Package: AnalyzeFMRI (via r-universe)

October 26, 2026

Version 1.1-26

Date 2026-06-23

Title Functions for Analysis of fMRI Datasets Stored in the ANALYZE or 'NIFTI' Format

Author Pierre Lafaye De Micheaux [aut, cre], Jonathan L Marchini [aut], Cleve Moler [cph] (LAPACK/BLAS routines in src), Jack Dongarra [cph] (LAPACK/BLAS routines in src), Richard Hanson [cph] (LAPACK/BLAS routines in src), Sven Hammarling [cph] (LAPACK/BLAS routines in src), Jeremy Du Croz [cph] (LAPACK/BLAS routines in src)

Maintainer Pierre Lafaye De Micheaux <lafaye@unsw.edu.au>

Depends R (>= 4.4.0), R.matlab, fastICA, tcltk

Suggests tkrplot

Description Functions for I/O, visualisation and analysis of functional Magnetic Resonance Imaging (fMRI) datasets stored in the ANALYZE or 'NIFTI' format. Note that the latest version of 'XQuartz' seems to be necessary under MacOS.

License GPL (>= 2)

Copyright For LAPACK/BLAS routines in src (Univ. of Tennessee, Univ. of California Berkeley, NAG Ltd., Courant Institute, Argonne National Lab, and Rice University).

NeedsCompilation yes

Repository https://cran.r-universe.dev

Date/Publication 2026-06-23 14:40:02 UTC

RemoteUrl https://github.com/cran/AnalyzeFMRI

RemoteRef HEAD

RemoteSha 120d9e50faf29bb9b76a2ca51e2884de0a5276a8

Contents

analyze2nifti 3

centering . . . 5
cluster.threshold . . . 6
cov.est . . . 7
diminfo2fps . . . 8
EC.3D . . . 9
eigenvalues . . . 10
f.analyze.file.summary . . . 11
f.analyzeFMRI.gui . . . 11
f.basic.hdr.list.create . . . 12
f.basic.hdr.nifti.list.create . . . 12
f.complete.hdr.nifti.list.create . . . 13
f.ica.fmri . . . 17
f.ica.fmri.gui . . . 19
f.icast.fmri . . . 20
f.icast.fmri.gui . . . 21
f.nifti.file.summary . . . 22
f.plot.ica.fmri . . . 23
f.plot.ica.fmri.jpg . . . 24
f.plot.volume.gui . . . 24
f.read.analyze.header . . . 25
f.read.analyze.slice . . . 28
f.read.analyze.slice.at.all.timepoints . . . 28
f.read.analyze.tpt . . . 29
f.read.analyze.ts . . . 30
f.read.analyze.volume . . . 31
f.read.header . . . 31
f.read.nifti.header . . . 32
f.read.nifti.slice . . . 36
f.read.nifti.slice.at.all.timepoints . . . 37
f.read.nifti.tpt . . . 37
f.read.nifti.ts . . . 38
f.read.nifti.volume . . . 39
f.read.volume . . . 39
f.spectral.summary . . . 40
f.spectral.summary.nifti . . . 41
f.write.analyze . . . 41
f.write.array.to.img.2bytes . . . 42
f.write.array.to.img.8bit . . . 43
f.write.array.to.img.float . . . 44
f.write.list.to.hdr . . . 44
f.write.list.to.hdr.nifti . . . 45
f.write.nifti . . . 46
f.write.nii.array.to.img.2bytes . . . 47
f.write.nii.array.to.img.8bit . . . 47
f.write.nii.array.to.img.float . . . 48
fourDto2D . . . 49
fps2diminfo . . . 49
GaussSmoothArray . . . 50

GaussSmoothKernel . . . 51
ICAspat . . . 52
ICAtemp . . . 52
ijk2xyz . . . 53
magicfield . . . 59
mat34.to.TRSZ . . . 60
mat34.to.TZSR . . . 61
model.2.cov.func . . . 62
model.2.est.gamma . . . 62
N2G . . . 63
N2G.Class.Probability . . . 64
N2G.Density . . . 65
N2G.Fit . . . 65
N2G.Inverse . . . 66
N2G.Likelihood . . . 67
N2G.Likelihood.Ratio . . . 68
N2G.Region . . . 68
N2G.Spatial.Mixture . . . 69
N2G.Transform . . . 70
nifti.quatern.to.mat44 . . . 71
NonLinearSmoothArray . . . 72
orientation . . . 73
Q2R . . . 74
R2Q . . . 74
reduction . . . 75
Sim.3D.GammaRF . . . 76
Sim.3D.GRF . . . 77
SmoothEst . . . 78
st2xyzt . . . 80
threeDto4D . . . 81
Threshold.Bonferroni . . . 82
Threshold.FDR . . . 83
Threshold.RF . . . 84
twoDto4D . . . 85
with_dir . . . 85
xyz2ijk . . . 86
xyzt2st . . . 87

Index **88**

`analyze2nifti` *Create a NIFTI file from an Analyze file*

Description

Create a NIFTI file from an Analyze file.

Usage

```
analyze2nifti(file.in,path.in=".",path.out=NULL,file.out=NULL,is.nii=TRUE,
qform.code=2,sform.code=2,data.type=rawToChar(raw(10)),db.name=rawToChar(raw(18)),
dim.info=rawToChar(raw(1)),dim=NULL,TR=0,slice.code=rawToChar(raw(1)),
xyzt.units=rawToChar(raw(1)),descrip=NULL,aux.file=rawToChar(raw(24)),
intent.name=rawToChar(raw(16)))
```

Arguments

file.in	character, filename of the Analyze file to be read
path.in	character, Directory path from where to take the .hdr,.img,.mat files
path.out	character, Directory path where to write the .hdr/.img or .nii file
file.out	character, filename of the NIFTI file to write (without extension). If NULL, same as file.in
is.nii	logical, if TRUE a NIFTI .nii file will be created, if FALSE a .hdr/.img NIFTI file will be created
qform.code	value in 0,...,4
sform.code	value in 0,...,4
data.type	char[10]. UNUSED in NIFTI-1 but could be filled with what you want
db.name	char[18]. UNUSED in NIFTI-1 but could be filled with what you want
dim.info	MRI slice ordering: This field encode which spatial dimension (1=x, 2=y or 3=z) corresponds to which acquisition dimension for MRI data. In fact, it contains three informations: freq.dim, phase.dim and slice.dim, all squished into the single byte field dim.info (2 bits each, since the values for each field are limited to the range 0..3). The R function fps2diminfo can be used to encode these values from the dim.info byte.
dim	vector (of length 8) of image dimensions. dim[1] specifies the number of dimensions. In NIFTI-1 files, dim[2], dim[3], dim[4] are for space, dim[5] is for time. The 5th dimension (dim[6]) of the dataset, if present (i.e., dim[1]=5 and dim[6] > 1), contains multiple values (for example a vector) to be stored at each spatiotemporal location. Uses of dim[7] and dim[8] are not specified in NIFTI-1 format.
TR	Time Repetition to be stored in pixdim[5]
slice.code	Slice timing order. If this is nonzero, AND if slice.dim is nonzero, AND if slice.duration is positive, indicates the timing pattern of the slice acquisition. The following codes are defined: 0 (NIFTI SLICE UNKNOWN), 1 (NIFTI SLICE SEQ INC), 2 (NIFTI SLICE SEQ DEC), 3 (NIFTI SLICE ALT INC), 4 (NIFTI SLICE ALT DEC)
xyzt.units	Units of pixdim[2:5]. Bits 1..3 of xyzt.units specify the (same) space unit of pixdim[2:4]. Bits 4..6 of xyzt.units specify the time unit of pixdim[5]. See 'xyzt-units.txt' in the niftidoc directory of the source package. The R function st2xyzt can be used to encode these values from the xyzt.units byte.
descrip	char[80]. This field may contain any text you like
aux.file	char[24]. This field is used to store an auxiliary filename.

intent.name	char[16]. name or meaning of data. If no data name is implied or needed, intent.name[1] should be set to 0.

Value

Nothing is returned. The NIFTI file is created in the specified path.out directory (default is current directory).

Examples

```
tmpdir <- tempdir()
analyze2nifti(path.in = system.file(package = "AnalyzeFMRI"),
              path.out = tmpdir, file.in = "example",
              file.out = "nifti-tmp", is.nii = TRUE)
unlink(tmpdir) # tidy up
```

---

centering *centering*

Description

This function center the data in the two dimensions, the first dimension being indicated by col.first argument

Usage

```
centering(X,col.first=TRUE)
```

Arguments

X	a matrix of size tm x vm which contains the functionnal images
col.first	Logical. Center the columns or the rows first

Value

Xcentred	the double centered matrix

See Also

reduction

Examples

```
X <- matrix(rnorm(5 * 4), nrow = 5, ncol = 4)
Xcentred <- centering(X, col.first = TRUE)$Xcentred
```

`cluster.threshold` *Cluster threshold an array.*

Description

Calculate contiguous clusters of locations in a 3D array that are above some threshold and with some minimum size.

Usage

```
cluster.threshold(x, nmat = NULL, level.thr = 0.5, size.thr)
```

Arguments

`x`	A 3D array
`nmat`	A matrix with 3 columns specifying the neighbourhood system. Default is 6 nearest neighbours in 3D.
`level.thr`	The level at which to threshold the array values. Default is 0.5 and is designed to cluster 0-1 arrays.
`size.thr`	The cluster size threshold.

Value

Returns an array of the same size as x with a 1 at all locations which have a value above level.thr and are in a cluster of similiar locations with size greater than size.thr.

Author(s)

J. L. Marchini

Examples

```
x <- array(0, dim = c(64, 64, 21))
x[10:20, 10:20, 1:5] <- 1
x[30:40, 30:40, 6:7] <- 1
x[50, 50, 8:9] <- 1

a <- cluster.threshold(x, size.thr = 400)
sum(x) ## should be 849
sum(a) ## should be 605
```

cov.est *Estimates the covariance between neighbouring voxels*

Description

Estimates the covariance between neighbouring voxels using a specified neighbourhood system.

Usage

```
cov.est(mat, mask, nmat)
```

Arguments

mat	3D array of voxel values.
mask	Array with sam dimension as mat that is 1/0 for voxels to be included/excluded.
nmat	Neighbourhood matrix.

Value

The estimated covariance

Author(s)

J. L. Marchini

Examples

```
ksize <- 9
d <- c(64, 64, 21)
FWHM <- 9
sigma <- diag(FWHM^2, 3) / (8 * log(2))
voxdim <- c(2, 2, 4)

filtermat <- GaussSmoothKernel(voxdim, ksize, sigma)

mask <- array(1, dim = d)
num.vox <- sum(mask)

mat <- Sim.3D.GRF(d = d, voxdim = voxdim, sigma = sigma,
                  ksize = ksize, mask = mask, type = "field")$mat

nmat <- expand.grid(-1:1, -1:1, -1:1)
nmat4 <- nmat[c(11, 13, 15, 17), ]

cov <- cov.est(mat, mask, nmat4)
```

diminfo2fps *diminfo2fps*

Description

Extract freq.dim, phase.dim and slice.dim fields from the one byte dim.info field of a NIFTI header file.

Usage

```
diminfo2fps(dim.info)
```

Arguments

dim.info	dim.info field of a NIFTI header file

Value

A list containing `freq.dim`, `phase.dim` and `slice.dim` fields.

These are provided to store some extra information that is sometimes important when storing the image data from an FMRI time series experiment. (After processing such data into statistical images, these fields are not likely to be useful.) These fields encode which spatial dimension (1,2, or 3) corresponds to which acquisition dimension for MRI data.

Examples:
Rectangular scan multi-slice EPI:
`freq_dim` = 1 `phase_dim` = 2 `slice_dim` = 3 (or some permutation)
Spiral scan multi-slice EPI:
`freq_dim` = `phase_dim` = 0 `slice_dim` = 3 since the concepts of frequency- and phase-encoding directions don't apply to spiral scan.

The fields `freq.dim`, `phase.dim`, `slice.dim` are all squished into the single byte field `dim.info` (2 bits each, since the values for each field are limited to the range 0..3). This unpleasantness is due to lack of space in the 348 byte allowance.

See Also

fps2diminfo

Examples

```
dim.info <- f.read.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))$dim.info
diminfo2fps(dim.info)
```

EC.3D *Expected Euler Characteristic for a 3D Random Field*

Description

Calculates the Expected Euler Characteristic for a 3D Random Field thesholded a level u.

Usage

```
EC.3D(u, sigma, voxdim = c(1, 1, 1), num.vox, type = c("Normal", "t"), df = NULL)
```

Arguments

u	The threshold for the field.
sigma	The spatial covariance matrix of the field.
voxdim	The dimensions of the cuboid 'voxels' upon which the discretized field is observed.
num.vox	The number of voxels that make up the field.
type	The marginal distribution of the Random Field (only Normal and t at present).
df	The degrees of freedom of the t field.

Details

The Euler Characteristic χ_u (Adler, 1981) is a topological measure that essentially counts the number of isolated regions of the random field above the threshold u minus the number of 'holes'. As u increases the holes disappear and χ_u counts the number of local maxima. So when u becomes close to the maximum of the random field Z_{max} we have that

$$P(\text{reject}H_0|H_0\text{true}) = P(Z_{\text{max}}) = P(\chi_u > 0) \approx E(\chi_u)$$

where H_0 is the null hypothesis that there is no signicant positive actiavtion/signal present in the field. Thus the Type I error of the test can be controlled through knowledge of the Expected Euler characteristic.

Value

The value of the expected Euler Characteristic.

Author(s)

J. L. Marchini

References

Adler, R. (1981) *The Geometry of Random Fields.*. New York: Wiley. Worlsey, K. J. (1994) Local maxima and the expected euler characteristic of excursion sets of χ^2, f and t fields. *Advances in Applied Probability*, **26**, 13-42.

See Also

Threshold.RF

Examples

```
EC.3D(4.6, sigma = diag(1, 3), voxdim = c(1, 1, 1), num.vox = 10000)

EC.3D(4.6, sigma = diag(1, 3), voxdim = c(1, 1, 1), num.vox = 10000, type = "t", df = 100)
```

---

eigenvalues *eigenvalues*

Description

This function computes the eigenvalues of a centered and reduced data matrix

Usage

```
eigenvalues(X, draw = FALSE)
```

Arguments

X	a matrix of size tm x vm which contains the functionnal images centered and reduced
draw	Logical. Should we plot the eigenvalues

Value

A list containing

eigenvalues	vector of the eigenvalues

Examples

```
X <- matrix(rnorm(5 * 4), nrow = 5, ncol = 4)
Xc <- centering(X, col.first = TRUE)$Xcentred
Xcr <- reduction(Xc, row.red = FALSE)$Xred
eigencr <- eigenvalues(Xcr, draw = FALSE)$eigenvalues
```

f.analyze.file.summary

prints summary of .img file contents

Description

Prints a summary of the contents of an ANALYZE .img file using the associated .hdr header file.

Usage

```
f.analyze.file.summary(file)
```

Arguments

file	The location of .img file to be read

Value

A print out containing information about the .img file. This includes File name, Data Dimension, X dimension, Y dimension, Z dimension, Time dimension, Voxel dimensions, Data type

See Also

f.read.analyze.header, f.read.analyze.slice, f.read.analyze.slice.at.all.timepoints, f.read.analyze.ts, f.write.analyze, f.read.analyze.volume, f.spectral.summary, f.write.array.to.img.2byt f.write.array.to.img.float, f.write.list.to.hdr, f.basic.hdr.list.create

Examples

```
f.analyze.file.summary(system.file("example.img", package = "AnalyzeFMRI"))
```

f.analyzeFMRI.gui

starts AnalyzeFMRI GUI

Description

Starts an R/tk interfaced GUI that allows the user to explore an fMRI dataset stored in an ANALYZE format file using the functions of the AnalyzeFMRI package.

Usage

```
f.analyzeFMRI.gui()
```

Value

No value is returned

`f.basic.hdr.list.create`

creates basic .hdr list in ANALYZE format

Description

Creates a basic list that can be used to write a .hdr file

Usage

```
f.basic.hdr.list.create(X, file.hdr)
```

Arguments

`X`	Array that is to be converted to a .img file
`file.hdr`	Name of the .hdr file that will be created

Value

Returns a list of all the fields needed to create a .hdr file (see the functions code for details).

See Also

`f.write.list.to.hdr`, `f.analyze.file.summary`

Examples

```
a <- array(rnorm(20 * 30 * 40 * 3), dim = c(20, 30, 40, 3))
file <- "temp.hdr"
hdr.list <- f.basic.hdr.list.create(a, file)
```

`f.basic.hdr.nifti.list.create`

creates basic .hdr list in NIFTI format

Description

Creates a basic list that can be used to write a .hdr file or the header part of a .nii file

Usage

```
f.basic.hdr.nifti.list.create(dim.mat, file)
```

Arguments

dim.mat	dim.mat vector of the dimensions of the image array associated with the header file to be written
file	file Name of the .hdr file that will be contained in the file field of the header

Value

Returns a list of all the fields needed to create a .hdr file (see the function code for details).

See Also

f.write.list.to.hdr.nifti, f.nifti.file.summary

Examples

```
dim.mat <- c(20, 30, 40, 3)
file <- "temp.hdr"
hdr.nifti <- f.basic.hdr.nifti.list.create(dim.mat, file)
```

---

f.complete.hdr.nifti.list.create

creates complete .hdr list in NIFTI format

Description

Creates a complete list that can be used to write a .hdr file or the header part of a .nii file

Usage

```
f.complete.hdr.nifti.list.create(file,dim.info=character(1),dim,
intent.p1=single(1),intent.p2=single(1),intent.p3=single(1),intent.code=integer(1),
datatype=integer(1),bitpix=integer(1),slice.start=integer(1),pixdim=single(8),
scl.slope=single(1),scl.inter=single(1),slice.end=integer(1),slice.code=character(1),
xyzt.units=character(1),cal.max=single(1),cal.min=single(1),slice.duration=single(1),
toffset=single(1),descrip=paste(rep(" ", 80), sep = "", collapse = ""),
aux.file=paste(rep(" ", 24), sep = "", collapse =""),qform.code=integer(1),
sform.code=integer(1),quatern.b=single(1),quatern.c=single(1),quatern.d=single(1),
qoffset.x=single(1),qoffset.y=single(1),qoffset.z=single(1),srow.x=single(4),
srow.y=single(4),srow.z=single(4),
intent.name=paste(rep(" ", 16), sep = "", collapse = ""))
```

Arguments

file	The .hdr filename. If file extension is ".nii", this will create a header file for a ".nii" NIFTI file, else for a .hdr/.img NIFTI pair

`dim.info` MRI slice ordering: This field encode which spatial dimension (1=x, 2=y, or 3=z) corresponds to which acquisition dimension for MRI data. In fact, it contains three informations: `freq.dim`, `phase.dim` and `slice.dim`, all squished into the single byte field dim.info (2 bits each, since the values for each field are limited to the range 0..3). The R function `fps2diminfo` can be used to encode these values into the `dim.info` character byte.

`dim` vector (of length 8) of image dimensions. `dim[1]` specifies the number of dimensions. In NIFTI-1 files, `dim[2]`, `dim[3]`, `dim[4]` are for space, `dim[5]` is for time. The 5th dimension (`dim[6]`) of the dataset, if present (i.e., `dim[1]`=5 and `dim[6]` > 1), contains multiple values (for example a vector) to be stored at each spatio-temporal location. Uses of `dim[7]` and `dim[8]` are not specified in NIFTI-1 format.

`intent.p1` 1st intent parameter: first auxiliary parameter for a possible statistical distribution specified in `intent.code`

`intent.p2` 2nd intent parameter: second auxiliary parameter for a possible statistical distribution specified in `intent.code`

`intent.p3` 3rd intent parameter: third auxiliary parameter for a possible statistical distribution specified in `intent.code`

`intent.code` NIFTI INTENT code: if 0, this is a raw dataset; if in range 2...24, this indicates that the numbers in the dataset should be interpreted as being drawn from a given distribution. Most such distributions have auxiliary parameters (given with `intent.p`?); if in range 1001...1011, this is an other meaning. See file '`intent-code.txt`' in the niftidoc directory of the source package. If the dataset DOES NOT have a 5th dimension (`dim[1]`=4), then the auxiliary parameters are the same for each voxel, and are given in header fields `intent.p1`, `intent.p2`, and `intent.p3`. If the dataset DOES have a 5th dimension (`dim[1]`=5), then the auxiliary parameters are different for each voxel.

`datatype` integer indicator of data storage type for each voxel. This could be 2 (unsigned char), 4 (signed short), 8 (signed int), 16 (32 bit float), 32 (64 bit complex = two 32 bit floats), 64 (64 bit float = double), 128 (3 8 bit bytes), 256 (signed char), 512 (unsigned short), 768 (unsigned int), 1024 (signed long long), 1280 (unsigned long long), 1536 (128 bit float = long double), 1792 (128 bit complex = 2 64 bit floats), 2048 (256 bit complex = 2 128 bit floats).

`bitpix` the number of bits per voxel. This field MUST correspond with the datatype field. The total number of bytes in the image data is `dim[2]`* ... * `dim[dim[1]+1]` * `bitpix` / 8

`slice.start` Indicates the start of the slice acquisition pattern, when `slice.code` is nonzero. These values are present to allow for the possible addition of "padded" slices at either end of the volume, which don't fit into the slice timing pattern. If there are no padding slices, then `slice.start`=0 and `slice.end=dim[slice.dim+1]-1` are the correct values. For these values to be meaningful, slice.start must be non-negative and slice.end must be greater than slice.start.

`pixdim` vector (of length 8). Grid spacings. When reading a NIFTI-1 header, `pixdim[1]` stores `qfac` (which is either -1 or 1). If `pixdim[1]`=0 (which should not occur), we take `qfac`=1. `pixdim[2]`, `pixdim[3]` and `pixdim[4]` give the voxel width along dimension x, y and z respectively. `pixdim[5]` gives the time step (=Time

Repetition=TR). The units of pixdim can be specified with the xyzt.units field.

scl.slope — Data scaling: If the scl.slope field is nonzero, then each voxel value in the dataset should be scaled as $y =$ scl.slope $* x +$ scl.inter, where x = voxel value stored and y = "true" voxel value

scl.inter — Data scaling: offset. Idem above.

slice.end — Indicates the end of the slice acquisition pattern, when slice.code is nonzero. These values are present to allow for the possible addition of "padded" slices at either end of the volume, which don't fit into the slice timing pattern. If there are no padding slices, then slice.start=0 and slice.end=dim[slice.dim+1]-1 are the correct values. For these values to be meaningful, slice.start must be non-negative and slice.end must be greater than slice.start.

slice.code — Slice timing order. If this is nonzero, AND if slice.dim is nonzero, AND if slice.duration is positive, indicates the timing pattern of the slice acquisition. The following codes are defined: 0 (NIFTI SLICE UNKNOWN), 1 (NIFTI SLICE SEQ INC), 2 (NIFTI SLICE SEQ DEC), 3 (NIFTI SLICE ALT INC), 4 (NIFTI SLICE ALT DEC)

xyzt.units — Units of pixdim[2:5]. Bits 1..3 of xyzt.units specify the (same) space unit of pixdim[2:4]. Bits 4..6 of xyzt.units specify the time unit of pixdim[5]. See 'xyzt-units.txt' in the niftidoc directory of the source package. The R function st2xyzt can be used to encode these values into the xyzt.units byte.

cal.max — Maximum display intensity (white) corresponds to dataset value cal.max. Dataset values above cal.max should display as white. cal.min and cal.max only make sense when applied to scalar-valued datasets (i.e., dim[1] < 5 or dim[6] = 1).

cal.min — Minimum display intensity (black) corresponds to dataset value cal.min. Dataset values below cal.min should display as black.

slice.duration — Time for 1 slice. If this is positive, AND if slice.dim is nonzero, indicates the amount of time used to acquire 1 slice.

toffset — Time axis shift: The toffset field can be used to indicate a nonzero start point for the time axis. That is, time point m is at $t =$ toffset $+ m *$ pixdim[5] for m=1, ..., dim[5]-1.

descrip — char[80]. This field may contain any text you like

aux.file — char[24]. This field is used to store an auxiliary filename.

qform.code — NIFTI code (in 0, ... ,4). 0: Arbitrary coordinates; 1: Scanner-based anatomical coordinates; 2: Coordinates aligned to another file's, or to anatomical "truth" (coregistration); 3: Coordinates aligned to Talairach-Tournoux Atlas; 4: MNI 152 normalized coordinates

sform.code — NIFTI code (in 0, ... ,4) with the same meaning as qform codes. The basic idea behind having two coordinate systems is to allow the image to store information about (1) the scanner coordinate system used in the acquisition of the volume (in the qform) and (2) the relationship to a standard coordinate system - e.g. MNI coordinates (in the sform). The qform allows orientation information to be kept for alignment purposes without losing volumetric information, since the qform only stores a rigid-body transformation (rotation and translation) which

preserves volume. On the other hand, the `sform` stores a general affine transformation (shear, scale, rotation and translation) which can map the image coordinates into a standard coordinate system, like Talairach or MNI, without the need to resample the image. By having both coordinate systems, it is possible to keep the original data (without resampling), along with information on how it was acquired (`qform`) and how it relates to other images via a standard space (`sform`). This ability is advantageous for many analysis pipelines, and has previously required storing additional files along with the image files. By using NIfTI-1 this extra information can be kept in the image files themselves. Note: the `qform` and `sform` also store information on whether the coordinate system is left-handed or right-handed and so when both are set they must be consistent, otherwise the handedness of the coordinate system (often used to distinguish left-right order) is unknown and the results of applying operations to such an image are unspecified.

`quatern.b` Quaternion b param. These b,c,d quaternion parameters encode a rotation matrix used when `qform.code` > 0 to obtain a rigid transformation that maps voxel indices (i, j, k) to spatial coordinates (x, y, z), typically anatomical coordinates assigned by the scanner. This transformation (*Method 2* in the '`nifti1.h`' documentation) is generated using also the voxel dimensions (`pixdim[1:4]`) and a 3D shift, i.e. a translation, (`qoffset.*`)

`quatern.c` Quaternion c param

`quatern.d` Quaternion d param

`qoffset.x` Quaternion x shift. If the (0020,0032) DICOM attribute is extracted into (px, py, pz), then `qoffset.x` = $-px$, `qoffset.y` = $-py$ and `qoffset.z` = pz is a reasonable setting when `qform.code`=NIFTI XFORM SCANNER ANAT.

`qoffset.y` Quaternion y shift

`qoffset.z` Quaternion z shift

`srow.x` vector of length 4. 1st row affine transform. These `srow.*` parameters contain an affine (non-rigid) transformation (*Method 3* in the '`nifti1.h`' documentation) that maps voxel indices (i, j, k) to spatial coordinates (x, y, z).

`srow.y` vector of length 4. 2nd row affine transform

`srow.z` vector of length 4. 3rd row affine transform

`intent.name` char[16]. name or meaning of data. If no data name is implied or needed, `intent.name[1]` should be set to 0.

Value

Returns a list of all the fields needed to create a .hdr file (see the function code for details).

See Also

`f.basic.hdr.nifti.list.create`, `f.write.list.to.hdr.nifti`, `f.nifti.file.summary`

Examples

```
dim.mat <- c(20, 30, 40, 3)
dim <- c(length(dim.mat), dim.mat, rep(0, 7 - length(dim.mat)))
filename <- "temp.hdr"
hdr.nifti <- f.complete.hdr.nifti.list.create(file = filename, dim = dim)
```

`f.ica.fmri` *Applies Spatial ICA (Independent Component Analysis) to fMRI datasets*

Description

Decomposes an fMRI dataset into a specified number of Spatially Independent Components maps and associated time-courses using the FastICA algorithm

Usage

```
f.ica.fmri(file.name, n.comp, norm.col=TRUE, fun="logcosh", maxit=1000,
alg.type="parallel", alpha=1, tol=1e-04, mask.file.name=NULL, slices=NULL)
```

Arguments

`file.name`	path to fMRI dataset (ANALYZE format .img file)
`n.comp`	number of components to extract
`norm.col`	a logical value indicating whether each voxel time series should be standardised to have zero mean and unit variance before the ICA algorithm is applied (default=TRUE recommended in practice)
`fun`	the functional form of the G function used in the approximation to negentropy (see details)
`maxit`	maximum number of iterations to perform
`alg.type`	if `alg.typ=="deflation"` the components are extracted one at a time (the default). if `alg.typ=="parallel"` the components are extracted simultaneously.
`alpha`	constant in range [1,2] used in approximation to negentropy when `fun=="logcosh"`
`tol`	a positive scalar giving the tolerance at which the un-mixing matrix is considered to have converged.
`mask.file.name`	Optional path to file containing a 0/1 mask for the dataset
`slices`	Optional vector of slices to be included

Details

The fMRI dataset is rearranged into a 2-dimensional data matrix X, where the column vectors are voxel time-series. A mask is used to specify which voxels are included. If this is not supplied by the user then a mask is constructed automatically using a 10% intensity threshold.

The data matrix is considered to be a linear combination of non-Gaussian (independent) components i.e. X = AS where rows of S contain the independent components and A is a linear mixing matrix. In short ICA attempts to 'un-mix' the data by estimating an un-mixing matrix U where UX = S.

Under this generative model the measured 'signals' in X will tend to be 'more Gaussian' than the source components (in S) due to the Central Limit Theorem. Thus, in order to extract the independent components/sources we search for an un-mixing matrix U that maximizes the non-gaussianity of the sources.

In FastICA, non-gaussianity is measured using approximations to negentropy (J) which are more robust than kurtosis based measures and fast to compute.

The approximation takes the form

$J(y) = [EG(y) - EG(v)]^2$ where v is a N(0,1) r.v

The following choices of G are included as options $G(u) = \frac{1}{\alpha} \log \cosh(\alpha u)$ and $G(u) = -\exp(\frac{-u^2}{2})$

The FastICA algorithm is used to 'un-mix' the data and recover estimates of the mixing matrix A and the source matrix S. Rows of the source matrix S represent spatially independent components of the dataset (these are arranged spatially in the output). Columns of A contain the associated time-courses of the independent components.

Pre-processing involves removing the mean of each row of the data matrix and (optionally) standardizing the columns of the data matrix to have zero mean and unit variance.

All computations are done using C code. This avoids reading the entire dataset into R and thus saves memory space.

Value

A list containing the following components

`A`	estimated mixing matrix
`S`	estimated source matrix that has been rearranged spatially i.e. S is a 4-D array and S[,,,i] contains the 3-D map of the ith component
`file`	the name of the data file
`mask`	the name of the mask file

Author(s)

J L Marchini <marchini@stats.ox.ac.uk> and C Heaton <chrisheaton99@yahoo.com>

References

A. Hyvarinen and E. Oja (2000) Independent Component Analysis: Algorithms and Applications, Neural Networks, 13(4-5):411-430

Beckmann C. (2000) Independent Component Analysis for fMRI. First Year D.Phil Report, Dept. of Engineering Science, University of Oxford.

See Also

f.ica.fmri.gui,f.plot.ica.fmri

`f.ica.fmri.gui` *tcltk GUI to apply ICA to fMRI datasets*

Description

The GUI provides a quick and easy to use interface for applying spatial ICA to fMRI datasets. Computations are done in C for speed and low memory usage.

Usage

```
f.ica.fmri.gui()
```

Details

The user is required to enter the location of the fMRI dataset (stored in the ANALYZE format) and (optionally) a mask for the dataset. If no mask is supplied then an option to create mask is available. There is option to normalize the columns of the data matrix and to exclude the top and bottom slices (which are sometimes affected by the registration procedures).

Once completed, the user has the option of saving the results to an R object or viewing the estimated components. The slices of each component map are plotted sequentially in a grid followed by the components associated time-course and that time-courses periodogram/power spectrum.

Value

`User named R object (optional)`
: Once completed, the user has the option of saving the results to an R object named by the user.

Author(s)

J L Marchini <marchini@stats.ox.ac.uk> and C Heaton <chrisheaton99@yahoo.com>

See Also

f.ica.fmri,f.plot.ica.fmri

`f.icast.fmri` *Applies Spatial or Temporal ICA (Independent Component Analysis) to fMRI NIFTI datasets*

Description

Decomposes an fMRI dataset into a specified number of Spatially or Temporally Independent Components maps and associated time-courses using the FastICA algorithm

Usage

```
f.icast.fmri(foncfile, maskfile, is.spatial, n.comp.compute = TRUE,
             n.comp = 0, hp.filter = TRUE, verbose = TRUE, path.out = "")
```

Arguments

`foncfile`	path and filename to fMRI dataset (NIFTI format .img or .nii file)
`maskfile`	path and filename to fMRI maskfile (0 and 1 values to determine if you are inside or outside the brain) dataset (NIFTI format .img or .nii file)
`is.spatial`	Logical. Should we perform a spatial or temporal ICA.
`n.comp.compute`	Logical. Should we estimate the number of components to exatract. If FALSE, n.comp value (>0) should be provided
`n.comp`	number of components to extract
`hp.filter`	Logical. Should we perform high-pass filtering on the data
`verbose`	`verbose` logical to display information on the progress of computations
`path.out`	The path where to write the file

Details

TODO!!! The fMRI dataset is rearranged into a 2-dimensional data matrix X, where the column vectors are voxel time-series. A mask is used to specify which voxels are included. If this is not supplied by the user then a mask is constructed automatically using a 10% intensity threshold.

The data matrix is considered to be a linear combination of non-Gaussian (independent) components i.e. X = AS where rows of S contain the independent components and A is a linear mixing matrix. In short ICA attempts to 'un-mix' the data by estimating an un-mixing matrix U where UX = S.

Under this generative model the measured 'signals' in X will tend to be 'more Gaussian' than the source components (in S) due to the Central Limit Theorem. Thus, in order to extract the independent components/sources we search for an un-mixing matrix U that maximizes the non-gaussianity of the sources.

In FastICA, non-gaussianity is measured using approximations to negentropy (J) which are more robust than kurtosis based measures and fast to compute.

The approximation takes the form

$J(y) = [EG(y) - EG(v)]^2$ where v is a N(0,1) r.v

The following choices of G are included as options $G(u) = \frac{1}{\alpha} \log \cosh(\alpha u)$ and $G(u) = -\exp(\frac{-u^2}{2})$

The FastICA algorithm is used to 'un-mix' the data and recover estimates of the mixing matrix A and the source matrix S. Rows of the source matrix S represent spatially independent components of the dataset (these are arranged spatially in the output). Columns of A contain the associated time-courses of the independent components.

Pre-processing involves removing the mean of each row of the data matrix and (optionally) standardizing the columns of the data matrix to have zero mean and unit variance.

All computations are done using C code. This avoids reading the entire dataset into R and thus saves memory space.

Value

Nothing for the moment ... TODO!! The spatial and temporal components are written on disk

Author(s)

P Lafaye de Micheaux <plafaye@club.fr>

References

A. Hyvarinen and E. Oja (2000) Independent Component Analysis: Algorithms and Applications, Neural Networks, 13(4-5):411-430

Beckmann C. (2000) Independent Component Analysis for fMRI. First Year D.Phil Report, Dept. of Engineering Science, University of Oxford.

See Also

`f.icast.fmri.gui`

`f.icast.fmri.gui` *tcltk GUI to apply Spatial or Temporal ICA to fMRI NIFTI datasets*

Description

The GUI provides a quick and easy to use interface for applying spatial or temporal ICA to fMRI NIFTI datasets. Computations WILL BE (NOT YET IMPLEMENTED) done in C for speed and low memory usage.

Usage

```
f.icast.fmri.gui()
```

Details

The user is required to enter the location of the fMRI dataset (stored in the NIFTI format) and (optionally) a mask for the dataset. If no mask is supplied then an option to create mask is available. TODO!!

Once completed, the user has the option of saving the results to an R object or viewing the estimated components. The slices of each component map are plotted sequentially in a grid followed by the components associated time-course and that time-courses periodogram/power spectrum.TODO!!

Value

`User named R object (optional)`

Once completed, the user has the option of saving the results to an R object named by the user.TODO!!

Author(s)

P Lafaye de Micheaux <plafaye@club.fr>

See Also

`f.icast.fmri`,`f.ica.fmri.gui`

`f.nifti.file.summary` *prints summary of .img file contents*

Description

Prints a summary of the contents of a NIFTI .img file using the associated .hdr header file.

Usage

```
f.nifti.file.summary(file)
```

Arguments

`file`	The location of .img file to be read

Value

A print out containing information about the .img file. This includes File name, Data Dimension, X dimension, Y dimension, Z dimension, Time dimension, Voxel dimensions, Data type

See Also

`f.read.nifti.header`, `f.read.nifti.slice`, `f.read.nifti.slice.at.all.timepoints`, `f.read.nifti.ts`, `f.write.nifti`, `f.read.nifti.volume`, `f.spectral.summary.nifti`, `f.write.array.to.img.2bytes`, `f.write.array.to.img.float`, `f.write.list.to.hdr.nifti`, `f.basic.hdr.nifti.list.create`

Examples

```
f.nifti.file.summary(system.file("example-nifti.img", package = "AnalyzeFMRI"))
```

`f.plot.ica.fmri` *Plots a specified component from the output of f.ica.fmri*

Description

Plots a specified component from the output of f.ica.fmri

Usage

```
f.plot.ica.fmri(obj.ica, comp, cols)
```

Arguments

`obj.ica`	R object returned by the function f.ica.fmri
`comp`	number of the component to plot
`cols`	optional vector of colours to use for plotting

Details

The slices of the specified component map are plotted sequentially in a grid followed by the components associated time-course and that time-courses periodogram/power spectrum

Value

No return value, just plot graphs

Author(s)

J L Marchini <marchini@stats.ox.ac.uk> and C Heaton <chrisheaton99@yahoo.com>

See Also

`f.ica.fmri`,`f.ica.fmri.gui`

f.plot.ica.fmri.jpg *Plot the components of the ouput of f.ica.fmri to a series of jpeg files*

Description

This function allows the compact graphical storage of the output of a spatial ICA decomposition of an fMRI dataset. each component is plotted to a jpeg.

Usage

```
f.plot.ica.fmri.jpg(ica.obj, file="", cols=heat.colors(100), width=700, height=700)
```

Arguments

ica.obj	Object that is the output of f.ica.fmri
file	The component i will be plotted in file 'file'.comp.i.jpeg
cols	Optional colour vector for plotting the components
width	Width of jpeg images
height	Height of jpeg images

Value

No return value, just write in jpeg files

Author(s)

J L Marchini

See Also

f.ica.fmri, jpeg

f.plot.volume.gui *tcltk GUI to display FMRI or MRI images*

Description

tcltk GUI to display FMRI or MRI images. This GUI is very usefull, for example, for investigating the results of an ICA performed with f.icast.fmri.gui(). But it can also be used to display an MRI or an FMRI image

Usage

```
f.plot.volume.gui(array.fonc = NULL, hdr.fonc = NULL)
```

Arguments

`array.fonc`	An optionnal array containing the MRI values
`hdr.fonc`	If array.fonc is not NULL, one must provide a list 'hdr.fonc' with a 'pixdim' field containing a vector of length 4 with the pixel dimensions

Details

One has the possibility to enter either a filename (with its path) or directly an R object in the file field. This function will only work if package 'tkrplot' is installed. It seems this package is not available for arm64 Macs.

Value

Nothing. Opens our Graphical User Interface

Author(s)

P Lafaye de Micheaux <plafaye@club.fr>

See Also

`f.icast.fmri.gui`

Examples

```
if (interactive()) f.plot.volume.gui()
```

`f.read.analyze.header` *read Analyze header file*

Description

Reads the ANALYZE image format .hdr header file into a list.

Usage

```
f.read.analyze.header(file)
```

Arguments

`file`	The .hdr file to be read

Value

A list containing the information in the fields of the .hdr file.

`file.name` name of the .img file

`swap` TRUE or FALSE variable indicating whether files are big or little endian

`....... HEADER KEY ..........`

`sizeof.hdr` This field implies that Analyze format was originally intended to be extensible, but in practice this did not happen, and instead the file size (and hence the value of this field) is 348. Software commonly tests the value in this field to detect whether the byte ordering is Big-Endian or Little-Endian.

`data.type` character vector indicating data storage type for each voxel

`db.name` database name

`extents` Should be 16384, the image file is created as contiguous with a minimum extent size

`session.error`

`regular` Must be 'r' to indicate that all images and volumes are the same size

`hkey.un0`

`....... IMAGE DIMENSION ..........`

`dim` vector of the image dimensions: dim[1] Number of dimensions in database, usually 4; dim[2] Image X dimension (slice width), number of pixels in an image row; dim[3] Image Y dimension (slice height), number of pixel rows in slice; dim[4] Volume Z dimension (volume depth), number of slices in a volume; dim[5] Time points, number of volumes in database dim[6] ?? dim[7] ?? dim[8] ??

`vox.units` 3 characters to specify the spatial units of measure for a voxel (mm., um., cm.)

`cal.units` 7 characters to specify the name of the calibration unit i.e. pixel,voxel

`unused1` ??

`datatype` integer indicator of data storage type for this image: 0 (None or Unknown), 1 (Binary), 2 (Unsigned-char), 4 (Signed-short), 8 (Signed-int), 16 (float), 32 (Complex), 64 (Double), 128 (RGB), 255 (All)

`bitpix` number of bits per pixel: 1 (packed binary, slices begin on byte boundaries), 8 (unsigned char, gray scale), 16 (signed short), 32 (signed integers or float), or 24 (RGB, 8 bits per channel)s

`dim.un0` unused

`pixdim` Parallel vector to dim, giving real world measurements in mm. and ms. pixdim[1]: ?? pixdim[2]: voxel width in mm. pixdim[3]: voxel height in mm. pixdim[4]: slice thickness (interslice distance) in mm. pixdim[5]: timeslice in ms. pixdim[6]: ?? pixdim[7]: ?? pixdim[8]: ??

`vox.offset` byte offset in the .img file at which voxels start. This value can be negative to specify that the absolute value is applied for every image voxel in the file

funused1	specify the range of calibration values. SPM extends the Analyze format by using a scaling factor for the image from the header
funused2	SPM2 image intensity zero intercept
funused3	??
cal.max	Max display intensity, calibration value, values of 0.0 for both fields imply that no calibration max and min values are used
cal.min	Min display intensity, calibration value
compressed	??
verified	??
glmax	The maximum pixel values for the entire database
glmin	The minimum pixel values for the entire database

..... DATA HISTORY .....

descrip	any text you like
aux.file	auxiliary filename
orient	planar slice orientation for this dataset: 0 transverse unflipped; 1 coronal unflipped; 2 sagittal unflipped; 3 transverse flipped; 4 coronal flipped; 5 sagittal flipped
originator	image central voxel coordinates. SPM uses this Analyze header field in an unorthodox way. originator[1]: SPM99 X near Anterior Commissure, originator[2]: SPM99 Y near Anterior Commissure, originator[3]: SPM99 Z near Anterior Commissure, originator[4]:??, originator[5]:??
generated	??
scannum	??
patient.id	??
exp.date	??
exp.time	??
hist.un0	??
views	??
vols.added	??
start.field	??
field.skip	??
omax	??
omin	??
smax	??
smin	??

See Also

f.analyze.file.summary

Examples

```
f.read.analyze.header(system.file("example.hdr", package="AnalyzeFMRI"))
```

f.read.analyze.slice *read one slice from a .img file*

Description

Reads in a specific slice from an ANALYZE .img image format file into an array.

Usage

```
f.read.analyze.slice(file, slice, tpt)
```

Arguments

file	The .img file to be read from
slice	The number of the slice (assumed to be the 3rd dimension)
tpt	The number of the scan that the slice is to be taken from

Details

The entire dataset is assumed to be 4D and a slice is extracted that is referenced by specifying the last two dimensions of the dataset i.e.slice and tpt.

Value

An array containing the slice

See Also

f.read.analyze.slice.at.all.timepoints, f.read.analyze.ts, f.read.analyze.volume

Examples

```
a <- f.read.analyze.slice(system.file("example.img", package = "AnalyzeFMRI"), 10, 1)
dim(a)
```

f.read.analyze.slice.at.all.timepoints *reads a slice at all time points from a .img file*

Description

Reads in a slice of a .img file at all time points into an array

Usage

```
f.read.analyze.slice.at.all.timepoints(file, slice)
```

Arguments

file	file The location of the .img file
slice	slice The number of the slice to be read in

Value

An array containing the slice at all time points

See Also

f.read.analyze.slice, f.read.analyze.ts, f.read.analyze.volume

Examples

```
a <- f.read.analyze.slice.at.all.timepoints(system.file("example.img", package = "AnalyzeFMRI"),10)
dim(a)
```

---

f.read.analyze.tpt *Read in a volume at one time point*

Description

Given a 4D ANALYZE .img/.hdr image pair this function can read in the 3D volume of measurements at a specific time point.

Usage

```
f.read.analyze.tpt(file, tpt)
```

Arguments

file	The .img file.
tpt	The time point to read in.

Details

Given a 4D ANALYZE .img/.hdr image pair this function can read in the 3D volume of measurements at a specific time point.

Value

A 3D array containing the volume.

See Also

f.read.analyze.slice, f.read.analyze.slice.at.all.timepoints, f.write.analyze,

Examples

```
a <- f.read.analyze.tpt(system.file("example.img", package = "AnalyzeFMRI"), 1)
```

`f.read.analyze.ts` *read in one voxel time series*

Description

Given a 4D ANALYZE .img/.hdr image pair this function can read in the time series from a specified position in 3D into a vector.

Usage

```
f.read.analyze.ts(file, x, y, z)
```

Arguments

`file`	The .img file
`x`	The x-coordinate
`y`	The y-coordinate
`z`	The z-coordinate

Details

Given a 4D ANALYZE .img/.hdr image pair this function can read in the time series from a specified position in 3D into a vector.

Value

A vector containing the time series

See Also

`f.read.analyze.slice`, `f.read.analyze.slice.at.all.timepoints`, `f.write.analyze`,

Examples

```
a <- f.read.analyze.ts(system.file("example.img", package = "AnalyzeFMRI"), 30, 30, 10)
```

f.read.analyze.volume *read whole .img file*

Description

Reads the ANALYZE image format .img file into an array.

Usage

```
f.read.analyze.volume(file)
```

Arguments

file	The location of the .img file to be read

Value

An array with the appropriate dimensions containing the image volume. A print out of the file information is also given. The function assumes that the corresponding .hdr file is in the same directory as the .img file.

See Also

f.read.analyze.slice, f.read.analyze.slice.at.all.timepoints, f.read.analyze.ts

Examples

```
a <- f.read.analyze.volume(system.file("example.img", package = "AnalyzeFMRI"))
dim(a)
```

f.read.header *read ANALYZE or NIFTI header file*

Description

Reads the ANALYZE or NIFTI image format .hdr (or .nii) header file into a list. The format type is determined by first reading the magic field.

Usage

```
f.read.header(file)
```

Arguments

file	The .hdr file to be read

Value

A list containing the information in the fields of the .hdr (.nii) file. See f.read.analyze.header of f.read.nifti.header to have the list of values.

See Also

`f.read.analyze.header` `f.read.nifti.header`

Examples

```
f.read.header(system.file("example.hdr", package = "AnalyzeFMRI"))
f.read.header(system.file("example-nifti.hdr", package = "AnalyzeFMRI"))
```

---

f.read.nifti.header *read Nifti header file*

Description

Reads the NIFTI image format .hdr (or .nii) header file into a list.

Usage

```
f.read.nifti.header(file)
```

Arguments

`file`	The .hdr (or .nii) file to be read

Value

A list containing the information in the fields of the .hdr (.nii) file.

`file.name`	path name of the .img file
`swap`	1 or 0 variable indicating whether files are big (=native) or little (=swapped) endian
`sizeof.hdr`	MUST be 348
`data.type`	char[10]. UNUSED
`db.name`	char[18]. UNUSED
`extents`	UNUSED
`session.error`	UNUSED
`regular`	UNUSED, but filled with 'r' as SPM does
`dim.info`	MRI slice ordering: This field encode which spatial dimension (1=x, 2=y, or 3=z) corresponds to which acquisition dimension for MRI data. In fact, it contains three informations: freq.dim, phase.dim and slice.dim, all squished into the single byte field dim.info (2 bits each, since the values for each field are limited to the range 0..3). The R function diminfo2fps can be used to extract these values from the dim.info byte.

`dim` vector (of length 8) of image dimensions. dim[1] specifies the number of dimensions. In NIFTI-1 files, dim[2], dim[3], dim[4] are for space, dim[5] is for time. The 5th dimension (dim[6]) of the dataset, if present (i.e., dim[1]=5 and dim[6] > 1), contains multiple values (for example a vector) to be stored at each spatiotemporal location. Uses of dim[7] and dim[8] are not specified in NIFTI-1 format.

`intent.p1` 1st intent parameter: first auxiliary parameter for a possible statistical distribution specified in intent.code

`intent.p2` 2nd intent parameter: second auxiliary parameter for a possible statistical distribution specified in intent.code

`intent.p3` 3rd intent parameter: third auxiliary parameter for a possible statistical distribution specified in intent.code

`intent.code` NIFTI INTENT code: if 0, this is a raw dataset; if in range 2...24, this indicates that the numbers in the dataset should be interpreted as being drawn from a given distribution. Most such distributions have auxiliary parameters (given with intent.p?); if in range 1001...1011, this is an other meaning. See file intent-code.txt in the niftidoc directory of the source package. If the dataset DOES NOT have a 5th dimension (dim[1]=4), then the auxiliary parameters are the same for each voxel, and are given in header fields intent.p1, intent.p2, and intent.p3. If the dataset DOES have a 5th dimension (dim[1]=5), then the auxiliary parameters are different for each voxel.

`datatype` integer indicator of data storage type for each voxel. This could be 0 (unknown), 2 (unsigned char = 1 byte), 4 (signed short = 2 bytes), 8 (signed int = 4 bytes), 16 (32 bit float = 4 bytes), 32 (64 bit complex = two 32 bit floats = 8 bytes), 64 (64 bits float = double = 8 bytes), 128 (RGB triple = three 8 bits bytes = 3 bytes), 256 (signed char = 1 byte), 512 (unsigned short = 2 bytes), 768 (unsigned int = 4 bytes), 1024 (signed long long = 8 bytes), 1280 (unsigned long long = 8 bytes), 1536 (128 bit float = long double = 16 bytes), 1792 (128 bit complex = 2 64 bit floats = 16 bytes), 2048 (256 bit complex = 2 128 bit floats = 32 bytes).

`bitpix` the number of bits per voxel. This field MUST correspond with the datatype field. The total number of bytes in the image data is dim[2]* ... * dim[dim[1]+1] * bitpix / 8

`slice.start` Indicates the start of the slice acquisition pattern, when slice.code is nonzero. These values are present to allow for the possible addition of "padded" slices at either end of the volume, which don't fit into the slice timing pattern. If there are no padding slices, then slice.start=0 and slice.end=dim[slice.dim+1]-1 are the correct values. For these values to be meaningful, slice.start must be non-negative and slice.end must be greater than slice.start.

`pixdim` vector (of length 8). Grid spacings. When reading a NIFTI-1 header, pixdim[1] stores qfac (which is either -1 or 1). If pixdim[1]=0 (which should not occur), we take qfac=1. pixdim[2], pixdim[3] and pixdim[4] give the voxel width along dimension x, y and z respectively. pixdim[5] gives the time step (=Time Repetition=TR). The units of pixdim can be specified with the xyzt.units field.

`vox.offset` Offset into .nii file. Should be 352 for a .nii file, 0 for a nifti .hdr/.img pair.

`scl.slope` Data scaling: If the scl.slope field is nonzero, then each voxel value in the dataset should be scaled as y = scl.slope*x + scl.inter, where x = voxel value stored and y = "true" voxel value

`scl.inter` Data scaling: offset. Idem above.

`slice.end` Indicates the end of the slice acquisition pattern, when slice.code is nonzero. These values are present to allow for the possible addition of "padded" slices at either end of the volume, which don't fit into the slice timing pattern. If there are no padding slices, then slice.start=0 and slice.end=dim[slice.dim+1]-1 are the correct values. For these values to be meaningful, slice.start must be non-negative and slice.end must be greater than slice.start.

`slice.code` Slice timing order. If this is nonzero, AND if slice.dim is nonzero, AND if slice.duration is positive, indicates the timing pattern of the slice acquisition. The following codes are defined: 0 (unknown), 1 (sequential increasing), 2 (sequential decreasing), 3 (alternating increasing), 4 (alternating decreasing), 5 (alternating increasing #2), 6 (alternating decreasing #2)

`xyzt.units` Units of pixdim[2:5]. Bits 1..3 of xyzt.units specify the (same) space unit of pixdim[2:4]. Bits 4..6 of xyzt.units specify the time unit of pixdim[5]. See xyzt-units.txt in the niftidoc directory of the source package. The R function xyzt2st can be used to extract these values from the xyzt.units byte.

`cal.max` Maximum display intensity (white) corresponds to dataset value cal.max. Dataset values above cal.max should display as white. cal.min and cal.max only make sense when applied to scalar-valued datasets (i.e., dim[1] < 5 or dim[6] = 1).

`cal.min` Minimum display intensity (black) corresponds to dataset value cal.min. Dataset values below cal.min should display as black.

`slice.duration` Time for 1 slice. If this is positive, AND if slice.dim is nonzero, indicates the amount of time used to acquire 1 slice.

`toffset` Time axis shift: The toffset field can be used to indicate a nonzero start point for the time axis. That is, time point m is at t=toffset+m*pixdim[5] for m=1, ..., dim[5]-1.

`glmax` UNUSED

`glmin` UNUSED

`descrip` char[80]. This field may contain any text you like

`aux.file` char[24]. This field is used to store an auxiliary filename.

`qform.code` NIFTI code (in 0, ... ,4). 0: Arbitrary coordinates; 1: Scanner-based anatomical coordinates; 2: Coordinates aligned to another file's, or to anatomical "truth" (coregistration); 3: Coordinates aligned to Talairach-Tournoux Atlas; 4: MNI 152 normalized coordinates

`sform.code` NIFTI code (in 0, ... ,4) with the same meaning as qform codes. The basic idea behind having two coordinate systems is to allow the image to store information about (1) the scanner coordinate system used in the acquisition of the volume (in the qform) and (2) the relationship to a standard coordinate system - e.g. MNI coordinates (in the sform). The qform allows orientation information to be kept for alignment purposes without losing volumetric information, since the qform only stores a rigid-body transformation (rotation and translation) which preserves volume. On the other hand, the sform stores a general affine transformation (shear, scale, rotation and translation) which can map the image coordinates into a standard coordinate system, like Talairach or MNI, without the need to resample the image. By having both coordinate systems, it is possible

to keep the original data (without resampling), along with information on how it was acquired (qform) and how it relates to other images via a standard space (sform). This ability is advantageous for many analysis pipelines, and has previously required storing additional files along with the image files. By using NIfTI-1 this extra information can be kept in the image files themselves. Note: the qform and sform also store information on whether the coordinate system is left-handed or right-handed and so when both are set they must be consistent, otherwise the handedness of the coordinate system (often used to distinguish left-right order) is unknown and the results of applying operations to such an image are unspecified.

`quatern.b` Quaternion b param. These b,c,d quaternion parameters encode a rotation matrix used when qform.code > 0 to obtain a rigid transformation that maps voxel indices (i,j,k) to spatial coordinates (x,y,z), typically anatomical coordinates assigned by the scanner. This transformation ("Method 2" in the nifti1.h documentation) is generated using also the voxel dimensions (pixdim[1:4]) and a 3D shift, i.e. a translation, (qoffset.*)

`quatern.c` Quaternion c param

`quatern.d` Quaternion d param

`qoffset.x` Quaternion x shift. If the (0020,0032) DICOM attribute is extracted into (px,py,pz), then qoffset.x = -px qoffset.y = -py qoffset.z = pz is a reasonable setting when qform.code=NIFTI XFORM SCANNER ANAT.

`qoffset.y` Quaternion y shift

`qoffset.z` Quaternion z shift

`srow.x` vector of length 4. 1st row affine transform. These srow.* parameters contain an affine (non-rigid) transformation ("Method 3" in the nifti1.h documentation) that maps voxel indices (i,j,k) to spatial coordinates (x,y,z).

`srow.y` vector of length 4. 2nd row affine transform

`srow.z` vector of length 4. 3rd row affine transform

`intent.name` char[16]. 'name' or meaning of data. If no data name is implied or needed, intent.name[1] should be set to 0.

`magic` MUST be "nix" or "n+x", where x in 0...9

`extension` By default,all 4 bytes of this array should be set to zero. In a .nii file, these 4 bytes will always be present, since the earliest start point for the image data is byte #352. In a separate .hdr file, these bytes may or may not be present. If not present (i.e., if the length of the .hdr file is 348 bytes), then a NIfTI-1 compliant program should use the default value of extension={0,0,0,0}. The first byte (extension[0]) is the only value of this array that is specified at present. The other 3 bytes are reserved for future use.

If extension[0] is nonzero, it indicates that extended header information is present in the bytes following the extension array. In a .nii file, this extended header data is before the image data (and vox_offset must be set correctly to allow for this). In a .hdr file, this extended data follows extension and proceeds (potentially) to the end of the file.

Examples

```
f.read.nifti.header(system.file("example-nifti.hdr", package = "AnalyzeFMRI"))
```

`f.read.nifti.slice` *read one slice from a .img or .nii file in NIFTI format*

Description

Reads in a specific slice from a NIFTI .img or .nii image format file into an array.

Usage

```
f.read.nifti.slice(file, slice, tpt)
```

Arguments

`file`	The .img file to be read from
`slice`	The number of the slice (assumed to be the 3rd dimension)
`tpt`	The number of the scan that the slice is to be taken from

Details

The entire dataset is assumed to be 4D and a slice is extracted that is referenced by specifying the last two dimensions of the dataset i.e.slice and tpt.

Value

An array containing the slice

See Also

`f.read.nifti.slice.at.all.timepoints`, `f.read.nifti.ts`, `f.read.nifti.volume`

Examples

```
a <- f.read.nifti.slice(system.file("example-nifti.img", package = "AnalyzeFMRI"), 10, 1)
dim(a)
```

f.read.nifti.slice.at.all.timepoints

reads a slice at all time points from a NIFTI .img or .nii file

Description

Reads in a slice of a .img or .nii file at all time points into an array

Usage

```
f.read.nifti.slice.at.all.timepoints(file, slice)
```

Arguments

file	file The location of the .img file
slice	slice The number of the slice to be read in

Value

An array containing the slice at all time points

See Also

f.read.nifti.slice, f.read.nifti.ts, f.read.nifti.volume

Examples

```
a <- f.read.nifti.slice.at.all.timepoints(system.file("example-nifti.img",
                                                      package = "AnalyzeFMRI"), 10)
dim(a)
```

f.read.nifti.tpt

Read in a volume at one time point

Description

Given a 4D NIFTI .img/.hdr image pair or a .nii file this function can read in the 3D volume of measurements at a specific time point.

Usage

```
f.read.nifti.tpt(file, tpt)
```

Arguments

file	The .img file.
tpt	The time point to read in.

Details

Given a 4D NIFTI .img/.hdr image pair or a .nii file this function can read in the 3D volume of measurements at a specific time point.

Value

A 3D array containing the volume.

See Also

f.read.nifti.slice, f.read.nifti.slice.at.all.timepoints, f.write.nifti,

Examples

```
f.read.nifti.tpt(system.file("example-nifti.img", package = "AnalyzeFMRI"), 1)
```

---

f.read.nifti.ts *read in one voxel time series*

Description

Given a 4D NIFTI .img/.hdr image pair or a .nii file this function can read in the time series from a specified position in 3D into a vector.

Usage

```
f.read.nifti.ts(file, x, y, z)
```

Arguments

file	The .img file
x	The x-coordinate
y	The y-coordinate
z	The z-coordinate

Details

Given a 4D NIFTI .img/.hdr image pair or a .nii file this function can read in the time series from a specified position in 3D into a vector.

Value

A vector containing the time series

See Also

f.read.nifti.slice, f.read.nifti.slice.at.all.timepoints, f.write.nifti,

Examples

```
f.read.nifti.ts(system.file("example-nifti.img", package = "AnalyzeFMRI"), 30, 30, 10)
```

---

f.read.nifti.volume *read whole image file*

Description

Reads the NIFTI image file into an array.

Usage

```
f.read.nifti.volume(file)
```

Arguments

file	The location of the image file to be read

Value

An array with the appropriate dimensions containing the image volume. A print out of the file information is also given. The function assumes that the corresponding .hdr file is in the same directory as the .img file (but if a .nii file is provided).

See Also

f.read.nifti.slice, f.read.nifti.slice.at.all.timepoints, f.read.nifti.ts

Examples

```
a <- f.read.nifti.volume(system.file("example-nifti.img", package = "AnalyzeFMRI"))
dim(a)
```

---

f.read.volume *read whole image file*

Description

Reads the ANALYZE or NIFTI image format image file into an array. Autodetects format type.

Usage

```
f.read.volume(file)
```

Arguments

file	The location of the image file to be read

Value

An array with the appropriate dimensions containing the image volume. A print out of the file information is also given. The function assumes that the corresponding .hdr file is in the same directory as the .img file. (but if it is a .nii file)

See Also

f.read.nifti.slice, f.read.nifti.slice.at.all.timepoints, f.read.nifti.ts

Examples

```
a <- f.read.volume(system.file("example-nifti.img", package = "AnalyzeFMRI"))
dim(a)
```

`f.spectral.summary` *plots graphical summary of spectral properties of an fMRI dataset*

Description

For an analyze .img file the periodogram of the time series are divided by a flat spectral estimate using the median periodogram ordinate. The resulting values are then combined within each Fourier frequency and quantiles are plotted against frequency. This provides a fast look at a fMRI dataset to identify any artifacts that reside at single frequencies.

Usage

```
f.spectral.summary(file, mask.file, ret.flag=FALSE)
```

Arguments

`file`	`file` The location of .img file
`mask.file`	`mask.file` Optional location of a .img file containing a mask. If not given then one is created.
`ret.flag`	`ret.flag` flag specifying whether to return the array of quantiles at each frequency

Value

If `ret.flag` = `TRUE` the an array of quantiles at each frequency is returned

See Also

f.analyze.file.summary

f.spectral.summary.nifti

plots graphical summary of spectral properties of an fMRI dataset

Description

For a NIFTI .img file the periodogram of the time series are divided by a flat spectral estimate using the median periodogram ordinate. The resulting values are then combined within each Fourier frequency and quantiles are plotted against frequency. This provides a fast look at a fMRI dataset to identify any artifacts that reside at single frequencies.

Usage

```
f.spectral.summary.nifti(file, mask.file, ret.flag = FALSE, verbose = TRUE)
```

Arguments

file	file The location of .img file
mask.file	mask.file Optional location of a .img file containing a mask. If not given then one is created.
ret.flag	ret.flag flag specifying whether to return the array of quantiles at each frequency
verbose	verbose logical to display information on the progress of computations

Value

If ret.flag = TRUE the an array of quantiles at each frequency is returned

See Also

f.nifti.file.summary

f.write.analyze

writes an array to a .img/.hdr pair in ANALYZE format

Description

Creates a .img and .hdr pair of files from a given array

Usage

```
f.write.analyze(mat,file,size,pixdim,vox.units,cal.units,originator,path.out=NULL)
```

Arguments

mat	An array
file	The name of the file to be written to, without .img or .hdr suffix
size	Specify the format of the .img file. Either "float" (for 4 byte floats) or "int" (2 byte integers) or "char" (1 byte integers).
pixdim	A vector of length 3 specifying the voxel dimensions in mm
vox.units	String specifying the spatial units of measure for a voxel
cal.units	String specifying the name of calibration unit
originator	vector of length 5, only the three first values are used. Put the last two equal to zero
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.write.array.to.img.8bit, f.write.array.to.img.2bytes, f.write.array.to.img.float

Examples

```
a <- array(rnorm(20 * 30 * 40 * 3), dim = c(20, 30, 40, 3))
file <- "temp"
tmpdir <- tempdir()
f.write.analyze(a, file, size = "float", path.out = tmpdir)
unlink(tmpdir) # tidy up
```

---

f.write.array.to.img.2bytes

write array of 2 byte integers

Description

Writes an array to a .img file of 2 byte integers

Usage

```
f.write.array.to.img.2bytes(mat,file,path.out=NULL)
```

Arguments

mat	An array
file	The name of the file to be written, preferably with .img suffix
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.write.analyze f.write.array.to.img.float

`f.write.array.to.img.8bit`
write array of 1 byte integers

Description

Writes an array to a .img file of 1 byte integers

Usage

```
f.write.array.to.img.8bit(mat,file,path.out=NULL)
```

Arguments

`mat`	An array
`file`	The name of the file to be written, preferably with .img suffix
`path.out`	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.write.analyze, f.write.array.to.img.float, f.write.array.to.img.2bytes

f.write.array.to.img.float *write array of 4 byte floats*

Description

Writes an array to a .img file of 4 byte floats

Usage

```
f.write.array.to.img.float(mat,file,path.out=NULL)
```

Arguments

mat	An array
file	The name of the file to be written, preferably with .img suffix
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.write.analyze, f.write.array.to.img.2bytes , f.write.array.to.img.8bit

f.write.list.to.hdr *writes a .hdr file in ANALYZE format*

Description

Writes a list of attributes to a .hdr file

Usage

```
f.write.list.to.hdr(L,file,path.out=NULL)
```

Arguments

L	A list of all the fields included in a .hdr file
file	The name of the file to write, preferably with .hdr suffix
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.basic.hdr.list.create

Examples

```
a <- array(rnorm(20 * 30 * 40 * 3), dim = c(20, 30, 40, 3))
file <- "temp.hdr"
b <- f.basic.hdr.list.create(a, file)
tmpdir <- tempdir()
f.write.list.to.hdr(b, file, path.out = tmpdir)
unlink(tmpdir) # tidy up
```

---

f.write.list.to.hdr.nifti

writes a .hdr file in NITI format

Description

Writes a list of attributes to a .hdr file

Usage

```
f.write.list.to.hdr.nifti(L,file, path.out=NULL)
```

Arguments

L	A list of the all the fields included in a .hdr file
file	The name of the file to write to, preferably with .hdr suffix
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.basic.hdr.nifti.list.create

Examples

```
a <- array(rnorm(20 * 30 * 40 * 3), dim = c(20, 30, 40, 3))
file <- "temp.hdr"
b <- f.basic.hdr.nifti.list.create(dim(a), file)
tmpdir <- tempdir()
f.write.list.to.hdr.nifti(b, file, path.out = tmpdir)
unlink(tmpdir) # tidy up
```

`f.write.nifti` *writes an array to a .img/.hdr pair in NIFTI format or to a .nii file*

Description

Creates a .img/.hdr pair of files or a .nii file from a given array

Usage

```
f.write.nifti(mat,file,size,L,nii,path.out=NULL)
```

Arguments

`mat`	An array
`file`	The name of the file to be written, without .img or .hdr suffix
`size`	Specify the format of the .img file. Either "float" (for 4 byte floats) or "int" (2 byte integers) or "char" (1 byte integers).
`L`	if NULL, the list is created by the function, else it should be provided. This list contains the header part of a NIFTI image.
`nii`	should we write only one .nii file or a .hdr/.img pair of files
`path.out`	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

`f.write.array.to.img.8bit`, `f.write.array.to.img.2bytes`, `f.write.array.to.img.float`
`f.write.nii.array.to.img.8bit`, `f.write.nii.array.to.img.2bytes`, `f.write.nii.array.to.img.float`

Examples

```
a<-array(rnorm(20*30*40*3),dim=c(20,30,40,3))
file<-"temp"
tmpdir <- tempdir()
f.write.nifti(a,file,size="float",nii=TRUE, path.out = tmpdir)
unlink(tmpdir) # tidy up
```

`f.write.nii.array.to.img.2bytes`
write array of 2 byte integers and add at the begining of the file the NIFTI header part

Description

Writes an array to a .img file of 2 byte integers and add at the begining of the file the NIFTI header part

Usage

```
f.write.nii.array.to.img.2bytes(mat,L,file,path.out=NULL)
```

Arguments

`mat`	An array
`L`	A list containing the header information
`file`	The name of the file to be written, preferably with .img suffix
`path.out`	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

`f.write.nifti` `f.write.nii.array.to.img.float`

`f.write.nii.array.to.img.8bit`
write array of 1 byte integers and add at the begining of the file the NIFTI header part

Description

Writes an array to a .img file of 1 byte integers and add at the begining of the file the NIFTI header part

Usage

```
f.write.nii.array.to.img.8bit(mat,L,file,path.out)
```

Arguments

mat	An array
L	A list containing the header information
file	The name of the file to be written, preferably with .img suffix
path.out	The path where to write the file

Value

Nothing is returned

See Also

f.write.nifti, f.write.nii.array.to.img.float, f.write.nii.array.to.img.2bytes

---

f.write.nii.array.to.img.float

write array of 4 byte floats and add at the begining of the file the NIFTI header part

---

Description

Writes an array to a .img file of 4 byte floats and add at the begining of the file the NIFTI header part

Usage

```
f.write.nii.array.to.img.float(mat,L,file,path.out=NULL)
```

Arguments

mat	An array
L	A list containing the header information
file	The name of the file to be written, preferably with .img suffix
path.out	The path where to write the file (mandatory argument, even if the default value is NULL for backward compatibility)

Value

Nothing is returned

See Also

f.write.nifti, f.write.nii.array.to.img.2bytes , f.write.nii.array.to.img.8bit

fourDto2D *fourDto2D*

Description

This function transforms a 4D image array into a 2D image matrix by unrolling space. This is usefull to perform a subsequent ICA.

Usage

```
fourDto2D(volume.4d, tm)
```

Arguments

volume.4d	a 4D array to be transformed
tm	number of time dimensions

Value

a matrix of size tm x vm which contains the tm images

See Also

threeDto4D twoDto4D

Examples

```
volume.4d <- array(rnorm(96), dim = c(2, 4, 4, 3)) # a fake 4D image array
x.2d <- fourDto2D(volume.4d, tm = 3)
```

fps2diminfo *fps2diminfo*

Description

Encode freq.dim, phase.dim and slice.dim fields into the one byte dim.info field of a NIFTI header file.

Usage

```
fps2diminfo(freq.dim,phase.dim,slice.dim)
```

Arguments

freq.dim	freq.dim field of a NIFTI file
phase.dim	phase.dim field of a NIFTI file
slice.dim	slice.dim field of a NIFTI file

Value

A list containing `dim.info` field.

See Value Section of the help file of function diminfo2fps().

See Also

`diminfo2fps`

Examples

```
dim.info <- f.read.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))$dim.info
mylist <- diminfo2fps(dim.info)
fps2diminfo(mylist$freq.dim,mylist$phase.dim,mylist$slice.dim)
```

`GaussSmoothArray` *Spatially smooth an array with Gaussian kernel.*

Description

Applies a stationary Gaussian spatial smoothing kernel to a 3D or 4D array.

Usage

```
GaussSmoothArray(x, voxdim=c(1, 1, 1), ksize=5, sigma=diag(3, 3),
                 mask=NULL, var.norm=FALSE)
```

Arguments

`x`	The array to be smoothed.
`voxdim`	The dimensions of the *vol*ume *el*ements (voxel) that make up the array.
`ksize`	The dimensions (in number of voxels) of the 3D discrete smoothing kernel used to smooth the array.
`sigma`	The covariance matrix of the 3D Gaussian smoothing kernel. This matrix doesn't have to be non-singular; zero on the diagonal of sigma indicate no smoothing in that direction.
`mask`	A 3D 0-1 mask that delimits where the smoothing occurs.
`var.norm`	Logical flag indicating whether to normalize the variance of the smoothed array.

Value

The smoothed array is returned.

Author(s)

J. L. Msrchini

See Also

GaussSmoothKernel

Examples

```
d <- c(10, 10, 10, 20)
mat <- array(rnorm(cumprod(d)[length(d)]), dim = d)
mat[, , 6:10, ] <- mat[, , 6:10, ] + 3
mask <- array(0, dim = d[1:3])
mask[3:8, 3:8, 3:8] <- 1
b <- GaussSmoothArray(mat, mask = mask, voxdim = c(1, 1, 1), ksize = 5, sigma = diag(1, 3))
```

GaussSmoothKernel *Calculates a discrete Gaussian smoothing kernel.*

Description

Calculates a simple, discrete Gaussian smoothing kernel of a specfic size given the covariance matrix of the Gaussian.

Usage

```
GaussSmoothKernel(voxdim=c(1, 1, 1), ksize=5, sigma=diag(3, 3))
```

Arguments

voxdim	Dimensions of each voxel.
ksize	Dimensions of the discrete kernel size.
sigma	The covariance matrix of the Gaussian kernel.

Value

An array of dimension (ksize,ksize,ksize) containing the smoothing kernel.

Author(s)

J. L. Marchini

Examples

```
a <- GaussSmoothKernel(voxdim=c(1,1,1), ksize=5, sigma=diag(1,3))
```

ICAspat *ICAspat*

Description

This function performs a spatial ICA

Usage

```
ICAspat(X,n.comp,alg.typ="parallel",centering=TRUE,hp.filter=TRUE)
```

Arguments

X	a matrix of size tm x vm which contains the functionnal images
n.comp	number of maximally independent components to extract
alg.typ	if 'alg.typ == "parallel"' the components are extracted simultaneously (the default). if 'alg.typ == "deflation"' the components are extracted one at a time.
centering	Logical. Should we center the data first. Centering will be performed by firstly removing the column mean.
hp.filter	Logical. Should we perform high-pass filtering on the data

Value

A list containing

time.series	estimated mixing matrix of size tm x n.comp
spatial.components	estimated source matrix of size n.comp x vm

See Also

ICAtemp

ICAtemp *ICAtemp*

Description

This function performs a temporal ICA

Usage

```
ICAtemp(X,n.comp,alg.typ="parallel",centering=TRUE,hp.filter=TRUE)
```

Arguments

`X`	a matrix of size vm x tm which contains the functionnal images
`n.comp`	number of maximally independent components to extract
`alg.typ`	if 'alg.typ == "parallel"' the components are extracted simultaneously (the default). if 'alg.typ == "deflation"' the components are extracted one at a time.
`centering`	Logical. Should we center the data first. Centering will be performed by firstly removing the column mean.
`hp.filter`	Logical. Should we perform high-pass filtering on the data

Value

A list containing

`time.series`	estimated source matrix of size n.comp x tm
`spatial.components`	estimated mixing matrix of size vm x n.comp

See Also

`ICAspat`

`ijk2xyz` *ijk2xyz*

Description

This function maps from data coordinates (e.g. column i, row j, slice k), into some real world (x,y,z) positions in space. These positions could relate to Talairach-Tournoux (T&T) space, MNI space, or patient-based scanner coordinates.

Usage

```
ijk2xyz(ijk=c(1,1,1),method=2,L)
```

Arguments

`ijk`	matrix. Each column of ijk should contain a voxel index coordinates (i,j,k) to be mapped to its (x,y,z) real coordinates in some other space
`method`	1 (qform.code=sform.code=0),2 (qform.code>0, rigid transformation) or 3 (sform.code>0, affine transformation).
`L`	header list of a NIFTI file

Details

The NIfTI format allows storage on disk to be in either a left- or right-handed coordinate system. However, the format includes an implicit spatial transformation into a RIGHT-HANDED coordinate system. This transform maps from data coordinates (e.g. column i, row j, slice k), into some real world (x, y, z) positions in space. These positions could relate to Talairach-Tournoux (T&T) space, MNI space, or patient-based scanner coordinates. For T&T, and MNI coordinates, x increases from left to right, y increases from posterior to anterior, and z increases in the inferior to superior direction. Directions in the scanner coordinate system are similar. MRI data is usually exported as DICOM format, which encodes the positions and orientations of the slices. When data are converted from DICOM to NIfTI-1 format, the relevant information can be determined from the Pixel Spacing, Image Orientation (Patient) and Image Position (Patient) fields of the DICOM files. NIfTI-1 also allows the space of one image to be mapped to that of another (via a rigid or affine transform). This is to enable on-the-fly resampling of registered images. This would allow intra-subject images, collected with lots of different orientations or resolutions, to be treated as if they are all in register.

Neurological and radiological conventions only relate to visualization of axial images. They are unrelated to how the data are stored on disk, or even how the real-world coordinates are represented. It is more appropriate to consider whether the real-world coordinate system is left- or right-handed (see below). Talairach and Tournoux use a right-handed system, whereas the storage convention of ANALYZE files is usually considered as left-handed. These coordinate systems are mirror images of each other (if you are a psychologist, try explaining why mirror images appear to be left-right flipped, rather than flipped up-down, or back-front). Transforming between left- and right-handed coordinate systems involves flipping, and can not be done by rotations alone.

x=thumb, y=index finger (forefinger), z=left (resp. right) hand's middle finger for left-handed persons (resp. right-handed persons).

Volume orientation is given by a transformation that maps voxel indices (i, j, k) to spatial coordinates (x, y, z), typically anatomical coordinates assigned by the scanner. This transformation (*Method 2* in the '`nifti1.h`' documentation) is generated using the voxel dimensions, a quaternion encoding a rotation matrix, and a 3D shift, all stored in the NIfTI-1 header; details can be found in the '`nifti1.h`' comments. The NIfTI-1 header also provides for a general affine transformation, separate from that described by *Method 2*. This transformation (*Method 3*) also maps voxel indices (i, j, k) to (x, y, z), which in this case are typically coordinates in a standard space such as the Talairach space. The elements of this transformation matrix are stored in the NIfTI-1 header. For example, the *Method 2* transformation can be constructed from the attributes from a set of DICOM files; the *Method 3* transform can be computed offline and inserted into the header later. The exact "meaning" of the coordinates given by the *Method 2* and *Method 3* transformations is recorded in header fields `qform.code` and `sform.code`, respectively. Code values can indicate if the (x, y, z) axes are

- Anatomical coordinates from the scanner (e.g., the DICOM header)
- Aligned to some anatomical "truth" or standard
- Aligned and warped to Talairach-Tournoux coordinates
- Aligned and warped to MNI-152 coordinates

It is possible that neither transformation is specified (i.e., `qform.code=sform.code=0`), in which case we are left with the voxel size in `pixdim[]`, and no orientation is given or assumed. This use (*Method 1*) is discouraged.

The basic idea behind having two coordinate systems is to allow the image to store information about (1) the scanner coordinate system used in the acquisition of the volume (in the `qform`) and (2) the relationship to a standard coordinate system - e.g. MNI coordinates (in the `sform`). The `qform` allows orientation information to be kept for alignment purposes without losing volumetric information, since the `qform` only stores a rigid-body transformation which preserves volume. On the other hand, the `sform` stores a general affine transformation which can map the image coordinates into a standard coordinate system, like Talairach or MNI, without the need to resample the image. By having both coordinate systems, it is possible to keep the original data (without resampling), along with information on how it was acquired (`qform`) and how it relates to other images via a standard space (`sform`). This ability is advantageous for many analysis pipelines, and has previously required storing additional files along with the image files. By using NIfTI-1 this extra information can be kept in the image files themselves.
Note: the `qform` and `sform` also store information on whether the coordinate system is left-handed or right-handed (see Q15) and so when both are set they must be consistent, otherwise the handedness of the coordinate system (often used to distinguish left-right order) is unknown and the results of applying operations to such an image are unspecified.

There are 3 different methods by which continuous coordinates can be attached to voxels. The discussion below emphasizes 3D volumes, and the continuous coordinates are referred to as (x, y, z). The voxel index coordinates (i.e., the array indexes) are referred to as (i, j, k), with valid ranges:

- $i = 0, \ldots,$`dim[1]-1`
- $j = 0, \ldots,$`dim[2]-1` (if `dim[0]` $>= 2$)
- $k = 0, \ldots,$`dim[3]-1` (if `dim[0]` $>= 3$)

The (x, y, z) coordinates refer to the CENTER of a voxel. In methods 2 and 3, the `(x,y,z)` axes refer to a subject-based coordinate system, with

$+x$ = Right $+y$ = Anterior $+z$ = Superior.

This is a right-handed coordinate system. However, the exact direction these axes point with respect to the subject depends on `qform.code` (*Method 2*) and `sform.code` (*Method 3*).

N.B.: The i index varies most rapidly, j index next, k index slowest. Thus, voxel (i, j, k) is stored starting at location

$(i + j *$ `dim[1]` $+ k *$ `dim[1]` $*$ `dim[2]`$) *$ (`bitpix`$/8)$

into the dataset array.

N.B.: The ANALYZE 7.5 coordinate system is

$+x$ = Left $+y$ = Anterior $+z$ = Superior

which is a left-handed coordinate system. This backwardness is too difficult to tolerate, so this NIFTI-1 standard specifies the coordinate order which is most common in functional neuroimaging.

N.B.: The 3 methods below all give the locations of the voxel centers in the (x, y, z) coordinate system. In many cases, programs will wish to display image data on some other grid. In such a case, the program will need to convert its desired (x, y, z) values into (i, j, k) values in order to extract (or interpolate) the image data. This operation would be done with the inverse transformation to those described below.

N.B.: *Method 2* uses a factor qfac which is either -1 or 1; qfac is stored in the otherwise unused pixdim[0]. If pixdim[0]=0.0 (which should not occur), we take qfac=1. Of course, pixdim[0] is only used when reading a NIFTI-1 header, not when reading an ANALYZE 7.5 header.

N.B.: The units of (x, y, z) can be specified using the xyzt.units field.

- *METHOD 1* (the "old" way, used only when qform.code = 0):
  The coordinate mapping from (i, j, k) to (x, y, z) is the ANALYZE 7.5 way. This is a simple scaling relationship:
  x = pixdim[1] * i
  y = pixdim[2] * j
  z = pixdim[3] * k
  No particular spatial orientation is attached to these (x, y, z) coordinates. (NIFTI-1 does not have the ANALYZE 7.5 orient field, which is not general and is often not set properly.) This method is not recommended, and is present mainly for compatibility with ANALYZE 7.5 files.
- *METHOD 2* (used when qform.code > 0, which should be the "normal" case):
  The (x, y, z) coordinates are given by the pixdim[] scales, a rotation matrix, and a shift. This method is intended to represent "scanner-anatomical" coordinates, which are often embedded in the image header (e.g., DICOM fields (0020,0032), (0020,0037), (0028,0030), and (0018,0050)), and represent the nominal orientation and location of the data. This method can also be used to represent "aligned" coordinates, which would typically result from some post-acquisition alignment of the volume to a standard orientation (e.g., the same subject on another day, or a rigid rotation to true anatomical orientation from the tilted position of the subject in the scanner). The formula for (x, y, z) in terms of header parameters and (i, j, k) is:

```
[ x ]       [ R11 R12 R13 ] [          pixdim[1] * i ]       [ qoffset.x ]
[ y ]   =   [ R21 R22 R23 ] [          pixdim[2] * j ]   +   [ qoffset.y ]
[ z ]       [ R31 R32 R33 ] [ qfac * pixdim[3] * k ]       [ qoffset.z ]
```

  The qoffset.* shifts are in the NIFTI-1 header. Note that the center of the $(i, j, k) = (0, 0, 0)$ voxel (first value in the dataset array) is just $(x, y, z) =$ (qoffset.x, qoffset.y, qoffset.z).
  The rotation matrix R is calculated from the quatern.* parameters. This calculation is described below.
  The scaling factor qfac is either 1 or -1. The rotation matrix R defined by the quaternion parameters is "proper" (has determinant 1). This may not fit the needs of the data; for example, if the image grid is
  i increases from Left-to-Right
  j increases from Anterior-to-Posterior
  k increases from Inferior-to-Superior
  Then (i, j, k) is a left-handed triple. In this example, if qfac=1, the R matrix would have to be
  [1 0 0]
  [0 -1 0] which is "improper" (determinant = -1).
  [0 0 1]
  If we set qfac=-1, then the R matrix would be
  [1 0 0]
  [0 -1 0] which is proper.
  [0 0 -1]

This R matrix is represented by quaternion $[a, b, c, d] = [0, 1, 0, 0]$ (which encodes a 180 degree rotation about the x-axis).

- *METHOD 3* (used when `sform.code` > 0):

  The (x, y, z) coordinates are given by a general affine transformation of the (i, j, k) indexes:

```
x = srow.x[0] * i + srow.x[1] * j + srow.x[2] * k + srow.x[3]
y = srow.y[0] * i + srow.y[1] * j + srow.y[2] * k + srow.y[3]
z = srow.z[0] * i + srow.z[1] * j + srow.z[2] * k + srow.z[3]
```

  The `srow.*` vectors are in the NIFTI.1 header. Note that no use is made of `pixdim[]` in this method.

- WHY 3 METHODS?

  Method 1 is provided only for backwards compatibility. The intention is that *Method 2* (`qform.code` > 0) represents the nominal voxel locations as reported by the scanner, or as rotated to some fiducial orientation and location. *Method 3*, if present (`sform.code` > 0), is to be used to give the location of the voxels in some standard space. The `sform.code` indicates which standard space is present. Both methods 2 and 3 can be present, and be useful in different contexts (*method 2* for displaying the data on its original grid; *method 3* for displaying it on a standard grid).

  In this scheme, a dataset would originally be set up so that the *Method 2* coordinates represent what the scanner reported. Later, a registration to some standard space can be computed and inserted in the header. Image display software can use either transform, depending on its purposes and needs.

  In *Method 2*, the origin of coordinates would generally be whatever the scanner origin is; for example, in MRI, (0,0,0) is the center of the gradient coil.

  In *Method 3*, the origin of coordinates would depend on the value of `sform.code`; for example, for the Talairach coordinate system, (0,0,0) corresponds to the Anterior Commissure.

- QUATERNION REPRESENTATION OF ROTATION MATRIX (*METHOD 2*)

  The orientation of the (x, y, z) axes relative to the (i, j, k) axes in 3D space is specified using a unit quaternion $[a, b, c, d]$, where $a * a + b * b + c * c + d * d = 1$. The (b, c, d) values are all that is needed, since we require that $a = \sqrt{(1.0 - (b * b + c * c + d * d))}$ be nonnegative. The (b, c, d) values are stored in the (`quatern.b`,`quatern.c`,`quatern.d`) fields.

  The quaternion representation is chosen for its compactness in representing rotations. The (proper) 3x3 rotation matrix that corresponds to $[a, b, c, d]$ is

```
     [  a*a+b*b-c*c-d*d   2*b*c-2*a*d       2*b*d+2*a*c       ]
R =  [  2*b*c+2*a*d       a*a+c*c-b*b-d*d   2*c*d-2*a*b       ]
     [  2*b*d-2*a*c       2*c*d+2*a*b       a*a+d*d-c*c-b*b   ]
```

```
     [ R11  R12  R13 ]
  =  [ R21  R22  R23 ]
     [ R31  R32  R33 ]
```

  If (p, q, r) is a unit 3-vector, then rotation of angle h about that direction is represented by the quaternion

$[a,b,c,d] = [cos(h/2), p*sin(h/2), q*sin(h/2), r*sin(h/2)]$.
Requiring $a >= 0$ is equivalent to requiring $-Pi <= h <= Pi$. (Note that $[-a,-b,-c,-d]$ represents the same rotation as $[a,b,c,d]$; there are 2 quaternions that can be used to represent a given rotation matrix R.) To rotate a 3-vector (x,y,z) using quaternions, we compute the quaternion product
$[0,x',y',z'] = [a,b,c,d] * [0,x,y,z] * [a,-b,-c,-d]$
which is equivalent to the matrix-vector multiply

```
[ x' ]          [ x ]
[ y' ]  = R     [ y ] (equivalence depends on a*a+b*b+c*c+d*d=1)
[ z' ]          [ z ]
```

Multiplication of 2 quaternions is defined by the following:
$[a,b,c,d] = a*1 + b*I + c*J + d*K$
where
$I*I = J*J = K*K = -1$ (I,J,K are square roots of -1)
$I*J = K$, $J*K = I$, $K*I = J$
$J*I = -K$, $K*J = -I$, $I*K = -J$ (not commutative!).
For example

$$[a,b,0,0] * [0,0,0,1] = [0,0,-b,a]$$

since this expands to

$$(a+b*I)*(K) = (a*K + b*I*K) = (a*K - b*J).$$

The above formula shows how to go from quaternion (b,c,d) to rotation matrix and direction cosines. Conversely, given R, we can compute the fields for the NIFTI-1 header by
$a = 0.5 * \sqrt{(1 + R11 + R22 + R33)}$ (not stored)
$b = 0.25 * (R32 - R23)/a$ => `quatern.b`
$c = 0.25 * (R13 - R31)/a$ => `quatern.c`
$d = 0.25 * (R21 - R12)/a$ => `quatern.d`
If `a=0` (a 180 degree rotation), alternative formulas are needed. See the '`nifti1.io.c`' function mat44.to.quatern() for an implementation of the various cases in converting R to $[a,b,c,d]$.
Note that R-transpose (= R-inverse) would lead to the quaternion $[a,-b,-c,-d]$.
The choice to specify the `qoffset.x` (etc.) values in the final coordinate system is partly to make it easy to convert DICOM images to this format. The DICOM attribute "Image Position (Patient)" (0020,0032) stores the (Xd,Yd,Zd) coordinates of the center of the first voxel. Here, (Xd,Yd,Zd) refer to DICOM coordinates, and $Xd = -x$, $Yd = -y$, $Zd = z$, where (x,y,z) refers to the NIFTI coordinate system discussed above. (i.e., DICOM $+Xd$ is Left, $+Yd$ is Posterior, $+Zd$ is Superior,
whereas $+x$ is Right, $+y$ is Anterior , $+z$ is Superior.)
Thus, if the (0020,0032) DICOM attribute is extracted into (px,py,pz), then
`qoffset.x` = $-px$ `qoffset.y` = $-py$ `qoffset.z` = pz
is a reasonable setting when `qform.code`=NIFTI.XFORM.SCANNER.ANAT.
That is, DICOM's coordinate system is 180 degrees rotated about the z-axis from the neuroscience/NIFTI coordinate system. To transform between DICOM and NIFTI, you just have to negate the x- and y-coordinates.

The DICOM attribute (0020,0037) "Image Orientation (Patient)" gives the orientation of the x- and y-axes of the image data in terms of 2 3-vectors. The first vector is a unit vector along the x-axis, and the second is along the y-axis. If the (0020,0037) attribute is extracted into the value (xa, xb, xc, ya, yb, yc), then the first two columns of the R matrix would be
[-xa -ya]
[-xb -yb]
[xc yc]
The negations are because DICOM's x- and y-axes are reversed relative to NIFTI's. The third column of the R matrix gives the direction of displacement (relative to the subject) along the slice-wise direction. This orientation is not encoded in the DICOM standard in a simple way; DICOM is mostly concerned with 2D images. The third column of R will be either the cross-product of the first 2 columns or its negative. It is possible to infer the sign of the 3rd column by examining the coordinates in DICOM attribute (0020,0032) "Image Position (Patient)" for successive slices. However, this method occasionally fails for reasons that I (RW Cox) do not understand.

Value

A list containing the matrix xyz of the positions of the points specified in ijk.

See Also

xyz2ijk Q2R R2Q

Examples

```
L <- f.read.header(system.file("example-nifti.hdr",
package="AnalyzeFMRI"))
ijk <- matrix(c(1,1,1,2,3,7),byrow=FALSE,nrow=3)
ijk2xyz(ijk=ijk,method=2,L)
```

`magicfield` *Get magicfield from the header of an image file*

Description

Determine the type of a file : NIFTI .nii format, NIFTI .hdr/.img pair format, ANALYZE format.

Usage

```
magicfield(file, verbose = TRUE)
```

Arguments

`file`	character, filename of an image (or header) file
`verbose`	`verbose` logical to display information in the console

Value

A list containing the magic and dim fields.

Examples

```
magicfield(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
```

`mat34.to.TRSZ` *Affine 4x4 (or 3x4) matrix to Translation, Rotation, Shear and Scale*

Description

Extract in that order Translation, Rotation, Shear and Scale from a 4x4 (or 3x4) affine matrix from a NIFTI header list (srow.x, srow.y, srow.z).

Usage

```
mat34.to.TRSZ(M)
```

Arguments

`M`	the affine matrix

Details

Decomposes M using the convention: M = translation * scale * skew * rotation. Be careful that rotation can be improper.

Value

A list containing Translation, Scale, Shear and Rotation. Rotation decomposition is also provided (rotation = RotZ*RotY*RotX*Ref where Ref is a Reflexion if the rotation is improper or is Identity if the rotation is proper).

See Also

`R2Q` `Q2R` `mat34.to.TZSR`

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
M <- rbind(L$srow.x,L$srow.y,L$srow.z)
mat34.to.TRSZ(M)
```

`mat34.to.TZSR` *Affine 4x4 (or 3x4) matrix to Translation, Scale, Shear and Rotation*

Description

Extract in that order Translation, Scale, Shear and Rotation from a 4x4 (or 3x4) affine matrix from a NIFTI header list (srow.x, srow.y, srow.z).

Usage

```
mat34.to.TZSR(M)
```

Arguments

`M`	the affine matrix

Details

Decomposes M using the convention: M = translation * scale * skew * rotation. Be careful that rotation can be improper.

Value

A list containing Translation, Scale, Shear and Rotation. Rotation decomposition is also provided (rotation = RotZ*RotY*RotX*Ref where Ref is a Reflexion if the rotation is improper or is Identity if the rotation is proper).

See Also

`R2Q` `Q2R` `mat34.to.TRSZ`

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
M <- rbind(L$srow.x,L$srow.y,L$srow.z)
mat34.to.TZSR(M)
```

model.2.cov.func *Calculates covariance from Hartvig Model 2*

Description

Calculates covariance from Hartvig Model 2

Usage

```
model.2.cov.func(g, par)
```

Arguments

g	The value of gamma
par	A vector of parameters from the N2G model

Value

The calculated covariance

Author(s)

J. L. Marchini

References

Hartvig, N. V. and Jensen, J. L (2000) Spatial Mixture Modelling of fMRI Data, Human Brain Mapping 11:233–248

model.2.est.gamma *Estimate gamma for Model 2 of Hartvig and Jensen (2000)*

Description

Estimate gamma for Model 2 of Hartvig and Jensen (2000)

Usage

```
model.2.est.gamma(cov, par)
```

Arguments

cov	An estimate of the spatial covariance
par	N2G model parameter estimates

Value

The estimate of gamma

Author(s)

J. L. Marchini

References

Hartvig, N. V. and Jensen, J. L (2000) Spatial Mixture Modelling of fMRI Data, Human Brain Mapping 11:233–248

`N2G` *Fits the N2G model*

Description

Fits the N2G model (1 Normal and 2 Gamma's mixture model) to a dataset using Maximum Likelihhod.

Usage

```
N2G(data, par.start = c(4, 2, 4, 2, 0.9, 0.05))
```

Arguments

`data`	The dataset.
`par.start`	The starting values for the optimization to maximize the likelihood. The parameters of the model are ordered in the vector par.start in the following way (refer to the model below) c(a, b, c, d, p1, p2)

Details

The mixture model considered is a mixture of a standard normal distribution and two Gamma functions. This model is denoted N2G.

x ~ p1 * N(0, 1) + p2 * Gamma(a, b) + (1 - p1 - p2) * -Gamma(c, d)

Value

A list with components

`par`	The fitted parameter values.
`lims`	The upper and lower thresholds for the Normal component of the fitted model

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density , N2G.Likelihood , N2G.Transform, N2G.Fit , N2G.Inverse , N2G.Region

Examples

```
par <- c(3, 2, 3, 2, .3, .4)
data <- c(rnorm(10000), rgamma(2000, 10, 1), -rgamma(1400, 10, 1))
hist(data, n = 100, freq = FALSE)

q <- N2G.Fit(data, par, maxit = 10000, method = "BFGS")
p <- seq(-50, 50, .1)
lines(p, N2G.Density(p, q), col = 2)
```

N2G.Class.Probability *Posterior Probabilities for N2G model*

Description

Calculates the Posterior Probability of data points being in each class given the parameters of the N2G model.

Usage

```
N2G.Class.Probability(data, par)
```

Arguments

data	The dataset (usually a vector)
par	The paarmeters of the model

Value

Returns the Posterior Probability of data points being in each class given the parameters of the N2G model.

Author(s)

J. L. Marchini

See Also

N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density, N2G.Likelihood , N2G.Transform, N2G.Fit , N2G , N2G.Inverse , N2G.Region

N2G.Density — *Calculates the density function for the N2G model*

Description

Calculates the density function for the N2G model

Usage

```
N2G.Density(data, par)
```

Arguments

data	The dataset (usually a vector)
par	The parameters of the model.

Details

Calculates the density function for the N2G model

Value

Returns the density at each point of the datasets

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Likelihood , N2G.Transform, N2G.Fit , N2G , N2G.Inverse , N2G.Region

N2G.Fit — *Optimzation function for N2G model*

Description

Function that carries out the likelihood optimzation for the N2G model.

Usage

```
N2G.Fit(data, par.start, maxit, method)
```

Arguments

data	The dataset (usually a vector)
par.start	Starting values for the parameters
maxit	Maximum number of iterations
method	Optimzation method (passed to optim)

Details

Numerical optimization of the N2G model likelihood.

Value

Returns the optimized model parameters.

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Likelihood , N2G.Transform, N2G.Density , N2G , N2G.Inverse , N2G.Region

---

N2G.Inverse *Transform parameters of N2G model back to their real domains*

Description

Transform parameters of N2G model back to their real domains

Usage

```
N2G.Inverse(par)
```

Arguments

par	Parameter vector

Details

Transform parameters of N2G model back to their real domains

Value

Returns the transformed parameters.

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density , N2G.Likelihood , N2G.Transform, N2G.Fit , N2G , N2G.Region

---

N2G.Likelihood *Calculates the (negative) Likelihood of the N2G model*

Description

Calculates the (negative) Likelihood of the N2G model

Usage

```
N2G.Likelihood(inv.par, data)
```

Arguments

inv.par	A vector of transformed parameters for the N2G model
data	The dataset (usually a vector)

Details

Calculates the (negative) Likelihood of the N2G model

Value

Returns (negative) Likelihood at each point of the dataset.

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density , N2G.Transform, N2G.Fit , N2G , N2G.Inverse , N2G.Region

N2G.Likelihood.Ratio *N2G Likelihood Ratio's*

Description

Calculates the ratio of the likelihood that data came from the positive Gamma distribution (activation) to the likelihood that data came from the other two distributions (Normal and negative Gamma)

Usage

```
N2G.Likelihood.Ratio(data, par)
```

Arguments

data	The dataset (usually a vector)
par	The parameter vector for the N2G model

Value

Returns the vector of likelihood ratio's

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Spatial.Mixture, N2G.Density , N2G.Likelihood , N2G.Transform, N2G.Fit , N2G , N2G.Inverse , N2G.Region

N2G.Region *N2G Normal component interval*

Description

Calculates the interval within which observations are classified as belonging to the Normal component of an N2G model.

Usage

```
N2G.Region(par1)
```

Arguments

par1	The parameters of the N2G model.

Value

Avector containing the upper and lower boundaries of the interval.

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density , N2G.Likelihood , N2G.Transform, N2G.Fit , N2G , N2G.Inverse

---

N2G.Spatial.Mixture *fMRI Spatial Mixture Modelling*

Description

Fits the spatial mixture model of Hartvig and Jensen (2000)

Usage

```
N2G.Spatial.Mixture(data, par.start = c(4, 2, 4, 2, 0.9, 0.05),
                    ksize, ktype = c("2D", "3D"), mask = NULL)
```

Arguments

data	The dataset (usually a vector)
par.start	Starting values for N2G model
ksize	Kernel size (see paper)
ktype	Format of kernel "2D" or "3D"
mask	Mask for dataset.

Value

p.map = a1, par = fit$par, lims = fit$lims Returns a list with following components

p.map	Posterior Probability Map of activation
par	Fitted parameters of the underlying N2G model
lims	Normal component interval for fitted model

Author(s)

J. L. Marchini

References

Hartvig and Jensen (2000) Spatial Mixture Modelling of fMRI Data

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Density, N2G.Likelihood, N2G.Transform, N2G.Fit, N2G, N2G.Inverse, N2G.Region

Examples

```
## simulate image
d <- c(100, 100, 1)
y <- array(0, dim = d)
m <- y
m[, , ] <- 1

z.init <- 2 * m
z.init[20:40, 20:40, 1] <- 1
z.init[50:70, 50:70, 1] <- 3

y[z.init == 1] <- -rgamma(sum(z.init == 1), 4, 1)
y[z.init == 2] <- rnorm(sum(z.init == 2))
y[z.init == 3] <- rgamma(sum(z.init == 3), 4, 1)

mask <- 1 * (y < 1000)

## fit spatial mixture model
ans <- N2G.Spatial.Mixture(y, par.start = c(4, 2, 4, 2, 0.9, 0.05),
                           ksize = 3, ktype = "2D", mask = m)

## plot original image, standard mixture model estimate and spatial mixture
## model estimate
oldpar <- par(mfrow = c(1, 3))
image(y[, , 1])
image(y[, , 1] > ans$lims[1]) ## this line plots the results of a Non-Spatial Mixture Model
image(ans$p.map[, , 1] > 0.5) ## this line plots the results of the Spatial Mixture Model
par(oldpar)
```

N2G.Transform *Transform parameters of N2G model so as to lie on the real line.*

Description

Transform parameters of N2G model so as to lie on the real line

Usage

```
N2G.Transform(par)
```

Arguments

par	Parameter vector for N2G model.

Details

Transformation required for optimization.

Value

Returns the transformed parameters.

Author(s)

J. L. Marchini

See Also

N2G.Class.Probability, N2G.Likelihood.Ratio, N2G.Spatial.Mixture, N2G.Density , N2G.Likelihood , N2G.Fit , N2G , N2G.Inverse , N2G.Region

---

nifti.quatern.to.mat44

Quaternion (etc..) to affine 4x4 matrix

Description

Generate a 4x4 affine matrix from a NIFIT header list.

Usage

```
nifti.quatern.to.mat44(L)
```

Arguments

L	a NIFTI header list

Value

The 4x4 affine matrix.

See Also

R2Q Q2R

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
nifti.quatern.to.mat44(L)
```

NonLinearSmoothArray *Non-linear spatial smmothing of 3D and 4D arrays.*

Description

Smooths the values in an array spatially using a weighting kernel that doesn't smooth across boundaries.

Usage

```
NonLinearSmoothArray(x, voxdim=c(1, 1, 1), radius=2, sm=3, mask=NULL)
```

Arguments

x	The array to be smoothed.
voxdim	The voxel dimensions of the array.
radius	The radius of the spatial smoothing
sm	The standard deviation of the Gaussian smoothing kernel.
mask	Optional mask for smoothing.

Details

For a 3D array the smoothed values are obtained through a weighted sum of the surrounding voxel values within the specfied radius. The weights are calculated using a Gaussian kernel function applied to the differences between the voxel and its surrounding voxels. In this way the smoothing is anisotropic.

For a 4D array the first 3 dimensions represent space and the fourth represents time. Therefore, each spatial location contains a time series of values. These time series are smoothed spatially in an anisotropic fashion. The sum of squared differences between each pair of time series are used to define the smoothing weights.

Value

The smoothed array is returned.

Author(s)

J. L. Marchini

See Also

GaussSmoothArray

Examples

```
#3D array
d<-rep(10,3)
a<-array(3,dim=d)
a[,5:10,5:10]<-7
a<-a+array(rnorm(n=1000,sd=1),dim=d)

h<-NonLinearSmoothArray(a,voxdim=c(1,1,1),radius=2,sm=3)
oldpar <- par(mfrow=c(2,2))
image(a[1,,],zlim=c(-1,12));title("Before smoothing")
image(h[1,,],zlim=c(-1,12));title("After smoothing")
persp(a[1,,],zlim=c(-1,12))
persp(h[1,,],zlim=c(-1,12))

#4D array
d<-c(10,10,10,20)
a<-array(1,dim=d)
a[,,6:10,]<-2
a<-a+array(rnorm(20000,sd=.1),dim=d)

h<-NonLinearSmoothArray(a,voxdim=c(1,1,1),radius=2,sm=3)

par(mfrow=c(2,2),mar=c(0,0,0,0))
for(i in 1:10){
    for(j in 10:1){
        plot(a[1,i,j,],type="l",ylim=c(0,3),axes=FALSE);box()
        lines(h[1,i,j,],col=2)
    }}
par(oldpar)
```

`orientation` *Orientation storage*

Description

To determine if data is stored in Radiological or Neurological order.

Usage

```
orientation(L)
```

Arguments

`L`	a NIFTI header list

Value

-1 for Radiological and 1 for Neurological.

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
orientation(L)
```

Q2R *Quaternion to rotation*

Description

Generate a (proper) rotation matrix from a quaternion.

Usage

```
Q2R(Q,qfac)
```

Arguments

Q	quaternion vector
qfac	qfac nifti field. It is pixdim[1]

Value

The rotation.

See Also

R2Q

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
Q <- c(L$quatern.b,L$quatern.c,L$quatern.d)
Q2R(Q,L$pixdim[1])
```

R2Q *Rotation to quaternion*

Description

Convert from (proper) rotation matrix to quaternion form.

Usage

```
R2Q(R,qfac=NULL)
```

Arguments

R	Rotation matrix
qfac	qfac nifti field. It is pixdim[1]. If NULL, R is transformed to have determinant 1

Value

The quaternion.

See Also

Q2R

Examples

```
L <- f.read.nifti.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))
Q <- c(L$quatern.b,L$quatern.c,L$quatern.d)
R <- Q2R(Q,L$pixdim[1])
Q
R2Q(R)
```

---

reduction *reduction*

Description

This function reduces the data in the row or col dimension.

Usage

```
reduction(X, row.red = TRUE)
```

Arguments

X	a matrix of size tm x vm which contains the functionnal images
row.red	Logical. Reduces the columns or the rows

Value

Xred	the reduced matrix

See Also

centering

Examples

```
X <- matrix(rnorm(5 * 4), nrow = 5, ncol = 4)
Xred <- reduction(X, row.red = TRUE)$Xred
```

`Sim.3D.GammaRF` *Simulate Gamma distributed Random Field*

Description

Simulates a Gamma distributed random field by simulating a Gaussian Random Field and transforming it to be Gamma distributed.

Usage

```
Sim.3D.GammaRF(d, voxdim, sigma, ksize, mask, shape, rate)
```

Arguments

`d`	A vector specifying the dimensions of a 3D or 4D array.
`voxdim`	The dimensions of each voxel.
`sigma`	The 3D covariance matrix of the field.
`ksize`	The size (in voxels) of the kernel with which to filter the independent field.
`mask`	A 3D mask for the field.
`shape`	The shape parameter of the Gamma distribution.
`rate`	The rate parameter of the Gamma distribution.

Value

A 3D array containing the simulated field

Author(s)

J. L. Marchini

Examples

```
d <- c(64, 64, 21)
FWHM <- 9
sigma <- diag(FWHM^2, 3) / (8 * log(2))
voxdim <- c(2, 2, 4)
m <- array(1, dim = d)

a <- Sim.3D.GammaRF(d = d, voxdim = voxdim, sigma = sigma,
                    ksize = 9, mask = m, shape = 6, rate = 1)
```

`Sim.3D.GRF` *Simulate a GRF*

Description

Simulates a Gaussian Random Field with specified dimensions and covariance structure.

Usage

```
Sim.3D.GRF(d, voxdim, sigma, ksize, mask = NULL, type = c("field", "max"))
```

Arguments

`d`	A vector specifying the dimensions of a 3D or 4D array.
`voxdim`	The dimensions of each voxel.
`sigma`	The 3D covariance matrix of the field.
`ksize`	The size (in voxels) of the kernel with which to filter the independent field.
`mask`	A 3D mask for the field.
`type`	If type == "field" then the simulated field together with the maximum of the field is returned. If type == "max" then the maximum of the field is returned.

Details

The function works by simulating a Gaussian r.v at each voxel location and then smoothing the field with a discrete filter to obtain a field with the desired covariance structure.

Value

`mat`	Contains the simulated field if type == "field", else NULL
`max`	The maximum value of the simulated field.

Author(s)

J. L. Marchini

See Also

`GaussSmoothArray`,`GaussSmoothKernel`

Examples

```
d <- c(64, 64, 21)
FWHM <- 9
sigma <- diag(FWHM^2, 3) / (8 * log(2))
voxdim <- c(2, 2, 4)
msk <- array(1, dim = d)

field <- Sim.3D.GRF(d = d, voxdim = voxdim, sigma = sigma, ksize = 9, mask = msk, type = "max")
```

`SmoothEst` *Estimate the variance-covariance matrix of a Gaussian random field*

Description

Estimate the variance-covariance matrix of a Gaussian random field

Usage

```
SmoothEst(mat, mask, voxdim, method = "Forman")
```

Arguments

`mat`	3D array that is the Gaussian Random Field.
`mask`	3D mask array.
`voxdim`	Vector of length 3 containing the voxel dimensions.
`method`	The estimator to use. method = "Forman" (the default) uses the estimator proposed in [1]. method = "Friston" uses the estimator proposed in [2, 3], but tis can be biased when the amount of smoothing is small compared to the size of each voxel (see [1] for more details and example below)

Details

Calculates the varaince-covariance matrix using the variance covariance matrix of partial derivatives.

Value

A (3x3) diagonal matrix.

Author(s)

J. L. Marchini

References

[1] Stephen D. Forman et al. (1995) Improved assessment of significant activation in functional magnetic resonance imaging (fMRI): Use of a cluster-size threshold. Magnetic Resonance in Medicine, 33:636-647.

[2] Karl J. Friston et al. (1991) Comparing functional (PET) images: the assessment of significant change. J. Cereb. Blood Flow Metab. 11:690-699.

[3] Stefan J. Kiebel et al. (1999) Robust smoothness estimation in statistical parametric maps using standardized residuals from the general linear model. NeuroImage, 10:756-766.

Examples

```
###############
## EXAMPLE 1 ##
###############
## example that illustrates the bias of the Friston
## method when smoothing is small compared to voxel size
## NB. The presence of bias becomes clearer if the
##     simulations below are run about 100 times and
##     the results averaged

ksize <- 13
d <- c(64, 64, 64)
voxdim <- c(1, 1, 1)
FWHM <- 2 ## using a small value of FWHM (=2) compared to voxel size (=1)
sigma <- diag(FWHM^2, 3) / (8 * log(2))
mask <- array(1, dim = d)
num.vox <- sum(mask)

grf <- Sim.3D.GRF(d = d, voxdim = voxdim, sigma = sigma,
                  ksize = ksize, mask = mask, type = "field")$mat

sigma
SmoothEst(grf, mask, voxdim, method = "Friston")
SmoothEst(grf, mask, voxdim, method = "Forman") ## compared to sigma
##the Forman estimator is better (on average) than the Friston estimator

###############
## EXAMPLE 2 ##
###############
## increasing the amount of smoothing decreases the bias of the Friston estimator

ksize <- 13
d <- c(64, 64, 64)
voxdim <- c(1, 1, 1)
FWHM <- 5 ## using a large value of FWHM (=5) compared to voxel size (=1)
sigma <- diag(FWHM^2, 3) / (8 * log(2))
mask <- array(1, dim = d)
num.vox <- sum(mask)

grf <- Sim.3D.GRF(d = d, voxdim = voxdim, sigma = sigma,
                  ksize = ksize, mask = mask, type = "field")$mat
```

```
SmoothEst(grf, mask, voxdim, method = "Friston")
SmoothEst(grf, mask, voxdim, method = "Forman")
sigma
```

---

st2xyzt *st2xyzt*

Description

Encode and assemble a space code with a time code dimension into the combined one byte xyzt.units field of a NIFTI header file.

Usage

```
st2xyzt(space,time)
```

Arguments

space	space field of a NIFTI file
time	time field of a NIFTI file

Value

A list containing `xyzt.units` field.

Bits 0..2 of `xyzt.units` specify the units of `pixdim[2..4]` (e.g., spatial units are values 0,1,2,...,7). Bits 3..5 of `xyzt.units` specify the units of `pixdim[5]` (e.g., temporal units are multiples of 8: 0,8,16,24,32,40,48,56).

This compression of 2 distinct concepts into 1 byte is due to the limited space available in the 348 byte ANALYZE 7.5 header.

Some NIFTI codes: 0 (unspecified units), 1 (meters), 2 (millimeters), 3 (micrometers), 8 (seconds), 16 (milliseconds), 24 (microseconds), 32 (Hertz), 40 (ppm, part per million) and 48 (radians per second).

See Also

xyzt2st

Examples

```
xyzt.units <- f.read.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))$xyzt.units
mylist <- xyzt2st(xyzt.units)
st2xyzt(mylist$space,mylist$time)
```

threeDto4D *threeDto4D*

Description

To read tm functionnal images files in ANALYZE or NIFTI format, and concatenate them to obtain one 4D image file in Analyze (hdr/img pair) or Nifti format (hdr/img pair or single nii) which is written on disk. Note that this function outputs the files in the format sent in. If desired, one can use the function `analyze2nifti` to create NIFTI files from ANALYZE files.

Usage

```
threeDto4D(outputfile, path.in = NULL, prefix = NULL, regexp = NULL, times = NULL,
     list.of.in.files = NULL, path.out = NULL, is.nii.pair = FALSE, hdr.number = 1)
```

Arguments

`outputfile`	character. Name of the outputfile without extension
`path.in`	character with the path to the directory containing the image files
`prefix`	character. Common prefix to each file
`regexp`	character. Regular expression to get all the files, e.g., "????.img"
`times`	vector. Numbers (indices) of the image files to retrieve
`list.of.in.files`	names of img files to concatenate (with full path)
`path.out`	where to write the output hdr/img pair files. Will be taken as path.in if not provided.
`is.nii.pair`	logical. Should we write a single nii NIFTI file or a hdr/img NIFTI pair file
`hdr.number`	Number of the original 3D Analyze or NIFTI image file from which to take the header that should serve as the final header of the newly 4D created image file

Value

None. Writes an Analyze or a NIFTI file on the disk.

See Also

`twoDto4D` `fourDto2D`

Examples

```
a <- f.read.analyze.volume(system.file("example.img", package = "AnalyzeFMRI"))
dim(a)
tmpdir <- tempdir()
b <- threeDto4D(outputfile = "fourdfile", path.in =
          paste0(system.file(package = "AnalyzeFMRI"), "/"),
      prefix = NULL, regexp = NULL, times = 2, list.of.in.files = rep("example.img", 2),
```

```
      path.out = tmpdir, is.nii.pair = FALSE, hdr.number = 1)
dim(b)
unlink(tmpdir) # tidy up
```

`Threshold.Bonferroni` *Calculates Bonferroni Threshold*

Description

Calculate the Bonferroni threshold for n iid tests that results in an overall p-value of p.val. The tests can be distributed as Normal, t or F.

Usage

```
Threshold.Bonferroni(p.val, n, type = c("Normal", "t", "F"), df1 = NULL, df2 = NULL)
```

Arguments

`p.val`	The required overall p-value.
`n`	The number of tests.
`type`	The distribution of the tests. One of "Normal", "t" or "F"
`df1`	The degrees of freedom of the t-distribution or the first degrees of freedom parameter for the F distribution.
`df2`	The second degrees of freedom parameter for the F distribution.

Value

Returns the Bonferroni threshold.

Examples

```
Threshold.Bonferroni(0.05, 1000)

Threshold.Bonferroni(0.05, 1000, type = c("t"), df1 = 20)

Threshold.Bonferroni(0.05, 1000, type = c("F"), df1 = 3, df2 = 100)
```

`Threshold.FDR` *False Discovery Rate (FDR) Threshold*

Description

Calculates the False Discovery Rate (FDR) threshold for a given vector of statistic values.

Usage

```
Threshold.FDR(x, q, cV.type = 2, type = c("Normal", "t", "F"), df1 = NULL, df2 = NULL)
```

Arguments

`x`	A vector of test statistic values.
`q`	The desired False Discovery Rate threshold.
`cV.type`	A flag that specifies the assumptions about the joint distribution of p-values. Choose cV.type = 2 for fMRI data (see Genovese et al (2001)
`type`	The distribution of the statistic values. Either "Normal", "t" or "F".
`df1`	The degrees of freedom of the t-distribution or the first degrees of freedom parameter for the F distribution.
`df2`	The second degrees of freedom parameter for the F distribution.

Value

Returns the FDR threshold.

Author(s)

J. L. Marchini

References

Genovese et al. (2001) Thresholding of Statistical Maps in Functional NeuroImaging Using the False Discovery Rate.

Examples

```
x <- c(rnorm(1000), rnorm(100, mean = 3))
Threshold.FDR(x = x, q = 0.20, cV.type = 2)
```

`Threshold.RF` *Random Field Theory Thersholds.*

Description

Calculates the Random Field theory threshold to give that results in a specified p-value.

Usage

```
Threshold.RF(p.val, sigma, voxdim = c(1, 1, 1), num.vox,
             type = c("Normal", "t"), df = NULL)
```

Arguments

`p.val`	The required p-value.
`sigma`	The 3D covariance matrix of the random field.
`voxdim`	The dimesnions of a voxel.
`num.vox`	The number of voxels that constitute the random field.
`type`	The type of random field, "Normal" or "t".
`df`	The degrees of the t distributed field.

Details

Calculates the threshold that produces an expected Euler characteristic equal to the required p-value.

Value

Returns the Random Field threshold.

Author(s)

J. L. Marchini

See Also

`EC.3D`

Examples

```
a <- Threshold.RF(p.val = 0.05, sigma = diag(1, 3), voxdim = c(1, 1, 1), num.vox = 10000)

EC.3D(a, sigma = diag(1, 3), voxdim = c(1, 1, 1), num.vox = 10000)
```

twoDto4D *twoDto4D*

Description

This function transform a 2D matrix of size tm x vm containing 3D images in each row into a 4D array image.

Usage

```
twoDto4D(x.2d, dim)
```

Arguments

x.2d	a 2D matrix to be transformed
dim	vector of length 4 containing the dimensions of the array. dim[1:3] are the space dimensions. dim[4] is the time dimension

Value

a 4D array image

See Also

threeDto4D fourDto2D

Examples

```
# contains t = 3 (fake) 3D "images" each of size 2 x 4 x 4
x.2d <- matrix(rnorm(3 * 32), nrow = 3, ncol = 32)
volume.4d <- twoDto4D(x.2d, dim = c(2, 4, 4, 3))
dim(volume.4d)
```

with_dir *Working directory*

Description

Temporarily change the current working directory.

Usage

```
with_dir(new, code)
```

Arguments

new	[character(1)] New working directory
code	[any] Code to execute in the temporary environment

Value

[any]
The results of the evaluation of the code argument.

See Also

setwd()

Examples

```
getwd()

with_dir(tempdir(), getwd())
```

---

xyz2ijk *xyz2ijk*

Description

This function maps from some real world (x,y,z) positions in space into data coordinates (e.g. column i, row j, slice k). These original positions could relate to Talairach-Tournoux (T&T) space, MNI space, or patient-based scanner coordinates.

Usage

```
xyz2ijk(xyz=c(1,1,1),method=2,L)
```

Arguments

xyz	matrix. Each column of xyz should contain a voxel real world index coordinates (x,y,z) to be mapped to its (i,j,k) voxel index coordinates in the dataset
method	1 (qform.code=sform.code=0),2 (qform.code>0, rigid transformation) or 3 (sform.code>0, affine transformation).
L	header list of a NIFTI file

Details

See help page of function ijk2xyz().

Value

A list containing the matrix xyz of the positions of the points specified in ijk.

See Also

ijk2xyz Q2R R2Q

Examples

```
L <- f.read.header(system.file("example-nifti.hdr",
package="AnalyzeFMRI"))
xyz <- matrix(c(1,1,1,2,3,7),byrow=FALSE,nrow=3)
xyz2ijk(xyz=xyz,method=2,L)
```

---

xyzt2st *xyzt2st*

Description

Extract space and time dimension codes from the one byte xyzt.units field of a NIFTI header file.

Usage

```
xyzt2st(xyzt.units, verbose = TRUE)
```

Arguments

xyzt.units	xyzt.units field of a NIFTI header file
verbose	verbose logical to display information about the units

Value

A list containing space and time fields.

See also the Value Section of the help file of function st2xyzt().

See Also

st2xyzt

Examples

```
xyzt.units <- f.read.header(system.file("example-nifti.hdr", package="AnalyzeFMRI"))$xyzt.units
xyzt2st(xyzt.units)
```

Index

∗ **utilities**

analyze2nifti, 3
centering, 5
cluster.threshold, 6
cov.est, 7
diminfo2fps, 8
EC.3D, 9
eigenvalues, 10
f.analyze.file.summary, 11
f.analyzeFMRI.gui, 11
f.basic.hdr.list.create, 12
f.basic.hdr.nifti.list.create, 12
f.complete.hdr.nifti.list.create, 13
f.ica.fmri, 17
f.ica.fmri.gui, 19
f.icast.fmri, 20
f.icast.fmri.gui, 21
f.nifti.file.summary, 22
f.plot.ica.fmri, 23
f.plot.ica.fmri.jpg, 24
f.plot.volume.gui, 24
f.read.analyze.header, 25
f.read.analyze.slice, 28
f.read.analyze.slice.at.all.timepoints, 28
f.read.analyze.tpt, 29
f.read.analyze.ts, 30
f.read.analyze.volume, 31
f.read.header, 31
f.read.nifti.header, 32
f.read.nifti.slice, 36
f.read.nifti.slice.at.all.timepoints, 37
f.read.nifti.tpt, 37
f.read.nifti.ts, 38
f.read.nifti.volume, 39
f.read.volume, 39
f.spectral.summary, 40
f.spectral.summary.nifti, 41
f.write.analyze, 41
f.write.array.to.img.2bytes, 42
f.write.array.to.img.8bit, 43
f.write.array.to.img.float, 44
f.write.list.to.hdr, 44
f.write.list.to.hdr.nifti, 45
f.write.nifti, 46
f.write.nii.array.to.img.2bytes, 47
f.write.nii.array.to.img.8bit, 47
f.write.nii.array.to.img.float, 48
fourDto2D, 49
fps2diminfo, 49
GaussSmoothArray, 50
GaussSmoothKernel, 51
ICAspat, 52
ICAtemp, 52
ijk2xyz, 53
magicfield, 59
mat34.to.TRSZ, 60
mat34.to.TZSR, 61
model.2.cov.func, 62
model.2.est.gamma, 62
N2G, 63
N2G.Class.Probability, 64
N2G.Density, 65
N2G.Fit, 65
N2G.Inverse, 66
N2G.Likelihood, 67
N2G.Likelihood.Ratio, 68
N2G.Region, 68
N2G.Spatial.Mixture, 69
N2G.Transform, 70
nifti.quatern.to.mat44, 71
NonLinearSmoothArray, 72
orientation, 73
Q2R, 74
R2Q, 74

reduction, 75
Sim.3D.GammaRF, 76
Sim.3D.GRF, 77
SmoothEst, 78
st2xyzt, 80
threeDto4D, 81
Threshold.Bonferroni, 82
Threshold.FDR, 83
Threshold.RF, 84
twoDto4D, 85
xyz2ijk, 86
xyzt2st, 87

analyze2nifti, 3

centering, 5, *75*
cluster.threshold, 6
cluster_mass *(*cluster.threshold*)*, 6
cov.est, 7
covariance_est *(*cov.est*)*, 7

diminfo2fps, 8, *50*

EC.3D, 9, *84*
eigenvalues, 10

f.analyze.file.summary, 11, *12*, *27*, *40*
f.analyzeFMRI.gui, 11
f.basic.hdr.list.create, *11*, 12, *45*
f.basic.hdr.nifti.list.create, 12, *16*, *22*, *45*
f.complete.hdr.nifti.list.create, 13
f.ica.fmri, 17, *19*, *23*, *24*
f.ica.fmri.gui, 19, *19*, *22*, *23*
f.icast.fmri, 20, *22*
f.icast.fmri.gui, 21, *21*, *25*
f.nifti.file.summary, *13*, *16*, 22, *41*
f.plot.ica.fmri, *19*, 23
f.plot.ica.fmri.jpg, 24
f.plot.volume.gui, 24
f.read.analyze.header, *11*, 25, *32*
f.read.analyze.slice, *11*, 28, *29–31*
f.read.analyze.slice.at.all.timepoints, *11*, 28, *28–31*
f.read.analyze.tpt, 29
f.read.analyze.ts, *11*, *28*, *29*, 30, *31*
f.read.analyze.volume, *11*, *28*, *29*, 31
f.read.header, 31
f.read.nifti.header, *22*, 32, *32*
f.read.nifti.slice, *22*, 36, *37–40*
f.read.nifti.slice.at.all.timepoints, *22*, *36*, 37, *38–40*
f.read.nifti.tpt, 37
f.read.nifti.ts, *22*, *36*, *37*, 38, *39*, *40*
f.read.nifti.volume, *22*, *36*, *37*, 39
f.read.volume, 39
f.spectral.summary, *11*, 40
f.spectral.summary.nifti, *22*, 41
f.write.analyze, *11*, *29*, *30*, 41, *43*, *44*
f.write.array.to.img.2bytes, *11*, *22*, 42, *42–44*, *46*
f.write.array.to.img.8bit, *42*, 43, *44*, *46*
f.write.array.to.img.float, *11*, *22*, *42*, *43*, 44, *46*
f.write.list.to.hdr, *11*, *12*, 44
f.write.list.to.hdr.nifti, *13*, *16*, *22*, 45
f.write.nifti, *22*, *38*, 46, *47*, *48*
f.write.nii.array.to.img.2bytes, *46*, 47, *48*
f.write.nii.array.to.img.8bit, *46*, 47, *48*
f.write.nii.array.to.img.float, *46*, *47*, 48, *48*
fourDto2D, 49, *81*, *85*
fps2diminfo, *8*, 49

gaussfilter1 *(*GaussSmoothKernel*)*, 51
gaussfilter2 *(*GaussSmoothKernel*)*, 51
GaussSmoothArray, 50, *72*, *77*
GaussSmoothKernel, 51, *51*, *77*

ica_fmri_JM *(*ICAspat*)*, 52
ICAspat, 52, *53*
ICAtemp, 52, *52*
ijk2xyz, 53, *87*

jpeg, *24*

magicfield, 59
mat34.to.TRSZ, 60, *61*
mat34.to.TZSR, *60*, 61
model.2.cov.func, 62
model.2.est.gamma, 62

N2G, 63, *64–71*
N2G.Class.Probability, 64, *64–71*
N2G.Density, *64*, 65, *66–71*
N2G.Fit, *64*, 65, *65*, *67–71*

N2G.Inverse, *64, 65*, 66, *66–71*
N2G.Likelihood, *64–66*, 67, *67–71*
N2G.Likelihood.Ratio, *64–67*, 68, *69–71*
N2G.Region, *64–67*, 68, *68*, *70, 71*
N2G.Spatial.Mixture, *64–68*, 69, *69*, *71*
N2G.Transform, *64–69*, 70, *70*
nifti.quatern.to.mat44, 71
non_lin_gauss_smooth *(*NonLinearSmoothArray*)*, 72
NonLinearSmoothArray, 72

orientation, 73

Q2R, *59–61*, *71*, 74, *75*, *87*

R2Q, *59–61*, *71*, 74, *74*, *87*
read2byte_v1_JM *(*f.read.analyze.volume*)*, 31
read4byte_v1_JM *(*f.read.analyze.volume*)*, 31
read_analyze_header_wrap_JM *(*f.read.analyze.volume*)*, 31
read_nifti_header_wrap_JM *(*f.read.nifti.header*)*, 32
read_nifti_magic_wrap *(*f.read.nifti.header*)*, 32
readchar_v1_JM *(*f.read.analyze.volume*)*, 31
readdouble_v1_JM *(*f.read.analyze.volume*)*, 31
readfloat_v1_JM *(*f.read.analyze.volume*)*, 31
reduction, *5*, 75

setwd(), *86*
Sim.3D.GammaRF, 76
Sim.3D.GRF, 77
sim_grf *(*Sim.3D.GRF*)*, 77
SmoothEst, 78
spatial_mixture *(*N2G.Spatial.Mixture*)*, 69
st2xyzt, 80, *87*
swaptest_wrap_JM *(*f.read.analyze.volume*)*, 31

temporal_non_lin_gauss_smooth *(*NonLinearSmoothArray*)*, 72
threeDto4D, *49*, 81, *85*
Threshold.Bonferroni, 82
Threshold.FDR, 83
Threshold.RF, *10*, 84
twoDto4D, *49*, *81*, 85

with_dir, 85
write2byte_JM *(*f.read.analyze.volume*)*, 31
write2byteappend_JM *(*f.write.analyze*)*, 41
write8bit_JM *(*f.write.analyze*)*, 41
write8bitappend_JM *(*f.write.analyze*)*, 41
write_analyze_header_wrap_JM *(*f.write.analyze*)*, 41
write_nifti_header_wrap_JM *(*f.write.analyze*)*, 41
writefloat_JM *(*f.write.analyze*)*, 41
writefloatappend_JM *(*f.write.analyze*)*, 41

xyz2ijk, *59*, 86
xyzt2st, *80*, 87