStats(MRIaggr.Pat1_red, "spatial_res")

## some calc methods automatically save results in the ls_descStats slot
# find spatial groups
calcGroupsMask(MRIaggr.Pat1_red,mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"),
            W.range = 6, W.spatial_res = selectDescStats(MRIaggr.Pat1_red,"spatial_res"),
               update.object = TRUE, overwrite = TRUE)

# extract spatial groups
selectDescStats(MRIaggr.Pat1_red, "GroupsLesion")
```

allocHemisphere *Allocate the position of the mid-saggital plan*

Description

Allocate information about the position of the mid-saggital plan to a MRIaggr object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocHemisphere(object,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

object	an object of class MRIaggr. REQUIRED.
value	a *list* of *data.frame*. Names must be among "midplane", "hemispheres" "data". See the Details section. REQUIRED.
overwrite	if the characteristics of a mid-saggital plan are already stored in object, can they be overwritten ? *logical*.
verbose	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
The "midplane" element indicates the position of the observations of the mid-saggital plan. It has to be a two columns *data.frame* with the coordinates ("i","j") in columns and the observations in rows.

The "data" element must contains the position of each voxel with respect to the mid-sagittal plan (column "i_hemisphere" and "j_hemisphere") the hemisphere ("left", "right" or "undefined") to which the voxel belongs (column "hemisphere").

The "hemispheres" element indicates in which hemisphere is the lesion (denoted by "lesion"). The others hemispheries are denoted by "contralateral". It has to be a one line two columns *data.frame* with names "left" "right".

Value

None.

See Also

calcHemisphere to obtain the position of the mid-saggital plan and the position of the lesion in the hemispheres.
selectParameter to extract the previous elements.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## automatic allocation
resHemi <- calcHemisphere(MRIaggr.Pat1_red, param = "T2_GRE_t0", gridSearch = FALSE, num = 1,
                          update.object = TRUE, overwrite = TRUE)

## manual allocation
allocHemisphere(MRIaggr.Pat1_red, overwrite = TRUE) <- list(midplane = resHemi$midplane,
                                                            data = resHemi$data)

## display
index1 <- data.frame(selectMidplane(MRIaggr.Pat1_red), 15)
names(index1) <- c("i","j","k")
multiplot(MRIaggr.Pat1_red, param = "T2_GRE_t0", num = 1, midplane = TRUE, window = FALSE,
             index1 = list(coords = index1, pch = 20, cex = 2, col = "purple")

)

selectMidplane(MRIaggr.Pat1_red)
selectHemispheres(MRIaggr.Pat1_red)
```

`allocNormalization` *Allocate normalization values*

Description

Allocate normalization values to a `MRIaggr` object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocNormalization(object,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`value`	the normalisation values. A *list* of *data.frame*. REQUIRED.
`overwrite`	if normalization values are already stored in `object`, can they be overwritten ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
To enable other methods of the package to use the normalization values the `value` argument should match the result of the `calcNormalization` function. This function only partially control the validity of the `value` argument.
A valid `value` argument should be composed a list of *data.frame* with the following names :

- `"norm_global"` : it should have 6 rows named `"mu_both"` `"mu_left"` `"mu_right"` `"sigma_both"` `"sigma_left"` `"sigma_right"`.
- `"normMu_slice_both"` :
- `"normSigma_slice_both"`
- `"normMu_slice_left"`
- `"normSigma_slice_left"`
- `"normMu_slice_right"`
- `"normSigma_slice_right"`
- `"normMu_3slices_both"`
- `"normSigma_3slices_both"`
- `"normMu_3slices_left"`
- `"normSigma_3slices_left"`
- `"normMu_3slices_right"`
- `"normSigma_3slices_right"`.

All the elements of the list must have column names that match the contrast parameters present in the `object`.
Appart from the first element of the list, all elements should have as many rows as slices contained in the object.

Value

None.

See Also

`calcNormalization` to compute the normalisation values.
`selectNormalization` to extract the normalisation values.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## parameters to normalize
param <- c("DWI_t0","T2_FLAIR_t2","T2_GRE_t0","TTP_t0")

## manual allocation
resNormalization <- calcNormalization(MRIaggr.Pat1_red, param = param)
```

```
allocNormalization(MRIaggr.Pat1_red, overwrite = TRUE) <- resNormalization

## automatic allocation
resNormalization <- calcNormalization(MRIaggr.Pat1_red, param = param,
                                      update.object = TRUE, overwrite = TRUE)
```

---

allocOptionsMRIaggr *Allocate new default values*

Description

Allocate default value(s) to `optionsMRIaggr`. For internal use.

Usage

```
allocOptionsMRIaggr(field, n.args)
```

Arguments

`field`	a named list containing the value(s) to be allocated. *list*
`n.args`	the number of default value(s) to be allocated. *positive integer*

---

allocTable *Allocate volumic information*

Description

Allocate volumic information to a `MRIaggr` object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocTable(object, type,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`value`	the volumic information. *data.frame*. REQUIRED.
`type`	the type of volumic information. Can be `"lesion"` `"reperfusion"` `"hypoperfusion"`. REQUIRED.
`overwrite`	if tables are already stored in `object@normalization`, can they be overwritten ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS :
The validity of the value object is not checked. A valid format should match the result of the calcTableHypoReperf and calcTableLesion functions.

Value

None.

See Also

calcTableHypoReperf to compute the hypoperfusion and reperfusion tables.
calcTableLesion to compute the lesion table.
selectTable to extract the tables.
plotTableLesion to display the lesion volume by slice.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

#### 1- lesion ####
## manual allocation
maskN <- c("MASK_DWI_t0","MASK_T2_FLAIR_t2")
resTable <- calcTableLesion(MRIaggr.Pat1_red, maskN = maskN, numeric2logical = TRUE)
allocTable(MRIaggr.Pat1_red, type = "lesion", overwrite = TRUE) <- resTable

## automatic allocation
resTable <- calcTableLesion(MRIaggr.Pat1_red, maskN = maskN,
     numeric2logical = TRUE, update.object = TRUE, overwrite = TRUE)

## display
selectTable(MRIaggr.Pat1_red, type = "lesion")

#### 2- hypoperfusion and reperfusion ####
## manual allocation
resTable <- calcTableHypoReperf(MRIaggr.Pat1_red, param = c("TTP","MTT"), timepoint=c("t0","t1"))
allocTable(MRIaggr.Pat1_red, type = "hypoperfusion", overwrite = TRUE) <- resTable$volume_hypo
allocTable(MRIaggr.Pat1_red, type = "reperfusion", overwrite = TRUE) <- resTable$volume_reperf

## automatic allocation
resTable <- calcTableHypoReperf(MRIaggr.Pat1_red, param = c("TTP","MTT"),
      timepoint = c("t0","t1"), update.object = TRUE, overwrite = TRUE)

## display
selectTable(MRIaggr.Pat1_red, type = "reperfusion")
```

allocW *Allocate a neighbourhood matrix*

Description

Allocate a neighbourhood matrix to a MRIaggr object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
allocW(object, type,
         overwrite = FALSE, verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

object	an object of class MRIaggr. REQUIRED.
value	a list containing the neighbourhood matrix and associated elements. *list*. See the details section. REQUIRED.
type	the type of element to be allocated. Can be "Wmatrix", "Wblocks" or "upper". REQUIRED.
overwrite	if an element with the same name is already stored in object@ls_descStats, can it be overwritten ? *logical*.
verbose	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
Argument value must be a list containing named elements :

- Element "Wmatrix" must be matrix or a sparse matrix.
- Element "Wblocks" must be consistent with the output of the calcBlockW method.
- Element "verbose" must be logical or NULL method.

FUNCTION:
This method enable to store a neighbourhood matrix (type="Wmatrix"), a block decomposition of the matrix (type="Wblocks") and its format (type="upper") : entire matrix (NULL), upper-triagonal (TRUE) or lower-triagonal (FALSE).

Value

None.

See Also

calcW to compute the neighboring matrix.
calcBlockW to decompose the neighboring matrix in independant blocks.
selectW to extract the neighboring matrix or its related elements.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## search for a neighbourhood matrix
# selectW(MRIaggr.Pat1_red) # no neighbourhood matrix

## compute the neighbourhood matrix
resW <- calcW(object = MRIaggr.Pat1_red, range = sqrt(2))

## store it
allocW(MRIaggr.Pat1_red, type = "Wmatrix") <- list(Wmatrix = resW$W)

## it is automatically performed when specifying update.object = TRUE
calcW(object = MRIaggr.Pat1_red, range = sqrt(2), update.object = TRUE, overwrite = TRUE)
```

---

array2df *Array to data.frame converter*

Description

Convert observations stored in the array format into the data.frame format.

Usage

```
array2df(array, coords = NULL, name_newparam = "res",
         names_coords = letters[9:(8+ncol(coords))], na.rm = TRUE)
```

Arguments

array	the array that should be converted into a data.frame. *array* or *matrix*. REQUIRED.
coords	the spatial coordinates of the observations contained in array. *matrix* or NULL.
name_newparam	the name of the contrast parameter to which cooresponds array. *character*.
names_coords	the name of the coordinates. *character vector*.
na.rm	should observations with missing values be removed ? *logical*.

Details

ARGUMENTS :
If coords is set to NULL, the coordinates will be defined by the position of the observations in array.
If na.rm is set to TRUE, the coord argument will only contains the coordinates of the non-NA observations of array.

Value

A dataframe with in columns the coordinates and the parameter values, and in rows the observations.

Examples

```
## load a NIFTI file (array format)
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
dim(nifti.Pat1_TTP_t0)

## conversion to data frame format
res128 <- array2df(array = nifti.Pat1_TTP_t0, name_newparam = "TTP_t0")
dim(res128)
head(res128)

## conversion to data frame format with specific coordinates
res256 <- array2df(array = nifti.Pat1_TTP_t0, name_newparam = "TTP_t0",
                   coords = expand.grid(129:206, 129:228, 1:3, 1))
dim(res256)
head(res256)
```

---

boxplotMask *Boxplot spatial group characteristics*

Description

Display a boxplot of the contrast parameter values inside and outside a spatial group.

Usage

```
## S4 method for signature 'MRIaggr'
boxplotMask(object, param, mask, num = NULL, rescale = TRUE,
        ylim = NULL, col = c("white", "purple"), main = NULL, mgp = c(2, 0.5, 0),
 x.legend = "topright", y.legend = NULL, cex.legend = 0.8,
         filename = paste(object@identifier, "boxplotMask", sep = "_"), ...)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameter(s) associated with the lesion mask(s). *character vector*. REQUIRED.
mask	the binary contrast parameter(s) defining the spatial group(s). *character vector*. REQUIRED.
num	the slices to consider. *numeric vector* or NULL.

rescale	should the contrast parameters be scaled ? *logical*.
ylim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the y limits.
col	the colors of the boxplots for observations inside and outside the mask(s). *character vector of size 2*.
main	an overall title for the plot. *character*.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
x.legend	the x coordinates of the legend. *numeric* or *character*.
y.legend	the y coordinates of the legend. *numeric* or *character*.
cex.legend	the expansion factor of the legend. *positive numeric*.
filename	the name of the file used to export the plot. *character*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the mgp argument can be found in par.

Information about the x.legend, y.legend, cex.legend arguments can be found in legend (cex.legend is the cex argument of legend).

Arguments ... must correspond to some of the following arguments : height, hemisphere, norm_mu, norm_sigma, numeric2logical, path, res, unit, width, window.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## display
boxplotMask(MRIaggr.Pat1_red, param = c("DWI_t0","TTP_t0","MTT_t0"),mask = "MASK_T2_FLAIR_t2",
     numeric2logical = TRUE)
```

`calcAUPRC` *Area under the PR curve*

Description

Compute the area under the precision recall curve by numerical integration.

Usage

```
calcAUPRC(x, y, subdivisions = 10000, performance = NULL, ci = TRUE, alpha = 0.05,
          method = "Kronrod", reltol = .Machine$double.eps^0.25)
```

Arguments

`x`	the biomarker values. *numeric vector*. REQUIRED.
`y`	the class labels. *numeric vector*, *character vector* or *logical vector*. REQUIRED.
`subdivisions`	the maximum number of subintervals used for the integration. *positive integer*. Only used if `method="integrate"`.
`performance`	an object of class `performance` can be supplied instead of arguments `x` and `y`.
`ci`	should the confidence interval be computed ? *logical*.
`alpha`	the type 1 error rate. *numeric*.
`method`	the integration method used to compute the area under the curve. Any of `"integrate"`, `"Kronrod"`, `"Richardson"` `"Clenshaw"`, `"Simpson"` or `"Romberg"`.
`reltol`	the relative accuracy requested. Positive *numeric*.

Details

This function requires to have installed the *ROCR* package to work.
The numeric integration of the precision over the recall values can be performed either using the *integrate* function of the stats package (if `method="integrate"`) or using the *integral* function of the pracma package. In the latter case, the method argument is used to define the integration procedure (see the documentation of *integral* for more details).
The confidence interval is computed using the first order delta method and the logistic transformation :

$$IC(AUPRC) = \left[\frac{e^{\mu_\eta - 1.96\tau}}{1 + e^{\mu_\eta - 1.96\tau}} \; ; \; \frac{e^{\mu_\eta + 1.96\tau}}{1 + e^{\mu_\eta + 1.96\tau}} \right]$$

$$\mu_\eta = logit(\widehat{AUPRC})$$

$$\tau = \frac{1}{\sqrt{n * \widehat{AUPRC} * (1 - \widehat{AUPRC})}}$$

See section 3.2 of (Boyd, 2013) for more details.

ARGUMENTS:
`y` must have exactly two levels.

If `performance` is set to `NULL`, the codex and `y` will be used to form the `performance` object.

Value

If `ci=FALSE` a *numeric* between 0 and 1.
If `ci=TRUE` a *numeric vector* of length 3 containing the punctual estimation, the lower and the upper bound of the confidence interval.

References

Kendrick Boyd1, Kevin H. Eng, and C. David Page. *Area Under the Precision-Recall Curve: Point Estimates and Confidence Intervals*. Machine Learning and Knowledge Discovery in Databases, 2013:451-466.

Examples

```
data(MRIaggr.Pat1_red, package = "MRIaggr")

## select parameter and binary outcome
cartoT2 <- selectContrast(MRIaggr.Pat1_red, param = "T2_FLAIR_t2", format = "vector")
cartoMASK <- selectContrast(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2", format = "vector")

## compute AUPRC
T2.AUPRC <- calcAUPRC(x = cartoT2, y = cartoMASK)

## compute AUC
## Not run:
if(require(pROC)){
T2.AUC <- auc(roc(cartoMASK ~ cartoT2))
}


## display
multiplot(MRIaggr.Pat1_red,param = "T2_FLAIR_t2", num = 1,
          index1 = list(coords = "MASK_T2_FLAIR_t2", outline = TRUE)
)

## End(Not run)

#### 2- with simulated data ####
n0 <- 1000
n1 <- c(10,100,1000)
for(iter_n in 1:length(n1)){
  X <- c(rnorm(n0,0),rnorm(n1[iter_n],2))
  Y <- c(rep(0,n0),rep(1,n1[iter_n]))
  print(calcAUPRC(X,Y))
}

## alternative way using a performance object
perfXY <- ROCR::performance(ROCR::prediction(X,Y), x.measure = "rec", measure = "prec")
calcAUPRC(performance = perfXY, subdivisions = 10000)
```

`calcBlockW` *Find disjoint spatial blocks of sites*

Description

Partition the space into disjoint spatial blocks of sites. Call the C++ function `calcOrderSite_hpp`. For internal use.

Usage

```
calcBlockW(W, site_order = NULL, dist.center = NULL, dist.max = Inf,
         verbose = optionsMRIaggr("verbose"))
```

Arguments

`W`	the neighbourhood matrix. *dgCMatrix*. REQUIRED.
`site_order`	a specific order to go all over the sites. *integer vector*.
`dist.center`	the distance between each point and a reference point. *numeric vector*.
`dist.max`	the neighbourhood range. *numeric vector*.
`verbose`	Should the process be verbose over iterations ? *logical*.

Details

This function requires to have installed the *Matrix* and the *spam* package to work. If no specific order is set, sites are visitating from the first to the last, according to the neighbourhood matrix.

Value

An *list* containing :

- `[[ls_groups]]` : a *list* containing the index of the sites for each independant group.
- `[[size_groups]]` : a *vector* containing the size of each independant group.
- `[[n_groups]]` : an *integer* giving the number of independant groups.

Examples

```
#### spatial field
## Not run:
n <- 100

## End(Not run)

coords <- data.frame(which(matrix(0, nrow = n, ncol = n) == 0,arr.ind = TRUE), 1)
optionsMRIaggr(quantiles.legend = FALSE, axes = FALSE, num.main = FALSE, bg = "white")

#### 1- neighbourhood matrix (king) ####
W_king <- calcW(coords, range = 1.001, row.norm = TRUE)$W
```

```
#### find independant groups
Block_king <- calcBlockW(W_king)

## check groups
# diagonal : percent of neighborhing sites whithin group
# extra-diagonal : percent of neighborhing sites between groups
sapply(1:Block_king$n_groups, function(x){
  sapply(1:Block_king$n_groups, function(y){
   sum(spam::rowSums(W_king[Block_king$ls_groups[[x]], Block_king$ls_groups[[y]]] > 0) > 0)
  }) / length(Block_king$ls_groups[[x]])
}
)

## diplay sparse matrix
spam::image(W_king)
spam::image(W_king[unlist(Block_king$ls_groups), unlist(Block_king$ls_groups)])

## display site blocks
col_sites <- unlist(lapply(1:Block_king$n_groups, function(x){
rep(rainbow(Block_king$n_groups)[x], Block_king$size_groups[x])
}))

multiplot(coords[unlist(Block_king$ls_groups),],
          xlim = c(0,30),ylim = c(0,30),
          col = col_sites, legend = FALSE)


#### 2- neighbourhood matrix (Queen) ####
W_queen <- calcW(coords, range = sqrt(2) + 0.001, row.norm = TRUE)$W

#### find independant groups
Block_queen <- calcBlockW(W_queen)

## check groups
# diagonal : percent of neighborhing sites whithin group
# extra-diagonal : percent of neighborhing sites between groups
sapply(1:Block_queen$n_groups, function(x){
  sapply(1:Block_queen$n_groups, function(y){
   sum(spam::rowSums(W_queen[Block_queen$ls_groups[[x]], Block_queen$ls_groups[[y]]] > 0) > 0)
  }) / length(Block_queen$ls_groups[[x]])
}
)

## diplay sparse matrix
spam::image(W_queen)
spam::image(W_queen[unlist(Block_queen$ls_groups), unlist(Block_queen$ls_groups)])

## display site blocks
col_sites <- unlist(lapply(1:Block_queen$n_groups, function(x){
rep(rainbow(Block_queen$n_groups)[x], Block_queen$size_groups[x])
}))
```

```
multiplot(coords[unlist(Block_queen$ls_groups),],
          xlim = c(0,30), ylim = c(0,30),
          col = col_sites, legend = FALSE)

#### 3- neighbourhood matrix (Regional) ####
W_Regional <- calcW(coords, range = 3, row.norm = TRUE)$W

#### find independant groups
system.time(
  Block_Regional <- calcBlockW(W_Regional)
)

system.time(
Block_Regional_test1 <- calcBlockW(W_Regional,
     dist.center = sqrt(spam::rowSums(sweep(coords, MARGIN = 2,
                                  STATS = apply(coords, 2, median), FUN = "-")^2))
     )
)
system.time(
  Block_Regional_test2 <- calcBlockW(W_Regional,
     dist.center = sqrt(spam::rowSums(sweep(coords, MARGIN = 2,
                                   STATS = apply(coords, 2, median), FUN = "-")^2)),
     dist.max = 3
  )
)
# all(unlist(Block_Regional_test1$ls_groups) == unlist(Block_Regional_test2$ls_groups))


## check groups
# diagonal : percent of neighborhing sites whithin group
# extra-diagonal : percent of neighborhing sites between groups
sapply(1:Block_Regional$n_groups,function(x){
   sapply(1:Block_Regional$n_groups,function(y){
    if(length(Block_Regional$ls_groups[[x]]) > 1){
      sum(spam::rowSums(as.matrix(W_Regional[Block_Regional$ls_groups[[x]],
                                   Block_Regional$ls_groups[[y]]]) > 0) > 0)
    }else{
      sum(W_Regional[Block_Regional$ls_groups[[x]],
                 Block_Regional$ls_groups[[y]]] > 0) > 0
    }
  }) / length(Block_Regional$ls_groups[[x]])
}
)
# clustering could be improved

## diplay sparse matrix
spam::image(W_Regional)
spam::image(W_Regional[unlist(Block_Regional$ls_groups), unlist(Block_Regional$ls_groups)])

## display site blocks
col_sites <- unlist(lapply(1:Block_Regional$n_groups, function(x){
rep(rainbow(Block_Regional$n_groups)[x], Block_Regional$size_groups[x])
}))
```

```
multiplot(coords[unlist(Block_Regional$ls_groups),],
          xlim = c(0,30), ylim = c(0,30),
          col = col_sites, legend = FALSE)
```

`calcBlockW_cpp` *Find disjoint spatial blocks of sites*

Description

C++ function called by `calcBlockW` to identify independant spatial blocks of sites according to a neighbourhood matrix.

Usage

```
calcBlockW_cpp(W_i, W_p, site_order, dist_center, dist_max, verbose)
```

Arguments

`W_i`	the 0-based row numbers for each non-zero element in the sparse neighbourhood matrix. *integer vector*.
`W_p`	the pointers to the initial (zero-based) index of elements in the colum of the sparse neighbourhood matrix. *integer vector*.
`site_order`	a specific order to go all over the sites. *integer vector*.
`dist_center`	the distance between each point and a reference point. *numeric vector*.
`dist_max`	the neighbourhood range. *numeric vector*.
`verbose`	Should the process be verbose over iterations ? *logical*.

`calcBrainMask` *Brain-Background discrimination*

Description

Seek to form two groups of observations (brain observations and background observations) using a threshold approach or a k-means algorithm.

Usage

```
## S4 method for signature 'MRIaggr'
calcBrainMask(object, param, type = "kmeans", th.breaks = 100,
        th.smoothing = TRUE, th.select_optima = 1, th.upper = TRUE, plot = TRUE,
         kmeans.n_groups = 2:4, kmeans.Neighborhood = 3, skull.param = NULL,
 skull.n_groups = 3,
 filename = paste("calcBrainMask", type, object@identifier, sep = "_"),
         update.object = FALSE, overwrite = FALSE, ...)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the contrast parameter(s) that should be used to identify the brain observations. *character vector*. REQUIRED.
`type`	the method to use. Can be `"threshold"` or `"kmeans"`.
`th.breaks`	the number of thresholds to use. *postive integer*.
`th.smoothing`	should the derivative be smoothed ? *logical*.
`th.select_optima`	the rank of the optimum to retain. *postive integer*.
`th.upper`	should the observations above the selected threshold be retained ? Else the observations bellow will the selected threshold be retained. *logical*.
`plot`	should the results be plotted ? *logical*.
`kmeans.n_groups`	the number of groups to use in the kmeans algorithm. *postive integer vector*.
`kmeans.Neighborhood`	the range of the neighbourhood. *postive integer*.
`skull.param`	the parameter used to identify the skull. *character*.
`skull.n_groups`	the number of groups to use in the kmeans algorithm to obtain the skull.
`filename`	the name of the file used to export the plot. *character*.
`update.object`	should the resulting mask be stored in `object` as a `"mask"` parameter ? *logical*.
`overwrite`	if a mask is already stored in `object@data`, can it be overwritten ? *logical*.
`...`	additional arguments to be passed to `optionsMRIaggr` for specifying the graphical parameters.

Details

ARGUMENTS:

Argument(s) `...` must correspond to some of the following arguments : `height`, `numeric2logical`, `path`, `res`, `unit`, `verbose`, `width`, `window`. Setting `skull.param` to `NULL` leads to skip the skull stripping step.

FUNCTION:
The threshold approach searches the best break point of the function that maps thresholds to the number of observations. For this, it find the optima of the first derivative of this function (possibly smoothed).
The arguments `th.breaks`, `th.smoothing`, `th.select_optima` and `th.upper` are only active if `type` is `"threshold"`.
`th.smoothing` can be set to an *integer* to specify the width of the smoothing kernel.

The k-means approach seeks the most spatially coherent partition of the observations, among the possible partitions defined by the `kmeans.n_groups` argument. The mean number of neighbors averaged over observations (spatial potential) is used as a metric of the spatial coherence of the partition.

The arguments kmeans.n_groups, kmeans.Neighborhood and upper are only active if type is "kmeans".

The skull step consists in identifying the skull with an additionnal parameter (T1 sequence appears well suited for this purpose), and remove the corresponding observations from the brain mask. It is the partition that gives the best spatial coherence for the final brain mask that is retained, leaving the posibility of no skull stripping.

The plot is active only if type is "threshold".

Value

If type is "threshold", a *list* containing :

- [[analysis]] : A *matrix* containing the parameter thresholds (column "threshold"), the number of observations inside the mask (column "Nb"), the first and its smoothed version (column "dNb" and "dNb.filtered") and indicator of optima (column "optima"). *matrix*.
- [[th_opt]] : A *matrix* containing the number of observations inside the mask and its derivative (in lines) for each optimum (in columns).
- [[best_group]] : An indicator variable giving the observations that belong to the mask. *logical vector*.
- [[mask_name]] : the mask name. *character*.

If type is "kmeans", a *list* containing :

- [[kmeans]] : the optimal kmeans partition (result of the kmeans function). *list*.
- [[potential]] : the spatial potential for the various brain partitions. *matrix*.
- [[best_V]] : the highest potential. *numeric*.
- [[best_group]] : An indicator variable giving the observations that belong to the mask. *logical vector*.
- [[mask_name]] : the mask name. *character*.
- [[potential_skull]] : the spatial potential for the various skull partitions (only if skull.param is not NULL). *matrix*.

See Also

selectContrast to select the mask parameter.
calcSmoothMask to spatially regularized the obtained mask.

Examples

```
## load NIFTI files and convert them to MRIaggr
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
ls.array <- list(readMRI(file.path(path.Pat1, "T1_t0"), format = "nifti"),
                 readMRI(file.path(path.Pat1, "T2_GRE_t0"), format = "nifti"))
MRIaggr.Pat1 <- constMRIaggr(ls.array, identifier = "Pat1", param = c("T1_t0","T2_GRE_t0"))

#### 1- thresholding approach ####
res <- calcBrainMask(MRIaggr.Pat1, param = "T2_GRE_t0", type = "threshold",
                     th.select_optima = 2)
```

```
breaks <- res$analysis[,"threshold"]
res <- calcBrainMask(MRIaggr.Pat1, param = "T2_GRE_t0", type = "threshold",
                     th.breaks = breaks[breaks > 50], th.select_optima = 1,
                     overwrite = TRUE, update.object = TRUE)

## display
multiplot(MRIaggr.Pat1, param = "mask")

multiplot(MRIaggr.Pat1, param = "T2_GRE_t0", index1 = "mask")

## other parameter
## Not run:
res <- calcBrainMask(MRIaggr.Pat1, param = "T1_t0", type = "threshold",
                     th.breaks = 200)

res <- calcBrainMask(MRIaggr.Pat1, param = "T1_t0", type = "threshold",
                     th.breaks = seq(0, 400, length.out = 50), th.select_optima = 2,
                     overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1, param = "mask")

## End(Not run)

#### 2- k-means approach ####
## Not run:
res <- calcBrainMask(MRIaggr.Pat1, param = "T2_GRE_t0", type = "kmeans",
                     kmeans.n_groups = 2:4,
                     update.object = TRUE, overwrite = TRUE)

## End(Not run)

## display
multiplot(MRIaggr.Pat1, param = "T2_GRE_t0", index1 = "mask")
multiplot(MRIaggr.Pat1, param = "mask")
```

`calcContralateral` *Compute contraleral normalization values*

Description

Associate each voxel to an hemisphere and compute the difference between the voxel values and their contralateral correspondent.

Usage

```
## S4 method for signature 'MRIaggr'
calcContralateral(object, param, num = NULL, type = "mean",
     param.ref = NULL, distband = 1, lambda = 1,
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the contrast parameters to normalize. *character vector*. REQUIRED.
`num`	the slices to extract. *numeric vector* or `NULL`.
`type`	the method used to compute the contralateral correspondent of each voxel. Can be `"mean"`, `"median"` or `"1NN_penalised"`.
`param.ref`	the parameter to use as a reference for the identification of the contralateral voxel. *character* or `NULL` if no reference parameter available.
`distband`	the distance within which the contralateral values are considered. *postive numeric*.
`lambda`	the importance of the penalization. *numeric*.
`verbose`	should the execution of the function be traced ? *logical*.
`update.object`	should the resulting contralateral parameters be stored in `object` ? *logical*.
`overwrite`	if contrast parameters with the same names are already stored in `object@data`, can they be overwritten ? *logical*.

Details

ARGUMENTS:
Information about the `num` argument can be found in the details section of `initNum`.

FUNCTION:
To compute the contralateral correspondent of each voxel, the mean or median value of the contralateral observations present in the distband can be used. Otherwise, considering a reference parameter, the contralateral voxel that minimised the difference in contrast (denoted I) with the voxel of interest (denoted x) penalized by the distance is retained :

$$voxel_{contro} = argmin_y \frac{|I(x) - I(y)|}{\sigma} + \lambda * dist(x, y) where \sigma = sd(I)$$

The `param.ref` and `lambda` are active only if `type` is `"1NN_penalized"`. In this case `lambda` equal `0` means no penalization.

Value

An *list* containing :

- `[[data]]` : a *data.frame* containing the coordinates and the parameters normalized by the contralateral values.
- `[[index_plot]]` : two lists containing the observations used to compute the contralateral values, one for each hemisphere.

See Also

`calcHemisphere` to identify the hemispheres. `selectContrast` to select the contralateral normalized parameters or the hemisphere parameter.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## associate each voxel to its contralateral correspondant
## according T1 parameter and compute the normalized parameters
res <- calcContralateral(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"), num = NULL,
         type = "mean", param.ref = "T1_t0", distband = 1, lambda = 1, verbose = TRUE)

## display
multiplot(res$data[,c("i","j","k"),drop = FALSE],
             contrast = res$data$DWI_t0_contro
)

multiplot(res$data[,c("i","j","k"), drop = FALSE],
             contrast = res$data$DWI_t0_contro,
           index1 = res$data[res$index_plot$index.plot_lesionR,c("i","j","k"), drop = FALSE],
           index2 = res$data[res$index_plot$index.plot_lesionL,c("i","j","k"), drop = FALSE]
)
```

`calcContro_cpp` *Compute contralateral normalization*

Description

C++ function called by `calcContralateral` to seek the contralateral voxel and the contralateral values. For internal use.

Usage

```
calcContro_cpp(contrast, coords_px, index_k, index_k_contro, d_lim,
         lambda, param_ref, var_ref, type_moy, type_med, type_NN, verbose)
```

Arguments

`contrast`	the contrast value of each voxel on a given slice. *matrix*.
`coords_px`	the coordinates of the observations. *matrix*.
`index_k`	the index of the observations on the hemisphere of interest. *integer vector*.
`index_k_contro`	the index of the observations on the contralateral hemisphere. *integer vector*.
`d_lim`	the distance within which the contralateral values are considered. *numeric*.
`lambda`	the importance of the penalization by the distance. *numeric*.
`param_ref`	the parameter to be used as a reference to identify the contralateral voxel. *character*.
`var_ref`	the variance of the reference parameter. *numeric*.
`type_moy`	should the mean contralateral value be used ? *logical*.

type_med	should the median contralateral value be used ? *logical*.
type_NN	should the closest contralateral voxel according to the reference parameter be used ? *logical*.
verbose	should the execution of the function be traced ? *logical*.

---

calcCriteriaGR *Assessment of clustering quality*

Description

Compute several quality indexes of a two group clustering. For internal use.

Usage

```
calcCriteriaGR(contrast, groups, W = NULL, sigma = NULL, breaks = NULL,
       rm.warning = TRUE, criterion.transition = FALSE, criterion.sdfront = FALSE,
       criterion.entropy = TRUE, criterion.Kalinsky = TRUE, criterion.Laboure = TRUE)
```

Arguments

contrast	the contrast value of each observations. *numeric vector*. REQUIRED.
groups	the indicator of group membership. *logical vector*. REQUIRED.
W	the neighbourhood matrix. *dgCMatrix* or NULL leading to not compute the d1 criterion.
sigma	the sigma_max that have been used in the GR algorithm. *positive numeric vector*.
breaks	the break points to use to categorize the contrast distribution. *numeric vector*.
rm.warning	should warning be displayed. *logical*.
criterion.transition	should the boundary criterion based on the transition levels be computed ? *logical*.
criterion.sdfront	should the boundary criterion based on the standard deviation be computed ? *logical*.
criterion.entropy	should the region criterion based on the entropy be computed ? *logical*.
criterion.Kalinsky	should the region criterion based on the Kalinsky index be computed ? *logical*.
criterion.Laboure	should the region criterion based on the Laboure index be computed ? *logical*.

References

Chantal Revol-Muller, Francoise Peyrin, Yannick Carrillon and Christophe Odet. *Automated 3D region growing algorithm based on an assessment function*. Pattern Recognition Letters, 23:137-150,2002.

`calcDistMask` *Euclidean distance to a spatial group*

Description

Compute the euclidean distance to a spatial group.

Usage

```
## S4 method for signature 'MRIaggr'
calcDistMask(object, mask, name_newparam = paste("dist", mask, sep = "_"),
     spatial_res = c(1,1,1), numeric2logical = FALSE, Neighborhood = "3D_N10",
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`mask`	the binary contrast parameter(s) defining the spatial groups from which the distance will be computed. *character vector*. REQUIRED.
`name_newparam`	the name of the new distance parameters. *character vector*.
`spatial_res`	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
`numeric2logical`	should `mask` be convert to logical ? *logical*.
`Neighborhood`	the type of neighbourhood. *character*.
`verbose`	should the execution of the function be traced ? *logical*.
`update.object`	should the resulting distance parameters be stored in `object` ? *logical*.
`overwrite`	if contrast parameters with the same names are already stored in `object@data`, can they be overwritten ? *logical*.

Details

This function requires to have installed the *RANN* package to work.

ARGUMENTS:
Information about the `num` argument can be found in the details section of `initNum`.

The `Neighborhood` argument can be a *matrix* or an *array* defining directly the neighbourhood to use (i.e the weight of each neighbor) or a name indicating which type of neighbourhood should be used (see the details section of `initNeighborhood`).

FUNCTION:
This function relies on the `nn2` function of the *RANN* package.

Value

An *data.frame* containing in row the observations and in columns the distance parameters.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute distance to initial and final mask
res <- calcDistMask(MRIaggr.Pat1_red, mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"),
            update.object = TRUE, overwrite = TRUE)

multiplot(MRIaggr.Pat1_red, param = "dist_MASK_DWI_t0",
          index1 = list(coords = "MASK_DWI_t0", outline = TRUE))

## compute distance to initial and final mask correcting anisotropy
res <- calcDistMask(MRIaggr.Pat1_red, mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"),
                    spatial_res = c(1.875,1.875,6),
                    update.object = TRUE, overwrite = TRUE)

multiplot(MRIaggr.Pat1_red, param = "dist_MASK_DWI_t0",
          index1 = list(coords = "MASK_DWI_t0", outline = TRUE))
```

---

calcDistTissues *Compute descriptive statistics*

Description

Compute the four first order statistics of the contrast parameters by cerebral structure.

Usage

```
## S4 method for signature 'MRIaggr'
calcDistTissues(object, param, param.membership, num = NULL, hemisphere = "both")
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the contrast parameters to consider. *character vector*. REQUIRED.
`param.membership`	the parameters indicating the probabilistic membership of each observations to each cerebral structure. *character vector*.
`num`	the slices to use. *numeric vector* or `NULL`.
`hemisphere`	the hemisphere to use. *character*.

Details

ARGUMENTS :
Information about the `num` argument can be found in the details section of `initNum`.

Information about the `hemisphere` argument can be found in the details section of `selectContrast`.

Value

An *data.frame* containing in row the various cerebral structures and in columns the various moments for each contrast parameter.

See Also

calcTissueType to compute a probabilistic classification of the brain observations in WM/GM/CSF.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute the distribution of DWI and T2 FLAIR for the CSF, WM, GM and lesion observations
res <- calcDistTissues(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"),
                         param.membership = c("CSF","WM","GM","MASK_DWI_t0")
)
```

---

calcFilter *Image filtration*

Description

Apply a filter to an image.

Usage

```
## S4 method for signature 'array'
calcFilter(object, filter, norm.filter = TRUE, bilateral = FALSE, na.rm = FALSE)

## S4 method for signature 'MRIaggr'
calcFilter(object, param, filter, norm.filter = TRUE, bilateral = FALSE,
     na.rm = FALSE, name_newparam = NULL,
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

object	an array or an object of class MRIaggr. REQUIRED.
param	the contrast parameter to be filtered. *character vector*. REQUIRED.
filter	the filter to use. Can be a *matrix* or an *array*, or a name indicating which filter should be used. REQUIRED.
norm.filter	should the filtered correspond to a weighted mean over site ? (or a weighted sum). *logical*.
bilateral	should the influence of each neighbor be ponderated by the difference in signal with the considered observation ? *logical*.

na.rm	should observations with missing values in their neighbourhood be set to NA ? Otherwise the ponderation is adjusted. *logical*.
name_newparam	the name of the new parameters. *character vector*.
verbose	should the execution of the function be traced ? *logical*.
update.object	should the resulting filtered parameters be stored in object ? *logical*.
overwrite	if contrast parameters with the same names are already stored in object@data, can they be overwritten ? *logical*.

Details

ARGUMENTS:
Several types of pre-stored filters are availables and can be called by their name:

- filter for smoothing purpose (e.g. "3D_G5") : see the details section of initFilter.
- filter for neighbourhood definition purpose (e.g. "3D_N10") : see the details section of initNeighborhood.

norm.filter should be set to TRUE for gaussian, median or sobel filters to correct edge effects. This lead to weight the neighboring value in order to offset the incomplete neighbourhood.

Value

An *list* containing :

- [[res]] : a *data.frame* containing the coordinates and the filtered parameters.
- [[filter]] : the name of the filter that has been used. *character*.

See Also

selectContrast to select the filtered parameter(s). initFilter or codeinitNeighborhood to select pre-stored filters.

Examples

```
#### 1- array method ####
## load a NIFTI file
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
nifti.Pat1_DWI_t0 <- readMRI(file.path(path.Pat1, "DWI_t0"), format = "nifti")

## before filtering
graphics::image(nifti.Pat1_DWI_t0[,,1,1])

## after median filtering
niftiF.Pat1_DWI_t0 <- calcFilter(nifti.Pat1_DWI_t0[,,,1], filter = "2D_M3")$res
graphics::image(niftiF.Pat1_DWI_t0[,,1])

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")
```

```
## compute and allocate filtered parameter to the MRIaggr object
# gaussian filter
calcFilter(MRIaggr.Pat1_red, param = c("T2_FLAIR_t2","DWI_t0","TTP_t0"),
           filter = "2D_G3", bilateral = FALSE, na.rm = FALSE,
   update.object = TRUE, overwrite = TRUE)
selectParameter(MRIaggr.Pat1_red)

# median filter
calcFilter(MRIaggr.Pat1_red, param = c("T2_FLAIR_t2","DWI_t0","TTP_t0"),
           filter = "2D_M3", na.rm = FALSE, update.object = TRUE, overwrite = TRUE)

## display
par(mfrow = c(2,2), mar = c(2,2,2,2))
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
          num.main = FALSE, main = "raw",
          num = 1, window = NULL, breaks = c(-100,seq(0, 450),601))
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2_2D_G3",
          num.main = FALSE, main = "2D_G3",
          num = 1, legend = FALSE, window = NULL, breaks = c(-100,seq(0,450),601))
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2_2D_M3",
          num.main = FALSE, main = "2D_M3",
          num = 1, legend = FALSE, window = NULL, breaks = c(-100,seq(0, 450),601))

## see the results of the different filters
# G : Gaussian filter
resG <- calcFilter(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
                   filter = "2D_G3")

# M : median filter
resM <- calcFilter(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
                   filter = "2D_M3")

# S : Sobel filter
resS <- calcFilter(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
                  filter = "2D_Sx")

# I
resI.T <- calcFilter(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2",
                  filter = "2D_I3", norm.filter = TRUE)
resI.F <- calcFilter(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2",
                   filter = "2D_I3", norm.filter = FALSE)

# N
resN.T <- calcFilter(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2",
                     filter = "3D_N10", norm.filter = TRUE)
resN.F <- calcFilter(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2",
                     filter = "3D_N10", norm.filter = FALSE)

## display
par(mfrow = c(2,2), mar = rep(2,4), mgp = c(2,0.75,0))
breaks <- seq(-50,500,1)
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2", num = 3,
             breaks = breaks, window = NULL, legend = FALSE,
```

```
            main = "no filtering", num.main = FALSE, main.legend = "")
multiplot(resS$res[,c("i","j","k")], contrast = resS$res[,4],
            num = 1, window = NULL, legend = FALSE,
            palette = "cm.colors", breaks = seq(-100, 100),
            main = "sobelX filtering", num.main = FALSE)
multiplot(resG$res[,c("i","j","k")], contrast = resG$res[,4],
            num = 3, window = NULL, legend = FALSE, breaks = breaks,
            main = "gaussian filtering", num.main = FALSE)
multiplot(resM$res[,c("i","j","k")], contrast = resM$res[,4],
            num = 3, window = NULL, legend = FALSE, breaks = breaks,
            main = "median filtering", num.main = FALSE)

layout(matrix(1:6, nrow = 3, ncol = 2, byrow = TRUE), widths = c(2,1), heights = rep(1,3))
par(mar = rep(2,4), mgp = c(2,0.75,0))
multiplot(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2", num = 3,
          window = NULL, legend = TRUE, main = "raw", num.main = FALSE, main.legend = "")
multiplot(resI.T$res[,c("i","j","k")], contrast = resI.T$res[,4],
            num = 3, window = NULL, legend = TRUE,
            main = "Influence filtering - norm = TRUE", num.main = FALSE)
multiplot(resI.F$res[,c("i","j","k")], contrast = resI.F$res[,4],
            num = 3, window = NULL, legend = TRUE,
            main = "Influence filtering - norm = FALSE", num.main = FALSE)


layout(matrix(1:6, nrow = 3, ncol = 2, byrow = TRUE), widths = c(2,1), heights = rep(1,3))
par(mar = rep(2,4), mgp = c(2,0.75,0))
multiplot(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2", num = 3,
          window = NULL, legend = TRUE, main = "raw", num.main = FALSE, main.legend = "")
multiplot(resN.T$res[,c("i","j","k")], contrast = resN.T$res[,4],
            num = 3, window = NULL, legend = TRUE,
            main = "Neighborhood3D filtering - norm = TRUE", num.main = FALSE)
multiplot(resN.F$res[,c("i","j","k")], contrast = resN.F$res[,4],
            num = 3, window = NULL, legend = TRUE,
            main = "Neighborhood3D filtering - norm = FALSE", num.main = FALSE)
```

`calcGR` *Interface to the Growing Region algorithm*

Description

Call the GRalgo function to perform the Growing Region algorithm.

Usage

```
calcGR(contrast, W, seed, sigma_max, range = c(-Inf, +Inf), range.seed = c(-Inf, +Inf),
         breaks = 100, rescale = FALSE, iter_max = 100, sd.robust = FALSE,
 keep.lower = FALSE, keep.upper = FALSE, verbose = TRUE,
 history.sigma = FALSE, history.step = FALSE, history.front = FALSE)
```

Arguments

`contrast`	the contrast value of each observation. *numeric vector*. REQUIRED.
`W`	the neighbourhood matrix. *dgCMatrix*. REQUIRED.
`seed`	the index of the initial seeds or a binary indicator of the initial seeds. *positive integer vector* or *logical vector*. REQUIRED.
`sigma_max`	the maximum admissible value for the variability of the group contrast. *positive numeric*. REQUIRED.
`range`	the range of acceptable contrast values for the growing region group. *numeric vector of size 2*.
`range.seed`	the range of acceptable contrast values for the seeds. *numeric vector of size 2*.
`breaks`	the break points or the number of break points to use to categorize the contrast distribution. *numeric vector* or `postive integer`.
`rescale`	should the contrast be scaled ? *logical*.
`iter_max`	the maximum number of iterations for the expansion of the growing region. `postive integer`.
`sd.robust`	should the median absolute deviation be used to estimte the variability of the group contrast, or the standard deviation ? *logical*.
`keep.lower`	should removing observations with high intensity of the region be forbidden ? *logical*.
`keep.upper`	should removing observations with low intensity of the region be forbidden ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.
`history.sigma`	should the values of sigma be recorded ? *logical*.
`history.step`	should the number of observations included in the growing region set be recorded ? *logical*.
`history.front`	should the propagation front of the GR set be recorded ? *logical*.

Details

FUNCTION:
This implementation of the Growing Region algorithm was been proposed by (Revol et al. 1997).

Value

An *list* containing :

- `[[GR]]` : the index of the observations in the growing region. *integer vector*.
- `[[test.break]]` : whether the GR algorithm was interrupted an during execution. *logical*.
- `[[iter]]` : the number of the last iteration of the algorithm. *integer*.
- `[[test.id]]` : whether the GR set has stabilised during the last iteration. *logical*.
- `[[sigma]]` : if `history.sigma` was set to `TRUE`, the value of the homogeneity criterion at the begining and the end of each step (in columns) for all steps (in row). *numeric matrix*.

- `[[history_GR]]` : if `history.step` was set to `TRUE`, the step when each GR observation was included in the GR set. *integer vector*.
- `[[breaks]]` : if `history.front` was set to `TRUE`, the values used to categorize the contrast. *numeric vector*.

References

Chantal Revol and Michel Jourlin. *A new minimum varance region growing algorithm for image segmentation*. Pattern Recognition Letters, 18(3):249-258,1997.

See Also

`calcCriteriaGR` for an automatic estimate of the sigma value.

Examples

```
## load a MRIaggr object
data(MRIaggr.Pat1_red, package = "MRIaggr")

calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0"), threshold = 1:10,
                     name_newparam = c("TTP.th_t0","MTT.th_t0"),
                     update.object = TRUE, overwrite = TRUE)

## display raw parameter
multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = "MASK_DWI_t0", outline = TRUE))

## extract raw parameter, coordinates and compute the neighbourhood matrix
carto <- selectContrast(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion",
                        param = c("TTP.th_t0","TTP_t0","MASK_DWI_t0"))
coords <- selectCoords(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion")
W <- calcW(coords, range = sqrt(2))$W

## the seed is taken to be the point with the largest TTP in the lesion mask
indexN <- which(carto$MASK_DWI_t0 == 1)
seed <- indexN[which.max(carto[indexN,"TTP_t0"])]

## Display step by step the GR algorithm with sigma = 1
for(iter in c(0,1,2,5,10)){
  resGR1 <- calcGR(contrast = carto$TTP.th_t0, W = W,
                   seed = seed, sigma_max = 1, iter_max = iter, verbose = FALSE)

  multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3,hemisphere = "lesion", legend = FALSE,
         breaks = seq(0,10,0.1), numeric2logical = TRUE, cex = 2,
 main = paste("iteration=",iter," - slice ", sep = ""),
         index1 = list(coords = coords[resGR1$GR,], pch = 20, cex = 1),
         index2 = list(coords = coords[seed,], pch = 20, cex = 1)
  )
}

## Not run:
## GR with sigma = 2.2
```

```
resGR2 <- calcGR(contrast = carto$TTP.th_t0, W = W,
                 seed = seed, sigma_max = 2.2, iter_max = 50,
                 history.step = TRUE, history.front = TRUE)

## display
# display the GR over the raw contrast
multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, hemisphere = "lesion", legend = FALSE,
             breaks = seq(0,10,0.1), numeric2logical = TRUE, cex = 2,
             index1 = list(coords = coords[resGR2$GR,], pch = 20, cex = 1)
)

# display the step of inclusion in GR group for each observation
multiplot(coords[resGR2$GR,],
             resGR2$history.step,breaks = 0:10,
             index1=list(coords = coords[seed,]),
             palette = rainbow(10)
)

# display the front propagation
multiplot(coords[resGR2$GR,],
             resGR2$Mfront[,7],
             index1 = list(coords = coords[seed,])
)

## End(Not run)
```

`calcGroupsCoords` *Compute spatial groups*

Description

Compute the spatial groups using the coordinates of the observations.

Usage

```
calcGroupsCoords(coords, array = NULL, Neighborhood, max_groups = 10000,
         verbose = optionsMRIaggr("verbose"))
```

Arguments

`coords`	the spatial coordinates of the observations. *data.frame*. REQUIRED.
`array`	alternative specification of the spatial coordinates using an array where the non-NA values indicates the points of interest. *array* or `NULL` leading to consider the `coords` argument.
`Neighborhood`	the type of neighbourhood. *character*.
`max_groups`	the maximum number of groups. *postive integer*.
`verbose`	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
the `Neighborhood` argument can be a *matrix* or an *array* defining directly the neighbourhood to use (i.e the weight of each neighbor) or a name indicating which type of neighbourhood should be used (see the details section of `initNeighborhood`).

Value

An *list* containing :

- `[[ls.group]]` : `lists` of the observations of each spatial group.
- `[[df.group]]` : a *data.frame* indicating the position and the group of each observation.
- `[[group_size]]` : the size of each spatial group. *integer vector*.

See Also

`initFilter` for various pre-stored filters.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

optionsMRIaggr(outline.index = TRUE, num.main = FALSE)

## select data
MASK_DWI_t0 <- selectContrast(MRIaggr.Pat1_red, param = "MASK_DWI_t0")
coords <- selectCoords(MRIaggr.Pat1_red)

#### 1- compute spatial groups using coordinates ####
res3DN18 <- calcGroupsCoords(coords = coords[MASK_DWI_t0 == 1,], Neighborhood = "3D_N18")
res3DN18$group_size

## display the lesion spatial groups
multiplot(coords, contrast=MASK_DWI_t0, legend = FALSE, num=2,
          index1=coords[MASK_DWI_t0 == 1,][res3DN18$ls.group[[1]],])

#### 2-compute spatial groups using an array ####
A.MASK_DWI_t0 <- df2array(MASK_DWI_t0,coords = coords)$contrast[[1]]
A.MASK_DWI_t0[A.MASK_DWI_t0 == FALSE] <- NA

## display
graphics::image(A.MASK_DWI_t0[,,3])

## computation of the spatial groups
res3DN18.bis <- calcGroupsCoords(array = A.MASK_DWI_t0, Neighborhood = "3D_N18")

res3DN18$group_size - res3DN18.bis$group_size # same result
```

calcGroupsCoords_cpp *Find spatial groups*

Description

C++ function called by `calcGroupsCoords` to compute the spatial groups. For internal use.

Usage

```
calcGroupsCoords_cpp(coords_NNA, index_NNA, min_index_NNA, max_index_NNA,
         Neighborhood, coords_max, max_groups, verbose)
```

Arguments

coords_NNA	the spatial coordinates of the observations in C version (beginning at 0). *matrix*.
index_NNA	the index of the coordinates in a array in C version (beginning at 0). *numerical vector*.
min_index_NNA	the minimum value of index_NNA. *postive integer*.
max_index_NNA	the maximum number of index_NNA. *postive integer*.
Neighborhood	the type of neighbourhood. *character*.
coords_max	the maximum coordinate in each dimension. *numerical vector*
max_groups	the maximum number of groups. *postive integer*.
verbose	should the execution of the function be traced ? *logical*.

calcGroupsMask *Compute spatial groups*

Description

Compute the spatial groups defined by a binary parameter.

Usage

```
## S4 method for signature 'MRIaggr'
calcGroupsMask(object, mask, numeric2logical = FALSE,
     W = "ifany", W.range, W.spatial_res = c(1,1,1),
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = TRUE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`mask`	the binary contrast parameter that should be used to identifying the spatial groups. *character*. REQUIRED.
`numeric2logical`	should `mask` be convert to logical ? *logical*.
`W`	the neighbourhood matrix. *dgCMatrix* or `"ifany"` leading to use the neighbourhood matrix stored in the `object` if any and else computate this matrix.
`W.range`	the neighbourhood range. *postive numeric*. REQUIRED.
`W.spatial_res`	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
`verbose`	should the execution of the function be traced ? *logical*.
`update.object`	should the resulting spatial groups be stored in `object` ? *logical*.
`overwrite`	if spatial groups are already stored in `object@ls_descStats`, can they be overwritten ? *logical*.

Details

This function requires to have installed the *Matrix* and the *spam* package to work.

FUNCTION:
Call the `calcGroupsW` function.

Value

An *list* containing for each parameter:

- `[[group]]` : a *vector* containing the group index for each observation.
- `[[group_size]]` : a *vector* with the size of each spatial group.
- `[[group_number]]` : the number of spatial groups. *integer vector*.
- `[[group_max]]` : the index of the largest group. *integer vector*.

See Also

`selectDescStats` to select the spatial groups.
`calcW` to compute the neighbourhood matrix.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute spatial groups
calcGroupsMask(MRIaggr.Pat1_red, mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"),
               W.range = 6, W.spatial_res = c(1.875,1.875,6),
               update.object = TRUE, overwrite = TRUE)

## extract spatial groups
selectDescStats(MRIaggr.Pat1_red, "GroupsLesion")
```

calcGroupsW *Compute spatial groups*

Description

Compute the spatial groups using a neighbourhood matrix.

Usage

```
calcGroupsW(W, subset = NULL, max_groups = 10000,
         verbose = optionsMRIaggr("verbose"))
```

Arguments

W	the neighbourhood matrix. *dgCMatrix*. REQUIRED.
subset	the subset of observations to use. *positive integer vector* or NULL leading to use all observations.
max_groups	the maximum number of groups. *postive integer*.
verbose	should the execution of the function be traced ? *logical*.

Details

This function requires to have installed the *Matrix* and the *spam* package to work.

Value

An *list* containing :

- [[group]] : a *vector* containing the group index for each observation. Observations out of the subset are set to NA.
- [[subset]] : a *vector* containing the group index for each observation in subset.
- [[group_size]] : a *vector* with the size of each spatial group.
- [[group_number]] : the number of spatial groups. *integer vector*.
- [[group_max]] : the number of the largest group. *integer vector*.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## select data
MASK_DWI_t0 <- selectContrast(MRIaggr.Pat1_red, param = "MASK_DWI_t0")
coords <- selectCoords(MRIaggr.Pat1_red)

## select compute W
W <- calcW(object = as.data.frame(coords[MASK_DWI_t0 == 1,]),
           range = sqrt(2), row.norm = TRUE)$W
```

```
## find spatial groups
res.Groups <- calcGroupsW(W)
res.Groups$group_size

## display
multiplot(coords[MASK_DWI_t0 == 1,], contrast = res.Groups$group,
          legend = FALSE, cex=0.5,
          palette=rainbow(10)[-1])
```

`calcGroupsW_cpp` *Find spatial groups*

Description

C++ function called by `calcGroupsW` to compute the spatial groups. For internal use.

Usage

```
calcGroupsW_cpp(W_i, W_p, subset, max_groups, verbose)
```

Arguments

`W_i`	the 0-based row numbers for each non-zero element in the sparse neighbourhood matrix. *integer vector*.
`W_p`	the pointers to the initial (zero-based) index of elements in the colum of the sparse neighbourhood matrix. *integer vector*.
`subset`	the subset of observations to use. *positive integer vector*.
`max_groups`	the maximum number of groups. *postive integer*.
`verbose`	should the execution of the function be traced ? *logical*.

`calcHemisphere` *Find the mid-sagittal plan*

Description

Find a plane that distinguish the two cerebral hemispheres.

Usage

```
## S4 method for signature 'MRIaggr'
calcHemisphere(object, param, num = NULL, p = 1, subset = NULL,
          penalty = "symmetry", mask = NULL, numeric2logical = FALSE,
 n.points = 100, gridSearch = TRUE,
          i_test = seq(-20, 20, by = 5), angle.test = seq(-30, 30, by = 5),
 unit_angle = "degree", NelderMead = TRUE, maxit = 100, reltol = 0.001,
        plot = TRUE, filename = paste(object@identifier, "_calcHemisphere", sep = ""),
          update.object = FALSE, overwrite = FALSE, ...)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameter that should be used to distinguish the two hemispheres. *character*. REQUIRED.
num	the slices to use. *numeric vector* or NULL.
p	the type of distance for the penalization. *positive numeric*.
subset	the subset of observations to use. *positive integer vector* or NULL leading to use all observations.
penalty	the type of objective function. Can be "symmetry" or "asymmetry".
mask	the binary contrast parameter(s) indicating the lesion. *character vector* or NULL if no mask is available.
numeric2logical	should mask be convert to logical ? *logical*.
gridSearch	Should grid search be used to find the mid-sagittal plane ? *logical*.
i_test	the abscissa or the number of abscissa to test. *numeric vector* or *positive integer*.
angle.test	the angle or the number of angle to test. *numeric vector* or *postive integer*.
unit_angle	the unit in which the angle is given. Can be "radian" or "degree".
n.points	the number of points that represent the mid-sagittal plan to computed. positive integer.
NelderMead	Should the center of the grid search be searched using Nelder-Mead algorithm or set to the center of the image ? *logical*.
maxit	The maximum number of iterations. *postive integer*. See the details section of optim.
reltol	Relative convergence tolerance. *positive numeric*. See the details section of optim.
plot	should the results be plotted ? *logical*.
filename	the name of the file used to export the plot. *character*.
update.object	should the resulting midplane be stored in object ? *logical*.
overwrite	if a midplane is already stored in object@midplane, can it be overwritten ? *logical*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Setting p to 1 leads to use the absolute difference ; setting p to 2 leads to use the euclidean distance.

Arguments ... must correspond to some of the following arguments : height, numeric2logical, path, res, unit, verbose, width, window.

FUNCTION:
This function seeks the plane that minimize the difference between contralateral values of the two

hemispheres or maximize the similarity between the two hemispheres. There are 2 degree of freedom : one for the position of the center and one for the angle. The separation between the hemisphere is assumed to be identical for all slices. From our experience, using an objective function based on symmetry gives better results compared to asymmetry.

Value

An *list* containing :

- `[[penalty]]` : an *array* containing the objective function for the various configurations.
- `[[nb]]` : an *array* containing the number of observations used to compute the penalty function for the various configurations.
- `[[moy]]` : an *array* containing the mean value of the objective function for the various configurations.
- `[[optimum]]` : the parameters of the optimal midplane. *numeric vector*
- `[[midplane]]` : the position of the midplane points. *matrix*.
- `[[data]]` : the position of the observations with respect to the mid-sagittal plane. *matrix*.
- `[[cv]]` : Was the optimum reached inside the parameter space and not at a border ? *logical*.

See Also

`selectParameter` to select the midplane.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")



## Not run:
res <- calcHemisphere(MRIaggr.Pat1_red, param = "T2_GRE_t0",
                      verbose = TRUE, update.object = TRUE, overwrite = TRUE)

## display the mid-sagittal plan
multiplot(MRIaggr.Pat1_red, param = "T2_GRE_t0", num = 3, legend = FALSE,
             midplane = TRUE, main = "original coordinates - slice ")

## display with the new coordinates
multiplot(selectContrast(MRIaggr.Pat1_red, param = c("i_hemisphere","j_hemisphere","k")),
          contrast=selectContrast(MRIaggr.Pat1_red, param = "T2_GRE_t0"), num = 3,
        index1=cbind(0, seq(-50,50), 3), main = "new coordinates - slice ", legend = FALSE)

## compute the mid-sagittal plan and mark lesion and healthy hemispheres
res <- calcHemisphere(MRIaggr.Pat1_red, param = "T2_GRE_t0",
         mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"), numeric2logical = TRUE,
         verbose = TRUE, update.object = TRUE, overwrite = TRUE)

## End(Not run)
```

`calcHemi_cpp` *Mid-saggital plan search*

Description

C++ function called by `calcHemisphere` to compute the objective function that evaluates the consistency of the mid-sagittal plan. For internal use.

Usage

```
calcHemi_cpp(coordsI, coordsJ, ls_indexK, n_num, value, n,
         i_pos, j_pos, angle_pos,
         penaltyNA, sd_data, p, symetrie)
```

Arguments

`coordsI`	the coordinates of each voxel along the first coordinate. *integer vector*.
`coordsJ`	the coordinates of each voxel along the second coordinate. *integer vector*.
`ls_indexK`	the index of the voxels on each slice. *list*.
`n_num`	the number of slices. *integer*.
`value`	the contrast parameter value for each voxel. *numeric vector*.
`n`	the number of voxels. *integer*.
`i_pos`	the first coordinate of the center of the mid-sagittal plane. *numeric*.
`j_pos`	the second coordinate of the center of the mid-sagittal plane. *numeric*.
`angle_pos`	the angle between the axis of the second coordinate and the mid-sagittal plane. *numeric*.
`penaltyNA`	a penalty factor for the percent of voxel without controlateral correspondant. *numeric*.
`sd_data`	the standard deviation of the parameter. *numeric*.
`p`	the penalization factor. *positive numeric*.
`symetrie`	the type of objective function. `TRUE` correspond to `"symmetry"` and `FALSE` to `"asymmetry"`.

`calcMultiPotential` *Computation of the spatial potential*

Description

Compute the regional potential of a spatial field. Call the `calcMultiPotential_cpp` C++ function.

Usage

```
calcMultiPotential(W_SR, W_LR, distance.ref, sample, coords,
        threshold = 0.01, nbGroup_min = 100, check.args = TRUE, verbose = TRUE)
```

Arguments

`W_SR`	The local neighborhood matrix. *dgCMatrix*. Should be normalized by row (i.e. `rowSums(Wweight_SR)=1`). REQUIRED.
`W_LR`	The regional neighborhood matrix. *dgCMatrix*. Should contain the distances between the observations (`0` indicating infinite distance). REQUIRED.
`distance.ref`	The intervals of distance defining the several neighborhood orders in `W_LR`. *numeric vector*. REQUIRED.
`sample`	The probability membership to the group. *numeric vector* or *character vector*. REQUIRED.
`coords`	The voxel coordinates. *matrix*. REQUIRED.
`threshold`	The minimum value to consider non-negligible group membership. *numeric*. Default is `0.01`.
`nbGroup_min`	The minimum group size of the spatial groups required for computing the potential. *integer*. Default is `100`.
`check.args`	Should the validity of the arguments be checked ? *logical*.
`verbose`	Should the radius of the spatial groups be displayed ? *logical*.

Details

If `check.args` is set to `TRUE`, argument `ccords` must be a *matrix* (and not a *data.frame*) and the x slot of argument `W_LR` must contain categories of distances instead of distances. Distance categories must begin at `0` and end at `length(distance.ref)-1`.

Value

A *numeric vector* containing the regional potential.

Examples

```
# spatial field
## Not run:
n <- 30

## End(Not run)

G <- 3
coords <- data.frame(which(matrix(0, nrow = n * G, ncol = n * G) == 0, arr.ind = TRUE), 1)
optionsMRIaggr(quantiles.legend = FALSE, axes = FALSE, num.main = FALSE)

# neighborhood matrix
W_SR <- calcW(coords, range = sqrt(2), row.norm = TRUE)$W
W_LR <- calcW(coords, range = 10, row.norm = FALSE)$W
distance.ref <- seq(1, 10, 1)

# data
set.seed(10)
sample <- simulPotts(W = W_SR, G = G, rho = 3.5, iter_max = 500)

#
multiplot(coords,sample$simulation[,2])

V1 <- calcMultiPotential(W_SR = W_SR, W_LR = W_LR, distance.ref = seq(1, 10, 1),
     sample = sample$simulation[,2], coords = coords, verbose = TRUE)

multiplot(coords, V1)


#
sampleV <- (apply(sample$simulation, 1, which.max) - 1) / 2
multiplot(coords, sampleV)

system.time(
  V2 <- calcMultiPotential(W_SR = W_SR, W_LR = W_LR, distance.ref = seq(1, 10, 1),
     sample = sampleV, coords = coords, verbose = TRUE)
)

Wcat_LR <- W_LR
Wcat_LR@x <- findInterval(x = Wcat_LR@x, vec = distance.ref) - 1

system.time(
  V2 <- calcMultiPotential(W_SR = W_SR, W_LR = Wcat_LR, distance.ref = seq(1, 10, 1),
     sample = sampleV, coords = as.matrix(coords), verbose = TRUE, check.args = FALSE)
)
# quicker but arguments have to be correctly specified

multiplot(coords, V2)
```

calcMultiPotential_cpp *Computation of the spatial potential*

Description

C++ function that computes the regional potential of spatial groups. For internal use.

Usage

```
calcMultiPotential_cpp(W_SR, W_LR, sample, threshold, coords,
        distance_ref, nbGroup_min, multiV, neutre)
```

Arguments

W_SR	The local neighborhood matrix. *dgCMatrix*. Should be normalized by row (i.e. rowSums(W_SR)=1).
W_LR	The regional neighborhood matrix. *dgCMatrix*. Should contain the distances between the observations (0 indicating infinite distance).
sample	The group probability membership. *numeric vector*.
threshold	The minimum value to consider non-negligible group membership. *numeric*.
coords	The voxel coordinates. *matrix*.
distance_ref	The intervals of distance defining the several neighborhood orders in W_LR. *numeric vector*.
nbGroup_min	The minimum group size of the spatial groups required for performing regional regularization. *integer*.
multiV	Should the regional potential range be specific to each spatial group ? *logical*.
neutre	the value to which the regional potential is set when the group is too small to compute it. *numeric*.

See Also

calcMultiPotential which is the R interface to this C++ function.

calcNormalization *Compute normalization values*

Description

Compute the normalization values for each contrast parameter.

Usage

```
## S4 method for signature 'MRIaggr'
calcNormalization(object, param, mu_type = "mean", sigma_type = "sd",
         rm.CSF = FALSE, rm.WM = FALSE, rm.GM = FALSE,
       verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameters to normalize. *character vector*. REQUIRED.
mu_type	the type of centering. Can be "mean" or "median".
sigma_type	the type of scaling. Can be "sd" or "mad".
rm.CSF	should the cerebral spinal fluid observations be excluded ? *logical*.
rm.GM	should the grey matter observations be excluded ? *logical*.
rm.WM	should the white matter observations be excluded ? *logical*.
verbose	should the execution of the function be traced ? *logical*.
update.object	should the resulting normalization values be stored in object ? *logical*.
overwrite	if normalization values are already stored in object@normalization, can they be overwritten ? *logical*.

Details

FUNCTION:
If any of the rm.CSF, rm.WM or rm.GM is set to true, then the values of the parameters remaining to FALSE (among CSF, WM and GM) are summed. Voxels with value under 0.5 are discarded.
Note that rm.CSF, rm.GM and rm.WM cannot be set simultaneously to TRUE.

Value

An *list* containing the normalization values, one element for each type of normalization.

See Also

selectNormalization to select the normalization values.
calcTissueType to compute a probabilistic classification of the brain observations in WM/GM/CSF.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute normalization values
res <- calcNormalization(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"),
                         update.object = TRUE, overwrite = TRUE)

## display
par(mfrow = c(2,4), mar = rep(1.5,4), mgp = c(2,0.5,0))
```

```
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2", num = 1:3,
            legend = TRUE, window = NULL, main = "raw - slice ")
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2", num = 1:3,
 norm_mu="contralateral", norm_sigma="contralateral",
            legend = TRUE, window = NULL, main = "normalized - slice ")

## extract normalization
selectNormalization(MRIaggr.Pat1_red, type = "global", mu = TRUE, sigma = FALSE)
```

`calcPotts` *Spatial regularization using ICM*

Description

Interface to C++ functions that perform spatial regularization of probabilistic group membership using Iterated Conditional Means.

Usage

```
calcPotts(W_SR, sample, rho, prior = TRUE, site_order = NULL, W_LR = NULL,
         nbGroup_min = 100, coords = NULL, distance.ref = NULL, threshold = 0.1,
 multiV = TRUE, iter_max = 200, cv.criterion = 0.005, verbose = 2)
```

Arguments

`W_SR`	The local neighborhood matrix. *dgCMatrix*. Should be normalized by row (i.e. `rowSums(W_SR)=1`). REQUIRED.
`sample`	The initial group probability membership. *numeric vector*. REQUIRED.
`rho`	Value of the spatial regularisation parameters. *numeric vector*. REQUIRED.
`prior`	Should the `sample` values be used as a prior ? *logical*.
`site_order`	a specific order to go all over the sites. *integer vector*.
`W_LR`	The regional neighborhood matrix. *dgCMatrix*. Should contain the distances between the observations (`0` indicating infinite distance).
`nbGroup_min`	The minimum group size of the spatial groups required for performing regional regularization. *integer*.
`coords`	The voxel coordinates. *matrix*.
`distance.ref`	The intervals of distance defining the several neighborhood orders in `W_LR`. *numeric vector*.
`threshold`	The minimum value to consider non-negligible group membership. *numeric*.
`multiV`	Should the regional potential range be specific to each spatial group ? *logical*.
`iter_max`	Maximum number of ICM iterations. *integer*.
`cv.criterion`	Convergence criterion of the ICM . *numeric*.
`verbose`	should the ICM be be traced over iterations ? *logical*. 1 to display each iteration and 2 to display convergence diagnostics.

Details

The convergence criterion of the ICM is computed as maximum absolute difference between the group membership probability between two consecutive iterations.

Examples

```
optionsMRIaggr(legend=FALSE,axes=FALSE,num.main=FALSE,mar=c(0,2,2,0))

# spatial field
## Not run:
n <- 30

## End(Not run)

G <- 3
coords <- which(matrix(0, nrow = n * G, ncol = n * G) == 0,arr.ind = TRUE)

# neighborhood matrix
W_SR <- calcW(as.data.frame(coords), range = sqrt(2), row.norm = TRUE)$W
resW <- calcW(as.data.frame(coords), range = 10, row.norm = FALSE, calcBlockW = TRUE)
W_LR <- resW$W
site_order <- unlist(resW$blocks$ls_groups) - 1

# initialisation
set.seed(10)
system.time(
sample <- simulPotts(W_SR, G = 3, rho = 3.5, iter_max = 500,
     site_order = site_order)$simulation
)

intensity <- rnorm((n * G)^2, mean = apply(sample, 1, which.max), sd = 0.5)
likelihood <- matrix(unlist(lapply(1:3, function(x){dnorm(intensity, mean = x, sd = 0.5)})),
     ncol = G, nrow = (n * G)^2, byrow = FALSE)
likelihood_sqrt <- sqrt(likelihood)

probability <- sweep(likelihood_sqrt, MARGIN = 1, FUN = "/", STATS = rowSums(likelihood_sqrt))

multiplot(as.data.frame(coords), probability, palette = "rgb",
     main = "original image")

#### local image restoration
LocalRestoration <- calcPotts(W_SR = W_SR, sample = probability, rho = 4,
     site_order = site_order)

multiplot(as.data.frame(coords), LocalRestoration$predicted, palette = "rgb",
     main = "local restoration of the image")

#### regional image restoration
distance.ref <- seq(1, 10, 1)

RegionalRestoration <- calcPotts(W_SR = W_SR, sample = probability,
     rho = c(4,2), site_order = site_order,
```

```
     W_LR = W_LR, coords = coords, distance.ref = distance.ref)

# regional potential
multiplot(as.data.frame(coords),
    matrix(unlist(RegionalRestoration$Vregional), ncol = 3, nrow = (n * G)^2, byrow = FALSE),
     palette = "rgb", main = "regional potentials")

# final image
multiplot(as.data.frame(coords),
    matrix(unlist(RegionalRestoration$predicted), ncol = 3, nrow = (n * G)^2, byrow = FALSE),
     palette = "rgb", main = "local and regional \n restoration of the image")
```

`calcPottsParameter` *Estimation of the local regularization parameters*

Description

Estimation of the local regularization parameters using mean field approximation or a likelihood free method.

Usage

```
calcPottsParameter(Y, W_SR, coords = NULL, range = NULL, method = "MF", verbose = 3, ...)
```

Arguments

`Y`	a *matrix* containing the observations (by rows) for the various groups (by columns). REQUIRED.
`W_SR`	the local neighbourhood matrix. *dgCMatrix*. Should be normalized by row (i.e. `rowSums(W_SR)=1`). REQUIRED.
`coords`	if argument `W_SR` is not specified, the coordinates of the sites used to compute the local neighbourhood matrix. *data.frame*.
`range`	if argument `W_SR` is not specified, the range of the neighbourhood used to compute the local neighbourhood matrix. positive *double*.
`method`	the method used to estimate the regularization parameter. Can be either `"MF"` or `"Lvfree"`.
`verbose`	should the execution of the function be be traced ? Can be `0`, `1`, `2` or `3`.
`...`	additional arguments to be passed to the function called by `calcPottsParameter` to perform the estimation. See the arguments of `link{rhoMF}` or `link{rhoLvfree}`.

Details

FUNCTION:
This function call either `rhoMF` if argument `method` is `"MF"` or `rhoLvfree` if argument `method` is `"Lvfree"`. Estimation of the regional regularization parameter is only available with mean field approximation (`method="MF"`). The likelihood free estimation (Pereyra et al, 2013) should give a more accurate estimation despite a higher computational cost.

References

M. Pereyra, N. Dobigeon, H. Batatia, and J.Y. Tourneret. *Estimation the granularity coefficient of a Potts-Markov random field within an MCMC algorithm*. IEEE Trans. Image Porcessing, 22(6):2385-2397, 2013.

See Also

`calcW` to compute the neighbourhood matrix,
`simulPotts` to simulate from a Potts model.

Examples

```
# spatial field
## Not run:
n <- 50

## End(Not run)

G <- 3
coords <- as.data.frame(which(matrix(0, nrow = n * G, ncol = n * G) == 0, arr.ind = TRUE))

# neighbourhood matrix
resW <- calcW(as.data.frame(coords), range = sqrt(2), row.norm = TRUE, calcBlockW = TRUE)
W_SR <- resW$W
site_order <- unlist(resW$blocks$ls_groups) - 1

# initialisation
set.seed(10)
sample <- simulPotts(W_SR, G = 3, rho = 3.5, iter_max = 500,
                     site_order = site_order)$simulation

#### estimation using the neighbourhood matrix
rho_MF <- calcPottsParameter(Y = sample, W_SR = W_SR,  method = "MF")

## Not run:
rho_Lvfree <- calcPottsParameter(Y = sample, W_SR = W_SR,
                                 site_order = site_order, method = "Lvfree", verbose = 2)

## End(Not run)

#### estimation using the coordinates
rho_MF <- calcPottsParameter(Y = sample, coords = coords, range = sqrt(2), method = "MF")

## Not run:
rho_Lvfree <- calcPottsParameter(Y = sample, coords = coords, range = sqrt(2),
                                 site_order = site_order, method = "Lvfree", verbose = 2)

## End(Not run)
```

calcPotts_cpp *Iterated conditional means for spatial regularization*

Description

C++ function that performs spatial regularization of probabilistic group membership using Iterated Conditional Means. For internal use.

Usage

```
calcPotts_cpp(W_SR, W_LR, sample, rho, coords, site_order, iter_max, cv_criterion,
         test_regional, distance_ref, threshold, neutre, nbGroup_min, multiV,
     last_vs_others, prior, type_reg, verbose)
```

Arguments

W_SR	the local neighborhood matrix. *dgCMatrix*. Should be normalized by row (i.e. rowSums(W_SR)=1).
W_LR	the regional neighborhood matrix. *dgCMatrix*. Should contain the distances between the observations (0 indicating infinite distance).
sample	the initial group probability membership. *numeric vector*.
rho	the value of the spatial regularisation parameters. *numeric vector*.
coords	the voxel coordinates. *matrix*.
site_order	a specific order to go all over the sites. *integer vector*.
iter_max	the maximum number of iterations. *integer*.
cv_criterion	the convergence criterion. *numeric*.
test_regional	should regional regularisation be used. *logical*.
distance_ref	the distance defining the several neighborhood orders relatively to W_LR. *numeric vector*.
threshold	the minimum value to consider non-negligible group membership. *numeric*.
neutre	the value to which the regional potential is set when the group is too small to compute it. *numeric*.
nbGroup_min	the minimum group size of the spatial groups required for performing regional regularization. *integer*.
multiV	should the regional potential range be specific to each spatial group ? *logical*.
last_vs_others	should the regional potential be computed for each class or only for the last versus the others ? *logical*.
type_reg	the type image to regularize : continuous or categorical ? *logical*.
prior	should the sample values be used as a prior ? *logical*.
verbose	should the running be be traced over iterations ? *logical*. 1 to display each iteration and 2 to display convergence diagnostics.

See Also

calcPotts which is the R interface to this C++ function.

calcRadius_cpp *Compute geometric caracteristics of a spatial group*

Description

C++ function that computes the barycenter of a spatial group. For internal use.

Usage

```
calcRadius_cpp(coords, sample, threshold, subset_bary, verbose)
```

Arguments

coords	the spatial coordinates of the observations. *matrix* with a number of rows equal to the length of sample.
sample	the weight of each voxel in the computation of the barycenter. *positive numeric*.
threshold	observations with a sample value below the value of threshold are discarded. *numeric*.
subset_bary	an indicator of the observations that should be kept ? *logical vector*.
verbose	should the radius of the spatial group be printed ? *logical*.

calcRegionalContrast *Compute regional contrast parameters*

Description

Compute the regional contrast parameters.

Usage

```
## S4 method for signature 'MRIaggr'
calcRegionalContrast(object,param, bandwidth, power = 2, diagonal = FALSE,
       W = "ifany", W.range, W.spatial_res = c(1,1,1), num = NULL, hemisphere = "both",
         name_newparam = paste(param,"regional", sep = "_"),
       verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameter(s) from which the regional parameter(s) will be computed. *character vector*. REQUIRED.
bandwidth	the bandwidth of the kernel. *postive numeric*. REQUIRED.
power	the power of the kernel. *postive numeric*.

diagonal	should the diagonal be added to the neighbourhood matrix ? *logical*.
W	the neighbourhood matrix. *dgCMatrix* or "ifany" leading to use the neighbourhood matrix stored in the object if possible, else to compute this matrix.
W.range	the neighbourhood range. *postive numeric*. Required only if W have to be computed.
W.spatial_res	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
num	the slices to use. *numeric vector* or NULL.
hemisphere	the hemisphere to use. *character*.
verbose	should the execution of the function be traced ? *logical*.
name_newparam	the name of the new parameters. *character vector*.
update.object	should the resulting regional parameters be stored in object ? *logical*.
overwrite	if contrast parameters with the same names are already stored in object can they be overwritten ? *logical*.

Details

This function requires to have installed the *Matrix* and the *spam* package to work.

ARGUMENTS :
Information about the num argument can be found in the details section of initNum.

Information about the hemisphere arguments can be found in the details section of selectContrast.

Information about bandwidth and power arguments can be found in the details section of EDK.

Value

A *data.frame* containing in columns the regional parameters.

See Also

selectContrast to select the regional parameter(s).
calcW to compute the neighboring matrix.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute regional values
res  <- calcRegionalContrast(MRIaggr.Pat1_red, param = c("T2_FLAIR_t2","T1_t0"), bandwidth = 1.875,
                         W.spatial_res = c(1.875,1.875,6), W.range = 6,
                         update.object = TRUE, overwrite = TRUE)

## display
par(mfrow = c(2,4), mar = rep(1.5,4), mgp = c(2,0.5,0))
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2", num = 1:3,
             window = NULL, main = "raw - slice ")
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2_regional", num = 1:3,
             window = NULL, main = "regional - slice ")
```

`calcROCthreshold` *ROC analysis*

Description

Perform a ROC analysis on a continuous variable for identifying a binary outcome.

Usage

```
## S4 method for signature 'MRIaggr'
calcROCthreshold(object, param, mask, plot = "ROC_Youden", digit = 10,
       filename = paste(object@identifier, "calcROCthreshold", plot, sep = "_"),
         update.object = FALSE, overwrite = FALSE, ...)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the contrast parameter(s) that should be used to identify the observations inside the mask. *character vector*. REQUIRED.
`mask`	the binary contrast parameter that will be used as the outcome in the ROC analysis. *character vector*. REQUIRED.
`plot`	the type of the graphic to display? *character* or `FALSE`. See the details section.
`digit`	the number of digits used to round the data. *positive integer*. `NULL` indicates no rounding.
`filename`	the name of the file used to export the plot. *character*.
`update.object`	should the resulting threshold analysis be stored in `object@ls_descStats` ? *logical*.
`overwrite`	if a threshold analysis is already stored in `object@ls_descStats`, can it be overwritten ? *logical*.
`...`	additional arguments to be passed to `optionsMRIaggr` for specifying the graphical parameters.

Details

This function requires to have installed the *ROCR* package to work.

ARGUMENTS:
Arguments `...` must correspond to some of the following arguments : `height`, `numeric2logical`, `path`, `res`, `unit`, `verbose`, `width`, `window`.

Possible values for `plot` are:

- `"ROC_Youden"` : display the ROC curve with the optimal threshold according to the Youden index.
- `"ROC_prev"` : display the ROC curve with the optimal threshold according to the utility function.

- "boxplot_Youden" : display a boxplot of the contrast parameter for each outcome with the optimal threshold according to the Youden index.
- "boxplot_prev" : display a boxplot of the contrast parameter for each outcome with the optimal threshold according to the utility function.
- FALSE : no graphic is displayed.

Value

An *data.frame* containing for each mask the AUC and AUPRC value, the optimal threshold and the corresponding sensitivity and specificity for the Youden criteria and a utility function taking into account the prevalence.

See Also

selectDescStats to select the mask caracteristics.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## ROC analysis
res <- calcROCthreshold(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"),
                      mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"), numeric2logical = TRUE)

res <- calcROCthreshold(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"),
                      mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"), numeric2logical = TRUE,
                         plot = "boxplot_Youden",
update.object = TRUE, overwrite = TRUE)

selectDescStats(MRIaggr.Pat1_red, "Mask_threshold")
```

---

calcSigmaGR *Automatic Growing Region algorithm*

Description

Evaluate the quality of the Growing Region partition regarding several homogeneity parameters.

Usage

```
calcSigmaGR(contrast, W, seed, sigma, criterion.transition = FALSE,
       criterion.sdfront = FALSE, criterion.entropy = TRUE, criterion.Kalinsky = TRUE,
         criterion.Laboure = TRUE, verbose = TRUE, ...)
```

Arguments

`contrast`	the contrast value of each observation. *numeric vector*. REQUIRED.
`W`	the neighbourhood matrix. *dgCMatrix*. REQUIRED.
`seed`	the index of the initial seeds or a binary indicator of the initial seeds. *positive integer vector* or *logical vector*. REQUIRED.
`sigma`	the sequence of maximum admissible values for the group variability *positive numeric vector*. REQUIRED.
`criterion.transition`	should the boundary criterion based on the transition levels be computed ? *logical*.
`criterion.sdfront`	should the boundary criterion based on the standard deviation be computed ? *logical*.
`criterion.entropy`	should the region criterion based on the entropy be computed ? *logical*.
`criterion.Kalinsky`	should the region criterion based on the Kalinsky index be computed ? *logical*.
`criterion.Laboure`	should the region criterion based on the Laboure index be computed ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.
`...`	arguments to be passed to `calcGR` for specifying the settings of the growing region algorithm : `range`, `range.seed`, `breaks`, `scale`, `iter_max`, `sd.robust`, `keep.lower` and `keep.upper`.

Details

ARGUMENTS:
Information about the `window`, `filename`, `width`, `height`, `path`, `unit` and `res` arguments can be found in the details section of `initWindow`.

Information about the `mar` and `mgp` arguments can be found in `par`.

FUNCTION:
This implementation of the automated Growing Region algorithm was proposed by (Revol et al. 2002) : `criterion.transition` corresponds to w2, `criterion.sdfront` corresponds to w3 where m=f(x), `criterion.entropy` corresponds to S(sigma_max) and `criterion.Laboure` corresponds to InvDGL(sigma). `criterion.Kalinsky` corresponds to the Kalinsky criterion which is the ratio of the variance between groups over the variance withing groups.

Value

An *list* containing :

- `[[df.criterion]]` : the value of the clustering criterion (in columns) for each sigma value (in rows). *numeric matrix*.
- `[[list.GR]]` : the list of the optimal GR sets, one for each clustering criterion.
- `[[best]]` : the optimal value of each clustering criterion. *data.frame*.
- `[[n.max]]` : the number of observations. *integer*.

References

Chantal Revol-Muller, Francoise Peyrin, Yannick Carrillon and Christophe Odet. *Automated 3D region growing algorithm based on an assessment function*. Pattern Recognition Letters, 23:137-150,2002.

See Also

plotSigmaGR for a graphical display of the quality criteria.

Examples

```
## Not run:
## load an \code{MRIaggr} object
data(MRIaggr.Pat1_red, package = "MRIaggr")

calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0"), threshold = 1:10,
                     name_newparam = c("TTP.th_t0","MTT.th_t0"),
                     update.object = TRUE, overwrite = TRUE)

## display raw parameter
multiplot(MRIaggr.Pat1_red, param="TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = "MASK_DWI_t0", outline = TRUE))

## extract raw parameter, coordinates and compute the neighbourhood matrix
carto <- selectContrast(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion",
                        param = c("TTP.th_t0","TTP_t0","MASK_DWI_t0"))
coords <- selectCoords(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion")
W <- calcW(coords, range = sqrt(2))$W

## the seed is taken to be the point with the largest TTP in the lesion mask
indexN <- which(carto$MASK_DWI_t0 == 1)
seed <- indexN[which.max(carto[indexN,"TTP_t0"])]

## find optimal sigma
resGR_sigma <- calcSigmaGR(contrast = carto$TTP.th_t0, W = W, seed = seed,
                           sigma = seq(1,4,0.1), iter_max = 50,
                           keep.upper = TRUE)

## display quality criteria according to sigma
plotSigmaGR(resGR_sigma)

## display retained region
multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = coords[resGR_sigma$list.GR$entropy,], outline = TRUE))

multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = coords[resGR_sigma$list.GR$Kalinsky,], outline = TRUE))

## find optimal sigma
resGR_sigma <- calcSigmaGR(contrast = carto$TTP.th_t0, W = W, seed = seed,
                           sigma = seq(1,4,0.1), iter_max = 50,
                           keep.upper = TRUE, keep.lower = TRUE)
```

```
## display quality criteria according to sigma
plotSigmaGR(resGR_sigma)

## display retained region
multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = coords[resGR_sigma$list.GR$entropy,], outline = TRUE))

multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = coords[resGR_sigma$list.GR$Kalinsky,], outline = TRUE))

## End(Not run)
```

`calcSmoothMask` *Spatial regularization*

Description

Perform a spatial regularization of a binary mask.

Usage

```
## S4 method for signature 'MRIaggr'
calcSmoothMask(object, mask = "mask", numeric2logical = FALSE,
          size_2Dgroup = 50, Neighborhood_2D = "3D_N8", rm.2Dhole = FALSE,
          size_3Dgroup = "unique", Neighborhood_3D = "3D_N10", rm.3Dhole = TRUE,
          erosion.th = 0.75, Vmask_min = 0.25, Vbackground_max = 0.75,
          Neighborhood_V = "3D_N10", verbose = optionsMRIaggr("verbose"),
 update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`mask`	the binary contrast parameter that should be smoothed. *character*.
`numeric2logical`	should `mask` be convert to logical ? *logical*.
`size_2Dgroup`	the minimum size of the 2D groups. *positive integer* or `"unique"`.
`Neighborhood_2D`	the type of 2D neighbourhood. *character*.
`rm.2Dhole`	should the 2D wholes inside the mask be removed ? *logical*.
`size_3Dgroup`	the minimum size of the 3D groups. *positive integer* or `"unique"`.
`Neighborhood_3D`	the type of 3D neighbourhood. *character*.
`rm.3Dhole`	should the 3D wholes inside the mask be removed ? *logical*.
`erosion.th`	the threshold below which the observations will be removed by the erosion. *numeric between 0 and 1*.

Vmask_min	mask observations with a proportion of neighbors belonging to the mask lower than Vmask_min are attributed to the background. *numeric between 0 and 1.*
Vbackground_max	background observations with a proportion of neighbors belonging to the mask higher than Vbackground_max are attributed to the mask. *numeric between 0 and 1.*
Neighborhood_V	the type of neighbourhood to use for the spatial regularization. *character.*
verbose	should the execution of the function be traced ? *logical.*
update.object	should the resulting regularized mask be stored in object ? *logical.*
overwrite	if a mask is already stored in object@data, can it be overwritten ? *logical.*

Details

ARGUMENTS:
the Neighborhood_2D or Neighborhood_3D arguments can be a *matrix* or an *array* defining directly the neighbourhood to use (i.e the weight of each neighbor) or a name indicating which type of neighbourhood should be used (see the details section of initNeighborhood).

FUNCTION:
This function applies 6 smoothing steps :

- exclusion of the small 2D groups from the mask (to skip set size_2Dgroup to FALSE). Note that size_2Dgroup = "unique" lead to keep the largest 2D group of each slice.
- filling of the small 2D holes in the mask (to skip set rm.2Dhole to FALSE).
- exclusion of the small 3D groups from the mask (to skip set size_3Dgroup to FALSE). Note that size_3Dgroup = "unique" lead to keep only the largest 3D group.
- erosion that first temporarily remove observations from the mask that have less than erosion.th percent of their neighbourhood in the mask. Then it computes the new 3D groups and remove permanently all the new 3D groups from the mask. To skip set erosion.th to FALSE.
- filling of the small 3D holes in the mask (to skip set rm.3Dhole to FALSE).
- spatial regularization that homogenize the local neighbourhood (to skip set both Vmask_min and Vbackground_max to FALSE).

Value

An *data.frame* containing the mask and the coordinates in columns.

See Also

selectContrast to select the smoothed mask. calcBrainMask to compute an indicator of the brain observations.

Examples

```
## load data and build MRIaggr
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
ls.array <- list(readMRI(file.path(path.Pat1,"T2_GRE_t0"), format = "nifti"))
MRIaggr.Pat1 <- constMRIaggr(ls.array,identifier="Pat1", param = "T2_GRE_t0")
```

```
## create the cerebral mask
res <- calcBrainMask(MRIaggr.Pat1, param = "T2_GRE_t0", type = "kmeans",
                     kmeans.n_groups = 2:4,
                     update.object = TRUE, overwrite = TRUE)

## smooth the cerebral mask
res <- calcSmoothMask(MRIaggr.Pat1, update.object = TRUE, overwrite = TRUE)

## display
multiplot(MRIaggr.Pat1,param = "mask", legend = FALSE)
```

`calcTableHypoReperf` *Compute reperfusion and hypoperfusion tables*

Description

Compute of the reperfusion and hypoperfusion values.

Usage

```
## S4 method for signature 'MRIaggr'
calcTableHypoReperf(object, param, timepoint, threshold = 1:10, sep = "_",
     norm_mu = FALSE, norm_sigma = FALSE, mask = NULL, numeric2logical = FALSE,
     param.update = "reperf",
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the perfusion parameter(s). *character vector*. REQUIRED.
`timepoint`	one or two time points. *character vector*. REQUIRED.
`threshold`	the value of the hypoperfusion thresholds. *numeric vector*.
`sep`	the separator between the parameter names and the time points. *character*.
`mask`	the binary contrast parameter indentifing the lesion at timepoint[1]. *character* or `NULL` if no mask is available.
`numeric2logical`	should `mask` be converted to logical ? *logical*.
`norm_mu`	the type of centering to apply on the parameter values. *character*.
`norm_sigma`	the type of scaling to apply on the parameter values. *character*.
`verbose`	should the execution of the function be traced ? *logical*.
`param.update`	which type of parameter should be stored in the object ? Any of `"shift"` `"reperf"` `"reperf_pc"` `"deperf"` `"deperf_pc"`.
`update.object`	should the resulting values be stored in `object` ? *logical*.
`overwrite`	if reperfusion or hypoperfusion values are already stored in `object@table_reperfusion` or `object@table_hypofusion`, can they be overwritten ? *logical*.

Details

ARGUMENTS:
Information about the norm_mu and norm_sigma arguments can be found in the details section of selectContrast.

FUNCTION:
If mask is set to NULL, no mismatch volume will not be computed.

Value

An *list* containing :

- [[voxel]] : a *data.frame* containing the coordinates and the reperfusion values.
- [[volume_hypo]] : the number of hypoperfused observations for the various thresholds.
- [[volume_reperf]] : the number of reperfused observations for the various thresholds.

See Also

calcThresholdMRIaggr to process contrast parameters.
selectTable to select the reperfusion/hypoperfusion tables.
calcW to compute the neighboring matrix.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

#### 1- directly ####
res <- calcTableHypoReperf(MRIaggr.Pat1_red, param = c("TTP","MTT"), timepoint=c("t0","t1"),
     mask="MASK_DWI_t0", numeric2logical = TRUE,
     update.object = TRUE, overwrite = TRUE)

carto_TTP_t0 <- selectContrast(MRIaggr.Pat1_red, param = "TTP_t0")
carto_TTP_t1 <- selectContrast(MRIaggr.Pat1_red, param = "TTP_t1")

## hypoperfusion
sum( (carto_TTP_t0 >= 4) )
selectTable(MRIaggr.Pat1_red, "hypoperfusion")["4","Vhypo.TTP_t0"]

## mismatch
testN <- (selectContrast(MRIaggr.Pat1_red, param = "MASK_DWI_t0") == 0)

sum( (carto_TTP_t0 >= 4) * testN )
selectTable(MRIaggr.Pat1_red, "hypoperfusion")["4","Vmismatch.TTP"]

sum( (carto_TTP_t0 >= 4) * testN ) / sum( testN == FALSE )
selectTable(MRIaggr.Pat1_red, "hypoperfusion")["4","PCmismatch.TTP"]


## reperfusion
sum((carto_TTP_t0 >= 4) * (carto_TTP_t1 < 4))
```

```
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","Vreperf.TTP"]

sum((carto_TTP_t0 >= 4) * (carto_TTP_t1 < 4)) / sum( (carto_TTP_t0>=4) )
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","PCreperf.TTP"]

## W reperfusion
carto_TTPth_t0 <- carto_TTP_t0
carto_TTPth_t0[carto_TTPth_t0 > 10] <- 10
carto_TTPth_t0[carto_TTPth_t0 < 0] <- 0

carto_TTPth_t1 <- carto_TTP_t1
carto_TTPth_t1[carto_TTP_t1 > 10] <- 10
carto_TTPth_t1[carto_TTP_t1 < 0] <- 0

weight <- (carto_TTPth_t0 - carto_TTPth_t1) / carto_TTPth_t0
weight[ ((carto_TTPth_t0 == 0) + (carto_TTP_t0 < 4) + (carto_TTP_t1 >= 4)) > 0 ] <- 0

sum((carto_TTP_t0 >= 4) * (carto_TTP_t1 < 4) * weight)
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","VreperfW.TTP"]

sum((carto_TTP_t0 >= 4) * (carto_TTP_t1 < 4) * weight) / sum( (carto_TTP_t0 >= 4) )
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","PCreperfW.TTP"]


## deperfusion
sum((carto_TTP_t0 < 4) * (carto_TTP_t1 >= 4))
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","Vdeperf.TTP"]

sum((carto_TTP_t0 < 4) * (carto_TTP_t1 >= 4)) / sum( (carto_TTP_t0 >= 4) )
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","PCdeperf.TTP"]

## shift
sum((carto_TTPth_t0 - carto_TTPth_t1 >= 4))
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","Vshift_reperf.TTP"]

sum((carto_TTPth_t0-carto_TTPth_t1 >= 4)) / sum( (carto_TTP_t0>=4) )
selectTable(MRIaggr.Pat1_red, "reperfusion")["4","PCshift_reperf.TTP"]

#### 2- via calcThresholdMRIaggr ####
## perform segmentation  (call mritc)
## Not run:
calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0","TTP_t1","MTT_t1"),
       threshold=1:10, name_newparam = c("TTP.GR_t0","MTT.GR_t0","TTP.GR_t1","MTT.GR_t1"),
         rm.CSF=TRUE, hemisphere = "lesion",
         GRalgo=TRUE, seed = c("MASK_T2_FLAIR_t2","MASK_DWI_t0"), W.range  = sqrt(2),
         update.object = TRUE, overwrite = TRUE)

res <- calcTableHypoReperf(MRIaggr.Pat1_red, param = c("TTP.GR","MTT.GR"),
         timepoint = c("t0","t1"), mask = "MASK_DWI_t0", numeric2logical = TRUE,
         update.object = TRUE, overwrite = TRUE)

## display
selectTable(MRIaggr.Pat1_red, "hypoperfusion")["4","Vhypo.TTP.GR_t0"]
```

```
par(mfrow = c(2,4), mar = rep(1.5,4), mgp = c(2,0.5,0))
multiplot(MRIaggr.Pat1_red, param = "TTP_t0", num = 1:3,
             palette = rainbow(10), window = NULL, main = "raw - slice ",
 breaks=(0:10)-10^{-10})
multiplot(MRIaggr.Pat1_red, param = "TTP.GR_t0", num = 1:3,
             palette = rainbow(10), window = NULL, main = "GR - slice ",
 breaks=(0:10)-10^{-10})

## End(Not run)
```

`calcTableLesion` *Vertical distribution of the lesion*

Description

Compute the number of lesion observations at each slice.

Usage

```
## S4 method for signature 'MRIaggr'
calcTableLesion(object, maskN, mask = NULL, numeric2logical = FALSE,
       verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`maskN`	the binary contrast parameter indicating the lesion. *character*. REQUIRED.
`mask`	the binary contrast parameter indicating the brain. *character* or `NULL` if no mask is available.
`numeric2logical`	should `mask` and `maskN` values be converted to logical ? *logical*.
`verbose`	should the execution of the function be traced ? *logical*.
`update.object`	should the resulting lesion table be stored in `object` ? *logical*.
`overwrite`	if a lesion table is already stored in `object@table_lesion`, can it be overwritten ? *logical*.

Value

A *data.frame* containing the number of observation within the mask at each slice.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute table
res <- calcTableLesion(MRIaggr.Pat1_red, maskN = c("MASK_DWI_t0", "MASK_T2_FLAIR_t2"),
     numeric2logical = TRUE, update.object = TRUE, overwrite = TRUE)

## extract table
res <- selectTable(MRIaggr.Pat1_red, "lesion")
```

---

calcThreshold *Image thresholding*

Description

Threshold a contrast parameter at one or several values.

Usage

```
## S4 method for signature 'MRIaggr'
calcThresholdMRIaggr(object, param, hemisphere = "both", rm.CSF = FALSE,
     threshold = 1:10, decreasing = FALSE,
 GRalgo = FALSE, W = "ifany", seed = NULL, numeric2logical = FALSE, W.range,
 W.spatial_res = rep(1,3), name_newparam = paste(param,"Th", sep = "_"),
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)

calcThreshold(contrast, param, hemisphere = NULL, rm.CSF = FALSE, threshold = 1:10,
     decreasing = FALSE, GRalgo = FALSE, W = NULL, seed = NULL,
     numeric2logical = FALSE, verbose = optionsMRIaggr("verbose"))
```

Arguments

object	an object of class MRIaggr. REQUIRED.
contrast	the dataset containing the contrast parameter to be thresholded. *matrix*. REQUIRED.
param	the contrast parameters to be thresholded. *character vector*. REQUIRED.
hemisphere	the hemisphere to consider. *character* or NULL.
rm.CSF	should the cerebral spinal fluid observations be excluded ? *logical* or *character*.
threshold	the thresholds to be used for the discretization of the contrast parameter. *numeric vector*.
decreasing	should the increasing thresholding (FALSE) or decreasing thresholding (TRUE) be used. *logical*.
GRalgo	should a Growing Region algorithm be used to clean the thresholded parameter ? *logical*.

W	the neighbourhood matrix. *dgCMatrix* or "ifany" leading to use the neighbourhood matrix stored in the object if any and else computate this matrix.
seed	the index of the seeds for the growing region algorithm . *positive integer vector*.
numeric2logical	should seed be converted to logical ? *logical*.
W.range	only distances smaller than W.range are stored in W. *postive numeric*.
W.spatial_res	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
name_newparam	the name of the new parameters. *character*.
verbose	should the execution of the function be traced ? *logical*.
update.object	should the resulting thresholded parameters be stored in object ? *logical*.
overwrite	contrast parameters with the same names are already stored in object@data, can it be overwritten ? *logical*.

Details

These functions require to have installed the *Matrix* and the *spam* package to work when argument GRalgo is set to TRUE.

ARGUMENTS:
[data.frame method] the hemisphere argument must be one of the levels present in the column named "hemisphere" in data. NULL leading to use all observations.
[MRIaggr method] Information about the hemisphere arguments can be found in the details section of selectContrast.

FUNCTION:
[data.frame method] By default the indicator of CSF will be extract from the column named CSF. If it is contained in another column the user must specify rm.CSF with the name of this column.
[MRIaggr method] Setting rm.CSF to TRUE require to have the corresponding parameter (by default CSF) stored in the object. It can be done using calcTissueType.

References

Chantal Revol and Michel Jourlin. *A new minimum varance region growing algorithm for image segmentation*. Pattern Recognition Letters, 18(3):249-258,1997.

See Also

selectContrast to select contrast parameters in the MRIaggr object.

Examples

```
## load a MRIaggr object
data(MRIaggr.Pat1_red, package = "MRIaggr")

#### 1- MRIaggr method
## raw parameter
multiplot(MRIaggr.Pat1_red, param = "TTP_t0", legend = FALSE, main = "TTP_t0 - slice ",
            palette=rainbow(10), breaks = seq(0,10) - 10^{-10})
```

```
## thresholded parameter
calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0"),threshold = 1:10,
                name_newparam = c("TTP.th_t0","MTT.th_t0"),
                update.object = TRUE, overwrite = TRUE)

multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", main = "TTP.th_t0 - slice",
             legend = FALSE,palette = rainbow(10), breaks = (0:10) - 10^{-10})

## Not run:
## 1st correction
calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0"),threshold = 1:10,
                rm.CSF = TRUE, hemisphere = "lesion",
 name_newparam = c("TTP.red_t0","MTT.red_t0"),
 update.object = TRUE, overwrite = TRUE)

multiplot(MRIaggr.Pat1_red, param = "TTP.red_t0", main = "TTP.red_t0 - slice",
             legend = FALSE, palette = rainbow(10), breaks = (0:10) - 10^{-10})

## 2nd correction
calcThresholdMRIaggr(MRIaggr.Pat1_red, param = c("TTP_t0","MTT_t0"), threshold = 1:10,
        rm.CSF = TRUE, hemisphere = "lesion", name_newparam = c("TTP.GR_t0","MTT.GR_t0"),
         GRalgo = TRUE, seed = c("MASK_T2_FLAIR_t2","MASK_DWI_t0"), W.range  = sqrt(2),
         update.object = TRUE, overwrite = TRUE)

multiplot(MRIaggr.Pat1_red, param = "TTP.GR_t0", main = "TTP.GR_t0 - slice",
             legend = FALSE, palette = rainbow(10), breaks = (0:10) - 10^{-10})


#### 2- data.frame function ####

## raw parameter
multiplot(MRIaggr.Pat1_red, param = "TTP_t0", legend = FALSE, main = "TTP_t0 - slice ",
         palette=rainbow(10), breaks = seq(0,10) - 10^{-10})

## thresholded parameter
data <- selectContrast(MRIaggr.Pat1_red,
                       param = c("TTP_t0","MTT_t0","hemisphere","CSF","WM","GM"))
hypo_Th1_10 <- calcThreshold(data,param = c("TTP_t0","MTT_t0"), threshold = 1:10)

multiplot(selectCoords(MRIaggr.Pat1_red), main = "TTP_t0_th - slice ",
          hypo_Th1_10[,1], legend = FALSE, palette = rainbow(10), breaks = (0:10) - 10^{-10})

## 1st correction
data$CSF <- as.numeric(apply(data[,c("CSF","WM","GM")], 1, which.max) == 1)

hypoC_Th1_10 <- calcThreshold(data,param = c("TTP_t0","MTT_t0"), threshold = 1:10,
                              hemisphere = "left", rm.CSF = TRUE)

multiplot(selectCoords(MRIaggr.Pat1_red), main = "TTP_t0_thC - slice",
        hypoC_Th1_10[,1], legend = FALSE, palette = rainbow(10), breaks = (0:10) - 10^{-10})

## 2nd correction
maskN <- c("MASK_T2_FLAIR_t2","MASK_DWI_t0")
```

```
data[, maskN] <- selectContrast(MRIaggr.Pat1_red, param = maskN)
W <- calcW(MRIaggr.Pat1_red, range  = sqrt(2 * 1.875^2 + 0.001), row.norm = TRUE, upper = NULL,
           spatial_res = c(1.875,1.875,6))$W
max(spam::rowSums(W > 0))

hypoCC_Th1_10 <- calcThreshold(data, param = c("TTP_t0","MTT_t0"), threshold = 1:10,
                    hemisphere = "left", rm.CSF = TRUE,
                    GRalgo = TRUE, seed = c("MASK_T2_FLAIR_t2","MASK_DWI_t0"), W = W)

multiplot(selectCoords(MRIaggr.Pat1_red), main = "TTP_t0_thCC  - slice",
        hypoCC_Th1_10[,1], legend = FALSE, palette = rainbow(10), breaks = (0:10) - 10^{-10})


## End(Not run)
```

`calcTissueType` *Probabilistic tissue type segmentation*

Description

Perform a probabilistic segmentation of the voxel in Cerebro Spinal Fluid, White Matter and Grey Matter classes.

Usage

```
## S4 method for signature 'MRIaggr'
calcTissueType(object, param, niter = 100, nnei = 6,
       beta = if(sub == TRUE){0.3}else{0.7}, sub = TRUE, digit = 0, verbose = TRUE,
       name_newparam = c("CSF", "GM", "WM"), update.object = FALSE, overwrite = FALSE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`param`	the contrast parameter that should be used to distinguish the WM, the GM and the CSF. *character*. REQUIRED.
`niter`	the number of iterations used by `mritc.bayes`. *positive integer*.
`nnei`	the number of neighbors. *positive integer*.
`beta`	the parameter 'inverse temperature' of the Potts model. *numeric*.
`sub`	if `TRUE`, use the higher resolution model; otherwise, use the whole voxel method. *logical*.
`digit`	the number of decimal places to use for the initialization. *positive integer*.
`verbose`	indicate the level of output as the algorithm runs. *logical*.
`name_newparam`	the name of the new paramaters containing the probabilistic segmentation. *character vector of size 3*.
`update.object`	should the resulting tissue types be stored in `object` ? *logical*.
`overwrite`	if tissue types are already stored in `object@data`, can they be overwritten ? *logical*.

Details

This function requires to have installed the *mritc* package to work.

ARGUMENTS:
Information about the `nnei` and `sub` arguments can be found in `makeMRIspatial`.
Information about the `niter` and `beta` arguments can be found in `mritc`.

FUNCTION:
This function uses the `mritc.bayes` function of the *mritc* package to compute the probabilistic segmentation. T1 sequence is the recommanded sequence to identify the various tissue types but T2 gradient echo may also be used.

The initialization function `initOtsu` was found much more slower when the contrast parameter values have a large number of digit. The `digit` argument enable to round the contrast values ONLY for the computation of the initialization values.

Value

An *list* containing :

- `[[prob]]` : the group membership of each voxel for each class. *matrix*.
- `[[mu]]` : the mean value of each class. *numeric vector*.
- `[[sigma]]` : the standard deviation of each class. *numeric vector*.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## Not run:
## perform segmentation  (call mritc)
calcTissueType(MRIaggr.Pat1_red, param = "T1_t0", update.object = TRUE, overwrite = TRUE)

## display
multiplot(MRIaggr.Pat1_red, num = 1,
             param = c("CSF","WM","GM"), legend = FALSE,
             palette = "rgb")

## End(Not run)
```

---

calcW *Compute the neighbourhood matrix*

Description

Computate a neighbourhood matrix using spatial coordinates.

Usage

```
## S4 method for signature 'data.frame'
calcW(object, range, method = "euclidean", upper = NULL,
        format = "dgCMatrix", row.norm = FALSE, spatial_res = rep(1,ncol(object)),
  calcBlockW = FALSE)

## S4 method for signature 'MRIaggr'
calcW(object, range, spatial_res = c(1,1,1), num = NULL,
     hemisphere = "both", subset = NULL, upper = TRUE, format = "dgCMatrix",
 row.norm = FALSE, calcBlockW = FALSE,
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

object	a *data.frame* containing the coordinates of the observations or an object of class MRIaggr. REQUIRED.
range	only distances smaller than range are stored in W. postive numeric. REQUIRED.
method	the distance measure to be used. *character*. This must be one of "euclidean", "maximum", "minkowski" or "greatcircle".
spatial_res	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
num	the slices to use. *numeric vector* or NULL.
hemisphere	the hemisphere to use. *character*.
subset	the subset of observations to use. *positive integer vector* or NULL leading to use all observations.
upper	should the entire matrix (NULL) or only the upper-triagonal (TRUE) or only the lower-triagonal (FALSE) values be calculated ?
format	the format of the output. Could be "spam" or "dgCMatrix".
row.norm	should the resulting matrix be row-normalized ? TRUE/FALSE.
calcBlockW	should the partition into disjoint spatial blocks of sites be computed ? *logical*.
verbose	should the execution of the function be traced ? *logical*.
update.object	should the resulting neighbourhood matrix be stored in object ? *logical*.
overwrite	if a neighbourhood matrix is already stored in object@ls_descStats, can it be overwritten ? *logical*.

Details

These functions require to have installed the *Matrix* and the *spam* package to work.

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

The range argument cooresponds to the delta argument of the nearest.dist function from the *spam* package.

Information about the hemisphere argument can be found in the details section of selectContrast.

The row.norm argument is ignored if format is set to "spam".

FUNCTION:
This function relies on the nearest.dist function of the *spam* package.

Each of the num, hemisphere and subset argument define a subset of the total set of observations. It is the intersection of all these three subsets that is extracted.

Value

Invisible. A list containing :

- [[W]] : a *spam* or *dgCMatrix* object.
- [[block]] : the output of the calcBlockW function if calcBlockW is set to TRUE, NULL otherwise.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

#### 1- data.frame method ####
coords <- selectCoords(MRIaggr.Pat1_red, num = 1:3, hemisphere = "lesion")

## full W
W <- calcW(object = coords, range = sqrt(2))$W
W[1:10,1:10]
table(spam::rowSums(W))

## full W normalized by row
W <- calcW(object = coords, range = sqrt(2), row.norm = TRUE)$W
W[1:10,1:10]
table(spam::rowSums(W))

## upper W
W <- calcW(object = coords, range = sqrt(2), upper = TRUE)$W
W[1:10,1:10]

#### 2- MRIaggr method ####

## compute W (regular lattice)
W <- calcW(MRIaggr.Pat1_red, range = sqrt(2), upper = NULL, num = 1:3, hemisphere = "lesion")$W
table(spam::rowSums(W > 0))

## compute W (irregular lattice)
W <- calcW(MRIaggr.Pat1_red, range = sqrt(2 * 1.875^2), upper = NULL, num = 1:3,
         hemisphere = "lesion", spatial_res=c(1.875, 1.875, 6))$W
table(spam::rowSums(W > 0))
```

Carto3D-class *class "Carto3D"*

Description

Patient-specific storage of univariate contrast data.

Arguments

identifier	the patient identifier. *character*.
parameter	the name of the contrast parameter. *character*.
contrast	the contrast and the spatial coordinates of each voxel. *data.frame* with four columns named "i" "j" "k" and the name of the contrast parameter.
fieldDim	the dimension of the lattice containing the observations expressed in number of voxels. *data.frame*.
voxelDim	the voxel size with its unit of measure. A four columns *data.frame* with names "i", "j", "k" and "unit".
default_value	the reference values of the contrast parameters (e.g. the background value). *character*.

S4 methods

Getters :

selectContrast return the contrast parameters.

selectCoords return the coordinates.

selectIdentifier return the identifier of the patient

selectN return the number of voxels

selectParameter return the names of the parameters stored in the object

selectFieldDim return the dimension of the lattice containing the observations

selectVoxelDim return the dimension of a voxel

Displayers :

multiplot display the contrast data.

Constructors :

constMRIaggr aggregate several Carto3D objects into a single objet MRIaggr.

Initializers :

initNum check the validity of the num argument.

See Also

MRIaggr class that aggregates several carto3D objects. Carto3D2MRIaggr to convert carto3D objects into MRIaggr objects.

Carto3D2MRIaggr *Carto3D to MRIaggr converter*

Description

Construct a MRIaggr object by aggregating Carto3D objects.

Usage

```
Carto3D2MRIaggr(ls.Carto3D, rm.Carto3D = FALSE, tol = 10^{-10},
         num = NULL, verbose = optionsMRIaggr("verbose"))
```

Arguments

ls.Carto3D	a list of Carto3D objects. REQUIRED.
rm.Carto3D	should the object on which the ls.Carto3D argument points be removed form the global environment ? *logical*.
tol	numeric precision for the consistency check. *positive numeric*.
num	the slices to extract. *numeric vector* or NULL.
verbose	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Value

a MRIaggr object.

Examples

```
## load NIFTI files
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")

Pat1.TTP.t0.nifti <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Pat1.DWI.t0.nifti <- readMRI(file.path(path.Pat1, "DWI_t0"), format = "nifti")
Pat1.MASK_DWI.t0.nifti <- readMRI(file.path(path.Pat1, "MASK_DWI_t0"), format = "nifti")
Pat1.MASK_T2_FLAIR.t2.nifti <- readMRI(file.path(path.Pat1, "MASK_T2_FLAIR_t2"),
                                       format = "nifti")

## convert them to Carto3D
Pat1.TTP.t0.Carto3D <- constCarto3D(Pat1.TTP.t0.nifti,
             identifier = "Pat1", param = "TTP_t0", default_value = NA)
Pat1.DWI.t0.Carto3D <- constCarto3D(Pat1.DWI.t0.nifti,
             identifier = "Pat1", param = "DWI_t0", default_value = NA)
Pat1.MASK_DWI.t0.Carto3D <- constCarto3D(Pat1.MASK_DWI.t0.nifti,
             identifier = "Pat1", param = "MASK_DWI_t0", default_value = NA)
```

```
Pat1.MASK_T2_FLAIR.t2.Carto3D <- constCarto3D(Pat1.MASK_T2_FLAIR.t2.nifti,
            identifier = "Pat1", param = "MASK_T2_t2", default_value = NA)

## convert Carto3D to MRIaggr
MRIaggr.Pat1 <- Carto3D2MRIaggr(list(Pat1.TTP.t0.Carto3D,
                                     Pat1.DWI.t0.Carto3D,
                                     Pat1.MASK_DWI.t0.Carto3D,
                                     Pat1.MASK_T2_FLAIR.t2.Carto3D)
)
```

`constCarto3D` *Array constructor for Carto3D objects*

Description

Creates a `Carto3D` object from an array.

Usage

```
constCarto3D(array, identifier, param, default_value = NULL,
         pos_default_value = c(1,1,1), voxelDim = NULL, rm.array = FALSE)
```

Arguments

`array`	the values of the contrast parameter. `array`. REQUIRED.
`identifier`	the identifier of the patient from which the data originated. *character*. REQUIRED.
`param`	the contrast parameter. *character*. REQUIRED.
`default_value`	the reference values of the contrast parameters (e.g. the background value). *character* or `NULL` leading to search the reference value in `array[pos_default_value]`.
`pos_default_value`	the coordinates of the observation that contains the reference value. *numeric vector* with length the number of dimension of array.
`voxelDim`	the voxel size with its unit of measure. A four columns *data.frame* with names `"i"`, `"j"`, `"k"` and `"unit"`.
`rm.array`	should the object on which *array* argument points be removed form the global environment ? *logical*.

Details

ARGUMENTS:
`pos_default_value` is active only if `default_value` is set to `NULL`.

Value

An object of class `Carto3D`.

Examples

```
## load NIFTI files
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")

## convert them to Carto3D
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")
class(Carto3D.Pat1_TTP_t0)
```

`constCompressMRIaggr` *Compress a MRIaggr object*

Description

Generate from an existing `MRIaggr` object a new one with lower spatial resolution.

Usage

```
## S4 method for signature 'MRIaggr'
constCompressMRIaggr(object, compression.factor, param = NULL,
         mask = NULL, threshold = 0.49, verbose = optionsMRIaggr("verbose"))
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`compression.factor`	the compression factor. *postive integer*. REQUIRED.
`param`	the contrast parameters to load in the new MRIaggr object. *character vector* or `NULL`.
`mask`	the binary contrast parameter(s). *character vector*.
`threshold`	the value above which the local mean of the binary parameters is assigned to 1 (and otherwise to 0). *numeric between 0 and 1*.
`verbose`	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
Information about the `param` argument can be found in the details section of `initParameter`.

FUNCTION:
The fonction uses a local mean to compress the initial parameters maps to a lower resolution.

Value

a `MRIaggr` object.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compress the MRIaggr object
MRIaggr.compressed <- constCompressMRIaggr(MRIaggr.Pat1_red, compression.factor = 2,
                                    param = c("DWI_t0","T2_FLAIR_t2","MASK_T2_FLAIR_t2"),
                                            mask = "MASK_T2_FLAIR_t2")

## display
par(mfrow = c(2,4), mar = rep(1.75,4), mgp = c(2,0.75,0))
multiplot(MRIaggr.Pat1_red, param = "DWI_t0", window = NULL, breaks = seq(0,350,1),
             midplane = TRUE, main = "before - slice ")
multiplot(MRIaggr.compressed, param = "DWI_t0", window = NULL, breaks = seq(0,350,1),
             midplane = TRUE, main = "after - slice ")

multiplot(MRIaggr.Pat1_red, param = "MASK_T2_FLAIR_t2", main = "before - slice ")
multiplot(MRIaggr.compressed, param = "MASK_T2_FLAIR_t2", main = "after - slice ")
```

`constLatex` *Constructor for Latex report*

Description

Construct a Latex report that includes plots generated by the `multiplot` function.

Usage

```
constLatex(directory, filename = NULL, identifier = NULL, param = NULL, tabular = NULL,
         extra_text = NULL, subsection = NULL, label = NULL,
         width = 0.9, trim = c(0,0,0,0), plotPerPage = 3,
         width.legend = 0.35, trim.legend=  c(0,0,0,0),
         title = "", date="", author = "", verbose = TRUE)
```

Arguments

`directory`	the path to the root directory. This directory should contains subsdirectories themself containing the image files. *character*.
`filename`	the name of the latex file that should be generated. *character* or *NULL*.
`identifier`	the identifiers of the patients for which the graphics should be displayed. *character vector* or `NULL` leading to use all patients.
`param`	the names of subdirectories containing the images. *character vector* or `NULL` leading to use all subdirectories.
`tabular`	a list of data.frame to display in the table format. *list of data.frame* or `NULL` if there is no table to display.
`extra_text`	additionnal text to display. *list of character vector* or `NULL` if there is no extra text to display.

subsection	the names of subsections for the latex document. *character vector* or NULL leading to use param for naming the subsections.
label	the legend that will be displayed under each figure of a given parameter. *character vector* or NULL leading to use param for the legend.
verbose	should the execution of the function be traced ? *logical*.
width	the width of each image relative to the linewidth. *positive numeric*.
trim	the length in mm with which the imported images will be cropped (left, bottom, right top). *numeric vector of size 4*.
plotPerPage	the number of image that should be displayed on the same page. *strictly positive interger*.
width.legend	the width of the legend image relative to the linewidth. *numeric between 0 and 1*.
trim.legend	the length in mm with which the legend of the images will be cropped. *numeric vector of size 4*.
title	the title of the latex document. *character*.
date	the date on the latex document. *character*.
author	the author of the latex document. *character*.

Details

ARGUMENTS:
tabular must contains in its first column the patient identifiers. This column will not be display.

FUNCTION:
Patient identifiers are read from the file names : it is the character string preceding the first underscore ("_").

The function generate latex code that reads the images using the includegraphicx latex function. In particular arguments width, trim, width.legend and trim.legend are used by this function to adjust the display of the images.

The latex files require the following (latex) packages to compile: inputenc, amssymb, amsmath, titlesec, geometry, enumitem, graphicx, color, space, hyperref and caption. If filename is set to NULL the latex file will not be created.

Value

A *list* containing :

- [[text.preamble]] : a *character* containing the preamble of a latex document.
- [[text.begin]] : a *character* for beginning the latex report.
- [[ls.text]] : a *list of character*, each containing the content of the report for each patient.
- [[text.end]] : a *character* for ending the latex report.

Examples

```
## Not run:
## generate MRIaggr objects
data("MRIaggr.Pat1_red", package = "MRIaggr")

path.Pat2 <- system.file("anlz", package = "oro.nifti")
analyse.avg <- readMRI(file.path(path.Pat2, "avg152T1"), format = "analyze")
# analyse.avg@descrip
MRIaggr.Pat2 <- constMRIaggr(analyse.avg, param = "T1_t0", identifier = "Pat2")

path.Pat3 <- system.file("nifti", package = "oro.nifti")
nifti.ffd <- readMRI(file.path(path.Pat3, "filtered_func_data"), format = "nifti")
# nifti.ffd@descrip
MRIaggr.Pat3 <- constMRIaggr(lapply(1:5, function(x){nifti.ffd[,,,x, drop = FALSE]}),
                             param=paste("T2_GRE_t", 0:4, sep = ""),identifier = "Pat3")

## create directories
path <- "."
dir.create(file.path(path, "Display"))
dir.create(file.path(path, "Display/T1_t0"))
dir.create(file.path(path, "Display/T2_GRE_t0"))
dir.create(file.path(path, "Display/MASK_DWI_t0"))

## export images
multiplot(MRIaggr.Pat1_red, param = "T1_t0",
          mfrow = c(2,4), axes = FALSE, mar = c(0,0,1.5,0),
          window = "png", path = "Display/T1_t0/")
multiplot(MRIaggr.Pat2, param = "T1_t0",
          mfrow = c(3,6), axes = FALSE, mar = c(0,0,1.5,0), legend = FALSE,
          window = "png", path = "Display/T1_t0/")

multiplot(MRIaggr.Pat1_red, param = "T2_GRE_t0",
          mfrow = c(2,4), axes = FALSE, mar = c(0,0,1.5,0),
          window = "png", path = "Display/T2_GRE_t0/")
multiplot(MRIaggr.Pat3, param = "T2_GRE_t0",
          mfrow = c(2,4), axes = FALSE, mar = c(0,0,1.5,0),
          window = "png", path = "Display/T2_GRE_t0/")

multiplot(MRIaggr.Pat1_red, param = "MASK_DWI_t0",
          mfrow = c(2,4), axes = FALSE, mar = c(0,0,1.5,0),
          window = "png", path = "Display/MASK_DWI_t0/")

## create latex file (default)
res <- constLatex(directory = "Display/", filename = "sweaveDisplay")

## create latex file (personalized)
res <- constLatex(directory = "Display/", filename = "sweaveDisplay",
                  param = c("T1_t0","T2_GRE_t0","MASK_DWI_t0"),
                  tabular = list(data.frame(cbind(Id = c("Pat1","Pat2","Pat3"),
                                                  Age = c(28,54,32),
                                                  Gender = c("Male","Female","Male")))
                                 ),
```

```
                    label = c("T1 sequence","T2 gradient echo sequence","lesion mask"),
                    extra_text = list("Patient 1 suffers from stroke diseases ...",
                                      "Patient 2 has ...",
                                      "Patient 3 has ...")
)

## End(Not run)
```

---

constMRIaggr *Array constructor for MRIaggr object*

Description

Construct a MRIaggr object from a list of array, each array corresponding to a different contrast parameters.

Usage

```
constMRIaggr(ls.array, identifier, param, default_value = NULL,
     pos_default_value = c(1,1,1), tol = 10^{-10},
 voxelDim = NULL, verbose = optionsMRIaggr("verbose"), rm.ls.array = FALSE)
```

Arguments

ls.array	the value of the contrast parameter(s) for each observation. list of array. REQUIRED.
identifier	the identifier of the patient to which belong the contrast parameters. *character*. REQUIRED.
param	the contrast parameter(s). *character vector* or NULL. REQUIRED.
default_value	the reference values of the contrast parameters (e.g. the background value). *character* or NULL leading to search the reference value in array[pos_default_value].
pos_default_value	the coordinates of the observations that contains the reference value. *numeric vector*.
tol	numeric precision for the consistency check. *positive numeric*.
voxelDim	the voxel size with its unit of measure. A four columns *data.frame* with names "i", "j", "k" and "unit".
verbose	should the execution of the function be traced ? *logical*.
rm.ls.array	should the object on which the ls.array argument points be removed form the global environment ? *logical*.

Details

ARGUMENTS:
All the array in ls.array in must have the same dimensions.

Information about the param argument can be found in the details section of initParameter.

pos_default_value is active only if default_value is set to NULL.

Value

a MRIaggr object.

Examples

```
#### 1- 1st method ####
## load NIFTI files
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")

nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
nifti.Pat1_DWI_t0 <- readMRI(file.path(path.Pat1, "DWI_t0"), format = "nifti")
nifti.Pat1_MASK_DWI_t0 <- readMRI(file.path(path.Pat1, "MASK_DWI_t0"), format = "nifti")
nifti.Pat1_MASK_T2_FLAIR_t2 <- readMRI(file.path(path.Pat1, "MASK_T2_FLAIR_t2"),
                                       format = "nifti")

## convert them to MRIaggr
MRIaggr.Pat1 <- constMRIaggr(list(nifti.Pat1_TTP_t0, nifti.Pat1_DWI_t0,
                                  nifti.Pat1_MASK_DWI_t0, nifti.Pat1_MASK_T2_FLAIR_t2),
            identifier= "Pat1", param=c("TTP_t0","DWI_t0","MASK_DWI_t0","MASK_T2_FLAIR_t2"))


#### 2- 2nd method ####
## load nifti files
param <- c("DWI_t0.nii","MASK_DWI_t0.nii","MTT_t0.nii","TTP_t0.nii","T1_t0.nii","T2_GRE_t0.nii",
           "MTT_t1.nii","TTP_t1.nii","T2_FLAIR_t2.nii","MASK_T2_FLAIR_t2.nii")

ls.array <- list()
for(iter_param in 1:length(param)){
  ls.array[[iter_param]] <- readMRI(file.path(path.Pat1,param[iter_param]), format = "nifti")
}

## convert them to MRIaggr
param <- gsub(".nii", "", param)

MRIaggr.Pat1 <- constMRIaggr(ls.array, identifier = "Pat1", param = param)

#### additionnal examples
## Not run:
## load an analyse file (example of oro.nifti::readANALYZE)
path.Pat2 <- system.file("anlz", package = "oro.nifti")
analyse.avg <- readMRI(file.path(path.Pat2, "avg152T1"), format = "analyze")
MRIaggr.Pat2 <- constMRIaggr(analyse.avg, param = "avg", identifier = "Pat2")

# display
multiplot(MRIaggr.Pat2, param = "avg",
          mfrow = c(4,6), axes = FALSE, main = "",
          mar = c(0,0,0.75,0), mar.legend = c(0,0,1,0))

### load a nifti file (example of oro.nifti::readNIfTI)
path.Pat3 <- system.file("nifti", package = "oro.nifti")
nifti.ffd <- readMRI(file.path(path.Pat3, "filtered_func_data"), format = "nifti")
MRIaggr.Pat3 <- constMRIaggr(lapply(1:dim(nifti.ffd)[4],
```

```
                                    function(x){nifti.ffd[,,,x, drop = FALSE]}),
              param=paste("ffd", 1:dim(nifti.ffd)[4], sep = "_"), identifier = "Pat3")

# display
multiplot(MRIaggr.Pat3, param = "ffd_1")

## load a dicom file (examples of oro.dicom::readDICOMFile)
path.Pat4 <- system.file("dcm", package = "oro.dicom")
dicom.Abdo <- readMRI(file.path(path.Pat4, "Abdo.dcm"), format = "dicom")
path.Pat4 <- constMRIaggr(dicom.Abdo, param = "Abdo",identifier = "Pat4")

# display
multiplot(path.Pat4, param = "Abdo")
multiplot(path.Pat4, xlim = c(100, 200), param = "Abdo")


## End(Not run)
```

`constReduceMRIaggr` *Reduce a MRIaggr*

Description

Construct a `MRIaggr` object restricted to a subset of observations.

Usage

```
## S4 method for signature 'MRIaggr'
constReduceMRIaggr(object, mask, numeric2logical = FALSE, keep.index = TRUE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`mask`	the binary contrast parameter or a vector indicating the observations to be kept. *character* or *logical vector* with length equal to the number of observations in `object`. REQUIRED.
`numeric2logical`	should `mask` be converted to logical ? *logical*.
`keep.index`	should the previous `index` parameter be saved in the `ls_descStats` slot ? *logical*.

Value

a `MRIaggr` object.

See Also

`calcBrainMask` to compute an indicator of the brain observations.

Examples

```
## load NIFTI files and convert them to MRIaggr
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
ls.array <- list(readMRI(file.path(path.Pat1, "T2_GRE_t0"), format = "nifti"))
MRIaggr.Pat1 <- constMRIaggr(ls.array, identifier = "Pat1", param = "T2_GRE_t0")

## create the cerebral mask

## Not run:
res <- calcBrainMask(MRIaggr.Pat1, param = "T2_GRE_t0", type = "kmeans",
                     kmeans.n_groups = 2:4,
                     update.object = TRUE, overwrite = TRUE)

res <- calcSmoothMask(MRIaggr.Pat1, update.object = TRUE, overwrite = TRUE)

## End(Not run)
## display
multiplot(MRIaggr.Pat1, param = "mask", legend = FALSE)

## construct the reduced object
MRIaggr.Pat1_red <- constReduceMRIaggr(MRIaggr.Pat1, mask = "mask")

## display
par(mfrow = c(2,4), mar = rep(1.75, 4), mgp = c(2,0.75,0))
multiplot(MRIaggr.Pat1, param = "T2_GRE_t0",
             window = NULL,breaks = seq(0, 300, 1), legend = FALSE)
multiplot(MRIaggr.Pat1_red, param = "T2_GRE_t0",
             window = NULL, breaks = seq(0, 300, 1), legend = FALSE)
```

---

df2array *data.frame to array converter*

Description

Convert observations stored in the data.frame format into the array format.

Usage

```
df2array(contrast, coords, format = "any", default_value = NA, range.coords = NULL)
```

Arguments

contrast	the dataset containing the observations in rows and the contrast parameters in columns. *vector* or *data.frame*. REQUIRED.
coords	the spatial coordinates of the observations. *matrix* with a number of rows equal to the number of rows of data. REQUIRED.
format	the format of the output. Can be "any","matrix","data.frame" or "list".
default_value	the element used to fill the missing observations. *numeric*.

`range.coords` the maximum coordinate in each dimension to be considered. *numeric vector* with length equal to the number of columns of `coords`.

Details

FUNCTION:
If `contrast` contains several parameters, they are treated one at a time and the result is returned in the form of a list. If `range.coords` is `NULL` then the maxima coordinates are those of the `coords` argument. If only one parameter is specified and the format is set to `"any"` then a *vector* is returned.

Value

a list containing :

- `[[contrast]]` : *list* containing the new contrast in the new format.
- `[[coords]]` : a *data.frame* containing the coordinates of each observation.
- `[[unique_coords.group]]` : two *list* containing the possibles coordinates in each dimension.

Examples

```
#### 1- with simulated data ####
## simulate
set.seed(10)
n <- 4
Y <- rnorm(n^2)

## conversion
res1 <- df2array(contrast = Y, coords = expand.grid(1:n + 0.5, 1:n + 0.5))
res2 <- df2array(contrast = Y, coords = expand.grid(1:n, 1:n), format = "matrix")
res3 <- df2array(contrast = Y, coords = expand.grid(2 * (1:n), 2 * (1:n)))
res4 <- df2array(contrast=cbind(Y ,Y, Y), coords = expand.grid(2 * (1:n), 2 * (1:n)),
                    range.coords = c(10,10))

## display
par(mfrow = c(2,2), mar = rep(2,4), mgp = c(1.5,0.5,0))
fields::image.plot(unique(res1$coords[,1]), unique(res1$coords[,2]), res1$contrast[[1]],
                   xlab = "", ylab = "")
fields::image.plot(unique(res2$coords[,1]), unique(res2$coords[,2]), res2$contrast,
                   xlab = "", ylab = "")
fields::image.plot(res3$contrast[[1]])
fields::image.plot(res4$contrast[[2]])

#### 2- with MRIaggr data ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")
carto <- selectContrast(MRIaggr.Pat1_red, param = "DWI_t0", format = "vector")
coords <- selectCoords(MRIaggr.Pat1_red)
coords[,1] <- coords[,1] + 30
coords[,2] <- coords[,2] + 15

## converion 1
array.DWI_t0 <- df2array(carto, coords = coords, default_value = 1000)$contrast[[1]]
```

```
# display
fields::image.plot(min(coords[,1]):max(coords[,1]), min(coords[,2]):max(coords[,2]),
                   array.DWI_t0[,,1], xlab = "i", ylab = "j")

## conversion 2
array.DWI_t0 <- df2array(contrast=carto, coords = coords, default_value = 1000,
                         range.coords = c(128,128,3))$contrast[[1]]

# display
fields::image.plot(1:128, 1:128, array.DWI_t0[,,1], xlab = "i", ylab = "k")
```

EDK *Gaussian kernel*

Description

Apply a gaussian kernel to observations. For internal use.

Usage

```
EDK(x, bandwidth, power = 2)
```

Arguments

x	the data on which the kernel will be applied. *numeric* or *numeric vector*.
bandwidth	the bandwidth of the kernel. *numeric*.
power	the power of the kernel. *numeric*.

Details

FUNCTION:
This function corresponds to the following kernel :

$$\frac{1}{\sqrt[power]{2 * pi * bandwidth^2}} * \exp(-(x/bandwidth)^power)$$

Setting power to 2 lead to use a gaussian kernel.

Value

A *numeric*.

filtrage2Dmed_cpp *2D median filtering*

Description

C++ function call by calcFilter that performs two dimensional median filtering. For internal use.

Usage

```
filtrage2Dmed_cpp(M_data, M_operateur, index_data, na_rm)
```

Arguments

M_data	matrix to which the filter will be applied.
M_operateur	the filter to be applied.
index_data	index of the non NA data.
na_rm	should the observations with missing values in their neighbourhood be removed ? Otherwise the ponderation is adjusted.

filtrage2D_cpp *2D filtering*

Description

C++ function call by calcFilter that performs two dimensional filtering. For internal use.

Usage

```
filtrage2D_cpp(M_data, M_operateur, index_data, bilateral, na_rm)
```

Arguments

M_data	matrix to which the filter will be applied.
M_operateur	the filter to be applied.
index_data	index of the non NA data.
bilateral	should the influence of each neighbor be ponderated by the difference in signal with the considered observation ?
na_rm	should the observations with missing values in their neighbourhood be removed ? Otherwise the ponderation is adjusted.

filtrage3Dmed_cpp *3D median filtering*

Description

C++ function call by `calcFilter` that performs three dimensional median filtering. For internal use.

Usage

```
filtrage3Dmed_cpp (Vec_data, p_data, Vec_operateur, p_operateur, index_data, na_rm)
```

Arguments

`Vec_data`	vector of data to which the filter will be applied.
`p_data`	spatial dimensions of the data.
`Vec_operateur`	vector representing the filter to be applied.
`p_operateur`	spatial dimensions of the filter.
`index_data`	index of the non NA data.
`na_rm`	should the observations with missing values in their neighbourhood be removed ? Otherwise the ponderation is adjusted.

filtrage3D_cpp *3D filtering*

Description

C++ function call by `calcFilter` that performs three dimensional filtering. For internal use.

Usage

```
filtrage3D_cpp(Vec_data, p_data, Vec_operateur, p_operateur, index_data,
         bilateral, na_rm)
```

Arguments

`Vec_data`	vector of data to which the filter will be applied.
`p_data`	spatial dimensions of the data.
`Vec_operateur`	vector representing the filter to be applied.
`p_operateur`	spatial dimensions of the filter.
`index_data`	index of the non NA data.
`bilateral`	should the influence of each neighbor be ponderated by the difference in signal with the considered observation ?
`na_rm`	should the observations with missing values in their neighbourhood be removed ? Otherwise the ponderation is adjusted.

GRalgo *Growing Region algorithm*

Description

Perform the Growing Region algorithm proposed by (Revol et al., 1997). For internal use.

Usage

```
GRalgo(contrast, W, seed, sigma_max, range, breaks, step, operator, iter_max,
         keep.lower, keep.upper, history.sigma, history.step, history.front)
```

Arguments

contrast	the contrast value of each observation. *numeric vector*.
W	the neighbourhood matrix. *dgCMatrix*.
seed	the index of the initial seeds or a binary indicator of the initial seeds. *positive integer vector* or *logical vector*.
sigma_max	the maximum admissible value for the variability of the group contrast. *positive numeric*.
range	the range of acceptable contrast values for the growing region group. *numeric vector of size 2*.
breaks	the break points to use to categorize the contrast distribution. *numeric vector*.
step	the step between two consecutive breaks. *numeric*.
iter_max	the maximum number of iterations for the expansion of the growing region. postive integer.
operator	should the median absolute deviation be used to estimte the variability of the group contrast ("mad") or the standard deviation ("sd").
keep.lower	should removing observations with high intensity of the region be forbidden ? *logical*.
keep.upper	should removing observations with low intensity of the region be forbidden ? *logical*.
history.sigma	should the values of sigma be recorded ? *logical*.
history.step	should the number of observations included in the GR set be recorded ? *logical*.
history.front	should the propagation front of the GR set be recorded ? *logical*.

References

Chantal Revol and Michel Jourlin. *A new minimum varance region growing algorithm for image segmentation*. Pattern Recognition Letters, 18(3):249-258,1997.

heatmapMRIaggr *Correlation between contrast parameters*

Description

Display a correlation map of the contrast parameters.

Usage

```
## S4 method for signature 'MRIaggr'
heatmapMRIaggr(object, param, num = NULL,
        rescale = TRUE, method = "pearson", points.values = TRUE, type = "image",
 breaks = NULL, col = cm.colors(256), main = NULL,
         mgp = c(2,0.5,0), mar = c(4,4,1,6), las = 1, cex.axis = 1,
         filename = paste(object@identifier, "heatmapMRIaggr", sep = "_"), ...)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameters used to compute the correlations. *character vector*. REQUIRED.
num	the slices to use. *numeric vector* or NULL.
rescale	should the contrast parameters be scaled ? *logical*.
method	the correlation coefficient which is to be computed. Can be "pearson", "kendall" or "spearman".
points.values	should the correlation values be printed on the plot ? *logical*.
type	the type of plot to display. Any of "image" or "image.plot" or FALSE meaning no plot.
breaks	the break points to use to generate the color intervals. *numeric vector* or NULL leading to automatic breakpoints generation.
col	the colors with which the correlations will be displayed. *character vector*.
main	an overall title for the plot. *character*.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
mar	the number of margin lines to be specified on the four sides of the plot. *positive numeric vector of size 4*.
las	the style of the axis labels. Any of 0, 1, 2 or 3.
cex.axis	the magnification to be used for axis annotation relative to the current setting of cex. *positive numeric*.
filename	the name of the file used to export the plot. *character*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

This function with argument type set to `image.plot` requires to have installed the *field* package to work.

Arguments `...` must correspond to some of the following arguments : `height`, `hemisphere`, `path`, `res`, `unit`, `width`, `window`.

ARGUMENTS:
Information about the `num` argument can be found in the details section of `initNum`.

If `breaks` is not `NULL`, it must be of length `length(col)+1`.

Information about the `mar`, `las` and `mgp` arguments can be found in `par`.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## pearson
heatmapMRIaggr(MRIaggr.Pat1_red, param = c("MASK_T2_FLAIR_t2","DWI_t0","TTP_t0","MTT_t0"),
           las = 1, type = "image", cex = 0.75,
           breaks = seq(-1, 1, length.out = 51),
           col = cm.colors(50))
## spearman
heatmapMRIaggr(MRIaggr.Pat1_red, param = c("MASK_T2_FLAIR_t2","DWI_t0","TTP_t0","MTT_t0"),
           las = 1, type = "image", cex = 0.75, method = "spearman",
           breaks = seq(-1, 1, length.out = 51),
           col = cm.colors(50))


## spearman with legend
  heatmapMRIaggr(MRIaggr.Pat1_red, param = c("MASK_T2_FLAIR_t2","DWI_t0","TTP_t0","MTT_t0"),
           las = 1, type = "image.plot", cex = 0.75, method = "spearman",
           breaks = seq(-1, 1, length.out = 51),
           col = cm.colors(50))
```

---

initCol *Color initialization*

Description

Check and initialize display arguments. For internal use.

Usage

```
initCol(contrast, coords, param = NULL, pch, col, palette,
        breaks, legend, type.breaks, method)
```

Arguments

contrast	the contrast value of each observation. *matrix*.
coords	the spatial coordinates of the observations. *data.frame*.
param	the contrast parameter to display. *character*.
pch	the symbol with which the observations will be displayed. *positive integer*.
col	the color with which the observations will be displayed. *character vector*.
palette	the colors or the palette to use when associating colors to intensities. *character vector* or *character*.
breaks	the break points or the number of breakpoints to use to generate the color intervals. *numeric vector* or *postive integer*.
legend	how should the legend be displayed ? *logical*, NULL or "only".
type.breaks	should the break points be equally space according the range of data values ("range"), centered ("range_center") or correspond to the quantile values ("quantile") ?
method	the name of the function that called the initializer. *character*.

Details

FUNCTION:
breaks and palette are active only if col is NULL.

---

initConstLatex *Initializateurs for the constLatex function*

Description

Initialize arguments of the constLatex function. For internal use.

Usage

```
initDirPat_constLatex(directory, param, identifier, verbose)

initSection_constLatex(directory.plot, names_dirs, param, subsection, label)

initPlot_constLatex(directory, names_dirs, directory.plot, identifier,
         tabular, plotPerPage)
```

Arguments

directory	the path to the root directory. This directory should contains subsdirectories themself containing the image files. *character*.
directory.plot	the paths to the subdirectory(ies) containing the image files. *character* or *character vector*.
names_dirs	the names of the subdirectory(ies). *character* or *character vector*.
identifier	the identifiers of the patients for which the graphics should be displayed. *character vector* or NULL leading to use all patients.
param	the names of directories containing the images. *character vector* or NULL leading to use all directories.
tabular	a list of data.frame to display in the table format. *list of data.frame* or NULL if there is no table to display.
subsection	the names of subsections for the latex document. *character vector* or NULL leading to use param for naming the subsections.
label	the legend that will be displayed under each figure of a given parameter. *character vector* or NULL leading to use param for the legend.
verbose	should the execution of the function be traced ? *logical*.
plotPerPage	the number of image that should be displayed on the same page. *strictly positive interger*.

---

initDisplayWindow *Device management*

Description

Display the device on which the plot will be displayed. For internal use.

Usage

```
initDisplayWindow(window, filename, path, width, height, scale, res,
         mfrow, bg, pty, mar, mgp, n.contrast = 1)
```

Arguments

window	the type of device on which the plot will be displayed. *logical*, NULL or *character*.
filename	the name of the file used to export the plot. *character*.
path	the directory where the plot file will be created. *character*.
width	the width of the device used to export the plot in inches. *postive numeric*.
height	the height of the device used to export the plot. *postive numeric*.
scale	the scaling factor to convert height and height to standard unit. *numeric*.

res	the nominal resolution in ppi which will be recorded in the bitmap file. *positive integer*.
mfrow	the division of the device in plot region. *numeric vector of size 2*.
bg	the color used for the background. *character*.
pty	the type of plot region to be used. Can be "s" or "m".
mar	the number of margin lines to be specified on the four sides of the plot. *positive numeric vector of size 4*.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
n.contrast	the number of contrast parameters. *postive integer*.

Details

ARGUMENTS:
Information about the window, filename, width, height, path, unit and res arguments can be found in the details section of initWindow.

Information about the bg, pty, mar and mgp arguments can be found in par.

---

initFilter *Initialization of a filter*

Description

Return a filter corresponding to the specified name.

Usage

```
initFilter(filter, method)
```

Arguments

filter	the filter to be initialized. *character*. REQUIRED.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS :
filter refers to classical filters that will be constructed by the function:
The first two characters refer to the dimension of the filter : "2D" or "3D".
The third character must be "_".
The fourth character refers to the type of filter among : "M", "G", "S" or "I" :

- "M" : median filter (by default a weight of 1 is attributed to each site)
- "G" : gaussian filter (sites are weight with a gaussian kernel)
- "S" : sobel filter (gradient in the i, j or k direction)

- "I" : influence filter (sites with distance to the central site inferior or equal to sqrt(p) have weight 1, otherwise 0)

The last one or two characters indicates the size p of the filter except for the sobel filter where it indicates the direction of the gradient ("x", "y" or "z")

Value

A *list* were :

- [[filter]] : contains the filter. *matrix of dimension p*p* or *array of dimension p*p*p*.
- [[filter_split]] : contains the decomposition of the filter name. *character vector*.

Examples

```
initFilter("2D_G3", method = "calcFilter")$filter
initFilter("2D_G5", method = "calcFilter")$filter
initFilter("3D_G3", method = "calcFilter")$filter

initFilter("2D_M9", method = "calcFilter")$filter
initFilter("3D_M3", method = "calcFilter")$filter

initFilter("2D_Sx", method = "calcFilter")$filter
initFilter("2D_Sy", method = "calcFilter")$filter
initFilter("3D_Sx", method = "calcFilter")$filter
initFilter("3D_Sz", method = "calcFilter")$filter

initFilter("2D_I3", method = "calcFilter")$filter
initFilter("2D_I5", method = "calcFilter")$filter
initFilter("2D_I7", method = "calcFilter")$filter
initFilter("2D_I9", method = "calcFilter")$filter
initFilter("3D_I5", method = "calcFilter")$filter
```

---

initGR *Growing Region initialization*

Description

Initialize arguments of the Growing Region algorithm. For internal use.

Usage

```
initGR(contrast, W, seed, sigma_max, range, range.seed, breaks, iter_max, sd.robust,
         keep.lower, keep.upper, history.sigma, history.step, history.front,
         rescale, verbose, method = "initGR")
```

Arguments

contrast	the contrast value of each observation. *numeric vector*.
W	the neighbourhood matrix. *dgCMatrix*.
seed	the index of the initial seeds or a binary indicator of the initial seeds. *positive integer vector* or *logical vector*.
sigma_max	the maximum admissible value for the variability of the group contrast. *positive numeric*.
range	the range of acceptable contrast values for the growing region group. *numeric vector of size 2*.
range.seed	the range of acceptable contrast values for the seeds. *numeric vector of size 2*.
breaks	the break points or the number of break points to use to categorize the contrast distribution. *numeric vector* or `postive integer`.
iter_max	the maximum number of iterations for the expansion of the growing region. `postive integer`.
sd.robust	should the median absolute deviation be used to estimte the variability of the group contrast, or the standard deviation ? *logical*.
keep.lower	should removing observations with high intensity of the region be forbidden ? *logical*.
keep.upper	should removing observations with low intensity of the region be forbidden ? *logical*.
history.sigma	should the values of sigma be recorded ? *logical*.
history.step	should the number of observations included in the growing region set be recorded ? *logical*.
history.front	should the propagation front of the GR set be recorded ? *logical*.
rescale	should the contrast be scaled ? *logical*.
verbose	should the execution of the function be traced ? *logical*.
method	the name of the function that called the initializer. *character*.

`initIndex` *Index initialization*

Description

Check and initialize index. For internal use.

Usage

```
initIndex(object, index, num, hemisphere = "both", numeric2logical, indexNum = NULL,
       outline.default, cex.default, pch.default, col.default, filter.default, method)
```

Arguments

object	an object of class MRIaggr or NULL .
index	the coordinates of additionnal points to display. *data.frame* or *list*.
num	the slices to display. *numeric vector* or NULL.
hemisphere	the hemisphere to display. *character*.
numeric2logical	if a parameter is specified for the index arguments, should it be converted to logical ? *logical*.
indexNum	the number associated to the index (for display). *numeric*.
outline.default	the default display for the spatial group (edge or entire group). *logical*.
cex.default	the default expansion factor used to plot the observations. *numeric*.
pch.default	the default label used to plot the observations. *numeric*.
col.default	the default color used to plot the observations. *character vector*.
filter.default	the default filter used to define the neighbourhood. *character*.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the hemisphere argument can be found in the details section of selectContrast.

Possible values for index are:

- NULL : the argument is skiped, no additional points are displayed.
- *character* : the name of a logical parameter contained in the object (if called by an MRIaggr method)
- *data.frame* with 3 columns named "i","j","k" containing the coordinates of the points to display.
- *list* with an element named "data" containing the coordinates of the points to display (see previous point) or the name of a logical parameter (if called by an MRIaggr method).
  It can also contain "cex", "pch" and "col" component to specify the how the points should be displayed.
  To only display the outline of the spatial group instead of the spatial group itself, the list must contains an element named "outline" that have for value TRUE.
  The type of neighbourhood used to determine the outline can be specified with a "filter" element in the list. The default neighbourhood is "2D_N4".

initMask *Initialization of the slice numbers*

Description

Check and initialize the mask argument of a MRIaggr method. For internal use.

Usage

```
## S4 method for signature 'MRIaggr'
initMask(object, mask, test = TRUE, init = TRUE,
         format = "data.frame", arg_name = "mask", long_name = "mask", method)
```

Arguments

object	an object of class Carto3D or MRIaggr.
mask	the binary contrast parameters to check or initialize. *character vector*.
test	should the mask be checked ? *logical*.
init	should the mask be initialized ? *logical*. See the details section.
format	the format of the output. Any of "vector", "matrix" and "data.frame".
arg_name	a short name for the error message. *character*.
long_name	the complete name for the error message. *character*.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS :
Setting init to TRUE leads to apply numeric2logical to each mask.

initNeighborhood *Initialization of a neighbourhood filter*

Description

Return the neighbourhood configuration corresponding to the specified name.

Usage

```
initNeighborhood(Neighborhood, method)
```

Arguments

Neighborhood	the name of neighbourhood configuration. Any of "2D_N4", "2D_N8" "3D_N6" "3D_N10" "3D_N18" "3D_N26". REQUIRED.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS :
Neighborhood refers to classical neighbourhood configurations :
The first two characters refer to the dimension d of the filter : "2D" or "3D".
The third character must be "_".
The fourth character refers to the type of filter and must be "N".
The last one or two characters indicates the number of neighbors (denoted n) in each neighbourhood.

Value

A n*d *matrix* with in line the coordinates of the neighbors relative to the current observation.

Examples

```
# 2D neighbourhood
initNeighborhood("2D_N4", method = "calcFilter") # rock neighbourhood
initNeighborhood("2D_N8", method = "calcFilter") # queen neighbourhood

# 3D neighbourhood
initNeighborhood("3D_N6", method=  "calcFilter")
initNeighborhood("3D_N10" ,method = "calcFilter")
initNeighborhood("3D_N18", method = "calcFilter")
initNeighborhood("3D_N26", method = "calcFilter")
```

---

initNum *Initialization of the slice numbers*

Description

Check and initialize the num argument of a Carto3D or a MRIaggr method. For internal use.

Usage

```
## S4 method for signature 'Carto3D'
initNum(object, num, test = TRUE, init = TRUE, method)

## S4 method for signature 'MRIaggr'
initNum(object, num, test = TRUE, init = TRUE, slice_var = "k", method)
```

Arguments

object	an object of class Carto3D or MRIaggr.
num	the slice numbers to check or initialize. *numeric vector* or NULL. See the details section.
test	should the slice numbers be checked ? *logical*.
init	should the slice numbers be initialized if num equals NULL ? *logical*.

slice_var	the type of slice to extract. "i" for sagittal, "j" for coronal and "k" for transverse. *character*.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS :
Setting num to NULL leads to load all available slices.

---

initParameter *parameter initialization*

Description

Check and initialize the param argument in MRIaggr methods. For internal use.

Usage

```
## S4 method for signature 'MRIaggr'
initParameter(object, param, test = TRUE, init = FALSE,
         accept.coords = TRUE, accept.mask = TRUE, accept.index = TRUE,
         arg_name = "param", long_name = "parameters", method)
```

Arguments

object	an object of class MRIaggr.
param	the contrast parameters to check or initialize. *character vector* or NULL.
test	should the parameters be checked ? *logical*.
init	should the parameters be initialized if param equals NULL ? *logical*.
accept.coords	should coordinates be accepted as parameters ? *logical*.
accept.mask	should mask be accepted as a parameter ? *logical*.
accept.index	should index be accepted as a parameter ? *logical*.
arg_name	a short name for the error message. *character*.
long_name	the complete name for the error message. *character*.
method	the name of the function that called the initializer. *character*.

Details

ARGUMENTS :
Setting param to NULL leads to load all available parameters including coodinates, mask and index if accept.coords, accept.mask or accept.index are respectively set to TRUE.

`initWindow` *Display initialization*

Description

Check and initialize display arguments. For internal use.

Usage

```
initWindow(window, filename, path, width, height, unit, res,
 n.plot, mfrow, xlim, ylim, method)
```

Arguments

`window`	the type of device on which the plot will be displayed. *logical*, `NULL` or *character*.
`filename`	the name of the file used to export the plot. *character*.
`path`	the directory where the plot file will be created. *character*.
`width`	the width of the device used to export the plot. *postive numeric*.
`height`	the height of the device used to export the plot. *postive numeric*.
`unit`	the units in which `height` and `width` are given. *character*.
`res`	the nominal resolution in ppi which will be recorded in the bitmap file. *positive integer*.
`n.plot`	the number of images to display. *integer*.
`mfrow`	the division of the device in plot region. *numeric vector of size 2*.
`xlim`	the x limits of the plot. *numeric vector of size 2*.
`ylim`	the y limits of the plot. *numeric vector of size 2*.
`method`	the name of the function that called the initializer. *character*.

Details

FUNCTION:
The arguments `filename`, `width`, `height`, `path`, `unit` and `res` are only active if `window` is `"eps"`, `"svg"` or `"png"`.

`legendMRI` *Diplay a legend of a contrast map*

Description

Display a legend from break values. For internal use.

Usage

```
legendMRI(breaks, palette, mar, cex, cex.main, main, quantiles, digit)

legendMRI2(param, palette, mar, cex, cex.main)
```

Arguments

`breaks`	the break points to use to generate the color intervals. *numeric vector*.
`palette`	the colors or the palette to use when associating colors to observation intensities. *character vector*.
`param`	the name of the contrast parameters. *character vector*.
`mar`	the number of margin lines to be specified on the four sides of the legend. *positive numeric vector of size 4*.
`cex.main`	the expansion factor for the legend title. *positive numeric*.
`cex`	the expansion factor for the legend labels. *positive numeric*.
`main`	an overall title for the legend. *character*.
`quantiles`	the quantiles values to display on the legend. *numeric vector of size 5* or `NULL`.
`digit`	the number of decimal places to use for the legend label. *positive integer*.

Details

ARGUMENTS :
Information about the `mar` argument can be found in `par`.

Information about the `palette` argument can be found in the details section of `initCol`.

FUNCTIONS :
`legendMRI` is used when there is only one contrast parameter whereas `legendMRI2` is used for multiparametric contrast.

logit *Logistic transform*

Description

Logistic and inverse logistic transform. For internal use.

Usage

```
logit(x)

inv.logit(x)
```

Arguments

x	a *numeric vector*, ranging between 0 and 1 for logit.

Value

A *numeric vector*.

MRIaggr-class *Class "MRIaggr"*

Description

Patient-specific storage of multivariate contrast data and clinical data.

Arguments

identifier	the patient identifier. *character*.
contrast	the value of the contrast parameters for each observation. *data.frame*.
clinic	the clinical data of the patient. *data.frame*.
fieldDim	the dimensions of the spatial field (expressed in number of voxels) containing the voxels. *data.frame*.
voxelDim	the voxel size dimensions its unit of measure. A four columns *data.frame* with names "i", "j", "k" and "unit".
default_value	a reference value for each contrast parameter. *data.frame*.
history	a list of the calc or const methods that have been applied to the MRIaggr object . *data.frame*.
normalization	the normalization values of each contrast parameter. *list*.
hemispheres	the presence or absence of lesion(s) in each hemisphere. *data.frame*.
midplane	the position of the mid-sagittal plane. *data.frame*.

`W` the neighbourhood matrix. *list*.

`table_lesion` the vertical distribution of the lesion volumes. *data.frame*.

`table_reperfusion` the volumic reperfusion data. *data.frame*.

`table_hypoperfusion` the volumic hypoperfusion data. *data.frame*.

`ls_statDesc` a slot to store additional data. *list*.

S4 methods

Getters :

`selectContrast` return the contrast parameters.

`selectCoords` return the coordinates.

`selectClinic` return the clinical data.

`selectDefault_value` return the default values of the contrast parameters.

`selectDescStats` return extra information.

`selectHemispheres` return the position of the lesion in each hemisphere.

`selectHistory` return the call of the methods applied on the object.

`selectIdentifier` return the identifier of the patient

`selectMidplane` return the position of the mid-sagittal plan

`selectN` return the number of voxels

`selectNormalization` return the normalization values of the contrast parameters

`selectParameter` return the names of the parameters stored in the object

`selectTable` return volumic information about the lesion or about the perfusion

`selectFieldDim` return the dimension of the lattice containing the observations

`selectVoxelDim` return the dimension of a voxel

Setters :

`allocContrast<-` allocate a contrast parameter

`allocClinic<-` allocate clinical data

`allocDescStats<-` allocate extra information

`allocHemisphere<-` allocate the position of the lesion in the hemisphere or the position of the mid-sagittal brain

`allocNormalization<-` allocate normalization values

`allocTable<-` allocate volumic information

`supprContrast<-` remove a contrast parameter

`supprDescStats<-` remove extra information

Calculators :

`calcBrainMask`

calcContralateral
calcDistMask
calcDistTissues
calcFilter
calcGroupsMask
calcHemisphere
calcROCthreshold
calcNormalization
calcRegionalContrast
calcSmoothMask
calcTableHypoReperf
calcTableLesion
calcThresholdMRIaggr
calcTissueType
calcW

Displayers :

boxplotMask
heatmapMRIaggr
multiplot
plotDistClass
pointsHemisphere
plotLesion3D
plotTableLesion
outlineMRIaggr
summary,MRIaggr-method

Constructors :

constCompressMRIaggr generate from an existing MRIaggr object a new one with lower spatial resolution

constReduceMRIaggr copy an existing MRIaggr object restricted to a subset of voxels

writeMRIaggr create an image file containing the values of a contrast parameter stored in the MRIaggr object.

Initializers :

initNum initialization of the num argument

initParameter intialization of the param argument

initMask initialization of the mask argument

See Also

constMRIaggr to build a MRIaggr object from a list of array.
readMRI to read image files.

MRIaggr.Pat1_red *Example of processed MRIaggr object*

Description

Example of processed MRIaggr object using the functionalities of the MRIaggr package.

Usage

```
MRIaggr.Pat1_red
```

Format

A MRIaggr object containing in addition to the contrast data :

- the position of the mid-sagittal plan.
- the normalization values for the contrast parameters.
- the spatial groups of lesion.
- the regional and contralateral normalised contrast values.
- the CSF/WM/GM probabilistic membership.
- the neighbourhood matrix.
- the lesion volumes per slice
- the hypoperfusion and reperfusion volumes for various time thresholds.

Source

I-know study : www.i-know-stroke.eu

multiplot *Slice by slice display*

Description

Make a slice by slice display of a *data.frame*, a Carto3D or a MRIaggr object.

Usage

```
## S4 method for signature 'data.frame'
multiplot(object, contrast = NULL, num = NULL,
         index1 = NULL, index2 = NULL, index3 = NULL,
       col = NULL, pch = NULL ,xlim = NULL, ylim = NULL, filename = "multiplot",...)

## S4 method for signature 'Carto3D'
multiplot(object, num = NULL,
         col=NULL, pch = NULL, xlim = NULL, ylim = NULL,
         filename = "multiplot",...)

## S4 method for signature 'MRIaggr'
multiplot(object, param, num = NULL,
         index1 = NULL, index2 = NULL, index3 = NULL,
         midplane = FALSE, col = NULL, pch = NULL, xlim = NULL, ylim = NULL,
 filename = "multiplot", ...)
```

Arguments

object	an object of class MRIaggr, or a Carto3D, or a 2 or 3 column data.frame containing the coordinates of the observations in columns. REQUIRED.
param	the contrast parameter to display. *character*. REQUIRED.
contrast	the intensities to display. *numerical vector* or NULL leading to use the same color for all observations.
num	the slices to display. *numeric vector* or NULL.
index1,index2,index3	the coordinates of additionnal points to display. *data.frame* or *list* or NULL.
midplane	should the mid-sagittal plan be displayed ? *logical*.
col	the color with which the observations will be displayed. *character vector* or NULL leading to determine the colors using the palette and breaks arguments.
pch	the symbol with which the observations will be displayed. *positive integer* or NULL leading to use the image function instead of plot.
xlim	the x limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the x limits.
ylim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the y limits.
filename	the name of the file used to export the plot. *character*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum. Information

about the index1, index2 and index3 arguments can be found in the details section of initIndex (argument index).

Information about the ... can be found in the documentation of the optionsMRIaggr function.

Value

Invisible, return a list containing :

breaks.plot	the breaks used to display the images.
palette.plot	the palette used to display the images.
breaks.legend	the breaks used to display the legend. *character*.
palette.legend	the palette used to display the legend. *character*.
quantiles.legend	the quantiles used to display the legend. *character*.

See Also

plotLesion3D for a 3D plot of the lesion.
slices3d of the *misc3d* package for a more interactive 3D plot.

Examples

```
#### 1- data.frame ####
## simulate
n <- 10
Y <- rnorm(n^2)

## display
multiplot(object = data.frame(expand.grid(1:n,1:n),1),
             contrast = Y, window = FALSE)

## load a MRIaggr object
data(MRIaggr.Pat1_red, package = "MRIaggr")

## select data
data <- selectContrast(MRIaggr.Pat1_red,
         param = c("DWI_t0","TTP_t0","MTT_t0","MASK_T2_FLAIR_t2"),
         hemisphere = "lesion", coords = TRUE)

## fit model
glm.1 <- glm(MASK_T2_FLAIR_t2 ~ DWI_t0 + TTP_t0 + MTT_t0, data = data,
         family = binomial(link = "logit"))

## display fitted values
multiplot(object = data[,c("i","j","k")],
             contrast = predict(glm.1, type = "response"), window = FALSE)

## display residuals
multiplot(object = data[,c("i","j","k")], num = 3,
             contrast = predict(glm.1, type = "response"), window = FALSE,
          index1 = list(coords = data[data$MASK_T2_FLAIR_t2,c("i","j","k")], outline = TRUE)
```

```
            )

#### 2- carto3D ####
## load NIFTI files and convert them to carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")

## display
multiplot(Carto3D.Pat1_TTP_t0)
## Not run:
multiplot(Carto3D.Pat1_TTP_t0, num = 1:2)
multiplot(Carto3D.Pat1_TTP_t0, num = 1:2, axes = FALSE)
multiplot(Carto3D.Pat1_TTP_t0, num = 1:2, axes = FALSE, legend = FALSE)
multiplot(Carto3D.Pat1_TTP_t0, num = 1:2, axes = FALSE, legend = FALSE,
         main = "", num.main = FALSE)
multiplot(Carto3D.Pat1_TTP_t0, num = 1:2, axes = FALSE, main = "", num.main = FALSE,
 palette = "gray.colors", breaks = seq(0,100))

## End(Not run)

#### 3- MRIaggr ####
## load a MRIaggr object
data(MRIaggr.Pat1_red, package = "MRIaggr")

# display 3 slices
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3)

## Not run:
# display 3 slices with no axes and white background
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3, axes = FALSE, bg = "white")

# remove the legend
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3, legend = FALSE)

## End(Not run)

## display an set of points
# using a binary parameter stored in the object
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3, index1 = list(coords = "MASK_DWI_t0")
)

## Not run:
# customize the display of the points
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3, index1 = list(coords = "MASK_DWI_t0", col = "pink", pch = 14)
)

# display only the edges of the set
```

```
multiplot(MRIaggr.Pat1_red, param = "DWI_t0", num = 3, legend = FALSE,
             index1 = list(coords = "MASK_DWI_t0", outline = TRUE)
)


# specify the index of points using coordinates
coordsIndex <- data.frame(i = c(40,60), j = c(80,100), k = c(3,3))

multiplot(MRIaggr.Pat1_red, param = "DWI_t0", num = 3, legend = FALSE,
             index2 = list(coords = coordsIndex, col = "black", pch = 15, cex = 4)
             )

# various possibilities for the display
multiplot(MRIaggr.Pat1_red, num = 1:3, param = "DWI_t0",
             legend = FALSE, window = FALSE)
multiplot(MRIaggr.Pat1_red, num = 1:3, param = "DWI_t0",
             legend = TRUE, window = FALSE)
multiplot(MRIaggr.Pat1_red, num = 1:3, param = "DWI_t0",
             legend = NULL, window = FALSE)
multiplot(MRIaggr.Pat1_red, num = 1:3, param = "DWI_t0",
             legend = "only", window = FALSE)

## End(Not run)
```

`optionsMRIaggr` *Set or Query Default Values for MRIaggr*

Description

`optionsMRIaggr` works in the same way as `options` : it can be used to set or query the default values used by the functions of the MRIaggr package.

Usage

```
optionsMRIaggr(..., reinit.options = FALSE)
```

Arguments

`...`	arguments in tag = value form, or a list of tagged values. *character* or *list*.
`reinit.options`	should the default values be reinitialized. *logical*

Value

If not argument is provided, a list containing all the graphical parameters is returned.
If one argument is provided with no value, the value of the corresponding graphical parameter.
If several arguments are provided, a list containing the corresponding graphical parameters is returned.
If one argument is provided with a value, nothing is returned.

optionsMRIaggr elements

The individual optionsMRIaggr elements are:

asp	the aspect ratio y/x. *numeric*.
axes	should the axes be plotted ? *logical*.
bg	the color used for the background. *character*.
breaks	the break points or the number of breakpoints to use to generate the color intervals . *numeric vector* or *pos*
cex	the expansion factor used to plot the observations. *positive numeric*.
cex.index	the expansion factor used to plot each additionnal points to display. *integer vector of size 3*.
cex.legend	the expansion factor of the legend. *positive numeric*.
cex.main	the expansion factor for the main title. *numeric*.
checkArguments	should the validity of thes argments be checked ? *logical*.
col.index	the color used to plot each additionnal points to display. *integer vector of size 3*.
col.midplane	the color used to display the midsagittal plan. *character*.
col.NA	the color to use to plot the NAs. *character*.
digit.legend	the number of decimal places to use when displaying graphics. *integer*.
digit.result	the number of decimal places to use when displaying results. *integer*.
digit.epsilon	the number of decimal places to use when displaying convergence criteria. *integer*.
digit.percentage	the number of decimal places to use when displaying percentage. *integer*.
filter.index	the filter used to define the edge of a spatial group. Any of `"2D_N4"`, `"2D_N8"`, `"3D_N4"`, `"3D_N6"`, `"3D_N8`
height	the height of the device used to export the plot. *postive numeric*.
hemisphere	the hemisphere to display. *character*.
legend	how should the legend be displayed ? *logical*, `NULL` or `"only"`.
main	an overall title for the plot. *character*.
main.legend	a main title for the legend. *character*.
mar	the number of margin lines to be specified on the four sides of the plot. *positive numeric vector of size 4*.
mar.legend	the number of margin lines to be specified on the four sides of the legend. *numeric vector of size 4*.
mfrow	the division of the device in plot region. *numeric vector of size 2* or `NULL` leading automatic adjustment.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
norm_mu	the type of centering to apply on the parameter values. *character*.
norm_sigma	the type of scaling to apply on the parameter values. *character*.
num.main	should the slice number be written over each plot. *logical*.
numeric2logical	if a parameter is specified for the index arguments, should it be converted to logical ? *logical*.
outline.index	should all the additionnal points be displayed or only those on the boundary. *logical*.
palette	the colors or the palette to use when associating colors to observation intensities. *character vector* or *chara*
path	the directory where the plot file will be created. *character*. If `path` is set to `NULL`, the image file is exported
pch.index	the label used to plot each additionnal points to display. *integer vector of size 3*.
pch.NA	the label to use to plot the NAs. *postive integer*.
pty	the type of plot region to be used. Can be `"s"` or `"m"`.
quantiles.legend	should the quantiles of the data be displayed on the legend ? *logical*.
res	the nominal resolution in ppi which will be recorded in the bitmap file. *positive integer*.
slice_var	the type of view to use. `"i"` for sagittal view, `"j"` for coronal view and `"k"` for transverse view. *character*.
type.breaks	should the break points be equally space according the range of data values (`"range"`), centered (`"range_`
unit	the units in which `height` and `width` are given. Can be any of `"px"`, `"in"`, `"cm"` or `"mm"`..
verbose	should the execution of the methods related to MRIaggr objects be traced ? *logical*.
xlab	a title for the x axis. *character*.
ylab	a title for the y axis. *character*.
width	the width of the device used to export the plot. *postive numeric*.

window the type of device on which the plot will be displayed. *logical*, NULL or *character*.

Details

ARGUMENTS:
palette must have be a *charater vector* of length length(breaks)-1 containing colors or a single *character* corresponding to a palette name : "rainbow", "grey.colors", "heat.colors", "terrain.colors", "topo.colors" or "cm.colors".

Possible values for legend are:

- TRUE : the legend is displayed in the current device.
- NULL : the legend is displayed it in a new device.
- FALSE : the legend is not displayed.
- "only" : the legend is displayed but not the slices.

Possible values for windows are:

- NULL : the plot is displayed on the current device with no reshape.
- FALSE : the plot is displayed on the current device with the appropriate reshape.
- TRUE : a new graphical device is open.
- "eps" : the plot is displayed in an image file using postscript.
- "svg" : the plot is displayed in an image file using svg.
- "png" : the plot is displayed in an image file using png.
- "pdf" : the plot is displayed in an image file using pdf.

Arguments filename, width, height, path, unit and res correspond to those of postscript, svg and png.

Arguments mfrow, mar, mar.legend, mgp pty, asp and bg acorrespond to those of par.

See the details section of selectContrast for more details about the hemisphere, norm_mu and norm_mu arguments.

See the initNeighborhood function for details about filter.index possible values.

ARGUMENTS:
The MRIaggr environment store an object called .ls_optionsMRIaggr that contains the list of default values. This object can be modified using the optionsMRIaggr function. The original default values can be restored using optionsMRIaggr(reinit.options=TRUE).

Examples

```
## load a MRIaggr object
data(MRIaggr.Pat1_red, package = "MRIaggr")

# display 3 slices
multiplot(MRIaggr.Pat1_red, param = "DWI_t0",
          num = 1:3, axes = FALSE, bg = "white", legend = FALSE)
```

```
##
# arguments :
# axes = FALSE, bg = "white", legend = FALSE
# can become the default argument of multiplot using :
##

optionsMRIaggr(axes = FALSE, bg = "white", legend = FALSE)

multiplot(MRIaggr.Pat1_red, param = "DWI_t0")

# now axes can be restored using either
multiplot(MRIaggr.Pat1_red, param = "DWI_t0", axes = TRUE)
# or
optionsMRIaggr(axes = TRUE)
multiplot(MRIaggr.Pat1_red, param = "DWI_t0")

## to get default arguments of multiplot
optionsMRIaggr()
optionsMRIaggr()$bg
optionsMRIaggr("bg")

## to restore the original defaut values
optionsMRIaggr(reinit.options = TRUE)
```

`outline` *Outline a region on a slice*

Description

Tool for graphical definition of a spatial region on an image.

Usage

```
outline(n=50,sequential = TRUE, min_dist = 1,
         col = c("blue","red","grey"), pch = 20, cex = c(0.75,1,0.75))
```

Arguments

`n`	maximum number of points to define the outline. *integer*.
`sequential`	should the region edge be updated on the graphical device after each point ? *logical*.
`min_dist`	if the distance between the new point and the initial point is inferior to `min_dist`, then the definition of the region ends. *numeric*. Only active if `sequential` is `TRUE`.
`col`	the colors in which the user-defined edge points, the interpolated edge points and the interior points should be ploted. *character vector of size 3*.
`pch`	the symbol with which the observations will be displayed. *positive integer*.
`cex`	the expansion factor used to plot the edge points, the interpolated edge points and the interior points. *positive numeric vector of size 3*.

Details

FUNCTION:
This function uses the locator function to obtain the coordinates of the cursor. It enable a point by point definition of a region where a linear interpolation is used between user-defined points to define the edge of the region.
In the non sequential mode, the definition of the points stop if the number of points exceed n or using Echap. In the sequential mode, the definition of the points stop if the number of points exceed n or if the new point is close enough to the initial point.

After defining the edge, the region is filled.

Value

A list of two elements :

- [[edge]] : a *data.frame* containing the position ("i" "j") of the edge of the region, the edge number ("edge"), whether its user-specified point or interpolated point ("points") and whether it was removed for the filling precedure ("valid").
- [[surface]] : a *data.frame* containing the position ("i" "j") of the points belonging to the region.

Examples

```
## Not run:
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")
num <- 3

## display 1
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
             num = num, legend = FALSE, window = FALSE)

## outline on display 1
res <- outline(sequential=TRUE,min_dist=3)

## display the results
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
             num = num, legend = FALSE,
             index1 = data.frame(k = num, res$edge[,c("i","j")]),
             index2 = data.frame(k = num, res$surface[,c("i","j")]),
             window = FALSE)

carto <- selectContrast(MRIaggr.Pat1_red, param = c("MASK_T2_FLAIR_t2","index"),
             num = num, coords = TRUE)
carto <- merge(carto, cbind(res$surface, outline = TRUE), all = TRUE)
carto[is.na(carto$outline),"outline"] <- FALSE
head(carto)

## display the results next to MASK_T2_FLAIR_t2
multiplot(MRIaggr.Pat1_red, param = "T2_FLAIR_t2",
             num = num, legend = FALSE,
             index1 = carto[carto$MASK_T2_FLAIR_t2,c("i","j","k")],
```

```
              index2 = carto[carto$outline,c("i","j","k")],
              window = FALSE)

## End(Not run)
```

---

outlineMRIaggr *Outline a region on a slice*

Description

Tool for graphical definition of a spatial region on an image.

Usage

```
## S4 method for signature 'MRIaggr'
outlineMRIaggr(object, param, index1 = NULL, num = NULL, hemisphere = "both",
     numeric2logical = FALSE, xlim = NULL, ylim = NULL, legend = FALSE,
     palette = "terrain.colors", col = NULL, breaks = 25, fill = TRUE, n = 50,
     sequential = TRUE, min_dist = 1, operator_index1 = "none",
     col.outline = c("blue","red","grey"), pch = 20, cex = c(0.75,1,0.75),
 name_newparam = "userMask",
 verbose = optionsMRIaggr("verbose"), update.object = FALSE, overwrite = FALSE)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameter map on which the outline will be drawn. *character*. REQUIRED.
index1	the coordinates of additionnal points to display. *data.frame* or *list* or NULL.
num	the slices on which the outline will be drawn. *numeric vector* or NULL.
hemisphere	the hemisphere to consider. *character*.
numeric2logical	if a parameter is specified for the index argument, should it be converted to logical ? *logical*.
xlim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the x limits.
ylim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the y limits.
legend	how should the legend be displayed ? *logical* or NULL.
palette	the colors or the palette to use when associating colors to observation intensities. *character vector* or *character*.
col	the color to use to plot the observations. *character vector* or NULL leading to automatic generation of the colors using the breaks and palette arguments.
breaks	the break points or the number of breakpoints to use to generate the color intervals . *numeric vector* or *postive integer*.

fill	should the spatial region be filled ? Otherwise only the edge is used. *logical*.
n	maximum number of points to define the outline. *integer*.
sequential	should the region edge be updated on the graphical device after each point ? *logical*.
min_dist	if the distance between the new point and the initial point is inferior to min_dist, then the definition of the region ends. *numeric*. Only active if sequential is TRUE.
operator_index1	the operator to apply between the index1 observations and the outlined observations. Can be "none" "intersection" "difference" or "union".
col.outline	the colors in which the user-defined edge points, the interpolated edge points and the interior points should be ploted. *character vector*[3].
pch	the symbol with which the observations will be displayed. *positive integer*.
cex	the expansion factor used to plot the edge points, the interpolated edge points and the interior points. *positive numeric vector of size 3*.
verbose	should the execution of the function be traced ? *logical*.
name_newparam	the name of the new parameter. *character*.
update.object	should the new parameter be stored in object ? *logical*.
overwrite	if a contrast parameter with the same names is already stored in object@data, can it be overwritten ? *logical*.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the index1 argument can be found in the details section of initIndex (argument index).

Information about the hemisphere argument can be found in the details section of selectContrast.

Information about the palette and legend arguments can be found in the details section of initCol.

FUNCTION:
This function uses the locator function to obtain the coordinates of the cursor. See outline for more details. It will display slice by slice the parameter map and on each slice a region should be drawn.
Press Echap and 1 to restart the draw on the same slice.
Press Echap and 0 to skip a slice.

Value

A matrix containing in columns :

- [,c("i","j","k")] : the coordinates of the observations.
- [,"index"] : the index of the observation in object.
- [,"edge"] : whether each observation is on an edge.
- [,"surface"] : whether each observation is in the interior.
- [,"userMask"] : whether each observation belong to index1 (if no not defined, always FALSE).

Examples

```
## Not run:
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## outline the area of interest
res <- outlineMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0",
             num = 1:3, sequential = TRUE, overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1_red, param = "userMask",
             num = 1:3)


## outline an edge of interest
res <- outlineMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0", index1 = "MASK_DWI_t0",
          fill = FALSE, num = 1:3, sequential = TRUE, overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1_red, param = "userMask", num = 1:3)

## define an new area as the union of the outlined area and the initial lesion mask
res <- outlineMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0", index1 = "MASK_DWI_t0",
            operator_index1 = "union", num = 3, sequential = TRUE,
overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1_red, param = "userMask", num = 3)

## define an new area as the intersection of the outlined area and the initial lesion mask
res <- outlineMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0", index1 = "MASK_DWI_t0",
            operator_index1 = "intersection", num = 3, sequential = TRUE,
overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1_red, param = "userMask", num = 3)

## define an new area as the difference between the outlined area and the initial lesion mask
res <- outlineMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0", index1 = "MASK_DWI_t0",
            operator_index1 = "difference", num = 3, sequential = TRUE,
overwrite = TRUE, update.object = TRUE)

multiplot(MRIaggr.Pat1_red, param = "userMask", num=3)

## End(Not run)
```

`plotDistClass` *Plot the distribution of the contrast parameter*

Description

Distribution of a contrast parameter according to the tissue class.

Usage

```
## S4 method for signature 'MRIaggr'
plotDistClass(object, param, param.membership, num = NULL,
        bw.adjust = 1, kernel = "gaussian", from = NULL, to = NULL, ylim = NULL,
 col = 1:6, main = NULL, mgp = c(2,0.5,0), type = "l",
       pch = 20, lwd = 1, x.legend = "topright", y.legend = NULL, cex.legend = 0.8,
         filename = paste(object@identifier, "plotDistClass", sep = "_"), ...)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the contrast parameter to display. *character*. REQUIRED.
param.membership	the parameters indicating the probabilistic membership to the tissue classes. *character vector*.
num	the slices to use. *numeric vector* or NULL.
bw.adjust	the smoothing bandwidth to use. *numeric*. See density for more details.
kernel	the smoothing kernel to use. *character*. See density for more details.
from,to	the left and right-most points of the grid at which the density is to be estimated. *numeric* or NULL leading to automatic adjustment. See density for more details.
ylim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the y limits.
col	the colors with which the distributions will be displayed. *character vector*.
main	an overall title for the plot. *character*.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
type	the type of plot to display. *character*. See plot.default for more details.
pch	the symbol with which the distribution will be displayed. *positive integer*.
lwd	the line width. *postive numeric*.
x.legend	the x coordinates of the legend. *numeric* or *character*.
y.legend	the y coordinates of the legend. *numeric* or *character*.
cex.legend	the expansion factor of the legend. *positive numeric*.
filename	the name of the file used to export the plot. *character*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the lwd argument can be found in par.

Information about the x.legend, y.legend, cex.legend arguments can be found in legend (cex.legend is the cex argument of legend).

Argument(s) ... must correspond to some of the following arguments : height, hemisphere, norm_mu, norm_sigma, path, res, unit width, window.

FUNCTION:
This method relies on the density function.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## display
plotDistClass(MRIaggr.Pat1_red, param = "DWI_t0",
              param.membership = c("MASK_T2_FLAIR_t2"))

# specify the smoothing bandwidth
plotDistClass(MRIaggr.Pat1_red, param = "DWI_t0", bw.adjust = 1,
              param.membership="MASK_T2_FLAIR_t2")

# specify the scale
plotDistClass(MRIaggr.Pat1_red, param = "DWI_t0", bw.adjust = 1,
              from = 200, to = 300, param.membership = "MASK_T2_FLAIR_t2")

# use several classes
## Not run:
plotDistClass(MRIaggr.Pat1_red, param = "TTP_t0", bw.adjust = 2,
              param.membership = c("CSF","WM","GM","MASK_T2_FLAIR_t2"))

plotDistClass(MRIaggr.Pat1_red, param = "DWI_t0", bw.adjust = 2,
              param.membership = c("CSF","WM","GM","MASK_T2_FLAIR_t2"))

## End(Not run)
```

---

plotLesion3D *3D plot of the lesion*

Description

Make a 3D plot of the lesion. (experimental version)

Usage

```
## S4 method for signature 'MRIaggr'
plotLesion3D(object, mask, edge = FALSE, Neighborhood = "3D_N6",
         numeric2logical = FALSE, spatial_res = c(1,1,1), xlim = NULL,
```

```
ylim = NULL, zlim = NULL, type.plot = "shapelist3d", px_max = 10000,
radius = 1, type = "s", col = "red", col.edge = "black")
```

Arguments

object	an object of class MRIaggr. REQUIRED.
mask	the binary contrast parameter indicating the lesion. *character*. REQUIRED.
edge	should the edges of the lesion be ploted instead of the core ? *logical*.
Neighborhood	the type of neighbourhood used to defined the edges. *character*.
numeric2logical	should mask be convert to logical ? *logical*.
spatial_res	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
xlim	the x limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the x limits.
ylim	the y limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the y limits.
zlim	the z limits of the plot. *numeric vector of size 2* or NULL leading to automatic setting of the z limits.
type.plot	the type of plot to be displayed. Can be "plot3d" or "shapelist3d".
px_max	the maximum number of points that can be ploted. *integer*.
radius	the radius of spheres. *numeric*. See plot3d for more details.
type	the type of item to plot. *character*. See plot3d for more details.
col	the color of the core of the lesion. *character*.
col.edge	the color of the edge of the lesion. *character*.

Details

ARGUMENTS:
the Neighborhood argument can be a *matrix* or an *array* defining directly the neighbourhood to use (i.e the weight of each neighbor) or a name indicating which type of neighbourhood should be used (see the details section of initNeighborhood).

FUNCTION:
This functions uses the plot3d or shapelist3d and thus require the *rgl* package to work. This package is not automatically loaded by the MRIaggr package.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## Not run:
```

```
if(require(rgl)){
  # global view
plotLesion3D(MRIaggr.Pat1_red, mask = "MASK_T2_FLAIR_t2", spatial_res = c(1.875,1.875,6),
     numeric2logical = TRUE)

  # by slice
plotLesion3D(MRIaggr.Pat1_red, mask = "MASK_T2_FLAIR_t2", spatial_res = c(1.875,1.875,6),
     type.plot = "plot3d",
     numeric2logical = TRUE)
}

## End(Not run)
```

---

plotMRI *Diplay a contrast parameter by coordinates*

Description

Display a single slice of a contrast parameter. For internal use.

Usage

```
plotMRI(contrast, coords, breaks, palette, col, asp,
        xlim, ylim, pch, cex, axes, col.NA, pch.NA, xlab, ylab, main, cex.main)
```

Arguments

contrast	the intensities to display. *numeric vector*.
coords	the spatial coordinates of the observations. *data.frame*.
breaks	the break points to use to generate the color intervals. *numeric vector*.
palette	the colors or the palette to use when associating colors to observation intensities. *character vector*.
col	the colors with which the observations will be displayed. *character vector*.
asp	the aspect ratio y/x. *numeric*.
xlim	the x limits of the plot. *numeric vector of size 2*.
ylim	the y limits of the plot. *numeric vector of size 2*.
pch	the symbol with which the observations will be displayed. *positive integer*.
cex	the expansion factor for the observation labels. *positive numeric*.
axes	should the axes be plotted ? *logical*.
col.NA	the color to use to plot the NAs. *character*.
pch.NA	the label to use to plot the NAs. *postive integer*.
xlab	a title for the x axis. *character*.
ylab	a title for the y axis. *character*.
main	an overall title for the plot. *character*.
cex.main	the expansion factor for the main title. *numeric*.

Details

ARGUMENTS:
Information about the `palette` argument can be found in the details section of `initCol`.

`plotSigmaGR` *Display quality criteria for the GR algorithm*

Description

Display the quality criteria for various values of sigma using the result of the `calcSigmaGR` function.

Usage

```
plotSigmaGR(calcSigmaGR, mar = c(3, 3, 2, 2), mgp = c(2, 0.75, 0), main = "",
     col = c("black", grDevices::rainbow(5)),
    criterion = c("n.obs", "transition", "sdfront", "entropy", "Kalinsky", "Laboure"),
    name_criteria = c("region size", "boundary transition", "boundary heterogeneity",
                      "region entropy", "region Kalinsky", "region Laboure"),
     filename = "calcSigmaGR", ...)
```

Arguments

`calcSigmaGR`	an object generated by the `calcSigmaGR` function. REQUIRED.
`mar`	the number of margin lines to be specified on the four sides of the plot. *positive numeric vector of size 4*.
`mgp`	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
`main`	an overall title for the plot. *character*.
`col`	the color to use to plot each criterion. *character vector*.
`criterion`	the criterion to be displayed. *character vector*.
`name_criteria`	the name to be used in the legend. *character vector*.
`filename`	the name of the file used to export the plot. *character*.
`...`	additional arguments for the graphical device : `window`, `width`, `height`, `path`, `unit`, `res` (see `optionsMRIaggr`).

See Also

`calcSigmaGR` to compute the quality criteria.

Examples

```
## load an \code{MRIaggr} object
data(MRIaggr.Pat1_red, package = "MRIaggr")

calcThresholdMRIaggr(MRIaggr.Pat1_red,param = c("TTP_t0","MTT_t0"), threshold = 1:10,
                     name_newparam = c("TTP.th_t0","MTT.th_t0"),
                     update.object = TRUE, overwrite = TRUE)

## display raw parameter
multiplot(MRIaggr.Pat1_red, param = "TTP.th_t0", num = 3, numeric2logical = TRUE,
          index1 = list(coords = "MASK_DWI_t0", outline = TRUE))

## extract raw parameter, coordinates and compute the neighbourhood matrix
carto <- selectContrast(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion",
                        param = c("TTP.th_t0","TTP_t0","MASK_DWI_t0"))
coords <- selectCoords(MRIaggr.Pat1_red, num = 3, hemisphere = "lesion")
W <- calcW(coords, range = sqrt(2))$W

## the seed is taken to be the point with the largest TTP in the lesion mask
indexN <- which(carto$MASK_DWI_t0 == 1)
seed <- indexN[which.max(carto[indexN,"TTP_t0"])]

## find optimal sigma
resGR_sigma <- calcSigmaGR(contrast = carto$TTP.th_t0, W = W, seed = seed,
                           sigma = seq(1,4,0.1), iter_max = 50,
                           keep.upper = TRUE)

## display quality criteria according to sigma
plotSigmaGR(resGR_sigma)

plotSigmaGR(resGR_sigma, criterion = "entropy")
```

---

plotTableLesion *Lesion volume displayed by slices*

Description

Display the lesion volume by slices using the table_lesion slot.

Usage

```
## S4 method for signature 'MRIaggr'
plotTableLesion(object, mask, num = NULL, type = "matplot",
         col = 1:5, lty = 1:5, lwd = 1, mgp = c(2,0.5,0), mar = rep(3,4),
         main = paste("lesion - ", object@identifier, sep = ""),
         cex.legend = 1, cex.main = 1, cex.axis = 1, cex.lab = 1,
        filename = paste(object@identifier, "plotTableLesion", sep = "_"), ...)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
mask	the binary contrast parameter indicating the lesion. *character*. REQUIRED.
num	the slices to display. *numeric vector* or NULL.
type	the type of plot to display. Can be "matplot" or "evolution".
col	the colors with which the volumes will be displayed. *character vector* or *numeric vector*.
lty	the line type used to represent the volume. *numeric vector*.
lwd	the line width. *postive numeric*.
main	an overall title for the plot. *character*.
mgp	the margin line for the axis title, axis labels and axis line. *positive numeric vector of size 3*.
mar	the number of margin lines to be specified on the four sides of the plot. *positive numeric vector of size 4*.
cex.main	the expansion factor for the main title. *numeric*.
cex.legend	the expansion factor of the legend. *positive numeric*.
cex.axis	the magnification to be used for axis annotation relative to the current setting of cex. *positive numeric*.
cex.lab	the magnification to be used for x and y labels relative to the current setting of cex. *positive numeric*.
filename	the name of the file used to export the plot. *character*.
...	additional arguments to be passed to optionsMRIaggr for specifying the graphical parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the lty, lwd, mar and mgp arguments can be found in par.

Argument(s) ... must correspond to some of the following arguments : height, numeric2logical, path, res, unit, width, window.

ARGUMENTS:
Require to have previously filled object@table_lesion. This can be done using the calcTableLesion method.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## matplot display of the lesion
plotTableLesion(MRIaggr.Pat1_red, num = 1:3, type = "matplot",
                 mask = c("MASK_DWI_t0", "MASK_T2_FLAIR_t2"))

## evolution display of the lesion
plotTableLesion(MRIaggr.Pat1_red, num = 1:3, type = "evolution",
                mask = c("MASK_DWI_t0", "MASK_T2_FLAIR_t2"))
```

---

pointsHemisphere *Add the position of the mid-sagittal plan*

Description

Add to an existing plot the mid-sagittal plan. For internal use.

Usage

```
## S4 method for signature 'MRIaggr'
pointsHemisphere(object, col = "red", lwd = 2, lty = 1)
```

Arguments

`object`	an object of class `MRIaggr`.
`col`	the color with which the mid-sagittal plan will be plotted. *character*.
`lwd`	the line width. *postive numeric*.
`lty`	the type of line used to represent the mid-sagittal plan. *numeric vector*.

Details

ARGUMENTS:
Information about the `lwd` and `lty` arguments can be found in `par`.

pointsOutline *Compute the outline of a spatial group*

Description

Return a the outline of a spatial group for a given neighbourhood. For internal use.

Usage

```
pointsOutline(coords, array = NULL, filter = "2D_N4")
```

Arguments

coords	the spatial coordinates of the observations. *data.frame*.
array	alternative specification of the spatial coordinates using an array where the non-NA values indicates the points of interest. *array* or NULL leading to use the coords argument.
filter	the type of filter, see calcFilter for more details. *character*.

Details

FUNCTION:
This function apply a filter to the spatial group and consider as the outline all the observations that do not reach the maximum value (i.e. do not have a complete neighbourhood).

Value

A *data.frame* with the spatial coordinates of the outline.

readMRI *Read an image file*

Description

Read an image file and convert it into an array.

Usage

```
readMRI(filename, format, na.value = 0,
what = "numeric", size = "NA_integer_", dimensions = NULL,
SPM = FALSE, reorient = FALSE, flipud = FALSE)
```

Arguments

filename	the file name of the image file. *character*. REQUIRED.
format	the format of the image file. Can be "raw.gz", "analyze", "nifti" or "dicom". REQUIRED.
na.value	the value with which NA values are replaced. *numeric* or NA.
what	an object whose mode will give the mode of the vector to be read, or a character vector of length one describing the mode: one of "numeric", "double", "integer", "int", "logical", "complex", "character", "raw". Only active if format equals rawb.gz.
size	the number of bytes per element in the byte stream. *integer*. See the documentation of the writeBin function for more details.
dimensions	the number of bytes per element in the byte stream. The default, NA_integer_, uses the natural size. Size changing is not supported for raw and complex vectors. Only active if format equals rawb.gz.
SPM	is a logical variable (default = FALSE) that forces the voxel data values to be rescaled using the funused1 ANALYZE header field. This is an undocumented convention of ANALYZE files processed using the Statistical Parametric Mapping (SPM) software. Only active if format equals analyse.
reorient	is a logical variable (default = TRUE) that enforces Qform/Sform transformations. Only active if format equals nifti.
flipud	is a logical variable for vertical flipping of the image (default is TRUE). Only active if format equals dicom.

Details

This function requires to have installed the *oro.nifti* package to work if argument format is set to "analyze" or "nifti". It requires to have installed the *oro.dicom* package to work if argument format is set to dicom.

ARGUMENTS : filename argument corresponds to:

- the con argument of the base::readBin function.
- the fname argument of the oro.nifti::readANALYZE function. It should be a pathname to .img or .hdr files without the suffix.
- the fname argument of the oro.nifti::readNIfTI function.
- the fname argument of the oro.dicom::readDICOMFile function. It should be the file name with suffix.

The what and dimensions correspond to the what and size argument of the base::readBin function.
The SPM corresponds to the SPM argument of the oro.nifti::readANALYZE function.
The reorient corresponds to the reorient argument of the oro.nifti::readNIfTI function.
The flipud corresponds to the flipud argument of the oro.dicom::readDICOMFile function.

FUNCTION :
This function is a copy of the readMRI function of the mritc package. It calls readANALYZE of

the oro.nifti package to read analyze files, readNIfTI of the oro.nifti package to read NIFTI files, readDICOMFile of the oro.dicom package to read dicom files and readBin of the base package to read rawb.gz.

Value

A nifti object if format="nifit", an anlz object if format="analyse" and else an array.

Examples

```
## load a NIFTI file
path.Pat1 <- system.file(file.path("nifti"), package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
dim(nifti.Pat1_TTP_t0)

## Not run:
## load an analyse file (example of oro.nifti::readANALYZE)
path.Pat2 <- system.file("anlz", package = "oro.nifti")
analyse.avg <- readMRI(file.path(path.Pat2, "avg152T1"), format = "analyze")
graphics::image(analyse.avg[,,45,1])

### load a NIFTI file (example of oro.nifti::readNIfTI)
path.Pat3 <- system.file("nifti", package = "oro.nifti")
nifti.ffd <- readMRI(file.path(path.Pat3, "filtered_func_data"), format = "nifti")
graphics::image(nifti.ffd[,,10,32])

## load a dicom file (examples of oro.dicom::readDICOMFile)
path.Pat4 <- system.file("dcm", package = "oro.dicom")
dicom.Abdo <- readMRI(file.path(path.Pat4,"Abdo.dcm"), format = "dicom")
graphics::image(dicom.Abdo[[2]], col = grey(0:64/64), axes = FALSE, xlab = "", ylab = "",
                main = "Abdo.dcm")

## End(Not run)
```

---

rhoLvfree *Estimation of the local regularization parameters*

Description

Estimation of the local regularization parameters on external data using an efficient MCMC procedure.

Usage

```
rhoLvfree(Y, W_SR, rho_max = 10, rho_init = "PML", sd_rho_init = "PML", site_order = NULL,
    dprior = function(x){stats::dunif(x,0,rho_max)}, epsilon = 0.001, update_epsilon = 1,
 n.sample = 2500, iter_max = 3, burn_in = round(0.25 * n.sample), thin = 1,
 n.chain = 1, verbose = 3, cpus = 1, export.coda = TRUE)
```

Arguments

Y	a *matrix* containing the observations (by rows) for the various groups (by columns). REQUIRED.
W_SR	the local neighbourhood matrix. *dgCMatrix*. Should be normalized by row (i.e. rowSums(W_SR)=1). REQUIRED.
rho_max	maximum possible rho value, minimum is 0. *double*.
rho_init	the initial rho value. *double*. If NULL, it is sample in a uniform law between 0 and rho_max
sd_rho_init	the standard deviation of the rho value for the proposal distribution. *double*. It is updated at the end of the burn in.
site_order	a specific order to go all over the sites. Can be NULL or an *integer vector*.
dprior	the prior distribution. *function*. Must be a density function between 0 and rho_max.
epsilon	the tolerance parameter regulating the acceptance of the parameter. *double*.
update_epsilon	if over 1 the value by which the epsilon parameter is devided every 5% of the total iterations, if under one the acceptance rate to reach. *double*.
n.sample	the number of iterations of the Gibbs sampler. *integer*.
iter_max	the number of Gibbs move for each simulation. *integer*.
burn_in	the number of iteration of the burn in phase. *integer*.
thin	the thinning interval between consecutive observations. *integer*.
n.chain	the number of chain to use. *integer*.
verbose	how should the execution of the function should be traced ? 1 traces display the initialization and the final result, 2 traces every 5% of the total iterations and 3 also displays the update performed every 5%.
cpus	the number of CPU to use. *integer*.
export.coda	should the results be convert to the mcmc format of the coda package ? *logical*.

Details

FUNCTION:
This function uses a likelihood-free Metropolis-Hastings method (proposed by Pereyra et al., 2013) to estimate the local regularization parameter. It should give a less biased estimation of the parameter but require a higher computational cost.

Value

A good initialisation (rho_init) may be obtained with the rhoMF function. It is automatically done if rho_init is set to "PML".

Setting sd_rho_init to "PML" leads to adjustement of the initial variance according to the rho value and the sample size.

References

M. Pereyra, N. Dobigeon, H. Batatia, and J.Y. Tourneret. *Estimation the granularity coefficient of a Potts-Markov random field within an MCMC algorithm*. IEEE Trans. Image Porcessing, 22(6):2385-2397, 2013.

See Also

calcW to compute the neighbourhood matrix,
simulPotts to simulate from a Potts model. rhoMF to estimate the regularization parameters using mean field approximation. calcPottsParameter general interface for estimating the regularization parameters.

Examples

```
# spatial field
## Not run:
n <- 50

## End(Not run)

G <- 3
coords <- which(matrix(0, nrow = n * G, ncol = n * G) == 0, arr.ind = TRUE)

# neighbourhood matrix
resW <- calcW(as.data.frame(coords), range = sqrt(2), row.norm = TRUE, calcBlockW = TRUE)
W_SR <- resW$W
site_order <- unlist(resW$blocks$ls_groups)-1

# initialisation
set.seed(10)
sample <- simulPotts(W_SR, G = 3, rho = 3.5, iter_max = 500,
                     site_order = site_order)$simulation

multiplot(as.data.frame(coords), sample,palette = "rgb")

# estimation

## Not run:
rho <- rhoLvfree(Y = sample, W_SR = W_SR, site_order = site_order)

## End(Not run)
```

---

rhoMF *Estimation of the local and regional spatial correlation*

Description

Estimation the spatial regularization parameters on external data using mean field approximation.

Usage

```
rhoMF(Y, W_SR, rho_max = 50, prior_prevalence = TRUE,
    test.regional = FALSE, W_LR, distance.ref, coords, threshold = 0.1,
nbGroup_min = 100, regionalGroups = "last_vs_others", multiV = TRUE)
```

Arguments

Y	a *matrix* containing the observations (by rows) for the various groups (by columns). REQUIRED.
W_SR	the local neighbourhood matrix. dgCMatrix. Should be normalized by row (i.e. rowSums(Wweight_SR)=1). REQUIRED.
rho_max	Maximum possible rho value (*numeric*), minimum is 0.
prior_prevalence	should a prior on class prevalence be including when estimating the regularisation parameters ? *logical*.
test.regional	Should regional regularization be considered. *logical*.
W_LR	the regional neighbourhood matrix. *dgCMatrix*. Should be contains the distances between the observations (0 indicating infinite distance).
distance.ref	the interval of distance defining the several neighbourhood orders in W_LR. *numeric vector*.
threshold	the minimum value of the posterior probability for group G for being considered as lesioned when forming the spatial groups. *double*.
nbGroup_min	the minimum group size of the spatial groups required for computing the regional potential. *integer*.
coords	coordinates of each site. *matrix*.
regionalGroups	how should the regional potential be computed : last group versus the others ("last_vs_others") or for each group ("each").
multiV	should the regional neighbourhood range be computed for each spatial group ? *logical*.

Value

A *numericVector* containing the estimated regularisation parameter(s).

See Also

calcW to compute the neighbourhood matrix,
simulPotts to draw simulations from a Potts model.
rhoLvfree to estimate the regularization parameters using mean field approximation. calcPottsParameter general interface for estimating the regularization parameters.

Examples

```
# spatial field
## Not run:
n <- 50

## End(Not run)

G <- 3
coords <- which(matrix(0, nrow = n * G, ncol = n * G) == 0,arr.ind = TRUE)

# neighbourhood matrix
W_SR <- calcW(as.data.frame(coords), range = sqrt(2), row.norm = TRUE)$W
W_LR <- calcW(as.data.frame(coords), range = 10, row.norm = FALSE)$W

# initialisation
set.seed(10)
sample <- simulPotts(W_SR, G = 3, rho = 3.5, iter_max = 500,
                     site_order = TRUE)$simulation

multiplot(as.data.frame(coords), sample,palette = "rgb")

# estimation
rho <- rhoMF(Y=sample, W_SR = W_SR)
rho

# the regional potential is computed for each group
rho <- rhoMF(Y = sample, W_SR = W_SR,
             test.regional = TRUE, W_LR = W_LR, distance.ref = seq(1, 10, 0.5),
 coords = coords, regionalGroups = "each")
rho

# the regional potential is computed for the last group vs. the others
rho <- rhoMF(Y = sample, W_SR = W_SR,
             test.regional = TRUE, W_LR = W_LR, distance.ref = seq(1, 10, 0.5),
 coords = coords, regionalGroups = "last_vs_others")
rho
```

`selectClinic` *Extract clinical data*

Description

Extract the clinical data from a `MRIaggr` object.

Usage

```
## S4 method for signature 'MRIaggr'
selectClinic(object, param = NULL)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	the clinical parameters to extract. *character vector* or NULL leading to extract all the clinical parameters.

Value

A one line *data.frame* containing the clinical data in columns.

See Also

allocClinic<- to allocate values in the clinic slot.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## select all clinical data
res <- selectClinic(MRIaggr.Pat1_red)

## select only the gender
res <- selectClinic(MRIaggr.Pat1_red, param = "Sex")
```

---

selectContrast *Extract contrast parameters*

Description

Extract the contrast parameters from a Carto3D or a MRIaggr object.

Usage

```
## S4 method for signature 'Carto3D'
selectContrast(object, num = NULL, na.rm = FALSE, coords = FALSE,
         format = "data.frame")

## S4 method for signature 'MRIaggr'
selectContrast(object, param = NULL, num = NULL, format = "data.frame",
         slice_var = "k", coords = FALSE, hemisphere = "both",
 norm_mu = FALSE, norm_sigma = FALSE, na.rm = FALSE, subset = NULL)
```

Arguments

object	an object of class Carto3D or MRIaggr. REQUIRED.
param	the contrast parameters to extract. *character vector* or NULL.
num	the slices to extract. *numeric vector* or NULL.
format	the format of the output. Can be "matrix", "data.frame" or "vector".
slice_var	the type of slice to extract. "i" for sagittal, "j" for coronal and "k" for transverse. *character*.
coords	the coordinates that sould be extracted. *logical* or any of "i" "j" "k".
hemisphere	the hemisphere to extract. *character*. See the details section.
norm_mu	the type of centering to apply on the parameter values. *character*. See the details section.
norm_sigma	the type of scaling to apply on the parameter values. *character*. See the details section.
na.rm	should observations with missing values be removed ? *logical*.
subset	the subset of observations to extract. *positive integer vector* or NULL leading to use all observations

Details

ARGUMENTS:
Information about the param argument can be found in the details section of initParameter.

Information about the num argument can be found in the details section of initNum.

Possible values for the hemisphere argument are:

- "both" : select all the observations.
- "left" : select the observations from the left hemisphere.
- "right" : select the observations from the right hemisphere.
- "lesion" : select the observations belonging to the hemisphere(s) that contain(s) the lesion (if any).
- "contralateral" : select the observations belonging to the hemisphere(s) that do not contain(s) the lesion (if any).

To select observations from a given hemisphere (all values except "both"), the parameter hemisphere must have been allocated to the object using, for instance, calcHemisphere. In addition for "lesion" and "contralateral" values, the slot @hemispheres has to be filled using, for instance, calcHemisphere.

Possible values for the centering argument (norm_mu) and the scaling argument (norm_sigma) are:

- "FALSE" : no normalization
- "global" : the centering or scaling value is computed using all the observations.
- "global_1slice" : the centering or scaling value is computed using all the observations that belong to the slice of the observation to normalize.

- "global_3slices" : the centering or scaling value is computed using all the observations that belong to the slice of the observation to normalize, the slice above (if any) and the slice below (if any).
- "contralateral" : the centering or scaling value is computed using the observations from the contralateral hemisphere.
- "contralateral_1slice" : the centering or scaling value is computed using the observations from the contralateral hemisphere that belong to the slice of the observation to normalize.
- "contralateral_3slices" : the centering or scaling value is computed using the observations from the contralateral hemisphere that belong to the slice of the observation to normalize, the slice above (if any) and the slice below (if any).
- "default_value" : the default value of the parameter stored in the slot @default_value is used for the centering (for norm_mu only).

If coords is set to TRUE the dataset containing the contrast parameters values will also contains all the coordinates. If coords is set to FALSE, it will not contain any coordinates.

Argument subset can be a *character* value that refers to a logical parameter in the object defining the subset of observation to extract.

FUNCTION:
Each of the num, hemisphere and subset argument define a subset of the total set of observations. It is the intersection of all these three subsets that is extracted.

When a normalisation is requested to center (resp. scale) the data, the normalisation value is extracted for each parameter in the element of the slot normalization that match the argument norm_mu (resp. norm_sigma). The parameters values are first centered by substraction with the value returned by norm_mu. Then they are scaled by division with the value returned by norm_sigma.

Value

A "vector", *matrix* or a *data.frame*. In the latter two cases, each row corresponds to a voxel and each column to a coordinate.

See Also

calcContralateral, calcRegionalContrast, calcFilter and calcTissueType to retreat and allocate the modified contrast parameters.
allocContrast<- to allocate new contrast parameters.
calcNormalization to compute and allocate the normalisation values.
allocNormalization<- to allocate the normalization values when obtained from an external source.
calcHemisphere and calcContralateral to compute and allocate the hemispheres.
allocHemisphere<- and allocContrast<- to allocate hemispheres obtained from an external source.

Examples

```
#### 1- Carto3D method ####
## load nifti files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")
```

```
## select all observations
carto1 <- selectContrast(Carto3D.Pat1_TTP_t0)
dim(carto1)

## select observations from slices 1 to 3 and return the result into a data.frame
carto2 <- selectContrast(Carto3D.Pat1_TTP_t0, num = 1:3, coords = FALSE, format = "data.frame")
dim(carto2)

## select observations from slices 1 to 3 and return the result into a vector
carto3 <- selectContrast(Carto3D.Pat1_TTP_t0, num = 1:3, coords = FALSE)
length(carto3)

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## select all parameters and all observations
carto <- selectContrast(MRIaggr.Pat1_red)
dim(carto)
head(carto)

## select a subset of parameters
carto <- selectContrast(MRIaggr.Pat1_red, param = c("DWI_t0","T2_FLAIR_t2"))
dim(carto)
head(carto)

## select a subset of parameters on slices 1 to 3
carto <- selectContrast(MRIaggr.Pat1_red, num=1:3, param=c("DWI_t0","T2_FLAIR_t2"))
dim(carto)
head(carto)

## select a subset of parameters on slices 1 to 3 and normalized the center
## the values using the contralateral
carto <- selectContrast(MRIaggr.Pat1_red, num=1:3, param=c("DWI_t0","T2_FLAIR_t2"),
                        norm_mu = "contralateral")
dim(carto)
head(carto)

## select only observations which are lesioned at admission (i.e. MASK_DWI_t0 = TRUE)
carto <- selectContrast(MRIaggr.Pat1_red, subset = "MASK_DWI_t0",
                        param = c("DWI_t0","T2_FLAIR_t2","MASK_DWI_t0"))
dim(carto)
head(carto)

## select only observations which are lesioned at admission (i.e. MASK_DWI_t0 = TRUE)
## with coordinates
carto <- selectContrast(MRIaggr.Pat1_red, subset = "MASK_DWI_t0",
                        param = c("DWI_t0","T2_FLAIR_t2","MASK_DWI_t0"), coords = TRUE)
dim(carto)
head(carto)

## select only observations for which i = 55
```

```
carto <- selectContrast(MRIaggr.Pat1_red, slice_var = "i",
                        num = 55, coords = TRUE)
dim(carto)
head(carto)
```

`selectCoords` *Extract spatial coordinates*

Description

Extract the coordinates from a `Carto3D` or from a `MRIaggr` object.

Usage

```
## S4 method for signature 'Carto3D'
selectCoords(object, coords=c("i","j","k"), num = NULL, format = "data.frame")

## S4 method for signature 'MRIaggr'
selectCoords(object, coords = c("i","j","k"), spatial_res = c(1,1,1),
         num = NULL, hemisphere = "both", subset = NULL,
 slice_var = "k", format = "data.frame")
```

Arguments

`object`	an `object` of class `Carto3D` or `MRIaggr`. REQUIRED.
`coords`	the coordinates that sould be extracted. Any of `"i"` `"j"` `"k"` or `"index"`.
`spatial_res`	a dilatation factor for the coordinates. *positive numeric vector of size 3*.
`num`	the slices to extract. *numeric vector* or `NULL`.
`hemisphere`	the hemisphere to extract. *character*.
`subset`	the subset of observations to extract. *positive integer vector* or `NULL` leading to use all observations
`slice_var`	the type of slice to extract. `"i"` for sagittal, `"j"` for coronal and `"k"` for transverse. *character*.
`format`	the format of the output. Can be `"vector"`, `"matrix"` or `"data.frame"`.

Details

ARGUMENTS:
Information about the `num` argument can be found in the details section of `initNum`.

Information about the `hemisphere` argument can be found in the details section of `selectContrast`.

FUNCTION:
Each of the `num`, `hemisphere` and `subset` argument define a subset of the total set of observations. It is the intersection of all these three subsets that is extracted.

Value

A "vector", *matrix* or a *data.frame*. In the latter two cases, each row corresponds to a voxel and each column to a coordinate.

See Also

calcHemisphere to identify the hemispheres.
allocHemisphere<- and allocContrast<- to allocate hemispheres obtained from an external source.

Examples

```
#### 1- Carto3D method ####
## load nifti files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")

## selection all coordinates
coords1 <- selectCoords(Carto3D.Pat1_TTP_t0)
dim(coords1)

## selection coordinates i and j from slices 1 to 3
coords2 <- selectCoords(Carto3D.Pat1_TTP_t0, num = 1:3, coords = c("i","j"))
dim(coords2)

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package="MRIaggr")

## select all coordinates for all observations
coords <- selectCoords(MRIaggr.Pat1_red)
dim(coords)
head(coords)

## select coordinate i for slices 1 and 3
coords <- selectCoords(MRIaggr.Pat1_red, coords = "i", num = c(1,3))
dim(coords)
head(coords)

## select coordinate i for observations in the hemishere containing the lesion
coords <- selectCoords(MRIaggr.Pat1_red, hemisphere = "lesion", num = c(1,3))
dim(coords)
head(coords)

## select coordinate i for observations in the right hemisphere
coords <- selectCoords(MRIaggr.Pat1_red, hemisphere = "right", num = c(1,3))
dim(coords)
head(coords)

## select all coordinates and rescale them
coords <- selectCoords(MRIaggr.Pat1_red, spatial_res = c(1.875,1.875,6))
```

```
dim(coords)
head(coords)

## select coordinate i and j and return a matrix
coords <- selectCoords(MRIaggr.Pat1_red, format = "matrix")
is(coords)
head(coords)
```

`selectDefault_value` *Extract reference values*

Description

Extract the reference values of the contrast parameters from a `Carto3D` or from a `MRIaggr` object.

Usage

```
## S4 method for signature 'Carto3D'
selectDefault_value(object)

## S4 method for signature 'MRIaggr'
selectDefault_value(object, param = NULL, character2numeric = FALSE)
```

Arguments

`object`	an `object` of class `Carto3D` or `MRIaggr`. REQUIRED.
`param`	the contrast parameters for which the reference values should be returned. *character vector* or `NULL` leading to extract reference values for all available contrast parameters.
`character2numeric`	should the default values be converted from character to numeric. *logical*.

Value

A *data.frame* with one line an several columns containing the default value of each parameter.

Examples

```
#### 1- Carto3D method ####
## load nifti files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")

## selection
selectDefault_value(Carto3D.Pat1_TTP_t0)
```

```
#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## select default values for all parameters
res <- selectDefault_value(MRIaggr.Pat1_red)

## select default values for T2_FLAIR_t2 and DWI_t0
res <- selectDefault_value(MRIaggr.Pat1_red, param=c("T2_FLAIR_t2", "DWI_t0"))

## select default values for T2_FLAIR_t2 and DWI_t0 and convert them in numeric
res <- selectDefault_value(MRIaggr.Pat1_red, param=c("T2_FLAIR_t2", "DWI_t0"),
         character2numeric = TRUE)
```

`selectDescStats` *Extract non-stardard elements*

Description

Extract elements in the `ls_descStats` slot of a `MRIaggr` object.

Usage

```
## S4 method for signature 'MRIaggr'
selectDescStats(object, name = NULL)
```

Arguments

`object`	an `object` of class `MRIaggr`. REQUIRED.
`name`	the name of the element to select. *character* or `NULL` leading to select all available elements.

Details

This function requires to have installed the *Matrix* and the *spam* package to work when argument `name` is set to `"W_euclidean"`.

See Also

`allocDescStats<-` to allocate elements in the `ls_descStats` slot.
`calcHemisphere` to identify the hemispheres.
`allocHemisphere<-` and `allocContrast<-` to allocate hemispheres obtained from an external source.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

calcGroupsMask(MRIaggr.Pat1_red,mask = c("MASK_DWI_t0","MASK_T2_FLAIR_t2"),
               W.range = 6, W.spatial_res = c(1.875,1.875,6),
               update.object = TRUE, overwrite = TRUE)

## select all elements in the slot @ls_descStats
ls_descStats <- selectDescStats(MRIaggr.Pat1_red)
names(ls_descStats)

## get the name of all elements present in the slot @ls_descStats
selectParameter(MRIaggr.Pat1_red, type = "ls_descStats")

## select a specific element
res <- selectDescStats(MRIaggr.Pat1_red, name = "GroupsLesion")
```

---

selectFieldDim *Extract the data dimension*

Description

Extract the dimensions of the spatial field from a Carto3D or from a MRIaggr object.

Usage

```
## S4 method for signature 'Carto3D'
selectFieldDim(object)

## S4 method for signature 'MRIaggr'
selectFieldDim(object)
```

Arguments

object	an object of class Carto3D or MRIaggr. REQUIRED.

Value

A *data.frame* with one line and three columns named "i","j","k" indicating the corresponding dimension.

Examples

```
#### 1- Carto3D method ####
## load NIFTI files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")
```

```
## selection
selectFieldDim(Carto3D.Pat1_TTP_t0)

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectFieldDim(MRIaggr.Pat1_red)
```

`selectHemispheres` *Extract the position of the lesion in each hemisphere*

Description

Extract the position of the lesion in each hemisphere from a `MRIaggr` object.

Usage

```
## S4 method for signature 'MRIaggr'
selectHemispheres(object, hemisphere = "both")
```

Arguments

`object`	an `object` of class `Carto3D` or `MRIaggr`. REQUIRED.
`hemisphere`	the hemisphere of interest. *character*. See the details section.

Details

Possible values for the `hemisphere` argument are:

- `"both"` : indicates the presence of the lesion in both hemispheres.
- `"left"` : indicates the presence of the lesion in the left hemisphere.
- `"right"` : indicates the presence of the lesion in the right hemisphere.
- `"lesion"` : indicates which hemispheres are lesioned.
- `"contralateral"` : indicates which hemispheres are intact.

Value

A *character vector* or a *data.frame*.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectHemispheres(MRIaggr.Pat1_red)
selectHemispheres(MRIaggr.Pat1_red, hemisphere = "right")
selectHemispheres(MRIaggr.Pat1_red, hemisphere = "left")
selectHemispheres(MRIaggr.Pat1_red, hemisphere = "lesion")
selectHemispheres(MRIaggr.Pat1_red, hemisphere = "contralateral")
```

---

selectHistory *Extract the call of the methods applied on the object*

Description

Extract the history of a MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectHistory(object)
```

Arguments

object	an object of class MRIaggr. REQUIRED.

Value

A *list* of each calc or const method that was applied to the object. For each method, it contains the call, the date of call and potential extra elements.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectHistory(MRIaggr.Pat1_red)
```

selectIdentifier *Extract the identifier*

Description

Extract the patient identifier from a Carto3D or from a MRIaggr object.

Usage

```
## S4 method for signature 'Carto3D'
selectIdentifier(object)

## S4 method for signature 'MRIaggr'
selectIdentifier(object)
```

Arguments

object	an object of class Carto3D or MRIaggr. REQUIRED.

Value

A *character*.

Examples

```
#### 1- Carto3D method ####
## load nifti files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")

## selection
selectIdentifier(Carto3D.Pat1_TTP_t0)

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectIdentifier(MRIaggr.Pat1_red)
```

selectMidplane *Extract the position of the mid-sagittal plan*

Description

Extract the position of the mid-sagittal plan from a MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectMidplane(object)
```

Arguments

object	an object of class MRIaggr. REQUIRED.

Value

A two column *matrix*.

Examples

```
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectMidplane(MRIaggr.Pat1_red)
```

selectN *Extract the number of observations*

Description

Extract the number of observations contained in a Carto3D or in a MRIaggr object.

Usage

```
## S4 method for signature 'Carto3D'
selectN(object, num = NULL)

## S4 method for signature 'MRIaggr'
selectN(object, num = NULL, hemisphere = "both", subset = NULL)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
num	the slices to consider. *numeric vector* or NULL.
hemisphere	the hemisphere to consider. *character*.
subset	the subset of observations to consider. *positive integer vector* or NULL leading to consider all observations.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the hemisphere argument can be found in the details section of selectContrast.

FUNCTION:
Each of the num, hemisphere and subset argument define a subset of the total set of observations. It is the length of the intersection of all these three subsets that is measured.

Value

An *integer*.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## total number of observations
res <- selectN(MRIaggr.Pat1_red)

## number of observations for the hemisphere that contains the lesion
res <- selectN(MRIaggr.Pat1_red, hemisphere = "lesion")

## number of observations in the first 1000 observations t
## that are in the hemisphere containing the lesion
res <- selectN(MRIaggr.Pat1_red, subset = 1:1000, hemisphere = "lesion")
```

---

selectNormalization *Extract the normalization values*

Description

Extract the normalization values from a MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectNormalization(object, type = NULL, mu = TRUE, sigma = TRUE,
         hemisphere = "both", num = NULL, param = NULL)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
type	the type of normalization. Must be on of "global", "slice", "3slices" or NULL leading to select all types of normalization.
mu	should the centering values for the normalization be returned. *logical*. Active only if type is "slice" or "3slices".
sigma	should the scaling values for the normalization be returned. *logical*. Active only if type is "slice" or "3slices".
num	the slices to extract. *numeric vector* or NULL.
hemisphere	the hemisphere to extract. *character*.
param	the contrast parameters for which the normalization values should be extracted. *character vector* or NULL indicating all available contrast parameters.

Details

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the hemisphere argument can be found in the details section of selectContrast.

Value

A *data.frame* or a *list*.

See Also

calcNormalization to compute and allocate the normalisation values.
allocNormalization<- to alloacte the normalization values when obtained from an external source.
calcHemisphere to identify the hemispheres.
allocHemisphere<- and allocContrast<- to allocate hemispheres obtained from an external source.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## select all normalization values
res <- selectNormalization(MRIaggr.Pat1_red)
names(res)

## select specific normalization normalization values
# computed on the whole brain
res <- selectNormalization(MRIaggr.Pat1_red, type = "global",
        mu = TRUE, sigma = FALSE, hemisphere = "both")
# idem but only for DWI_t0
res <- selectNormalization(MRIaggr.Pat1_red, type = "global",
        mu = TRUE, sigma = FALSE, param = "DWI_t0")
```

```
# computed by slice
res <- selectNormalization(MRIaggr.Pat1_red, type="slice",
         mu = TRUE,sigma = FALSE, hemisphere = "both")
# idem for slice 1
res <- selectNormalization(MRIaggr.Pat1_red, type = "slice",
         mu = TRUE, sigma = FALSE, num = 1)

# computed on 3 consecutive slices
res <- selectNormalization(MRIaggr.Pat1_red, type = "3slices", mu = FALSE, sigma = TRUE,
                           hemisphere = "left", num = 2, param = "T2_FLAIR_t2")
```

---

selectOptionsMRIaggr *Extract default values*

Description

Extract value(s) from `optionsMRIaggr`. For internal use.

Usage

```
selectOptionsMRIaggr(field = NULL)
```

Arguments

`field`	the field to be extracted. *character*

---

selectParameter *Extract parameters*

Description

Extract parameters from a `Carto3D` or from a `MRIaggr` object.

Usage

```
## S4 method for signature 'Carto3D'
selectParameter(object)

## S4 method for signature 'MRIaggr'
selectParameter(object, type = "contrast", mask = TRUE)
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`type`	the type of parameter to select. *character*. See the details section.
`mask`	should the mask be considered as an available contrast parameter ? *logical*.

Details

ARGUMENTS:
Possible values for type are:

- "clinic" : return the name of the clinical parameters.
- "contrast" : return the name of the contrast parameters.
- "ls_descStats" : return the name of the elements in @ls_descStats.

See Also

allocContrast<- to allocate new parameters.

Examples

```
#### 1- Carto3D method ####
## load NIFTI files and convert them to Carto3D
path.Pat1 <- system.file("nifti", package = "MRIaggr")
nifti.Pat1_TTP_t0 <- readMRI(file.path(path.Pat1, "TTP_t0"), format = "nifti")
Carto3D.Pat1_TTP_t0 <- constCarto3D(nifti.Pat1_TTP_t0, identifier = "Pat1", param = "TTP_t0")

## selection
selectParameter(Carto3D.Pat1_TTP_t0)

#### 2- MRIaggr method ####
## load a MRIaggr object
data("MRIaggr.Pat1_red", package="MRIaggr")

## extract all imaging parameters
res <- selectParameter(MRIaggr.Pat1_red)

## extract the names of the parameters in the slot @ls_descStats
res <- selectParameter(MRIaggr.Pat1_red, type = "ls_descStats")
```

---

selectTable *Extract volumic information*

Description

Extract the volumic information (lesion, hyperfusion or reperfusion) from a MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectTable(object, type, size = FALSE)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
type	the table to extract. Can be "lesion", "reperfusion" or "hypoperfusion". REQUIRED.
size	should the values in the table correspond to a number of voxels (FALSE) or a volume (TRUE).

Value

A *data.frame* containing volumetric measurements.

See Also

calcTableHypoReperf to compute and allocate the hypoperfusion and reperfusion tables.
calcTableLesion to compute and allocate the lesion table.
allocTable<- to allocate data in the table slots.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
res <- selectTable(MRIaggr.Pat1_red, type = "lesion")
res <- selectTable(MRIaggr.Pat1_red, type = "reperfusion")
res <- selectTable(MRIaggr.Pat1_red, type = "hypoperfusion")
```

---

selectVoxelDim *Extract the dimensions of a voxel*

Description

Extract the voxel size stored in the MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectVoxelDim(object, unit = TRUE)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
unit	should the unit be returned ? *logical*.

Value

A *data.frame* with one line and three or four columns containing the voxel size for each dimension and the size unit (if requested).

Examples

```
data("MRIaggr.Pat1_red", package = "MRIaggr")

## selection
selectVoxelDim(MRIaggr.Pat1_red)
```

---

selectW *Extract a neighbourhood matrix*

Description

Extract elements in the W slot of a MRIaggr object.

Usage

```
## S4 method for signature 'MRIaggr'
selectW(object, type = "Wmatrix", subset_W = NULL,
         hemisphere = "both", num = NULL, slice_var = "k", upper = NULL)
```

Arguments

object	an object of class MRIaggr. REQUIRED.
type	the type of the element to select. Can be "Wmatrix", "Wblocks" or "upper".
subset_W	the subset of observations to select. *integer vector* or NULL leading to select all the observations.
hemisphere	the hemisphere to use. *character*.
num	the slices to use. *numeric vector* or NULL.
slice_var	the type of slice to extract. "i" for sagittal, "j" for coronal and "k" for transverse. *character*.
upper	the format of the matrix to extract. Can be the full matrix (upper=NULL) or upper- or lower-triangular matrix (upper=TRUE or upper=FALSE).

Details

This function requires to have installed the *Matrix* and the *spam* package to work when argument name is set to "W_euclidean".

ARGUMENTS:
Information about the num argument can be found in the details section of initNum.

Information about the hemisphere argument can be found in the details section of selectContrast.

FUNCTION:
If subset_W is not NULL then arguments hemisphere and num are ignored.
Each of the num, hemisphere and subset argument define a subset of the total set of observations. It is the intersection of all these three subsets is extracted.

See Also

calcW to compute the neighboring matrix.
calcBlockW to decompose the neighboring matrix in independant blocks.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## compute and allocate a neighbourhood matrix
calcW(object=MRIaggr.Pat1_red, range = sqrt(2), update.object = TRUE)

## select the neighbourhood matrix
selectW(MRIaggr.Pat1_red)[1:10,1:10]
selectW(MRIaggr.Pat1_red, upper = TRUE)[1:10,1:10]
# selectW(MRIaggr.Pat1_red, upper = FALSE)[1:10,1:10] # will not work
```

---

simulPotts *Draw a sample from a Potts model*

Description

Draw a sample from a Potts model. Call the simulPotts_cpp or the simulPottsFast_cpp C++ function.

Usage

```
simulPotts(W, G, rho, initialization = NULL, site_order = NULL, iter_max = 2500,
         cv.criterion = 0, fast = FALSE, verbose = TRUE)
```

Arguments

W	the neighbourhood matrix. *dgCMatrix*. Should be normalized by row (i.e. rowSums(W)=1). REQUIRED.
G	the number of groups. *postive integer*. REQUIRED.
rho	the value of the regularization parameter (also called potts parameter or inverse temperature). REQUIRED.
initialization	the probability membership of each voxel to each group used for initialisation. *matrix*.
site_order	a specific order to go all over the sites. Can be NULL, TRUE or an *integer vector*.

iter_max	the number of iterations. *numeric*.
cv.criterion	maximum difference in group probability membership for convergence ? *double* between 0 and 1.
fast	should simulPottsFast_cpp be used ? *logical*. Faster but diseable tracing and convergence checking.
verbose	should the simulation be be traced over iterations ? *logical*.

Details

If no specific order is set, the order is sampled in a uniformed law which increase the computation time. If site_order is set to true, independant spatial block are identifyied using the calcBlockW function. Each independant spatial block is then iteratively considered.

Value

A *numeric vector* containing the regional potential.

Examples

```
# spatial field

## Not run:
n <- 30
iter_max <- 500

## End(Not run)
G <- 3
coords <- data.frame(which(matrix(0, nrow = n * G, ncol = n * G) == 0, arr.ind = TRUE), 1)
optionsMRIaggr(quantiles.legend = FALSE, axes = FALSE, num.main = FALSE)

# neighbourhood matrix
resW <- calcW(coords, range = sqrt(2), row.norm = TRUE, calcBlockW = TRUE)
W <- resW$W

# with no initial sample
res1 <- simulPotts(W, G = 3, rho = 3, iter_max = iter_max)
multiplot(coords,
          apply(res1$simulation, 1, which.max),
          breaks=seq(0.5, 3.5, 1))

res2 <- simulPotts(W, G = 4, rho = 3, iter_max = iter_max)
multiplot(coords,
          apply(res2$simulation, 1, which.max),
          breaks = seq(0.5, 4.5, 1))

res3 <- simulPotts(W, G = 4, rho = 6, iter_max = iter_max)
multiplot(coords,
          apply(res3$simulation, 1, which.max),
          breaks = seq(0.5, 4.5, 1))

# with specific initialisation
```

```
res3.bis <- simulPotts(W, rho = 6, initialization = res3$simulation, iter_max = iter_max / 2)
multiplot(coords,
          apply(res3.bis$simulation, 1, which.max),
          breaks=seq(0.5, 4.5, 1))

res3.ter <- simulPotts(W, rho = 6, initialization = res3$simulation, iter_max = iter_max)
multiplot(coords,
          apply(res3.ter$simulation, 1, which.max),
          breaks=seq(0.5, 4.5, 1))

#### defining site order save time
site_order <- unlist(resW$blocks$ls_groups) - 1

system.time(
  res <- simulPotts(W, iter_max = 100, G = 3, rho = 6, site_order = NULL,
                    fast = TRUE, verbose = FALSE)
)

system.time(
  res <- simulPotts(W, iter_max = 100, G = 3, rho = 6, site_order = site_order,
                    fast = TRUE, verbose = FALSE)
)

system.time(
  res <- simulPotts(W, iter_max = 100, G = 3, rho = 6, site_order = NULL,
                    fast = FALSE, verbose = FALSE)
)

system.time(
  res <- simulPotts(W, iter_max = 100, G = 3, rho = 6, site_order = site_order,
                    fast = FALSE, verbose = FALSE)
)
```

`simulPottsFast_cpp` *Potts model simulation*

Description

Simulation of isotropic Potts model with local interaction by Gibbs sampling. Slightly faster compared to `simulPotts_cpp`.

Usage

```
simulPottsFast_cpp(W_i, W_p, W_x, site_order, sample, rho, n, p, iter_nb)
```

Arguments

W_i	the 0-based row numbers for each non-zero element in the sparse neighbourhood matrix. *integer vector*.
W_p	the pointers to the initial (zero-based) index of elements in the colum of the sparse neighbourhood matrix. *integer vector*.
W_x	the non-zero elements of the matrix. *numeric vector*.
site_order	a specific order to go all over the sites. An *integer vector* or a negative number to indicate no specific order.
sample	A *matrix* containing the initial observations (by rows) for the various groups (by columns).
rho	A *numeric* corresponding to the Potts model parameter (i.e. intensity of the neighbourhood correlations).
n	The number of observations. *integer*.
p	The number of groups or colors. *integer*.
iter_nb	The maximum number of iteration of the Gibbs sampler. *integer*.

---

simulPotts_cpp *Potts model simulation*

Description

C++ function that simulates an isotropic Potts model by Gibbs sampling. Experimental.

Usage

```
simulPotts_cpp(W_SR, W_LR, sample, coords, site_order, rho,
         distance_ref, iter_max, cv_criterion, regional, verbose)
```

Arguments

W_SR	the local neighbourhood matrix. *dgCMatrix*. Should be normalized by row (i.e. `rowSums(W_SR)=1`).
W_LR	the regional neighbourhood matrix. *dgCMatrix*. Should contain the categroy of distance.
sample	the initial observations (by rows) for the various groups (by columns). *matrix*.
coords	the voxels coordinates. *matrix*.
site_order	a specific order to go all over the sites. An *integer vector* or a negative number to indicate no specific order.
rho	a *numeric* corresponding to the Potts model parameter (i.e. intensity of the neighbourhood correlations).
distance_ref	the distance defining the several neighbourhood orders relatively to `Wdist_LR`. *numeric vector*.

iter_max	the maximum number of iteration of the Gibbs sampler. *integer*.
cv_criterion	if the maximum absolute difference of the probabilistic membership between two iterations is below critere then the Gibbs sampler is stopped. *numeric*.
regional	should the regional potential be computed ?. *logical*.
verbose	should the convergence criterion be printed at each step ? *logical*.

---

summary-methods *Summary Method for Class "MRIaggr"*

Description

Summarize some information of an object of class "MRIaggr".

Usage

```
## S4 method for signature 'MRIaggr'
show(object)

## S4 method for signature 'MRIaggr'
summary(object, param = FALSE, clinic = FALSE,
        descStats = FALSE, history = FALSE, verbose = optionsMRIaggr("verbose"))
```

Arguments

object	an object of class MRIaggr. REQUIRED.
param	should detailed information be printed for the contrast parameters ? *logical*.
clinic	should detailed information be printed for the clinical attribute ? *logical*.
descStats	should detailed information be printed for the ls_descStats attribute ? *logical*.
history	should thecalc and const methods that have been applied to the object be listed ? *logical*.
verbose	should a summary of all attributes should be displayed or only those which are non-empty ? *logical*.

Value

None.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")
show(MRIaggr.Pat1_red)

# display all elements
summary(MRIaggr.Pat1_red)
```

```
# display non-empty elements
summary(MRIaggr.Pat1_red, verbose = FALSE)

# display a summary of the contrast parameters
summary(MRIaggr.Pat1_red, param = TRUE)

# display a summary of the clinical data
summary(MRIaggr.Pat1_red, clinic = TRUE)

# display a summary of the elements in descStats
summary(MRIaggr.Pat1_red, descStats = TRUE)

# display processing methods previously applied on the object
summary(MRIaggr.Pat1_red, history = TRUE)
```

`supprContrast` *Remove a contrast parameter*

Description

Remove a contrast parameter from a `MRIaggr` object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
supprContrast(object,
         verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`value`	the name of the parameter(s) that should be removed. *character vector*. REQUIRED.
`verbose`	should the execution of the function be traced ? *logical*.

Details

ARGUMENTS:
`value` can be a *numeric vector* indicating the position of the parameters to remove in the `data` slot.

Value

None.

See Also

`allocContrast<-` to allocate a contrast parameter.
`selectParameter` to display the contrast parameters.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## available contrast parameters
selectParameter(MRIaggr.Pat1_red)

## delete two contrast parameters
supprContrast(MRIaggr.Pat1_red) <- c("MTT_t0", "MASK_DWI_t0")

## remaining contrast parameters
selectParameter(MRIaggr.Pat1_red)
```

---

supprDescStats *Remove an element of ls_descStats*

Description

Remove an element from the `ls_descStat` attribute of a `MRIaggr` object.

Usage

```
## S4 replacement method for signature 'MRIaggr'
supprDescStats(object,
         verbose = optionsMRIaggr("verbose")) <- value
```

Arguments

`object`	an object of class `MRIaggr`. REQUIRED.
`value`	the name of the element(s) that should be removed. *character vector*. REQUIRED.
`verbose`	should the execution of the function be traced ? *logical*.

Value

None.

See Also

`allocDescStats<-` to allocate an element to the ls_descStat attribute.
`selectParameter` to display the elements of the ls_descStat attribute.

Examples

```
## load a MRIaggr object
data("MRIaggr.Pat1_red", package = "MRIaggr")

## existing elements in @ls_descStats
selectParameter(MRIaggr.Pat1_red, type = "ls_descStats")

## delete one element in @ls_descStats
supprDescStats(MRIaggr.Pat1_red) <- "index_sauve"

## remaining elements in @ls_descStats
selectParameter(MRIaggr.Pat1_red, "ls_descStats")
```

---

tnorm *Truncated Normal distribution*

Description

Density and random generation for the truncated Normal distribution with mean equal to mean and standard deviation equal to sd before truncation, and truncated on the interval [lower, upper]. For internal use.

Usage

```
dtnorm(x, mean = 0, sd = 1, lower = -Inf, upper = Inf, log = FALSE)
rtnorm(n, mean = 0, sd = 1, lower = -Inf, upper = Inf)
```

Arguments

x	vector of quantiles. *numeric vector*.
n	number of observations. *integer*.
mean	mean vector of means. *numeric*. Default is 0.
sd	vector of standard deviations. *numeric*. Default is 1.
lower	truncation point. *numeric*. Default is -Inf.
upper	upper truncation point. *numeric*. Default is Inf.
log	if TRUE, probabilities p are given as log(p). *logical*. Default is FALSE.

Details

These functions are a copy of the dtnorm and rtnorm of the *msm* package. See the corresponding help page for additional details.

valid *Checking argument validity*

Description

Check the validity of an argument. For internal use.

Usage

```
validCharacter(value, name = as.character(substitute(value)), validLength,
         validValues = "character", refuse.NULL = TRUE, method)

validClass(value, name = as.character(substitute(value)), validClass,
         superClasses = TRUE, method)

validDim_vector(value1, value2, name1 = as.character(substitute(value1)),
         name2 = as.character(substitute(value2)), type = "length", method)

validDim_matrix(value1, value2, name1 = as.character(substitute(value1)),
         name2 = as.character(substitute(value2)), type = "both", method)

validInteger(value, name = as.character(substitute(value)), validLength,
         validValues = NULL, min = NULL, max = NULL,
        refuse.NA = TRUE, refuse.NULL = TRUE, refuse.duplicates = FALSE, method)

validLogical(value, name = as.character(substitute(value)), validLength,
         refuse.NULL = TRUE, refuse.NA = TRUE, method)

validNames(value, name = as.character(substitute(value)),
         validLength = NULL, validValues = NULL, method)

validNumeric(value, name = as.character(substitute(value)), validLength,
         validValues = NULL , min = NULL, max = NULL,
        refuse.NA = TRUE, refuse.NULL = TRUE, refuse.duplicates = FALSE, method)

validPath(value, name = as.character(substitute(value)), method)
```

Arguments

value, value1, value2	the value of the argument(s).
name, name1, name2	the name(s) of the argument(s). *character*.
validLength	the valid lengthes for the argument. *integer vector*.
validClass	the valid classes for the argument. *character vector*.
validValues	the valid values for the argument. *character vector*.

refuse.NA	Is NA an invalid value ? *logical*.
refuse.NULL	Is NULL an invalid value ? *logical*.
refuse.duplicates	Are duplicated values invalid values. *logical*.
type	the operator used to compare the size. *character*. See the details section for more information.
min	the minimum value for the argument. *numeric* or *NULL*.
max	the maximum value for the argument. *numeric* or *NULL*.
superClasses	Should the super-classes of the object's class be also considered or only the object's class ? *logical*.
method	the name of the function that called the valid function. *character*.

Details

validCharacter checks whether value is a character vector.

validClass checks whether value belongs to the correct class of R objects.

validDim_vector checks whether value1 is a vector matching the dimensions of value2. Argument type indicates which operator should be used to measure the size of value2 among : "length", "nrow" and "ncol".

validDim_matrix checks whether value1 is a matrix matching the dimensions of value2. Argument type indicates which operator should be used to measure the size of value2 among : "nrow", "ncol" and "both", the latter indicating to use both nrow and ncol.

validInteger checks whether value is a integer vector.

validLogical checks whether value is a logical vector.

validNames checks whether the names value are valid.

validNumeric checks whether value is a numeric vector.

validPath checks whether value corresponds to a valid path.

---

writeMRI *Write an image file*

Description

Write an image file from an array.

Usage

```
writeMRI(data, filename, format,
       gzipped = TRUE, verbose = optionsMRIaggr("verbose"), size = "NA_integer_")
```

Arguments

data	the data to export. *three dimensional array*. REQUIRED.
filename	the name of the image file. *character*. REQUIRED.
format	the format of the image file. Can be "raw.gz", "analyze", "nifti" or "dicom". REQUIRED.
size	the number of bytes per element in the byte stream. *integer*. See the documentation of the writeBin function for more details.
gzipped	is a character string that enables exportation of compressed (.gz) files.
verbose	is a logical variable that allows text-based feedback during execution of the function.

Details

This function requires to have installed the *oro.nifti* package to work if argument format is set to "analyze" or "nifti".

ARGUMENTS : file argument corresponds to:

- the con argument of the base::readBin function.
- the filename argument of the oro.nifti::readANALYZE function. It should be a pathname to .img or .hdr files without the suffix.
- the filename argument of the oro.nifti::readNIfTI function.

FUNCTION :
This function is a copy of the writeMRI function of the mritc package. It calls writeANALYZE of the oro.nifti package to write analyze files, writeNIfTI of the oro.nifti package to write NIFTI files, and writeBin of the base package to write rawb.gz.

Value

None.

Examples

```
set.seed(10)
X <- rnorm(1000)

## Not run:
writeMRI(array(X, c(10,10,10)), filename = "test", format = "nifti", verbose = TRUE)
# range(array(X, c(10,10,10)) - readMRI("test", format = "nifti")[])

writeMRI(array(X, c(10,10,10)), filename = "test", format = "analyze", verbose = TRUE)
# range(array(X, c(10,10,10)) - readMRI("test", format = "analyze")[])

writeMRI(array(X, c(10,10,10)) , filename = "test", format = "rawb.gz", verbose = TRUE,
         size = "NA_integer_")
# range(array(X, c(10,10,10)) - readMRI("test",
#       format = "rawb.gz", size = "NA_integer_", dim = c(10,10,10)))
```

```
## End(Not run)
```

`writeMRIaggr` *Write an image file*

Description

Write an image file from a `MRIaggr` object.

Usage

```
## S4 method for signature 'MRIaggr'
writeMRIaggr(object, param, num = NULL,
         norm_mu = FALSE, norm_sigma = FALSE, range.coords = NULL,
 default_value = NA, filename, format, gzipped = TRUE,
         verbose = optionsMRIaggr("verbose"), size = "NA_integer_")
```

Arguments

`object`	an `object` of class `MRIaggr`. REQUIRED.
`param`	the contrast parameters to extract. *character vector* or `NULL`. REQUIRED.
`num`	the slices to extract. *numeric vector* or `NULL`.
`norm_mu`	the type of centering to apply on the parameter values. *character*. See the details section.
`norm_sigma`	the type of scaling to apply on the parameter values. *character*. See the details section.
`default_value`	the element used to fill the missing observations. *numeric*.
`range.coords`	the maximum coordinate in each dimension to be considered. *numeric vector* of length 3.
`filename`	the name of the image file. *character*. REQUIRED.
`format`	the format of the image file. Can be `"raw.gz"`, `"analyze"`, `"nifti"` or `"dicom"`. REQUIRED.
`size`	the number of bytes per element in the byte stream. *integer*. See the documentation of the `writeBin` function for more details.
`gzipped`	is a character string that enables exportation of compressed (.gz) files.
`verbose`	is a logical variable that allows text-based feedback during execution of the function.

Details

This function requires to have installed the *oro.nifti* package to work if argument format is set to `"analyze"` or `"nifti"`.

Value

None.

Examples

```
data("MRIaggr.Pat1_red", package = "MRIaggr")

## Not run:
writeMRIaggr(MRIaggr.Pat1_red, param = "DWI_t0", filename = "Pat1_DWIred", format = "nifti")
DWIred <- readMRI("Pat1_DWIred", format = "nifti")
image(DWIred[,,1])

## End(Not run)
```

Index