# Package ‘Morpho’

January 20, 2025

**Type** Package

**Title** Calculations and Visualisations Related to Geometric Morphometrics

**Version** 2.12

**Date** 2023-12-04

**Description** A toolset for Geometric Morphometrics and mesh processing. This includes (among other stuff) mesh deformations based on reference points, permutation tests, detection of outliers, processing of sliding semi-landmarks and semi-automated surface landmark placement.

**Suggests** car, lattice, shapes, testthat

**Depends** R (>= 3.2.0)

**Imports** Rvcg (>= 0.7), rgl (>= 0.100.18), foreach (>= 1.4.0), Matrix (>= 1.0-1), MASS, parallel, doParallel (>= 1.0.6), colorRamps, Rcpp, graphics, grDevices, methods, stats, utils, jsonlite, sf, bezier

**LinkingTo** Rcpp, RcppArmadillo (>= 0.4)

**Copyright** see COPYRIGHTS file for details

**License** GPL-2

**BugReports** `https://github.com/zarquon42b/Morpho/issues`

**LazyLoad** yes

**URL** `https://github.com/zarquon42b/Morpho`

**Encoding** UTF-8

**RoxygenNote** 7.2.3

**NeedsCompilation** yes

**Author** Stefan Schlager [aut, cre, cph],
Gregory Jefferis [ctb],
Dryden Ian [cph]

**Maintainer** Stefan Schlager `<zarquon42@gmail.com>`

**Repository** CRAN

**Date/Publication** 2023-12-06 15:20:07 UTC

# Contents

---

`Morpho-package` *A toolbox providing methods for data-acquisition, visualisation and statistical methods related to Geometric Morphometrics and shape analysis*

---

## Description

A toolbox for Morphometric calculations. Including sliding operations for Semilandmarks, importing, exporting and manipulating of 3D-surface meshes and semi-automated placement of surface landmarks.

## Details

| | |
|---|---|
| Package: | Morpho |
| Type: | Package |
| Version: | 2.12 |
| Date: | 2023-12-04 |
| License: | GPL |
| LazyLoad: | yes |

## Note

The pdf-version of Morpho-help can be obtained from CRAN on `https://cran.r-project.org/package=Morpho`

For more advanced operations on triangular surface meshes, check out my package Rvcg: `https://cran.r-project.org/package=Rvcg` or the code repository on github `https://github.com/zarquon42b/Rvcg`

## Author(s)

Stefan Schlager <`zarquon42@gmail.com`>

Maintainer: Stefan Schlager <`zarquon42@gmail.com`>

## References

Schlager S. 2013. Soft-tissue reconstruction of the human nose: population differences and sexual dimorphism. PhD thesis, Universitätsbibliothek Freiburg. URL: http://www.freidok.uni-freiburg.de/volltexte/9181/.

---

`align2procSym` *align new data to an existing Procrustes registration*

---

## Description

align new data to an existing Procrustes registration

## Usage

```
align2procSym(x, newdata, orp = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | result of a `procSym` call |
| `newdata` | matrix or array of with landmarks corresponding to the data aligned in x |
| `orp` | logical: allows to skip orthogonal projection, even if it was used in the `procSym` call. |

## Value

an array with data aligned to the mean shape in x (and projected into tangent space)

## Note

this will never yield the same result as a pooled Procrustes analysis because the sample mean is iteratively updated and new data would change the mean.

## Examples

```
require(Morpho)
data(boneData)
# run procSym on entire data set
proc <- procSym(boneLM)
# this is the training data
array1 <- boneLM[,,1:60]
newdata <- boneLM[,,61:80]
proc1 <- procSym(array1)
newalign <- align2procSym(proc1,newdata)
## compare alignment for one specimen to Proc. registration using all data
## Not run:
deformGrid3d(newalign[,,1],proc$orpdata[,,61])

## End(Not run)
```

## angle.calc *calculate angle between two vectors*

### Description

calculates unsigned angle between two vectors

### Usage

```
angle.calc(x, y)
```

### Arguments

| | |
|---|---|
| x | numeric vector (or matrix to be interpreted as vector) |
| y | numeric vector (or matrix to be interpreted as vector) of same length as x |

### Value

angle between x and y in radians.

### Examples

```
#calculate angle between two centered and
# superimposed landmark configuration
data(boneData)
opa <- rotonto(boneLM[,,1],boneLM[,,2])
angle.calc(opa$X, opa$Y)
```

## angleTest *Test whether the direction of two vectors is similar*

### Description

Test whether the direction of two vectors is similar

### Usage

```
angleTest(x, y)
```

### Arguments

| | |
|---|---|
| x | vector |
| y | vector |

## Details

Under the assumption of all (normalized) n-vectors being represented by an n-dimensional hypersphere, the probability of the angle between two vectors is <= the measured values can be estimated as the area of a cap defined by that angle and divided by the hypersphere's complete surface area.

## Value

a list with

| | |
|---|---|
| `angle` | angle between vectors |
| `p.value` | p-value for the probability that the angle between two random vectors is smaller or equal to the one calculatted from x and y |

## References

S. Li , 2011. Concise Formulas for the Area and Volume of a Hyperspherical Cap. Asian Journal of Mathematics & Statistics, 4: 66-70.

## Examples

```
x <- c(1,0); y <- c(1,1) # for a circle this should give us p = 0.25 as the angle between vectors
##  is pi/4 and for any vector the segment +-pi/4 covers a quarter of the circle
angleTest(x,y)
```

---

## `anonymize` *Replace ID-strings of data and associated files.*

---

## Description

Replace ID-strings with for digits - e.g. for blind observer error testing.

## Usage

```
anonymize(
  data,
  remove,
  path = NULL,
  dest.path = NULL,
  ext = ".ply",
  split = "_",
  levels = TRUE,
  prefix = NULL,
  suffix = NULL,
  sample = TRUE
)
```

### Arguments

| | |
|---|---|
| `data` | Named array, matrix or vector containing data. |
| `remove` | integer: which entry (separated by `split`) of the name is to be removed |
| `path` | Path of associated files to be copied to renamed versions. |
| `dest.path` | where to put renamed files. |
| `ext` | file extension of files to be renamed. |
| `split` | character: by which to split specimen-ID |
| `levels` | logical: if a removed entry is to be treated as a factor. E.g. if one specimen has a double entry, the anonymized versions will be named accordingly. |
| `prefix` | character: prefix before the alias string. |
| `suffix` | character: suffix after the alias ID-string. |
| `sample` | logical: whether to randomize alias ID-string. |

### Value

| | |
|---|---|
| `data` | data with names replaced |
| `anonymkey` | map of original name and replaced name |

### Examples

```
anonymize(iris,remove=1)
```

---

## `applyTransform` *apply affine transformation to data*

---

### Description

apply affine transformation to data

### Usage

```
applyTransform(x, trafo, ...)

## S3 method for class 'matrix'
applyTransform(x, trafo, inverse = FALSE, threads = 1, ...)

## S3 method for class 'mesh3d'
applyTransform(x, trafo, inverse = FALSE, threads = 1, ...)

## Default S3 method:
applyTransform(x, trafo, inverse = FALSE, threads = 1, ...)
```

## Arguments

| | |
|---|---|
| `x` | matrix or mesh3d |
| `trafo` | 4x4 transformation matrix or an object of class "tpsCoeff" |
| `...` | additional arguments, currently not used. |
| `inverse` | logical: if TRUE, the inverse of the transformation is applied (for TPS coefficients have to be recomputed) |
| `threads` | threads to be used for parallel execution in tps deformation. |

## Value

the transformed object

## See Also

`rotonto`, `link{rotmesh.onto}`, `tps3d`, `computeTransform`

## Examples

```
data(boneData)
rot <- rotonto(boneLM[,,1],boneLM[,,2])
trafo <- getTrafo4x4(rot)
boneLM2trafo <- applyTransform(boneLM[,,2],trafo)
```

---

`areaSphere` *compute the area of an n-dimensional hypersphere*

---

## Description

compute the area of an n-dimensional hypersphere

## Usage

```
areaSphere(n, r = 1)
```

## Arguments

| | |
|---|---|
| `n` | dimensionality of space the hypersphere is embedded in (e.g.3 for a 3D-sphere) |
| `r` | radius of the sphere |

## Value

returns the area

## Examples

```
areaSphere(2) #gives us the circumference of a circle of radius 1
```

## `areaSpherePart` *compute the area of an n-dimensional hypersphere cap*

### Description

compute the area of an n-dimensional hypersphere cap

### Usage

```
areaSpherePart(n, phi, r = 1)
```

### Arguments

| | |
|---|---|
| `n` | dimensionality of space the hypersphere is embedded in (e.g.3 for a 3D-sphere) |
| `phi` | angle between vectors defining the cone |
| `r` | radius of the sphere |

### Value

returns the area of the hypersphere cap

### Examples

```
areaSpherePart(2,pi/2) # covers half the area of a circle
```

## `armaGinv` *calculate Pseudo-inverse of a Matrix using RcppArmadillo*

### Description

a simple wrapper to call Armadillo's pinv function

### Usage

```
armaGinv(x, tol = NULL)
```

### Arguments

| | |
|---|---|
| `x` | numeric matrix |
| `tol` | numeric: maximum singular value to be considered |

### Value

Pseudo-inverse

### Examples

```
mat <- matrix(rnorm(12),3,4)
pinvmat <- armaGinv(mat)
```

---

## `array2list` *reverts list2array, converting an array to a list of matrices*

---

### Description

reverts list2array, converting an array to a list of matrices

### Usage

```
array2list(x)
```

### Arguments

| | |
|---|---|
| `x` | array |

### Value

returns a list containing the matrices

---

## `arrMean3` *calculate mean of an array*

---

### Description

calculate mean of a 3D-array (e.g. containing landmarks) (fast) using the Armadillo C++ Backend

### Usage

```
arrMean3(arr)
```

### Arguments

| | |
|---|---|
| `arr` | `k x m x n` dimensional numeric array |

### Value

matrix of dimensions `k x m`.

### Note

this is the same as `apply(arr, 1:2, mean)`, only faster for large configurations.

## Examples

```
data(boneData)
proc <- ProcGPA(boneLM, silent = TRUE)
mshape <- arrMean3(proc$rotated)
```

---

## asymPermute — *Assess differences in amount and direction of asymmetric variation (only object symmetry)*

---

### Description

Assess differences in amount and direction of asymmetric variation (only object symmetry)

### Usage

```
asymPermute(x, groups, rounds = 1000, which = NULL)
```

### Arguments

| | |
|---|---|
| `x` | object of class symproc result from calling `procSym` with `pairedLM` specified |
| `groups` | factors determining grouping. |
| `rounds` | number of permutations |
| `which` | select which factorlevels to use, if NULL, all pairwise differences will be assessed after shuffling pooled data. |

### Value

| | |
|---|---|
| `dist` | difference between vector lengths of group means |
| `angle` | angle (in radians) between vectors of group specific asymmetric deviation |
| `means` | actual group averages |
| `p.dist` | p-value obtained by comparing the actual distance to randomly acquired distances |
| `p.angle` | p-value obtained by comparing the actual angle to randomly acquired angles |
| `permudist` | vector containing differences between random group means' vector lenghts |
| `permuangle` | vector containing angles between random group means' vectors |
| `groupmeans` | array with asymmetric displacement per group |
| `levels` | character vector containing the factors used |

### Note

This test is only sensible if between-group differences concerning directional asymmetry have been established (e.g. by applying a MANOVA on the "asymmetric" PCscores (see also `procSym`) and one wants to test whether these can be attributed to differences in amount and/or direction of asymmetric displacement. Careful interpretation for very small amounts of directional asymmetry is advised. The Null-Hypothesis is that we have the same directional asymmetry in both groups. If you want to test whether the angle between groups is similar, please use `angleTest`.

### See Also

procSym

---

barycenter *calculates the barycenters for all faces of a triangular mesh*

---

### Description

calculates the barycenters for all faces of a triangular mesh

### Usage

```
barycenter(mesh)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class 'mesh3d' |

### Value

k x 3 matrix of barycenters for all k faces of input mesh.

### See Also

closemeshKD

### Examples

```
data(nose)
bary <- barycenter(shortnose.mesh)
## Not run:
require(rgl)
##visualize mesh
wire3d(shortnose.mesh)
# visualize barycenters
points3d(bary, col=2)
## now each triangle is equipped with a point in its barycenter

## End(Not run)
```

## `bindArr` *concatenate multiple arrays/matrices*

### Description

concatenate multiple 3-dimensional arrays and/or 2-dimensional matrices to one big array

### Usage

```
bindArr(..., along = 1, collapse = FALSE)
```

### Arguments

| | |
|---|---|
| `...` | matrices and/or arrays with appropriate dimensionality to combine to one array, or a single list containing suitable matrices, or arrays). |
| `along` | dimension along which to concatenate. |
| `collapse` | logical: if the resulting array is shallow (only 1 dimension deep), it is converted to a matrix. |

### Details

dimnames, if present and if differing between entries, will be concatenated, separated by a "_".

### Value

returns array of combined matrices/arrays

### See Also

`cbind`, `rbind`, `array`

### Examples

```
A <- matrix(rnorm(18),6,3)
B <- matrix(rnorm(18),6,3)
C <- matrix(rnorm(18),6,3)

#combine to 3D-array
newArr <- bindArr(A,B,C,along=3)
#combine along first dimension
newArr2 <- bindArr(newArr,newArr,along=1)
```

## boneData *Landmarks and a triangular mesh*

### Description

Landmarks on the osseous human nose and a triangular mesh representing this structure.

### Format

`boneLM`: A 10x3x80 array containing 80 sets of 3D-landmarks placed on the human osseous nose.

`skull_0144_ch_fe.mesh`: The mesh representing the area of the first individual of `boneLM`

## CAC *calculate common allometric component*

### Description

calculate common allometric component

### Usage

```
CAC(x, size, groups = NULL, log = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | datamatrix (e.g. with PC-scores) or 3D-array with landmark coordinates |
| `size` | vector with Centroid sizes |
| `groups` | grouping variable |
| `log` | logical: use `log(size)` |

### Value

| | |
|---|---|
| `CACscores` | common allometric component scores |
| `CAC` | common allometric component |
| `x` | (group-) centered data |
| `sc` | CAC reprojected into original space by applying `CAC %*% x` |
| `RSCscores` | residual shape component scores |
| `RSC` | residual shape components |
| `gmeans` | groupmeans |
| `CS` | the centroid sizes (log transformed if `log = TRUE`) |

## References

Mitteroecker P, Gunz P, Bernhard M, Schaefer K, Bookstein FL. 2004. Comparison of cranial ontogenetic trajectories among great apes and humans. Journal of Human Evolution 46(6):679-97.

## Examples

```
data(boneData)
proc <- procSym(boneLM)
pop.sex <- name2factor(boneLM,which=3:4)
cac <- CAC(proc$rotated,proc$size,pop.sex)
plot(cac$CACscores,cac$size)#plot scores against Centroid size
cor.test(cac$CACscores,cac$size)#check for correlation
#visualize differences between large and small on the sample's consensus
## Not run:
large <- restoreShapes(max(cac$CACscores),cac$CAC,proc$mshape)
small <- restoreShapes(min(cac$CACscores),cac$CAC,proc$mshape)
deformGrid3d(small,large,ngrid=0)

## End(Not run)
```

---

## `cExtract` *extract information about fixed landmarks, curves and patches from and atlas generated by "landmark"*

---

## Description

After exporting the pts file of the atlas from "landmark" and importing it into R via "read.pts" cExtract gets information which rows of the landmark datasets belong to curves or patches.

## Usage

```
cExtract(pts.file)
```

## Arguments

| | |
|---|---|
| `pts.file` | either a character naming the path to a pts.file or the name of an object imported via read.pts. |

## Value

returns a list containing the vectors with the indices of matrix rows belonging to the in "landmark" defined curves, patches and fix landmarks and a matrix containing landmark coordinates.

## Author(s)

Stefan Schlager

## See Also

`read.lmdta`,`read.pts`

## checkLM *Visually browse through a sample rendering its landmarks and corresponding surfaces.*

### Description

Browse through a sample rendering its landmarks and corresponding surfaces. This is handy e.g. to check if the landmark projection using placePatch was successful, and to mark specific specimen.

### Usage

```
checkLM(
  dat.array,
  path = NULL,
  prefix = "",
  suffix = ".ply",
  col = "white",
  pt.size = NULL,
  alpha = 1,
  begin = 1,
  render = c("w", "s"),
  point = c("s", "p"),
  add = FALSE,
  meshlist = NULL,
  Rdata = FALSE,
  atlas = NULL,
  text.lm = FALSE
)
```

### Arguments

| | |
|---|---|
| `dat.array` | array or list containing landmark coordinates. |
| `path` | optional character: path to files where surface meshes are stored locally. If not specified only landmarks are displayed. |
| `prefix` | prefix to attach to the filenames extracted from `dimnames(dat.array)[[3]]` (in case of an array), or `names(dat.array)` (in case of a list) |
| `suffix` | suffix to attach to the filenames extracted from `dimnames(dat.array)[[3]]` (in case of an array), or `names(dat.array)` (in case of a list) |
| `col` | mesh color |
| `pt.size` | size of plotted points/spheres. If `point="s"`. `pt.size` defines the radius of the spheres. If `point="p"` it sets the variable `size` used in `point3d`. |
| `alpha` | value between 0 and 1. Sets transparency of mesh 1=opaque 0= fully transparent. |
| `begin` | integer: select a specimen to start with. |
| `render` | if render="w", a wireframe will be drawn, else the meshes will be shaded. |

| | |
|---|---|
| `point` | how to render landmarks. "s"=spheres, "p"=points. |
| `add` | logical: add to existing rgl window. |
| `meshlist` | list holding meshes in the same order as `dat.array` (Overrides `path`). |
| `Rdata` | logical: if the meshes are previously stored as Rdata-files by calling save(), these are simply loaded and rendered. Otherwise it is assumed that the meshes are stored in standard file formats such as PLY, STL or OBJ, that are then imported with the function `file2mesh`. |
| `atlas` | provide object generated by `createAtlas` to specify coloring of surface patches, curves and landmarks |
| `text.lm` | logical: number landmarks. Only applicable when `atlas=NULL`. |

## Value

returns an invisible vector of indices of marked specimen.

## See Also

`placePatch`, `createAtlas`, `plotAtlas`,`file2mesh`

## Examples

```
data(nose)
###create mesh for longnose
longnose.mesh <- tps3d(shortnose.mesh,shortnose.lm,longnose.lm,threads=1)
### write meshes to disk
save(shortnose.mesh, file="shortnose")
save(longnose.mesh, file="longnose")

## create landmark array
data <- bindArr(shortnose.lm, longnose.lm, along=3)
dimnames(data)[[3]] <- c("shortnose", "longnose")
## Not run:
checkLM(data, path="./",Rdata=TRUE, suffix="")

## End(Not run)

## now visualize by using an atlas:
atlas <- createAtlas(shortnose.mesh, landmarks =
           shortnose.lm[c(1:5,20:21),],
patch=shortnose.lm[-c(1:5,20:21),])
if (interactive()){
checkLM(data, path="./",Rdata=TRUE, suffix="", atlas=atlas)
}
## remove data from disk
unlink("shortnose")
unlink("longnose")
```

## checkNA *check for NA values in a matrix (of landmarks)*

### Description

check for NA values in a matrix (of landmarks)

### Usage

```
checkNA(x)
```

### Arguments

| | |
|---|---|
| x | matrix containing landmarks |

### Value

returns a vector with missin landmarks and a vector of length=0 if none are missing

## classify *classify specimen based on between-group PCA or CVA or typprobClass*

### Description

classify specimen based on between-group PCA, CVA or typprobClass

### Usage

```
classify(x, cv = TRUE, ...)

## S3 method for class 'bgPCA'
classify(x, cv = TRUE, newdata = NULL, ...)

## S3 method for class 'CVA'
classify(x, cv = T, newdata = NULL, prior = NULL, ...)

## S3 method for class 'typprob'
classify(x, cv = TRUE, ...)
```

## Arguments

| | |
|---|---|
| `x` | result of groupPCA, CVA or typprobClass |
| `cv` | logical: use cross-validated scores if available |
| `...` | currently not used |
| `newdata` | use new data to predict scores and evaluate group affinity |
| `prior` | specify prior probability for CVA evaluation if NULL prior from CVA will be used. Be `m` your number of groups then to set the prior equally for all groups set `prior=rep(1,m)/m`. |

## Value

| | |
|---|---|
| `class` | classification result |
| `groups` | original grouping variable, only available if `newdata=NULL` |
| `posterior` | only for object of CVA and typprob, also the posterior probabilities are returned |

## See Also

`CVA`,`groupPCA`, `typprobClass`

---

`closemeshKD` *Project coordinates onto a target triangular surface mesh.*

---

## Description

For a set of 3D-coordinates the closest matches on a target surface are determined and normals at as well as distances to that point are calculated.

## Usage

```
closemeshKD(
  x,
  mesh,
  k = 50,
  sign = FALSE,
  barycoords = FALSE,
  cores = 1,
  method = 0,
  ...
)
```

## Arguments

| | |
|---|---|
| x | k x 3 matrix containing 3D-coordinates or object of class `mesh3d`. |
| mesh | triangular surface mesh stored as object of class `mesh3d`. |
| k | neighbourhood of kd-tree to search - the larger, the slower - but the more likely the absolutely closest point is hit. |
| sign | logical: if TRUE, signed distances are returned. |
| barycoords | logical: if `TRUE`, barycentric coordinates of the hit points are returned. |
| cores | integer: how many cores to use for the search algorithm. |
| method | integer: either 0 or 1, if 0 ordinary Euclidean distance is used, if 1, the distance suggested by Moshfeghi(1994) is calculated. |
| ... | additional arguments. currently unavailable. |

## Details

The search for the clostest point is designed as follows: Calculate the barycenter of each target face. For each coordinate of x, determine the k closest barycenters and calculate the distances to the closest point on these faces.

## Value

returns an object of class `mesh3d`. with:

| | |
|---|---|
| vb | 4xn matrix containing n vertices as homolougous coordinates |
| normals | 4xn matrix containing vertex normals |
| quality | vector: containing distances to target. In case of `method=1`, this is not the Euclidean distance but the distance of the reference point to the faceplane (orthogonally projected) plus the distance to the closest point on one of the face's edges (the target point). See the literature cited below for details. |
| it | 4xm matrix containing vertex indices forming triangular faces.Only available, when x is a mesh |

## Author(s)

Stefan Schlager

## References

Baerentzen, Jakob Andreas. & Aanaes, H., 2002. Generating Signed Distance Fields From Triangle Meshes. Informatics and Mathematical Modelling.

Moshfeghi M, Ranganath S, Nawyn K. 1994. Three-dimensional elastic matching of volumes IEEE Transactions on Image Processing: A Publication of the IEEE Signal Processing Society 3:128-138.

## See Also

`ply2mesh`

### Examples

```
data(nose)
out <- closemeshKD(longnose.lm,shortnose.mesh,sign=TRUE)
### show distances - they are very small because
###longnose.lm is scaled to unit centroid size.
hist(out$quality)
```

---

## `colors` *predefined colors for bone and skin*

---

### Description

predefined colors for bone and skin

### Details

available colors are:

bone1

bone2

bone3

skin1

skin2

skin3

skin4

---

## `computeArea` *Compute area enclosed within an irregular polygon*

---

### Description

Compute area enclosed within an irregular polygon - i.e. defined by curves

### Usage

```
computeArea(x)
```

### Arguments

| | |
|---|---|
| `x` | k x 2 or k x 3 matrix containing ordered coordinates forming the boundary of the area. For 3D-cases, the area should be closed to a 2D surface (see details below). |

## Details

For 3D coordinates, a PCA is computed and only the first two PCs are used to compute the area. This is a projection of the coordinates onto a 2D plane spanned by those PCs.

## Value

returns a list containing

| | |
|---|---|
| `area` | size of the enclosed area |
| `xpro2D` | projected coordinates of x in the 2D plane. |
| `poly` | object of class `sp` as defined by the `sp` package. |
| `xpro3D` | For 3D-cases, this contains the projected coordinates of x rotated back into the original coordinate system |

## Note

in case custom planes are preferred, the data can first be projected onto such a custom defined plane via `points2plane` first.

## Examples

```
require(shapes)
require(sf)
myarea <- computeArea(gorf.dat[c(1,6:8,2:5),,1])
myarea$area
plot(myarea$poly)


## 3D example
data(boneData)
myarea3D <- computeArea(boneLM[c(4,2,3,7,5,6,8),,1])
plot(myarea3D$poly)
cent <- colMeans(myarea3D$xpro2D)
text(cent[1],cent[2],labels=paste0("Area=",round(myarea3D$area,digits=2)))
```

---

## `computeTransform` *calculate an affine transformation matrix*

---

## Description

calculate an affine transformation matrix

## Usage

```
computeTransform(
  x,
  y,
  type = c("rigid", "similarity", "affine", "tps"),
  reflection = FALSE,
  lambda = 1e-08,
  weights = NULL,
  centerweight = FALSE,
  threads = 1
)
```

## Arguments

| | |
|---|---|
| `x` | fix landmarks. Can be a k x m matrix or mesh3d. |
| `y` | moving landmarks. Can be a k x m matrix or mesh3d. |
| `type` | set type of affine transformation: options are "rigid", "similarity" (rigid + scale) and "affine", |
| `reflection` | logical: if TRUE "rigid" and "similarity" allow reflections. |
| `lambda` | numeric: regularisation parameter of the TPS. |
| `weights` | vector of length k, containing weights for each landmark (only used in type="rigid" or "similarity"). |
| `centerweight` | logical or vector of weights: if weights are defined and centerweigths=TRUE, the matrix will be centered according to these weights instead of the barycenter. If centerweight is a vector of length `nrow(x)`, the barycenter will be weighted accordingly. |
| `threads` | number of threads to use in TPS interpolation. |

## Details

`x` and `y` can also be a pair of meshes with corresponding vertices.

## Value

returns a 4x4 (3x3 in 2D case) transformation matrix or an object of class "tpsCoeff" in case of type="tps".

## Note

all lines containing NA, or NaN are ignored in computing the transformation.

## See Also

`rotonto`, `link{rotmesh.onto}`, `tps3d`

## Examples

```
data(boneData)
trafo <- computeTransform(boneLM[,,1],boneLM[,,2])
transLM <- applyTransform(boneLM[,,2],trafo)
```

---

## covDist *calculates distances and PC-coordinates of covariance matrices*

---

### Description

calculates PC-coordinates of covariance matrices by using the Riemannian metric in their respective space.

### Usage

```
covDist(s1, s2)

covPCA(
  data,
  groups,
  rounds = 1000,
  bootrounds = 0,
  lower.bound = 0.05,
  upper.bound = 0.95
)
```

### Arguments

| | |
|---|---|
| `s1` | m x m covariance matrix |
| `s2` | m x m covariance matrix |
| `data` | matrix containing data with one row per observation |
| `groups` | factor: group assignment for each specimen |
| `rounds` | integer: rounds to run permutation of distances by randomly assigning group membership |
| `bootrounds` | integer: perform bootstrapping to generate confidence intervals (lower boundary, median and upper boundary) for PC-scores. |
| `lower.bound` | numeric: set probability (quantile) for lower boundary estimate from bootstrapping. |
| `upper.bound` | numeric: set probability (quantile) for upper boundary estimate from bootstrapping. |

## Details

`covDist` calculates the Distance between covariance matrices while `covPCA` uses a MDS (multidimensional scaling) approach to obtain PC-coordinates from a distance matrix derived from multiple groups. P-values for pairwise distances can be computed by permuting group membership and comparing actual distances to those obtained from random resampling. To calculate confidence intervals for PC-scores, within-group bootstrapping can be performed.

## Value

`covDist` returns the distance between s1 and s2

`covPCA` returns a list containing:

if `scores = TRUE`

| | |
|---|---|
| `PCscores` | PCscores |
| `eigen` | eigen decomposition of the centered inner product |

if `rounds > 0`

| | |
|---|---|
| `dist` | distance matrix |
| `p.matrix` | p-values for pairwise distances from permutation testing |

if `bootrounds > 0`

| | |
|---|---|
| `bootstrap` | list containing the lower and upper bound of the confidence intervals of PC-scores as well as the median of bootstrapped values. |
| `boot.data` | array containing all results generated from bootstrapping. |

## Author(s)

Stefan Schlager

## References

Mitteroecker P, Bookstein F. 2009. The ontogenetic trajectory of the phenotypic covariance matrix, with examples from craniofacial shape in rats and humans. Evolution 63:727-737.

Hastie T, Tibshirani R, Friedman JJH. 2013. The elements of statistical learning. Springer New York.

## See Also

`prcomp`

## Examples

```
cpca <- covPCA(iris[,1:4],iris[,5])
cpca$p.matrix #show pairwise p-values for equal covariance matrices
## Not run:
require(car)
```

```
sp(cpca$PCscores[,1],cpca$PCscores[,2],groups=levels(iris[,5]),
   smooth=FALSE,xlim=range(cpca$PCscores),ylim=range(cpca$PCscores))

data(boneData)
proc <- procSym(boneLM)
pop <- name2factor(boneLM, which=3)
## compare covariance matrices for PCscores of Procrustes fitted data
cpca1 <- covPCA(proc$PCscores, groups=pop, rounds = 1000)
## view p-values:
cpca1$p.matrix # differences between covariance matrices
# are significant
## visualize covariance ellipses of first 5 PCs of shape
spm(proc$PCscores[,1:5], groups=pop, smooth=FALSE,ellipse=TRUE, by.groups=TRUE)
## covariance seems to differ between 1st and 5th PC
## for demonstration purposes, try only first 4 PCs
cpca2 <- covPCA(proc$PCscores[,1:4], groups=pop, rounds = 1000)
## view p-values:
cpca2$p.matrix # significance is gone

## End(Not run)

#do some bootstrapping 1000 rounds
cpca <- covPCA(iris[,1:4],iris[,5],rounds=0, bootrounds=1000)
#plot bootstrapped data of PC1 and PC2 for first group
plot(t(cpca$boot.data[1,1:2,]),xlim=range(cpca$boot.data[,1,]),
                               ylim=range(cpca$boot.data[,2,]))
points(t(cpca$PCscores[1,]),col="white",pch=8,cex=1.5)##plot actual values

for (i in 2:3) {
  points(t(cpca$boot.data[i,1:2,]),col=i)##plot other groups
  points(t(cpca$PCscores[i,]),col=1,pch=8,cex=1.5)##plot actual values
}
```

## covW — *calculate the pooled within groups covariance matrix*

### Description

calculate the pooled within groups covariance matrix

### Usage

```
covW(data, groups, robust = c("classical", "mve", "mcd"), ...)
```

### Arguments

| | |
|---|---|
| data | a matrix containing data |
| groups | grouping variables |

| | |
|---|---|
| `robust` | character: determines covariance estimation methods in case `sep=TRUE`, when covariance matrices and group means can be estimated robustly using `MASS::cov.rob`. Default is the standard product-moment covariance matrix. |
| `...` | additional parameters passed to `MASS::cov.rob` for robust covariance and mean estimations. |

## Value

Returns the pooled within group covariance matrix. The attributes contain the entry means, containing the respective group means.

## Author(s)

Stefan Schlager

## See Also

`cov`, `typprobClass`

## Examples

```
data(iris)
poolCov <- covW(iris[,1:4],iris[,5])
```

---

`createAtlas` *Create an atlas needed in placePatch*

---

## Description

Create an atlas needed in placePatch

## Usage

```
createAtlas(
  mesh,
  landmarks,
  patch,
  corrCurves = NULL,
  patchCurves = NULL,
  keep.fix = NULL
)
```

## Arguments

| | |
|---|---|
| `mesh` | triangular mesh representing the atlas' surface |
| `landmarks` | matrix containing landmarks defined on the atlas, as well as on each specimen in the corresponding sample. |
| `patch` | matrix containing semi-landmarks to be projected onto each specimen in the corresponding sample. |
| `corrCurves` | a vector or a list containing vectors specifiyng the rowindices of `landmarks` to be curves that are defined on the atlas AND each specimen. e.g. if landmarks 2:4 and 5:10 are two distinct curves, one would specifiy `corrCurves = list(c(2:4), c(5:10))`. |
| `patchCurves` | a vector or a list containing vectors specifiyng the rowindices of `landmarks` to be curves that are defined ONLY on the atlas. E.g. if coordinates 5:10 and 20:40 on the `patch` are two distinct curves, one would specifiy `patchCurves = list(c(5:10),c(20:40))`. |
| `keep.fix` | in case corrCurves are set, specify explicitly which landmarks are not allowed to slide during projection (with `placePatch`) |

## Value

Returns a list of class "atlas". Its content is corresponding to argument names.

## Note

This is a helper function of `placePatch`.

## See Also

`placePatch`, `plotAtlas`

## Examples

```
data(nose)
atlas <- createAtlas(shortnose.mesh, landmarks =
           shortnose.lm[c(1:5,20:21),], patch=shortnose.lm[-c(1:5,20:21),])
```

---

`CreateL` *Create Matrices necessary for Thin-Plate Spline*

---

## Description

Create (Bending Engergy) Matrices necessary for Thin-Plate Spline, and sliding of Semilandmarks

## Usage

```
CreateL(
  matrix,
  lambda = 1e-08,
  output = c("K", "L", "Linv", "Lsubk", "Lsubk3"),
  threads = 1
)
```

## Arguments

| | |
|---|---|
| `matrix` | k x 3 or k x 2 matrix containing landmark coordinates. |
| `lambda` | numeric: regularization factor |
| `output` | character vector: select which matrices to create. Can be a vector containing any combination of the strings: `"K"`, `"L"`,`"Linv"`,`"Lsubk"`, `"Lsubk3"`. |
| `threads` | threads to be used for parallel execution calculating K. sliding of semilandmarks. |

## Value

depending on the choices in `output`:

| | |
|---|---|
| `L` | Matrix K as specified in Bookstein (1989) |
| `L` | Matrix L as specified in Bookstein (1989) |
| `Linv` | Inverse of matrix L as specified in Bookstein (1989) |
| `Lsubk` | uper left k x k submatrix of `Linv` |
| `Lsubk3` | Matrix used for sliding in `slider3d` and `relaxLM` |

.

## Note

This function is not intended to be called directly - except for playing around to grasp the mechansims of the Thin-Plate Spline.

## References

Gunz, P., P. Mitteroecker, and F. L. Bookstein. 2005. Semilandmarks in Three Dimensions, in Modern Morphometrics in Physical Anthropology. Edited by D. E. Slice, pp. 73-98. New York: Kluwer Academic/Plenum Publishers.

Bookstein FL. 1989. Principal Warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on pattern analysis and machine intelligence 11(6).

## See Also

`tps3d`

### Examples

```
data(boneData)
L <- CreateL(boneLM[,,1])
## calculate Bending energy between first and second specimen:
be <- t(boneLM[,,2])%*%L$Lsubk%*%boneLM[,,2]
## calculate Frobenius norm
sqrt(sum(be^2))
## the amount is dependant on on the squared scaling factor
# scale landmarks by factor 5 and compute bending energy matrix
be2 <- t(boneLM[,,2]*5)%*%L$Lsubk%*%(boneLM[,,2]*5)
sqrt(sum(be2^2)) # exactly 25 times the result from above
## also this value is not symmetric:
L2 <- CreateL(boneLM[,,2])
be3 <- t(boneLM[,,1])%*%L2$Lsubk%*%boneLM[,,1]
sqrt(sum(be3^2))
```

---

## createMissingList *create a list with empty entries to be used as missingList in slider3d*

---

### Description

create a list with empty entries to be used as missingList in slider3d

### Usage

```
createMissingList(x)
```

### Arguments

| | |
|---|---|
| x | length of the list to be created |

### Value

returns a list of length x filled with numerics of length zero.

### See Also

fixLMtps,fixLMmirror, slider3d

### Examples

```
## Assume in a sample of 10, the 9th individual has (semi-)landmarks 10:50
#   hanging in thin air (e.g. estimated using fixLMtps)
#   while the others are complete.
## create empty list
missingList <- createMissingList(10)
missingList[[9]] <- 10:50
```

## crossProduct *calculate the orthogonal complement of a 3D-vector*

### Description

calculate the orthogonal complement of a 3D-vector

### Usage

```
crossProduct(x, y, normalize = TRUE)

tangentPlane(x)
```

### Arguments

| | |
|---|---|
| x | vector of length 3. |
| y | vector of length 3. |
| normalize | logical: if TRUE, the resulting vector is normalized |

### Details

calculate the orthogonal complement of a 3D-vector or the 3D-crossproduct, finding an orthogonal vector to a plane in 3D.

### Value

tangentPlane:

crossProduct: returns a vector of length 3.

| | |
|---|---|
| y | vector orthogonal to x |
| z | vector orthogonal to x and y |

### Author(s)

Stefan Schlager

### Examples

```
require(rgl)

x <- c(1,0,0)
y <- c(0,1,0)

#example tangentPlane
z <- tangentPlane(x)
#visualize result
## Not run:
lines3d(rbind(0, x), col=2, lwd=2)
```

```
## show complement
lines3d(rbind(z$y, 0, z$z), col=3, lwd=2)

## End(Not run)
# example crossProduct
z <- crossProduct(x, y)
# show x and y
## Not run:
lines3d(rbind(x, 0, y), col=2, lwd=2)
# show z
lines3d(rbind(0, z), col=3, lwd=2)

## End(Not run)
```

---

`cSize` *calculate Centroid Size for a landmark configuration*

---

## Description

calculate Centroid Size for a landmark configuration

## Usage

```
cSize(x)
```

## Arguments

| | |
|---|---|
| `x` | k x 3 matrix containing landmark coordinates or mesh of class "mesh3d" |

## Value

returns Centroid size

## Examples

```
data(boneData)
cSize(boneLM[,,1])
```

## `cutMeshPlane` *cut a mesh by a hyperplane and remove parts above/below that plane*

### Description

cut a mesh by a hyperplane and remove parts above/below that plane

### Usage

```
cutMeshPlane(mesh, v1, v2 = NULL, v3 = NULL, normal = NULL, keep.upper = TRUE)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class "mesh3d" |
| `v1` | numeric vector of length=3 specifying a point on the separating plane |
| `v2` | numeric vector of length=3 specifying a point on the separating plane |
| `v3` | numeric vector of length=3 specifying a point on the separating plane |
| `normal` | plane normal (overrides specification by v2 and v3) |
| `keep.upper` | logical specify whether the points above or below the plane are should be kept |

### Details

see `cutSpace` for more details.

### Value

mesh with part above/below hyperplane removed

## `cutSpace` *separate a 3D-pointcloud by a hyperplane*

### Description

separate a 3D-pointcloud by a hyperplane

### Usage

```
cutSpace(pointcloud, v1, v2 = NULL, v3 = NULL, normal = NULL, upper = TRUE)
```

### Arguments

| | |
|---|---|
| `pointcloud` | numeric n x 3 matrix |
| `v1` | numeric vector of length=3 specifying a point on the separating plane |
| `v2` | numeric vector of length=3 specifying a point on the separating plane |
| `v3` | numeric vector of length=3 specifying a point on the separating plane |
| `normal` | plane normal (overrides specification by v2 and v3) |
| `upper` | logical specify whether the points above or below the plane are to be reported as TRUE. |

### Details

As above and below are specified by the normal calculated from $(v2 - v1) \times (v3 - v1)$, where $\times$ denotes the vector crossproduct. This means the normal points "upward" when viewed from the positon where v1, v2 and v3 are arranged counter-clockwise. Thus, which side is "up" depends on the ordering of v1, v2 and v3.

### Value

logical vector of length n. Reporting for each point if it is above or below the hyperplane

### Examples

```
data(nose)
v1 <- shortnose.lm[1,]
v2 <- shortnose.lm[2,]
v3 <- shortnose.lm[3,]
pointcloud <- vert2points(shortnose.mesh)
upper <- cutSpace(pointcloud, v1, v2, v3)
## Not run:
require(rgl)
normal <- crossProduct(v2-v1,v3-v1)
zeroPro <- points2plane(rep(0,3),v1,normal)
## get sign of normal displacement from zero
sig <- sign(crossprod(-zeroPro,normal))
d <- sig*norm(zeroPro,"2")
planes3d(normal[1],normal[2],normal[3],d=d)
points3d(pointcloud[upper,])

## End(Not run)
```

## CVA *Canonical Variate Analysis*

### Description

performs a Canonical Variate Analysis.

## Usage

```
CVA(
  dataarray,
  groups,
  weighting = TRUE,
  tolinv = 1e-10,
  plot = TRUE,
  rounds = 0,
  cv = FALSE,
  p.adjust.method = "none",
  robust = c("classical", "mve", "mcd"),
  prior = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `dataarray` | Either a k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. Or alternatively a n x m Matrix where n is the numeber of observations and m the number of variables (this can be PC scores for example) |
| `groups` | a character/factor vector containgin grouping variable. |
| `weighting` | Logical: Determines whether the between group covariance matrix and Grand-mean is to be weighted according to group size. |
| `tolinv` | Threshold for the eigenvalues of the pooled within-group-covariance matrix to be taken as zero - for calculating the general inverse of the pooled withing groups covariance matrix. |
| `plot` | Logical: determins whether in the two-sample case a histogramm ist to be plotted. |
| `rounds` | integer: number of permutations if a permutation test of the Mahalanobis distances (from the pooled within-group covariance matrix) and Euclidean distance between group means is requested If rounds = 0, no test is performed. |
| `cv` | logical: requests a Jackknife Crossvalidation. |
| `p.adjust.method` | method to adjust p-values for multiple comparisons see `p.adjust.methods` for options. |
| `robust` | character: determines covariance estimation methods, allowing for robust estimations using `MASS::cov.rob` |
| `prior` | vector assigning each group a prior probability. |
| `...` | additional parameters passed to `MASS::cov.rob` for robust covariance and mean estimations |

## Value

| | |
|---|---|
| `CV` | A matrix containing the Canonical Variates |

| | |
|---|---|
| CVscores | A matrix containing the individual Canonical Variate scores |
| Grandm | a vector or a matrix containing the Grand Mean (depending if the input is an array or a matrix) |
| groupmeans | a matrix or an array containing the group means (depending if the input is an array or a matrix) |
| Var | Variance explained by the Canonical Variates |
| CVvis | Canonical Variates projected back into the original space - to be used for visualization purposes, for details see example below |
| Dist | Mahalanobis Distances between group means - if requested tested by permutation test if the input is an array it is assumed to be superimposed Landmark Data and Procrustes Distance will be calculated |
| CVcv | A matrix containing crossvalidated CV scores |
| groups | factor containing the grouping variable |
| class | classification results based on posteriror probabilities. If cv=TRUE, this will be done by a leaving-one-out procedure |
| posterior | posterior probabilities |
| prior | prior probabilities |

## Author(s)

Stefan Schlager

## References

Cambell, N. A. & Atchley, W. R.. 1981 The Geometry of Canonical Variate Analysis: Syst. Zool., 30(3), 268-280.

Klingenberg, C. P. & Monteiro, L. R. 2005 Distances and directions in multidimensional shape spaces: implications for morphometric applications. Systematic Biology 54, 678-688.

## See Also

groupPCA

## Examples

```
## all examples are kindly provided by Marta Rufino

if (require(shapes)) {
# perform procrustes fit on raw data
alldat<-procSym(abind(gorf.dat,gorm.dat))
# create factors
groups<-as.factor(c(rep("female",30),rep("male",29)))
# perform CVA and test Mahalanobis distance
# between groups with permutation test by 100 rounds)
cvall<-CVA(alldat$orpdata,groups,rounds=10000)
## visualize a shape change from score -5 to 5:
cvvis5 <- 5*matrix(cvall$CVvis[,1],nrow(cvall$Grandm),ncol(cvall$Grandm))+cvall$Grandm
```

```
cvvisNeg5 <- -5*matrix(cvall$CVvis[,1],nrow(cvall$Grandm),ncol(cvall$Grandm))+cvall$Grandm
plot(cvvis5,asp=1)
points(cvvisNeg5,col=2)
for (i in 1:nrow(cvvisNeg5))
  lines(rbind(cvvis5[i,],cvvisNeg5[i,]))
}
### Morpho CVA
data(iris)
vari <- iris[,1:4]
facto <- iris[,5]

cva.1=CVA(vari, groups=facto)
## get the typicality probabilities and resulting classifications - tagging
## all specimens with a probability of < 0.01 as outliers (assigned to no class)
typprobs <- typprobClass(cva.1$CVscores,groups=facto)
print(typprobs)
## visualize the CV scores by their groups estimated from (cross-validated)
## typicality probabilities:
if (require(car)) {
scatterplot(cva.1$CVscores[,1],cva.1$CVscores[,2],groups=typprobs$groupaffinCV,
                  smooth=FALSE,reg.line=FALSE)
}
# plot the CVA
plot(cva.1$CVscores, col=facto, pch=as.numeric(facto), typ="n",asp=1,
   xlab=paste("1st canonical axis", paste(round(cva.1$Var[1,2],1),"%")),
   ylab=paste("2nd canonical axis", paste(round(cva.1$Var[2,2],1),"%")))

  text(cva.1$CVscores, as.character(facto), col=as.numeric(facto), cex=.7)

  # add chull (merge groups)
  for(jj in 1:length(levels(facto))){
        ii=levels(facto)[jj]
    kk=chull(cva.1$CVscores[facto==ii,1:2])
    lines(cva.1$CVscores[facto==ii,1][c(kk, kk[1])],
    cva.1$CVscores[facto==ii,2][c(kk, kk[1])], col=jj)
    }

  # add 80% ellipses
  if (require(car)) {
  for(ii in 1:length(levels(facto))){
    dataEllipse(cva.1$CVscores[facto==levels(facto)[ii],1],
    cva.1$CVscores[facto==levels(facto)[ii],2],
                    add=TRUE,levels=.80, col=c(1:7)[ii])}
  }
  # histogram per group
  if (require(lattice)) {
  histogram(~cva.1$CVscores[,1]|facto,
  layout=c(1,length(levels(facto))),
          xlab=paste("1st canonical axis", paste(round(cva.1$Var[1,2],1),"%")))
  histogram(~cva.1$CVscores[,2]|facto, layout=c(1,length(levels(facto))),
          xlab=paste("2nd canonical axis", paste(round(cva.1$Var[2,2],1),"%")))
  }
  # plot Mahalahobis
```

```
    dendroS=hclust(cva.1$Dist$GroupdistMaha)
    dendroS$labels=levels(facto)
    par(mar=c(4,4.5,1,1))
    dendroS=as.dendrogram(dendroS)
    plot(dendroS, main='',sub='', xlab="Geographic areas",
            ylab='Mahalahobis distance')


     # Variance explained by the canonical roots:
     cva.1$Var
     # or plot it:
     barplot(cva.1$Var[,2])

# another landmark based example in 3D:
data(boneData)
groups <- name2factor(boneLM,which=3:4)
proc <- procSym(boneLM)
cvall<-CVA(proc$orpdata,groups)
#' ## visualize a shape change from score -5 to 5:
cvvis5 <- 5*matrix(cvall$CVvis[,1],nrow(cvall$Grandm),ncol(cvall$Grandm))+cvall$Grandm
cvvisNeg5 <- -5*matrix(cvall$CVvis[,1],nrow(cvall$Grandm),ncol(cvall$Grandm))+cvall$Grandm
## Not run:
#visualize it
deformGrid3d(cvvis5,cvvisNeg5,ngrid = 0)

## End(Not run)

#for using (e.g. the first 5) PCscores, one will do:
cvall <- CVA(proc$PCscores[,1:5],groups)
#' ## visualize a shape change from score -5 to 5:
cvvis5 <- 5*cvall$CVvis[,1]+cvall$Grandm
cvvisNeg5 <- -5*cvall$CVvis[,1]+cvall$Grandm
cvvis5 <- restoreShapes(cvvis5,proc$PCs[,1:5],proc$mshape)
cvvisNeg5 <- restoreShapes(cvvisNeg5,proc$PCs[,1:5],proc$mshape)
## Not run:
#visualize it
deformGrid3d(cvvis5,cvvisNeg5,ngrid = 0)

## End(Not run)
```

## `data2platonic` *creates 3D shapes from data to be saved as triangular meshes*

### Description

creates 3D shapes from 3-dimensional data that can be saved as triangular meshes

### Usage

```
data2platonic(
  datamatrix,
  shape = Rvcg::vcgSphere(),
  col = "red",
  scale = FALSE,
  scalefactor = 1
)
```

### Arguments

| | |
|---|---|
| `datamatrix` | k x 3 data matrix |
| `shape` | a 3D shape |
| `col` | color value |
| `scale` | logical: whether to scale the data to unit sd. |
| `scalefactor` | scale the resulting shapes. |

### Value

returns all shapes merged into a single mesh

### Examples

```
mymesh <- data2platonic(iris[iris$Species=="setosa",1:3],scalefactor=0.1)
mymesh <- mergeMeshes(mymesh,data2platonic(iris[iris$Species=="versicolor",1:3],
                      shape=Rvcg::vcgIcosahedron(),scalefactor=0.1,col="green"))
mymesh <- mergeMeshes(mymesh,data2platonic(iris[iris$Species=="virginica",1:3],
                      shape=Rvcg::vcgTetrahedron(),scalefactor=0.1,col="blue"))
## Not run:
rgl::shade3d(mymesh)
## save to disk
Rvcg::vcgPlyWrite(mymesh,filename="3D_Data.ply")

## End(Not run)
```

---

## `deformGrid2d` *visualise differences between two superimposed sets of 2D landmarks*

---

### Description

visualise differences between two superimposed sets of 2D landmarks by deforming a square grid based on a thin-plate spline interpolation

### Usage

```
deformGrid2d(
  matrix,
  tarmatrix,
  ngrid = 0,
  lwd = 1,
  show = c(1:2),
  lines = TRUE,
  lcol = 1,
  lty = 2,
  col1 = 2,
  col2 = 3,
  pcaxis = FALSE,
  add = FALSE,
  wireframe = NULL,
  margin = 0.2,
  gridcol = "grey",
  gridlty = 1,
  cex1 = 1,
  cex2 = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `matrix` | reference matrix containing 2D landmark coordinates or mesh of class "mesh3d" |
| `tarmatrix` | target matrix containing 2D landmark coordinates or mesh of class "mesh3d" |
| `ngrid` | number of grid lines to be plotted; ngrid=0 suppresses grid creation. |
| `lwd` | width of lines connecting landmarks. |
| `show` | integer (vector): if c(1:2) both configs will be plotted, show = 1 only plots the reference and show = 2 the target. plotted. Options are combinations of 1,2 and 3. |
| `lines` | logical: if TRUE, lines between landmarks will be plotted. |
| `lcol` | color of lines |
| `lty` | line type |
| `col1` | color of "matrix" |
| `col2` | color of "tarmat" |
| `pcaxis` | logical: align grid by shape's principal axes. |
| `add` | logical: if TRUE, output will be drawn on existing plot. |
| `wireframe` | list/vector containing row indices to be plotted as wireframe (see `lineplot`.) |
| `margin` | margin around the bounding box to draw the grid |
| `gridcol` | color of the grid |
| `gridlty` | linetype for grid |
| `cex1` | control size of points belonging to `matrix` |
| `cex2` | control size of points belonging to `tarmatrix` |
| `...` | additional parameters passed to plot |

## Value

if `ngrid` > `1` the coordinates of the displaced grid knots are returned.

### Author(s)

Stefan Schlager

### See Also

tps3d

### Examples

```
if (require(shapes)) {
proc <- procSym(gorf.dat)
deformGrid2d(proc$mshape,proc$rotated[,,1],ngrid=5,pch=19)
}
```

---

## deformGrid3d *visualise differences between two superimposed sets of 3D landmarks*

---

### Description

visualise differences between two superimposed sets of 3D landmarks by deforming a cubic grid based on a thin-plate spline interpolation

### Usage

```
deformGrid3d(
  matrix,
  tarmatrix,
  ngrid = 0,
  align = FALSE,
  lwd = 1,
  showaxis = c(1, 2),
  show = c(1, 2),
  lines = TRUE,
  lcol = 1,
  add = FALSE,
  col1 = 2,
  col2 = 3,
  type = c("s", "p"),
  size = NULL,
  pcaxis = FALSE,
  ask = TRUE,
  margin = 0.2,
  createMesh = FALSE,
  slice1 = NULL,
  slice2 = NULL,
  slice3 = NULL,
  gridcol = 1,
  gridwidth = 1,
  ...
)
```

## Arguments

| | |
|---|---|
| `matrix` | reference matrix containing 3D landmark coordinates or mesh of class "mesh3d" |
| `tarmatrix` | target matrix containing 3D landmark coordinates or mesh of class "mesh3d" |
| `ngrid` | number of grid lines to be plotted; ngrid=0 suppresses grid creation. |
| `align` | logical: if TRUE, `tarmatrix` will be aligned rigidly to `matrix` |
| `lwd` | width of lines connecting landmarks. |
| `showaxis` | integer (vector): which dimensions of the grid to be plotted. Options are combinations of 1,2 and 3. |
| `show` | integer (vector): if c(1:2) both configs will be plotted, show = 1 only plots the reference and show = 2 the target |
| `lines` | logical: if TRUE, lines between landmarks will be plotted. |
| `lcol` | color of lines |
| `add` | logical: add to existing rgl window. |
| `col1` | color of "matrix" |
| `col2` | color of "tarmat" |
| `type` | "s" renders landmarks as spheres; "p" as points - much faster for very large pointclouds. |
| `size` | control size/radius of points/spheres |
| `pcaxis` | logical: align grid by shape's principal axes. |
| `ask` | logical: if TRUE for > 1000 coordinates the user will be asked to prefer points over spheres. |
| `margin` | margin around the bounding box to draw the grid |
| `createMesh` | logical: if TRUE, a triangular mesh of spheres and displacement vectors (can take some time depending on number of reference points and grid density). |
| `slice1` | integer or vector of integers: select slice(s) for the dimensions |
| `slice2` | integer or vector of integers: select slice(s) for the dimensions |
| `slice3` | integer or vector of integers: select slice(s) for the dimensions |
| `gridcol` | define color of grid |
| `gridwidth` | integer: define linewidth of grid |
| `...` | additional parameters passed to `rotonto` in case `align=TRUE` |

## Value

if `createMesh=TRUE`, a mesh containing spheres of reference and target as well as the displacement vectors is returned. Otherwise the knots of the displaced grid is returned.

### Author(s)

Stefan Schlager

### See Also

tps3d

### Examples

```
if (interactive()){
data(nose)
deformGrid3d(shortnose.lm,longnose.lm,ngrid=10)

## select some slices
deformGrid3d(shortnose.lm,longnose.lm,showaxis=1:3,ngrid=10,slice1=2,slice2=5,slice3=7)
}
```

---

## equidistantCurve *make a curve equidistant (optionally up/downsampling)*

---

### Description

make a curve equidistant (optionally up/downsampling)

### Usage

```
equidistantCurve(
  x,
  n = NULL,
  open = TRUE,
  subsample = 0,
  increment = 2,
  smoothit = 0,
  mesh = NULL,
  iterations = 1
)
```

### Arguments

| | |
|---|---|
| x | k x m matrix containing the 2D or 3D coordinates |
| n | integer: number of coordinates to sample. If NULL, the existing curve will be made equidistant. |
| open | logical: specifies whether the curve is open or closed. |
| subsample | integer: number of subsamples to draw from curve for interpolation. For curves with < 1000 points, no subsampling is required. |

| | |
|---|---|
| `increment` | integer: if > 1, the curve is estimated iteratively by incrementing the original points by this factor. The closer this value to 1, the smoother the line but possibly farther away from the control points. |
| `smoothit` | integer: smoothing iterations after each step |
| `mesh` | specify mesh to project point to |
| `iterations` | integer: how many iterations to run equidistancing. |

### Details

Equidistancy is reached by iteratively deforming (using TPS) a straight line with equidistantly placed points to the target using control points with the same spacing as the actual curve. To avoid singularity, the straight line containes a small amount of noise, which can (optionally) be accounted for by smoothing the line by its neighbours.

### Value

matrix containing equidistantly placed points

### Note

if n » number of original points, the resulting curves can show unwanted distortions.

### Examples

```
## Not run:
data(nose)
x <- shortnose.lm[c(304:323),]
xsample <- equidistantCurve(x,n=50,iterations=10,increment=2)

require(rgl)
points3d(xsample,size=5)
spheres3d(x,col=2,radius=0.3,alpha=0.5)

## End(Not run)
```

## exVar *calculate variance of a distribution stemming from prediction models*

### Description

calculates a quotient of the overall varriance within a predicted distribution to that from the original one. This function calculates a naive extension of the univariate R^2-value by dividing the variance in the predicted dat by the variance of the original data. No additional adjustments are made!!

## Usage

```
exVar(model, ...)

## S3 method for class 'lm'
exVar(model, ...)

## S3 method for class 'mvr'
exVar(model, ncomp, val = FALSE, ...)
```

## Arguments

| | |
|---|---|
| `model` | a model of classes "lm" or "mvr" (from the package "pls") |
| `...` | currently unused additional arguments. |
| `ncomp` | How many latent variables to use (only for mvr models) |
| `val` | use cross-vaildated predictions (only for mvr models) |

## Value

returns the quotient.

## Note

The result is only!! a rough estimate of the variance explained by a multivariate model. And the result can be misleading - especially when there are many predictor variables involved. If one is interested in the value each factor/covariate explains, we recommend a 50-50 MANOVA perfomed by the R-package "ffmanova", which reports this value factor-wise.

## Author(s)

Stefan Schlager

## References

Langsrud O, Juergensen K, Ofstad R, Naes T. 2007. Analyzing Designed Experiments with Multiple Responses Journal of Applied Statistics 34:1275-1296.

## Examples

```
lm1 <- lm(as.matrix(iris[,1:4]) ~ iris[,5])
exVar(lm1)
```

## fastKmeans *fast kmeans clustering for 2D or 3D point clouds*

### Description

fast kmeans clustering for 2D or 3D point clouds - with the primary purpose to get a spatially equally distributed samples

### Usage

```
fastKmeans(x, k, iter.max = 10, project = TRUE, threads = 0)
```

### Arguments

| | |
|---|---|
| `x` | matrix containing coordinates or mesh3d |
| `k` | number of clusters |
| `iter.max` | maximum number of iterations |
| `project` | logical: if x is a triangular mesh, the centers will be projected onto the surface. |
| `threads` | integer number of threads to use |

### Value

returns a list containing

| | |
|---|---|
| `selected` | coordinates closest to the final centers |
| `centers` | cluster center |
| `class` | vector with cluster association for each coordinate |

### Examples

```
require(Rvcg)
data(humface)
set.seed(42)
clust <- fastKmeans(humface,k=1000,threads=1)
## Not run:
require(rgl)

## plot the cluster centers
spheres3d(clust$centers)

## now look at the vertices closest to the centers
wire3d(humface)
spheres3d(vert2points(humface)[clust$selected,],col=2)

## End(Not run)
```

`file2mesh` *Import 3D surface mesh files*

## Description

Import 3D surface mesh files

## Usage

```
file2mesh(filename, clean = TRUE, readcol = FALSE)

obj2mesh(filename, adnormals = TRUE)

ply2mesh(
  filename,
  adnormals = TRUE,
  readnormals = FALSE,
  readcol = FALSE,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `filename` | character: path to file |
| `clean` | Logical: Delete dumpfiles. |
| `readcol` | Logical: Import vertex colors (if available). |
| `adnormals` | Logical: If the file does not contain normal information, they will be calculated in R: Can take some time. |
| `readnormals` | Logical: Import vertex normals (if available), although no face information is present. |
| `silent` | logical: suppress messages. |

## Details

imports 3D mesh files and store them as an R .object of class mesh3d

## Value

| | |
|---|---|
| `mesh` | list of class mesh3d - see rgl manual for further details, or a matrix containing vertex information or a list containing vertex and normal information |

## Examples

```
data(nose)
mesh2ply(shortnose.mesh)
mesh <- ply2mesh("shortnose.mesh.ply")

mesh2obj(shortnose.mesh)
mesh2 <- obj2mesh("shortnose.mesh.obj")
## cleanup
unlink(c("shortnose.mesh.obj","shortnose.mesh.ply"))
```

---

`find.outliers` *Graphical interface to find outliers and/or to switch mislabeld landmarks*

---

## Description

Graphical interface to find outliers and/or to switch mislabeld landmarks

## Usage

```
find.outliers(
  A,
  color = 4,
  lwd = 1,
  lcol = 2,
  mahalanobis = FALSE,
  PCuse = NULL,
  text = TRUE,
  reflection = FALSE
)
```

## Arguments

| | |
|---|---|
| `A` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. |
| `color` | color of Landmarks points to be plotted |
| `lwd` | linewidth visualizing distances of the individual landmarks from mean. |
| `lcol` | color of lines visualizing distances of the individual landmarks from mean. |
| `mahalanobis` | logical: use mahalanobis distance to find outliers. |
| `PCuse` | integer: Restrict mahalanobis distance to the first n Principal components. |
| `text` | logical: if `TRUE`, landmark labels (rownumbers) are displayed |
| `reflection` | logical: specify whether reflections are allowed for superimpositioning. |

## Details

This function performs a procrustes fit and sorts all specimen according to their distances (either Procrustes or Mahalanobis-distance) to the sample's consensus. It provides visual help for rearranging landmarks and/or excluding outliers.

## Value

| | |
|---|---|
| `data.cleaned` | array (in original coordinate system) containing the changes applied and outliers eliminated |
| `outlier` | vector with integers indicating the positions in the original array that have been marked as outliers |
| `dist.sort` | table showing the distance to mean for each observation - decreasing by distance |
| `type` | what kind of distance was used |

## Author(s)

Stefan Schlager

## See Also

`typprob`,`typprobClass`

## Examples

```
data(boneData)
## look for outliers using the mahalanobis distance based on the first
# 10 PCscores
# to perform the example below, you need,of course, uncomment the answers
if (interactive()){
outliers <- find.outliers(boneLM, mahalanobis= TRUE, PCuse=10)
# n # everything is fine
# n # proceed to next
# s # let's switch some landmarks (3 and 4)
# 3
# 4
# n # we are done
# y # yes, because now it is an outlier
# s #enough for now
}
```

---

`fixLMmirror` *estimate missing landmarks from their bilateral counterparts*

---

## Description

estimate missing landmarks from their bilateral counterparts

## Usage

```
fixLMmirror(x, pairedLM, ...)

## S3 method for class 'array'
fixLMmirror(x, pairedLM, ...)

## S3 method for class 'matrix'
fixLMmirror(x, pairedLM, ...)
```

## Arguments

| | |
|---|---|
| `x` | a matrix or an array containing landmarks (3D or 2D) |
| `pairedLM` | a k x 2 matrix containing the indices (rownumbers) of the paired LM. E.g. the left column contains the lefthand landmarks, while the right side contains the corresponding right hand landmarks. |
| `...` | additional arguments |

## Details

the configurations are mirrored and the relabled version is matched onto the original using a thin-plate spline deformation. The missing landmark is now estimated using its bilateral counterpart. If one side is completely missing, the landmarks will be mirrored and aligned by the unilateral landmarks.

## Value

a matrix or array with fixed missing bilateral landmarks.

## Note

in case both landmarks of a bilateral pair are missing a message will be issued. As well if there are missing landmarks on the midsaggital plane are detected.

## Examples

```
data(boneData)
left <- c(4,6,8)
## determine corresponding Landmarks on the right side:
# important: keep same order
right <- c(3,5,7)
pairedLM <- cbind(left, right)
exampmat <- boneLM[,,1]
exampmat[4,] <- NA #set 4th landmark to be NA
fixed <- fixLMmirror(exampmat, pairedLM=pairedLM)
## Not run:
deformGrid3d(fixed, boneLM[,,1],ngrid=0)
## result is a bit off due to actual asymmetry

## End(Not run)
## example with one side completely missing
```

```
oneside <- boneLM[,,1]
oneside[pairedLM[,1],] <- NA
onesidefixed <- fixLMmirror(oneside,pairedLM)
## Not run:
deformGrid3d(onesidefixed, boneLM[,,1],ngrid=0)
## result is a bit off due to actual asymmetry

## End(Not run)
```

---

## fixLMtps *estimate missing landmarks*

---

### Description

Missing landmarks are estimated by deforming a sample average or a weighted estimate of the configurations most similar onto the deficient configuration. The deformation is performed by a Thin-plate-spline interpolation calculated by the available landmarks.

### Usage

```
fixLMtps(data, comp = 3, weight = TRUE, weightfun = NULL)
```

### Arguments

| | |
|---|---|
| `data` | array containing landmark data |
| `comp` | integer: select how many of the closest observations are to be taken to calculate an initial estimate. |
| `weight` | logical: requests the calculation of an estimate based on the procrustes distance. Otherwise the sample's consensus is used as reference. |
| `weightfun` | custom function that operates on a vector of distances (see examples) and generates weights accordingly. |

### Details

This function tries to estimate missing landmark data by mapping weighted averages from complete datasets onto the missing specimen. The weights are the inverted Procrustes (see `proc.weight`) distances between the 'comp' closest specimen (using the available landmark configuration).

### Value

| | |
|---|---|
| `out` | array containing all data, including fixed configurations - same order as input |
| `mshape` | meanshape - calculated from complete datasets |
| `checklist` | list containing information about missing landmarks |
| `check` | vector containing position of observations in data where at least one missing coordinate was found |

## Note

Be aware that these estimates might be grossly wrong when the missing landmark is quite far off the rest of the landmarks (due to the radial basis function used in the Thin-plate spline interpolation.

## Author(s)

Stefan Schlager

## References

Bookstein FL. 1989. Principal Warps: Thin-plate splines and the decomposition of deformations IEEE Transactions on pattern analysis and machine intelligence 11.

## See Also

proc.weight, tps3d

## Examples

```
if (require(shapes)) {
data <- gorf.dat
### set first landmark of first specimen to NA
data[1,,1] <- NA
repair <- fixLMtps(data,comp=5)
### view difference between estimated and actual landmark
plot(repair$out[,,1],asp=1,pch=21,cex=0.7,col=2)#estimated landmark
points(gorf.dat[,,1],col=3,pch=20)#actual landmark
}
## 3D-example:
data(boneData)
data <- boneLM
### set first and 5th landmark of first specimen to NA
data[c(1,5),,1] <- NA
repair <- fixLMtps(data,comp=10)
##  view difference between estimated and actual landmark
## Not run:
deformGrid3d(repair$out[,,1], boneLM[,,1],ngrid=0)

## End(Not run)

## Now use a gaussian kernel to compute the weights and use all other configs
gaussWeight <- function(r,sigma=0.05) {
   sigma <- 2*sigma^2
   return(exp(-r^2/ sigma))
}
repair <- fixLMtps(data,comp=79,weightfun=gaussWeight)
```

## `getFaces` *find indices of faces that contain specified vertices*

### Description

find indices of faces that contain specified vertices

### Usage

```
getFaces(mesh, index)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class "mesh3d" |
| `index` | vector containing indices of vertices |

### Value

vector of face indices

## `getMeaningfulPCs` *get number of meaningful Principal components*

### Description

get number of meaningful Principal components

### Usage

```
getMeaningfulPCs(values, n, expect = 2, sdev = FALSE)
```

### Arguments

| | |
|---|---|
| `values` | eigenvalues from a PCA |
| `n` | sample size |
| `expect` | expectation value for chi-square distribution of df=2 |
| `sdev` | logical: if TRUE, it is assumed that the values are square roots of eigenvalues. |

### Details

This implements the method suggested by Bookstein (2014, pp. 324), to determine whether a PC is entitled to interpretation. I.e. a PC is regarded meaningful (its direction) if the ratio of this PC and its successor is above a threshold based on a log-likelihood ratio (and dependend on sample size).

## Value

| | |
|---|---|
| `tol` | threshold of ratio specific for `n` |
| `good` | integer vector specifying the meaningful Principal Components |

## References

Bookstein, F. L. Measuring and reasoning: numerical inference in the sciences. Cambridge University Press, 2014

## See Also

getPCtol

## Examples

```
data(boneData)
proc <- procSym(boneLM)
getMeaningfulPCs(proc$eigenvalues,n=nrow(proc$PCscores))
## the first 3 PCs are reported as meaningful
## show barplot that seem to fit the bill
barplot(proc$eigenvalues)
```

---

## getOuterViewpoints *Get viewpoints on a sphere around a 3D mesh*

---

## Description

Get viewpoints on a sphere around a 3D mesh to be used with virtualMeshScan

## Usage

```
getOuterViewpoints(
  x,
  n,
  inflate = 1.5,
  radius = NULL,
  subdivision = 3,
  PCA = FALSE
)
```

## Arguments

| | |
|---|---|
| `x` | triangular mesh of class 'mesh3d' |
| `n` | number of viewpoint to generate |
| `inflate` | factor for the size of the sphere: `inflate=1` means that the sphere around the object just touches the point farthest away from the mesh's centroid. |

| | |
|---|---|
| `radius` | defines a fix radius for the sphere (overrides arg `inflate`). |
| `subdivision` | parameter passed to `vcgSphere` |
| `PCA` | logical: if TRUE, the sphere will be deformed to match the principle axes of the mesh. NOTE: this may result in the sphere not necessarily completely enclosing the mesh. |

## Value

a list containing

| | |
|---|---|
| `viewpoints` | n x 3 matrix containing viewpoints. |
| `sphere` | sphere from which the points are sampled |
| `radius` | radius of the sphere |

## Examples

```
## Not run:
data(boneData)
vp <- getOuterViewpoints(skull_0144_ch_fe.mesh,n=100)

require(rgl)
shade3d(skull_0144_ch_fe.mesh,col="white")
spheres3d(vp$viewpoints)
wire3d(vp$sphere)

### Fit to principal axes
vppca <- getOuterViewpoints(skull_0144_ch_fe.mesh,n=100,PCA=TRUE,inflate=1.5)

require(rgl)
shade3d(skull_0144_ch_fe.mesh,col="white")
spheres3d(vppca$viewpoints)
wire3d(vppca$sphere)

## End(Not run)
```

---

`getPCscores` *Obtain PC-scores for new landmark data*

---

## Description

Obtain PC-scores for new landmark data

## Usage

```
getPCscores(x, PC, mshape)
```

### Arguments

| | |
|---|---|
| `x` | landmarks aligned (e.g. using `align2procSym` to the meanshape of data the PCs are derived from. |
| `PC` | Principal components (eigenvectors of the covariance matrix) |
| `mshape` | matrix containing the meanshape's landmarks (used to center the data) |

### Value

returns a matrix containing the PC scores

### See Also

`restoreShapes`

### Examples

```
## Not run:
data(boneData)
proc <- procSym(boneLM[,,-c(1:2)])
newdata <- boneLM[,,c(1:2)]
newdataAlign <- align2procSym(proc,newdata)
scores <- getPCscores(newdataAlign,proc$PCs,proc$mshape)

## End(Not run)
```

## getPCtol *determine the minimum ratio for two subsequent eigenvalues to be considered different*

### Description

determine the minimum ratio for two subsequent eigenvalues to be considered different

### Usage

```
getPCtol(n, expect = 2)
```

### Arguments

| | |
|---|---|
| `n` | sample size |
| `expect` | expectation value for chi-square distribution of df=2 |

### Value

returns the minimum ratio between two subsequent subsequent eigenvalues to be considered different.

## References

Bookstein, F. L. Measuring and reasoning: numerical inference in the sciences. Cambridge University Press, 2014

## See Also

getMeaningfulPCs

## Examples

```
## reproduce the graph from Bookstein (2014, p. 324)
## and then compare it to ratios for values to be considered
## statistically significant
myseq <- seq(from=10,to = 50, by = 2)
myseq <- c(myseq,seq(from=50,to=1000, by =20))
ratios <- getPCtol(myseq)
plot(log(myseq),ratios,cex=0,xaxt = "n",ylim=c(1,5.2))
ticks <- c(10,20,50,100,200,300,400,500,600,700,800,900,1000)
axis(1,at=log(ticks),labels=ticks)
lines(log(myseq),ratios)
abline(v=log(ticks), col="lightgray", lty="dotted")
abline(h=seq(from=1.2,to=5, by = 0.2), col="lightgray", lty="dotted")

## now we raise the bar and compute the ratios for values
## to be beyond the 95th percentile of
## the corresponding chi-square distribution:
ratiosSig <- getPCtol(myseq,expect=qchisq(0.95,df=2))
lines(log(myseq),ratiosSig,col=2)
```

---

# getPLSCommonShape — *Get the linear combinations associated with the common shape change in each latent dimension of a pls2B*

---

## Description

Get the linear combinations associated with the common shape change in each latent dimension of a pls2B

## Usage

```
getPLSCommonShape(pls)
```

## Arguments

| | |
|---|---|
| pls | object of class "pls2B" |

## Value

returns a list containing

| | |
|---|---|
| `shapevectors` | matrix with each containing the shapevectors (in column- major format) of common shape change associated with each latent dimension |
| `XscoresScaled` | Xscores scaled according to `shapevectors` |
| `YscoresScaled` | Yscores scaled according to `shapevectors` |
| `commoncenter` | Vector containing the common mean |
| `lmdim` | dimension of landmarks |

## References

Mitteroecker P, Bookstein F. 2007. The conceptual and statistical relationship between modularity and morphological integration. Systematic Biology 56(5):818-836.

## See Also

`plsCoVarCommonShape`

## Examples

```
data(boneData)
proc <- procSym(boneLM)
pls <- pls2B(proc$orpdata[1:4,,],proc$orpdata[5:10,,])
commShape <- getPLSCommonShape(pls)
## get common shape for first latent dimension at +-2 sd of the scores
## (you can do this much more convenient using \code{\link{plsCoVarCommonShape}}
scores <- c(-2,2) * sd(c(commShape$XscoresScaled[,1],commShape$YscoresScaled[,1]))
pred <- restoreShapes(scores,commShape$shapevectors[,1],matrix(commShape$commoncenter,10,3))
## Not run:
deformGrid3d(pred[,,1],pred[,,2])

## End(Not run)
```

---

## getPLSfromScores *compute changes associated with 2-Block PLS-scores*

---

## Description

compute changes associated with 2-Block PLS-scores

## Usage

```
getPLSfromScores(pls, x, y)
```

### Arguments

| | |
|---|---|
| `pls` | output of pls2B |
| `x` | scores associated with dataset x in original pls2B |
| `y` | scores associated with dataset y in original pls2B |

### Details

other than `predictPLSfromScores`, providing Xscores will not compute predictions of y, but the changes in the original data `x` that is associated with the specific scores

### Value

returns data in the original space associated with the specified values.

---

## `getPLSscores` *compute 2-Block PLS scores for new data*

---

### Description

compute 2-Block PLS scores for new data from an existing pls2B

### Usage

```
getPLSscores(pls, x, y)
```

### Arguments

| | |
|---|---|
| `pls` | output of pls2B |
| `x` | matrix or vector representing new dataset(s) - same kind as in original pls2B |
| `y` | matrix or vector representing new dataset(s) - same kind as in original pls2B |

### Value

returns a vector of pls-scores

### Note

either x or y must be missing

### See Also

`pls2B`, `predictPLSfromScores`,`predictPLSfromData`

## getPointAlongOutline *Get a point along a line with a given distance from the start of the line*

### Description

Get a point along a line with a given distance from the start of the line

### Usage

```
getPointAlongOutline(mat, dist = 15, reverse = FALSE)
```

### Arguments

| | |
|---|---|
| `mat` | matrix with rows containing sequential coordinates |
| `dist` | numeric: distance from the first point on the line. |
| `reverse` | logical: if TRUE start from the end of the line |

### Value

returns a vector containing the resulting coordinate

## getSides *try to identify bilateral landmarks and sort them by side*

### Description

try to identify bilateral landmarks and sort them by side

### Usage

```
getSides(x, tol = 3, pcAlign = TRUE, icpiter = 100, ...)
```

### Arguments

| | |
|---|---|
| `x` | matrix containing landmarks (see details) |
| `tol` | maximal distance allowed between original and mirrored set. |
| `pcAlign` | logical: if TRUE orginal and mirrored landmarks will be initally aligned by their PC-axes |
| `icpiter` | integer: number of iterations in ICP alignment. |
| `...` | more arguments passed to `mirror`. |

## Details

This function mirrors the landmark set and aligns it to the original. Then it tries to find pairs. If you have a sample, run a Procrustes registration first (without scaling to unit centroid size, or you later have to adapt `tol` - see examples) and then use the mean as it is usually more symmetrical.

## Value

returns a list containing

| | |
|---|---|
| `side1` | integer vector containing indices of landmarks on one side |
| `side2` | integer vector containing indices of landmarks on the other side |
| `unilat` | integer vector containing indices unilateral landmarks |

## Examples

```
data(boneData)
proc <- procSym(boneLM,CSinit=FALSE)
mysides <- getSides(proc$mshape)
if (interactive()){
#visualize bilateral landmarks
deformGrid3d(boneLM[mysides$side1,,1],boneLM[mysides$side2,,1])
## visualize unilateral landmarks
rgl::spheres3d(boneLM[mysides$unilat,,1],radius=0.5)
}
```

---

`getTrafo4x4` *get 4x4 Transformation matrix*

---

## Description

get 4x4 Transformation matrix

## Usage

```
getTrafo4x4(x)

## S3 method for class 'rotonto'
getTrafo4x4(x)
```

## Arguments

| | |
|---|---|
| `x` | object of class "rotonto" |

## Value

returns a 4x4 transformation matrix

### Examples

```
data(boneData)
rot <- rotonto(boneLM[,,1],boneLM[,,2])
trafo <- getTrafo4x4(rot)
```

---

## `getTrafoRotaxis` *compute a 4x4 Transformation matrix for rotation around an arbitrary axis*

---

### Description

compute a 4x4 Transformation matrix for rotation around an arbitrary axis

### Usage

```
getTrafoRotaxis(pt1, pt2, theta)
```

### Arguments

| | |
|---|---|
| `pt1` | numeric vector of length 3, defining first point on the rotation axis. |
| `pt2` | numeric vector of length 3, defining second point on the rotation axis. |
| `theta` | angle to rotate in radians. With pt1 being the viewpoint, the rotation is counter-clockwise. |

### Note

the resulting matrix can be used in `applyTransform`

---

## `getVisibleVertices` *find vertices visible from a given viewpoints*

---

### Description

find vertices visible from a given viewpoints

### Usage

```
getVisibleVertices(mesh, viewpoints, offset = 0.001, cores = 1)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class 'mesh3d' |
| `viewpoints` | vector or k x 3 matrix containing a set of viewpoints |
| `offset` | value to generate an offset at the meshes surface (see notes) |
| `cores` | integer: number of cores to use (not working on windows) |

## Value

a vector with (1-based) indices of points visible from at least one of the viewpoints

## Note

The function tries to filter out all vertices where the line connecting each vertex with the viewpoints intersects with the mesh itself. As, technically speaking this always occurs at a distance of value=0, a mesh with a tiny offset is generated to avoid these false hits.

## Examples

```
SCP1 <- file2mesh(system.file("extdata","SCP1.ply",package="Morpho"))
viewpoints <- read.fcsv(system.file("extdata","SCP1_Endo.fcsv",package="Morpho"))
visivert <- getVisibleVertices(SCP1,viewpoints)
```

---

groupPCA *Perform PCA based of the group means' covariance matrix*

---

## Description

Calculate covariance matrix of the groupmeans and project all observations into the eigenspace of this covariance matrix. This displays a low dimensional between group structure of a high dimensional problem.

## Usage

```
groupPCA(
  dataarray,
  groups,
  rounds = 10000,
  tol = 1e-10,
  cv = TRUE,
  mc.cores = parallel::detectCores(),
  weighting = TRUE
)
```

## Arguments

| | |
|---|---|
| dataarray | Either a k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. Or alternatively a n x m Matrix where n is the numeber of observations and m the number of variables (this can be PC scores for example) |
| groups | a character/factor vector containgin grouping variable. |
| rounds | integer: number of permutations if a permutation test of the euclidean distance between group means is requested.If rounds = 0, no test is performed. |
| tol | threshold to ignore eigenvalues of the covariance matrix. |

| | |
|---|---|
| `cv` | logical: requests leaving-one-out crossvalidation |
| `mc.cores` | integer: how many cores of the Computer are allowed to be used. Default is use autodetection by using detectCores() from the parallel package. Parallel processing is disabled on Windows due to occasional errors. |
| `weighting` | logical:weight between groups covariance matrix according to group sizes. |

## Value

| | |
|---|---|
| `eigenvalues` | Non-zero eigenvalues of the groupmean covariance matrix |
| `groupPCs` | PC-axes - i.e. eigenvectors of the groupmean covariance matrix |
| `Variance` | table displaying the between-group variance explained by each between group PC - this only reflects the variability of the group means and NOT the variability of the data projected into that space |
| `Scores` | Scores of all observation in the PC-space |
| `probs` | p-values of pairwise groupdifferences - based on permuation testing |
| `groupdists` | Euclidean distances between groups' averages |
| `groupmeans` | matrix with rows containing the Groupmeans, or a k x m x groupsize array if the input is a k x m x n landmark array |
| `Grandmean` | vector containing the Grand mean, or a matrix if the input is a k x m x n landmark array |
| `CV` | Cross-validated scores |
| `groups` | grouping Variable |
| `resPCs` | PCs orthogonal to the between-group PCs |
| `resPCscores` | Scores of the residualPCs |
| `resVar` | table displaying the residual variance explained by each residual PC. |

## Author(s)

Stefan Schlager

## References

Mitteroecker P, Bookstein F 2011. Linear Discrimination, Ordination, and the Visualization of Selection Gradients in Modern Morphometrics. Evolutionary Biology 38:100-114.

Boulesteix, A. L. 2005: A note on between-group PCA, International Journal of Pure and Applied Mathematics 19, 359-366.

## See Also

`CVA`

## Examples

```
data(iris)
vari <- iris[,1:4]
facto <- iris[,5]
pca.1 <-groupPCA(vari,groups=facto,rounds=100,mc.cores=1)

### plot scores
if (require(car)) {
scatterplotMatrix(pca.1$Scores,groups=facto, ellipse=TRUE,
        by.groups=TRUE,var.labels=c("PC1","PC2","PC3"))
}
## example with shape data
data(boneData)
proc <- procSym(boneLM)
pop_sex <- name2factor(boneLM, which=3:4)
gpca <- groupPCA(proc$orpdata, groups=pop_sex, rounds=0, mc.cores=2)
## Not run:
## visualize shape associated with first between group PC
dims <- dim(proc$mshape)
## calculate matrix containing landmarks of grandmean
grandmean <-gpca$Grandmean
## calculate landmarks from first between-group PC
#                   (+2 and -2 standard deviations)
gpcavis2sd<- restoreShapes(c(-2,2)*sd(gpca$Scores[,1]), gpca$groupPCs[,1], grandmean)
deformGrid3d(gpcavis2sd[,,1], gpcavis2sd[,,2], ngrid = 0,size=0.01)
require(rgl)
## visualize grandmean mesh

grandm.mesh <- tps3d(skull_0144_ch_fe.mesh, boneLM[,,1],grandmean,threads=1)
wire3d(grandm.mesh, col="white")
spheres3d(grandmean, radius=0.01)

## End(Not run)
```

## histGroup *plot histogram for multiple groups.*

### Description

plot a histogram for multiple groups, each group colored individually

### Usage

```
histGroup(
  data,
  groups,
  main = paste("Histogram of", dataname),
  xlab = dataname,
  ylab,
  col = NULL,
  alpha = 0.5,
  breaks = "Sturges",
  legend = TRUE,
  legend.x = 80,
  legend.y = 80,
  legend.pch = 15,
  freq = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | vector containing data. |
| `groups` | grouping factors |
| `main, xlab, ylab` | these arguments to title have useful defaults here. |
| `col` | vector containing color for each group. If NULL, the function "rainbow" is called. |
| `alpha` | numeric between 0 and 1. Sets the transparency of the colors |
| `breaks` | one of: • a vector giving the breakpoints between histogram cells, • a single number giving the number of cells for the histogram, • a character string naming an algorithm to compute the number of cells (see 'Details'), • a function to compute the number of cells. In the last three cases the number is a suggestion only. |
| `legend` | logical: if TRUE, a legend is plotted |
| `legend.x` | x position of the legend from the upper left corner |
| `legend.y` | y position of the legend from the upper left corners |
| `legend.pch` | integer: define the symbol to visualise group colors (`points`) |
| `freq` | logical: if TRUE, the histogram graphic is a representation of frequencies, the counts component of the result; if FALSE, probability densities are plotted for each group. |

## Details

Just a wrapper for the function hist from the "graphics" package

## Author(s)

Stefan Schlager

## See Also

`hist`

## Examples

```
data(iris)
histGroup(iris$Petal.Length,iris$Species)
```

---

`icpmat` *match two landmark configurations using iteratively closest point search*

---

### Description

match two landmark configurations using iteratively closest point search

### Usage

```
icpmat(
  x,
  y,
  iterations,
  mindist = 1e+15,
  subsample = NULL,
  type = c("rigid", "similarity", "affine"),
  weights = NULL,
  threads = 1,
  centerweight = FALSE
)
```

### Arguments

| | |
|---|---|
| `x` | moving landmarks |
| `y` | target landmarks |
| `iterations` | integer: number of iterations |
| `mindist` | restrict valid points to be within this distance |
| `subsample` | use a subsample determined by kmean clusters to speed up computation |
| `type` | character: select the transform to be applied, can be "rigid","similarity" or "affine" |
| `weights` | vector of length `nrow(x)` containing weights for each row in `x` |
| `threads` | integer: number of threads to use. |
| `centerweight` | logical: if weights are defined and centerweigths=TRUE, the matrix will be centered according to these weights instead of the barycenter. |

### Value

returns the rotated landmarks

## Examples

```
data(nose)
icp <- icpmat(shortnose.lm,longnose.lm,iterations=10)

## example using weights
## we want to assign high weights to the first three cordinates
icpw <- icpmat(shortnose.lm,longnose.lm,iterations=10,
               weights=c(rep(100,3),rep(1,620)),centerweight = TRUE)
## the RMSE between those four points and the target is now smaller:
require(Rvcg)
RMSE <- sqrt(sum(vcgKDtree(longnose.lm,icp[1:3,],k=1)$distance^2))
RMSEW<- sqrt(sum(vcgKDtree(longnose.lm,icpw[1:3,],k=1)$distance^2))
barplot(c(RMSE,RMSEW),names.arg=c("RMSE weighted","RMSE unweighted"))
## Not run:
## plot the differences between unweighted and weighted icp
deformGrid3d(icp,icpw)
## plot the first four coordinates from the icps:
spheres3d(icp[1:3,],col="red",radius = 0.5)
spheres3d(icpw[1:3,],col="green",radius = 0.5)
## plot the target
spheres3d(longnose.lm,col="yellow",radius = 0.2)

## End(Not run)
##2D example  using icpmat to determine point correspondences
if (require(shapes)) {
## we scramble rows to show that this is independent of point order
moving <- gorf.dat[sample(1:8),,1]
plot(moving,asp=1) ## starting config
icpgorf <- icpmat(moving,gorf.dat[,,2],iterations = 20)
points(icpgorf,asp = 1,col=2)
points(gorf.dat[,,2],col=3)## target

## get correspondences using nearest neighbour search
index <- mcNNindex(icpgorf,gorf.dat[,,2],k=1,cores=1)
icpsort <- icpgorf[index,]
for (i in 1:8)
lines(rbind(icpsort[i,],gorf.dat[i,,2]))
}
```

---

`invertFaces` *invert faces' orientation of triangular mesh*

---

## Description

inverts faces' orientation of triangular mesh and recomputes vertex normals

## Usage

```
invertFaces(mesh)
```

## Arguments

|  |  |
|---|---|
| `mesh` | triangular mesh of class `mesh3d` |

## Value

returns resulting mesh

## Author(s)

Stefan Schlager

## See Also

`updateNormals`

## Examples

```
data(nose)
## Not run:
rgl::shade3d(shortnose.mesh,col=3)

## End(Not run)
noseinvert <- invertFaces(shortnose.mesh)
## show normals
## Not run:
plotNormals(noseinvert,long=0.01)

## End(Not run)
```

---

## `kendalldist` *Calculates the Riemannian distance between two superimposed landmark configs.*

---

## Description

Calculates the Riemannian distance between two superimposed landmark configs.

## Usage

```
kendalldist(x, y)
```

## Arguments

|  |  |
|---|---|
| `x` | Matrix containing landmark coordinates. |
| `y` | Matrix containing landmark coordinates. |

## Value

returns Riemannian distance

## Examples

```
if(require(shapes)) {
OPA <- rotonto(gorf.dat[,,1],gorf.dat[,,2])
kendalldist(OPA$X,OPA$Y)
}
```

---

## `line2plane` *get intersection between a line and a plane*

---

## Description

get intersection between a line and a plane

## Usage

```
line2plane(ptLine, ptDir, planePt, planeNorm)
```

## Arguments

| | |
|---|---|
| `ptLine` | vector of length 3: point on line |
| `ptDir` | vector of length 3: direction vector of line |
| `planePt` | vector of length 3: point on plane |
| `planeNorm` | vector of length 3: plane normal vector |

## Value

hit point

## Note

in case you only have three points on a plane (named `pt1`, `pt2`, `pt3` you can get the plane's normal by calling `crossProduct(pt1-pt2,pt1-pt3)`.

`lineplot` *plot lines between landmarks*

## Description

add lines connecting landmarks to visualise a sort of wireframe

## Usage

```
lineplot(
  x,
  point,
  col = 1,
  lwd = 1,
  line_antialias = FALSE,
  lty = 1,
  add = TRUE
)
```

## Arguments

| | |
|---|---|
| `x` | matrix containing 2D or 3D landmarks |
| `point` | vector or list of vectors containing rowindices of x, determining which landmarks to connect. |
| `col` | color of lines |
| `lwd` | line width |
| `line_antialias` | logical: smooth lines |
| `lty` | line type (only for 2D case) |
| `add` | logical: add to existing plot |

## Note

works with 2D and 3D configurations

## Author(s)

Stefan Schlager

## See Also

`pcaplot3d`

## Examples

```
if (require(shapes)) {
##2D example
plot(gorf.dat[,,1],asp=1)
lineplot(gorf.dat[,,1],point=c(1,5:2,8:6,1),col=2)
}
##3D example
## Not run:
require(rgl)
data(nose)
points3d(shortnose.lm[1:9,])
lineplot(shortnose.lm[1:9,],point=list(c(1,3,2),c(3,4,5),c(8,6,5,7,9)),col=2)

## End(Not run)
```

---

`list2array` *converts a list of matrices to an array*

---

## Description

converts a list of matrices to an array

## Usage

```
list2array(x)
```

## Arguments

| | |
|---|---|
| `x` | a list containing matrices of the same dimensionality |

## Value

returns an array concatenating all matrices

---

`LPS2RAS` *convert data from LPS to RAS space and back*

---

## Description

convert data from LPS to RAS space and back

## Usage

```
LPS2RAS(x)
```

## Arguments

| | |
|---|---|
| `x` | mesh of class `mesh3d` or a matrix with 3D Landmarks |

## Details

As e.g. the Slicer versions >= 4.11 are using LPS space, it might be needed to transform data like fiducials and surface models from and back to that space.

## Value

returns the rotated data

---

`mcNNindex` *find nearest neighbours for 2D and 3D point clouds*

---

## Description

find nearest neighbours for point clouds using a kd-tree search. This is just a wrapper of the function vcgKDtree from package Rvcg. Wwraps the function `vcgKDtree` from package 'Rvcg' (for backward compatibility )

## Usage

```
mcNNindex(target, query, cores = parallel::detectCores(), k = k, ...)
```

## Arguments

| | |
|---|---|
| `target` | `k x m` matrix containing data which to search. |
| `query` | `l x m` matrix containing data for which to search. |
| `cores` | integer: amount of CPU-cores to be used. Only available on systems with OpenMP support. |
| `k` | integer: how many closest points are sought. |
| `...` | additional arguments - currently unused. |

## Value

`l x k` matrix containing indices of closest points.

## See Also

`closemeshKD`

### Examples

```
require(rgl)
data(nose)
# find closest vertex on surface for each landmark
clost <- mcNNindex(vert2points(shortnose.mesh),shortnose.lm, k=1,
mc.cores=1)
## Not run:
spheres3d(vert2points(shortnose.mesh)[clost,],col=2,radius=0.3)
spheres3d(shortnose.lm,radius=0.3)
wire3d(shortnose.mesh)

## End(Not run)
```

---

## mergeMeshes *merge multiple triangular meshes into a single one*

---

### Description

merge multiple triangular meshes into a single one, preserving color and vertex normals.

### Usage

```
mergeMeshes(...)
```

### Arguments

|     |     |
|-----|-----|
| `...` | triangular meshes of class `'mesh3d'` to merge or a list of triangular meshes. |

### Value

returns the meshes merged into a single one.

### See Also

`mesh2ply`, `file2mesh`, `ply2mesh`

### Examples

```
## Not run:
require(rgl)
data(boneData)
data(nose)
mergedMesh <- mergeMeshes(shortnose.mesh, skull_0144_ch_fe.mesh)
require(rgl)
shade3d(mergedMesh, col=3)

## End(Not run)
```

## mesh2grey *convert a colored mesh to greyscale.*

### Description

convert the colors of a colored mesh to greyscale values

### Usage

```
mesh2grey(mesh)
```

### Arguments

| | |
|---|---|
| mesh | Object of class mesh3d |

### Value

returns a mesh with material$color replaced by greyscale rgb values.

### Author(s)

Stefan Schlager

### See Also

ply2mesh,file2mesh

## mesh2obj *export mesh objects to disk*

### Description

export mesh objects to disk.

### Usage

```
mesh2obj(x, filename = dataname, writeNormals = TRUE)

mesh2ply(x, filename = dataname, col = NULL, writeNormals = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | object of class `mesh3d` - see rgl documentation for further details or a matrix containing vertices, this can either be a `k x 3` or a `3 x k` matrix, with rows or columns containing vertex coordinates. |
| `filename` | character: Path/name of the requested output - extension will be added atuomatically. If not specified, the file will be named as the exported object. |
| `writeNormals` | logical: if TRUE, existing normals of a mesh are written to file - can slow things down for very large meshes. |
| `col` | Writes color information to ply file. Can be either a single color value or a vector containing a color value for each vertex of the mesh. |

## Details

export an object of class `mesh3d` or a set of coordinates to a common mesh file.

## Note

meshes containing quadrangular faces will be converted to triangular meshes by splitting the faces. Additionally, mesh2obj is now simply a wrapper of `Rvcg::vcgObjWrite`.

## Author(s)

Stefan Schlager

## See Also

`ply2mesh`, `quad2trimesh`

## Examples

```
require(rgl)
vb <- c(-1.8,-1.8,-1.8,1.0,1.8,-1.8,-1.8,1.0,-1.8,1.8,-1.8,1.0,1.8,
1.8,-1.8,1.0,-1.8,-1.8,1.8,1.0,1.8,
-1.8,1.8,1.0,-1.8,1.8,1.8,1.0,1.8,1.8,1.8,1.0)
it <- c(2,1,3,3,4,2,3,1,5,5,7,3,5,1,2,2,6,5,6,8,7,7,5,6,7,8,4,4,3,7,4,8,6,6,2,4)
vb <- matrix(vb,4,8) ##create vertex matrix
it <- matrix(it,3,12) ## create face matrix
cube<-list(vb=vb,it=it)
class(cube) <- "mesh3d"
## Not run:
shade3d(cube,col=3) ## view the green cube

## End(Not run)
mesh2ply(cube,filename="cube") # write cube to a file called cube.ply
unlink("cube.ply")
```

## meshcube *calculate the corners of a mesh's bouning box*

### Description

calculate the corners of a mesh's bouning box

### Usage

```
meshcube(x)
```

### Arguments

| | |
|---|---|
| x | object of class 'mesh3d' |

### Value

returns a 8 x 3 matrix with the coordinates of the corners of the bounding box.

### Examples

```
require(rgl)
data(boneData)
mc <- meshcube(skull_0144_ch_fe.mesh)
## Not run:
spheres3d(mc)
wire3d(skull_0144_ch_fe.mesh)

## End(Not run)
```

## meshDist *calculates and visualises distances between surface meshes or 3D coordinates and a surface mesh.*

### Description

calculates and visualises distances between surface meshes or 3D coordinates and a surface mesh.

## Usage

```
meshDist(x, ...)

## S3 method for class 'mesh3d'
meshDist(
  x,
  mesh2 = NULL,
  distvec = NULL,
  from = NULL,
  to = NULL,
  steps = 20,
  ceiling = FALSE,
  rampcolors = colorRamps::blue2green2red(steps - 1),
  NAcol = "white",
  file = "default",
  imagedim = "100x800",
  uprange = 1,
  ray = FALSE,
  raytol = 50,
  raystrict = FALSE,
  save = FALSE,
  plot = TRUE,
  sign = TRUE,
  tol = NULL,
  tolcol = "green",
  displace = FALSE,
  shade = TRUE,
  method = c("vcglib", "morpho"),
  add = FALSE,
  scaleramp = TRUE,
  threads = 1,
  titleplot = "Distance in mm",
  ...
)

## S3 method for class 'matrix'
meshDist(
  x,
  mesh2 = NULL,
  distvec = NULL,
  from = NULL,
  to = NULL,
  steps = 20,
  ceiling = FALSE,
  rampcolors = colorRamps::blue2green2red(steps - 1),
  NAcol = "white",
  uprange = 1,
  plot = TRUE,
  sign = TRUE,
  tol = NULL,
  tolcol = "green",
  type = c("s", "p"),
  radius = NULL,
  displace = FALSE,
  add = FALSE,
  scaleramp = FALSE,
  titleplot = "Distance in mm",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | reference mesh; object of class "mesh3d" or a n x 3 matrix containing 3D coordinates. |
| `...` | additional arguments passed to `shade3d`. See `material3d` for details. |
| `mesh2` | target mesh: either object of class "mesh3d" or a character pointing to a surface mesh (ply, obj or stl file) |
| `distvec` | vector: optional, a vector containing distances for each vertex/coordinate of `x`, if distvec != NULL, `mesh2` will be ignored. |
| `from` | numeric: minimum distance to be colorised; default is set to 0 mm |
| `to` | numeric: maximum distance to be colorised; default is set to the maximum distance |
| `steps` | integer: determines break points for color ramp: n steps will produce n-1 colors. |
| `ceiling` | logical: if TRUE, the next larger integer of "to" is used |
| `rampcolors` | character vector: specify the colors which are used to create a colorramp. |
| `NAcol` | character: specify color for values outside the range defined by `from` and `to`. |
| `file` | character: filename for mesh and image files produced. E.g. "mydist" will produce the files mydist.ply and mydist.png |
| `imagedim` | character of type 100x200 where 100 determines the width and 200 the height of the image. |
| `uprange` | numeric between 0 and 1: restricts "to" to a quantile of "to", if to is NULL. |
| `ray` | logical: if TRUE, the search is along vertex normals. |
| `raytol` | maximum distance to follow a normal. |
| `raystrict` | logical: if TRUE, only outward along normals will be sought for closest points. |
| `save` | logical: save a colored mesh. |
| `plot` | logical: visualise result as 3D-plot and distance charts |
| `sign` | logical: request signed distances. Only meaningful, if mesh2 is specified or distvec contains signed distances. |
| `tol` | numeric: threshold to color distances within this threshold green. |
| `tolcol` | a custom color to color vertices below a threshold defined by `tol`. Default is green. |

| | |
|---|---|
| `displace` | logical: if TRUE, displacement vectors between original and closest points are drawn colored according to the distance. |
| `shade` | logical: if FALSE, the rendering of the colored surface will be supressed. |
| `method` | accepts: "vcglib" and "morpho" (and any abbreviation). vcglib uses a command line tool using vcglib headers, morpho uses fortran routines based on a kd-tree search for closest triangles. |
| `add` | logical: if TRUE, visualization will be added to the rgl window currently in focus |
| `scaleramp` | logical: if TRUE, the colorramp will be symmetrical for signed distances: spanning from `-max(from,to)` to `max(from,to)`. |
| `threads` | integer: number of threads to use. 0 = let system decide. |
| `titleplot` | character: axis description of heatmap. |
| `type` | character: "s" shows coordinates as spheres, while "p" shows 3D dots. |
| `radius` | determines size of spheres; if not specified, optimal radius size will be estimated by centroid size of the configuration. |

## Details

calculates the distances from a mesh or a set of 3D coordinates to another at each vertex; either closest point or along the normals

## Value

Returns an object of class "meshDist" if the input is a surface mesh and one of class "matrixDist" if input is a matrix containing 3D coordinates.

| | |
|---|---|
| `colMesh` | object of mesh3d with colors added |
| `dists` | vector with distances |
| `cols` | vector with color values |
| `params` | list of parameters used |

## Author(s)

Stefan Schlager

## References

Detection of inside/outside uses the algorithm proposed in:

Baerentzen, Jakob Andreas. & Aanaes, H., 2002. Generating Signed Distance Fields From Triangle Meshes. Informatics and Mathematical Modelling, .

## See Also

`render.meshDist`, , `export.meshDist`, `shade3d`

### Examples

```
data(nose)##load data
##warp a mesh onto another landmark configuration:
longnose.mesh <- tps3d(shortnose.mesh, shortnose.lm, longnose.lm,threads=1)
## Not run:
mD <- meshDist(longnose.mesh, shortnose.mesh)
##now change the color ramp
render(mD,rampcolors = c("white","red"))

## End(Not run)
#use unsigned distances and a ramp from blue to red
#color distances < 0.01 green:
## Not run:
meshDist(longnose.mesh, shortnose.mesh, rampcolors = c("blue", "red"),sign=FALSE, tol=0.5)

## End(Not run)
```

---

## `meshPlaneIntersect` *get intersections between mesh and a plane*

---

### Description

get intersections between mesh and a plane

### Usage

```
meshPlaneIntersect(mesh, v1, v2 = NULL, v3 = NULL, normal = NULL)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class "mesh3d" |
| `v1` | numeric vector of length=3 specifying a point on the separating plane |
| `v2` | numeric vector of length=3 specifying a point on the separating plane |
| `v3` | numeric vector of length=3 specifying a point on the separating plane |
| `normal` | plane normal (overrides specification by v2 and v3) |

### Value

returns the intersections of edges and the plane

### Examples

```
data(nose)
v1 <- shortnose.lm[1,]
v2 <- shortnose.lm[2,]
v3 <- shortnose.lm[3,]
intersect <- meshPlaneIntersect(shortnose.mesh,v1,v2,v3)
## Not run:
```

```
require(rgl)
wire3d(shortnose.mesh)
spheres3d(shortnose.lm[1:3,],col=2)#the plane
spheres3d(intersect,col=3,radius = 0.2)#intersections

## End(Not run)
```

## meshres *calculate average edge length of a triangular mesh*

### Description

calculate average edge length of a triangular mesh, by iterating over all faces.

### Usage

```
meshres(mesh)
```

### Arguments

| | |
|---|---|
| mesh | triangular mesh stored as object of class "mesh3d" |

### Value

returns average edge length (a.k.a. mesh resolution)

### Author(s)

Stefan Schlager

### Examples

```
data(boneData)
mres <- meshres(skull_0144_ch_fe.mesh)
```

`mirror` *mirror landmarks or triangular mesh in place*

## Description

mirror landmarks or triangular mesh in place

## Usage

```
mirror(
  x,
  icpiter = 50,
  subsample = NULL,
  pcAlign = FALSE,
  mirroraxis = 1,
  initPC = TRUE,
  initCenter = TRUE,
  v1 = NULL,
  v2 = NULL,
  v3 = NULL,
  normal = NULL,
  mc.cores = 2
)

## S3 method for class 'matrix'
mirror(
  x,
  icpiter = 50,
  subsample = NULL,
  pcAlign = FALSE,
  mirroraxis = 1,
  initPC = TRUE,
  initCenter = TRUE,
  v1 = NULL,
  v2 = NULL,
  v3 = NULL,
  normal = NULL,
  mc.cores = 2
)

## S3 method for class 'mesh3d'
mirror(
  x,
  icpiter = 50,
  subsample = NULL,
  pcAlign = FALSE,
  mirroraxis = 1,
  initPC = TRUE,
  initCenter = TRUE,
  v1 = NULL,
  v2 = NULL,
  v3 = NULL,
  normal = NULL,
  mc.cores = 2
)
```

## Arguments

| | |
|---|---|
| x | k x 3 matrix or mesh3d |
| icpiter | integer: number of iterations to match reflected configuration onto original one |
| subsample | integer: use only a subset for icp matching |
| pcAlign | if TRUE, the icp will be preceeded by an alignment of the principal axis (only used if icpiter > 0), currently only works for 3D data. |
| mirroraxis | integer: which axis to mirror at |
| initPC | logical: if TRUE the data will be prealigned by its principal axes. |
| initCenter | logical: if TRUE and `initPC=FALSE`, `x` will be translated to its centroid before mirroring. |
| v1 | point on plane |
| v2 | if normal=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |
| v3 | if normal=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |
| normal | plane normal (overrides specification by v2 and v3) |
| mc.cores | use parallel processing to find best alignment to original shape. |

## Details

reflect a mesh configuration at the plane spanned by its first 2 principal axis, then try to rigidily register the reflected configuration onto the original one using iterative closest point search to establish correspondences. Also, if a reflection plane is defined, `pcAlign`, `initPC`, `initCenter` and `mirroraxis` will be ignored and the object will be mirrored on the defined plane (and optionally aligned using an ICP approach).

## Value

returns the reflected object

## Examples

```
data(boneData)
boneMir <- mirror(boneLM[,,1],icpiter=50,mc.cores=2,mirroraxis=3)

### mirror on 3 midsaggital landmarks and then optimize it with an ICP
boneMirPlane <- mirror(boneLM[,,1],v1=boneLM[1,,1],v2=boneLM[2,,1],v3=boneLM[9,,1])

## 2D Example:
```

```
if (require(shapes)) {
gorfMir <- mirror(gorf.dat[,,1],mirroraxis=2,pcAlign=TRUE,icpiter = 0)
plot(gorfMir,asp = 1)
points(gorf.dat[,,1],col=3)
}
## Not run:
## now mirror a complete mesh
require(rgl)
skullMir <- mirror(skull_0144_ch_fe.mesh,icpiter=10,subsample = 30,
                   mc.cores=2,mirroraxis=3,pcAlign=TRUE)
###compare result to original
wire3d(skull_0144_ch_fe.mesh,col=3)
wire3d(skullMir,col=2)

## End(Not run)
```

---

## `mirror2plane` *mirror points or mesh on an arbitrary plane*

---

### Description

mirror points or mesh on an arbitrary plane

### Usage

```
mirror2plane(x, v1, normal = NULL, v2 = NULL, v3 = NULL)

## S3 method for class 'matrix'
mirror2plane(x, v1, normal = NULL, v2 = NULL, v3 = NULL)

## S3 method for class 'mesh3d'
mirror2plane(x, v1, normal = NULL, v2 = NULL, v3 = NULL)
```

### Arguments

| | |
|---|---|
| `x` | x 3D-vector or a k x 3 matrix with 3D vectors stored in rows. Or a triangular mesh of class mesh3d |
| `v1` | point on plane |
| `normal` | plane normal (overrides specification by v2 and v3) |
| `v2` | if pNorm=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |
| `v3` | if pNorm=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |

### Value

mirrored coordinates mesh

### Examples

```
# mirror mesh on plane spanned by 3 midsagital landmarks
data(boneData)
mirrmesh <- mirror2plane(skull_0144_ch_fe.mesh,v1=boneLM[1,,1],v2=boneLM[9,,1],v3=boneLM[10,,1])
```

---

## name2factor *extract data from array names*

---

### Description

extract data from array names

### Usage

```
name2factor(x, sep = "_", which, collapse = sep, as.factor = TRUE)

name2num(x, sep = "_", which, collapse = sep, dif = TRUE)
```

### Arguments

| | |
|---|---|
| x | data, can be a three-dimensional array, a matrix, a named list or a vector containing names to split |
| sep | character by which to split the strings |
| which | integer or vector of integers, if more entries are selected, they will be concatenated by the string specified with the option 'collapse'. |
| collapse | character by which to collapse data if two strings are to be concatenated |
| as.factor | logical: if TRUE, a factor vector will be returned, strings otherwise. |
| dif | logical: calculate difference if two fields containing numbers are selected. |

### Details

extract data from array names and convert to factors or numbers

If an array is used as input, the data info is expected to be in the 3rd dimension, for a matrix, rownames are used.

### Value

returns a vector containing factors or numbers

### Author(s)

Stefan Schlager

## Examples

```
data <- matrix(rnorm(200),100,2)
id <- paste("id",1:100,sep="")
pop <- c(rep("pop1",50),rep("pop2",50))
sex <- c(rep("male",50),rep("female",50))
age <- floor(rnorm(100,mean=50,sd=10))
rownames(data) <- paste(id,pop,sex,age,sep="_")
infos <- data.frame(pop=name2factor(data,which=2))
infos$age <- name2num(data,which=4)
infos$pop.sex <- name2factor(data,which=2:3)
```

## NNshapeReg *Estimate the shape by averaging the shape of the nearest neighbours.*

### Description

Estimate the shape of one set of landmarks by averaging the shape of the nearest neighbours obtained by a second set of landmarks. Weights are calculated either form Mahalanobis or Procrustes distances. This can be useful for data with missing landmarks.

### Usage

```
NNshapeReg(
  x,
  y = NULL,
  n = 3,
  mahalanobis = FALSE,
  mc.cores = parallel::detectCores()
)
```

### Arguments

| | |
|---|---|
| x | an array or matrix (one row per specim) with data used for estimating weights. |
| y | an array or matrix (one row per specim) with landmark data on which the weighted averaging is applied for prediction. If NULL, x will be used for both tasks. |
| n | amount of nearest neighbours to consider |
| mahalanobis | logical: use mahalanobis distance |
| mc.cores | integer: amount of cores used for parallel processing. |

### Details

This function calculates weights from one set of shape data and then estimates the shape of another (or same) set of landmarks. CAUTION: landmark data has to be registered beforehand.

### Value

matrix or array of estimates.

### See Also

proc.weight, fixLMtps

### Examples

```
if (require(shapes)) {
proc <- procSym(gorf.dat)
#use the closest 3 specimen based on the first 4 landmarks
#to estimate the shape
estim <- NNshapeReg(proc$rotated[1:4,,],proc$rotated,n=3,mc.cores=1)
#compare estimation and true config
plot(proc$rotated[,,1],asp=1)
points(estim[,,1],col=2)
}
```

## nose *landmarks and a triangular mesh representing a human nose*

### Description

triangular mesh representing a human nose and two matrices containing landmark data

### Format

`shortnose.mesh`: A triangular mesh of class 'mesh3d'.

`shortnose.lm`: matrix containing example landmark data placed on `shortnose.mesh`.

`longnose.lm`: matrix containing example landmark data representing a caricaturesquely deformed human nose.

## pcAlign *align two 3D-pointclouds/meshes by their principal axes*

### Description

align two 3D-pointclouds/meshes by their principal axes

## Usage

```
pcAlign(x, y, optim = TRUE, subsample = NULL, iterations = 10, mc.cores = 2)

## S3 method for class 'matrix'
pcAlign(x, y, optim = TRUE, subsample = NULL, iterations = 10, mc.cores = 2)

## S3 method for class 'mesh3d'
pcAlign(x, y, optim = TRUE, subsample = NULL, iterations = 10, mc.cores = 2)
```

## Arguments

| | |
|---|---|
| `x` | matrix or mesh3d |
| `y` | matrix or mesh3d, if missing, x will be centered by its centroid and aligned by its princial axis. |
| `optim` | logical if TRUE, the RMSE between reference and target will be minimized testing all possible axes alignments and (if iterations > 0) followed by a rigid ICP procedure. |
| `subsample` | integer: use subsampled points to decrease computation time of optimization. |
| `iterations` | integer: number of iterations for optimization (the higher the more accurate but also more time consuming). |
| `mc.cores` | use parallel processing to find best alignment to original shape. |

## Details

`x` and `y` will first be centered and aligned by their PC-axes. If `optim=TRUE`,all possible 8 ordinations of PC-axes will be tested and the one with the smallest RMSE between the transformed version of `x` and the closest points on `y` will be used. Then the rotated version of `x` is translated to the original center of mass of `y`.

## Value

rotated and translated version of x to the center and principal axes of y.

## Examples

```
data(boneData)
blm1 <- pcAlign(boneLM[,,1],boneLM[,,2])
## Not run:
require(rgl)
spheres3d(boneLM[,,1])#original position
spheres3d(blm1,col=2)#aligned configuration
spheres3d(boneLM[,,2],col=3)#target

## End(Not run)
```

`pcaplot3d` *visualization of shape variation*

## Description

visualization of shape change

## Usage

```
pcaplot3d(x, ...)

## S3 method for class 'symproc'
pcaplot3d(
  x,
  pcshow = c(1, 2, 3),
  mag = 3,
  color = 4,
  lwd = 1,
  sym = TRUE,
  legend = TRUE,
  type = c("spheres", "points"),
  ...
)

## S3 method for class 'nosymproc'
pcaplot3d(
  x,
  pcshow = c(1, 2, 3),
  mag = 3,
  color = 4,
  lwd = 1,
  legend = TRUE,
  type = c("spheres", "points"),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | a object derived from the function procSym calculated on 3D coordinates. |
| `...` | Additional parameters which will be passed to the methods. |
| `pcshow` | a vector containing the PCscores to be visualized. |
| `mag` | a vector or an integer containing which standard deviation of which PC has to be visualized. |
| `color` | color of the 3d points/spheres. |
| `lwd` | width of the lines representing the shape change. |

| | |
|---|---|
| `sym` | logical: if TRUE the symmetric component of shape is displayed. Otherwise the asymmetric one. |
| `legend` | logical: if TRUE a legend explaining the color coding of the PCs is plotted. |
| `type` | character: for `type="spheres"`, the landmarks will be rendered using rgl's `spheres3d` function and for `type="points"` by `points3d` respectivly. |

## Details

visualization of the shape changes explained by Principal components

## Value

returns an invisible array containing the shapes associated with the Principal components selected.

## See Also

`procSym`

## Examples

```
## Not run:
data(boneData)
proc <- procSym(boneLM)
pcaplot3d(proc,pcshow=1:3,mag=-3)#only one PC available

## End(Not run)
```

---

`PCdist` *correlation between a reduced space and the original space*

---

## Description

Calculates the correlation between distances in a reduced space and the original space

## Usage

```
PCdist(PCs, PCscores, x = 5, plot.type = "b")
```

## Arguments

| | |
|---|---|
| `PCs` | m x k matrix of Principal Components where m is the k is the number of PCs. |
| `PCscores` | n x m matrix of Principal Component scores where n is the number of observations. |
| `x` | integer: increment for every x-th PC the subspace to fullspace correlation will be calculated. |
| `plot.type` | "b"=barplot of correlation values, "s"=line between correlation values. |

### Value

a vector of R-squared values between subspace and fullspace distances and a barplot depicting the correlations belonging to the subspace.

### Author(s)

Stefan Schlager

### Examples

```
if (require(shapes)) {
a <- procSym(gorf.dat)
PCdist(a$PCs, a$PCscores, x = 2)
}
```

## permudist *performs permutation testing for group differences.*

### Description

This function compares the distance between two groupmeans to the distances obtained by random assignment of observations to this groups.

### Usage

```
permudist(
  data,
  groups,
  rounds = 1000,
  which = NULL,
  p.adjust.method = "none",
  median = FALSE
)
```

### Arguments

| | |
|---|---|
| `data` | array or matrix containing data |
| `groups` | factors determining grouping. |
| `rounds` | number of permutations |
| `which` | integer (optional): in case the factor levels are > 2 this determins which factor-levels to use |
| `p.adjust.method` | method to adjust p-values for multiple comparisons see `p.adjust.methods` for options. |
| `median` | logical: if TRUE, comparison will be median instead of mean. |

## Value

| | |
|---|---|
| `dist` | distance matrix with distances between actual group means |
| `p.adjust.method` | method used for p-value adjustion |
| `p.value` | distance matrix containing pairwise p-values obtained by comparing the actual distance to randomly acquired distances |

## Examples

```
data(boneData)
proc <- procSym(boneLM)
groups <- name2factor(boneLM,which=3)
perm <- permudist(proc$PCscores[,1:10], groups=groups, rounds=10000)

## now we concentrate only on sex dimorphism between Europeans
groups <- name2factor(boneLM,which=3:4)
levels(groups)
perm1 <- permudist(proc$PCscores, groups=groups,which=3:4, rounds=10000)
```

---

`permuvec` *perfom permutation testing on angles and distances between subgroups of two major groups.*

---

## Description

perform permutation test on length and angle of the vectors connecting the subgroup means of two groups: e.g. compare if length and angle between sex related differences in two populations differ significantly.

## Usage

```
permuvec(
  data,
  groups,
  subgroups = NULL,
  rounds = 9999,
  scale = TRUE,
  tol = 1e-10,
  mc.cores = parallel::detectCores()
)
```

## Arguments

| | |
|---|---|
| `data` | array or matrix containing data. |
| `groups` | factors of firs two grouping variables. |
| `subgroups` | factors of the subgrouping. |
| `rounds` | number of requested permutation rounds |
| `scale` | if TRUE: data will be scaled by pooled within group covarivance matrix. Otherwise Euclidean distance will be used for calculating distances. |
| `tol` | threshold for inverting covariance matrix. |
| `mc.cores` | integer: determines how many cores to use for the computation. The default is autodetect. But in case, it doesn't work as expected cores can be set manually.Parallel processing is disabled on Windows due to occasional errors. |

## Details

This function calculates means of all four subgroups and compares the residual vectors of the major grouping variables by angle and distance.

## Value

| | |
|---|---|
| `angle` | angle between the vectors of the subgroups means |
| `dist` | distances between subgroups |
| `meanvec` | matrix containing the means of all four subgroups |
| `permutangles` | vector containing angles (in radians) from random permutation |
| `permudists` | vector containing distances from random permutation |
| `p.angle` | p-value of angle between residual vectors |
| `p.dist` | p-value of length difference between residual vectors |
| `subdist` | length of residual vectors connecting the subgroups |

means.

## Examples

```
data(boneData)
proc <- procSym(boneLM)
pop <- name2factor(boneLM,which=3)
sex <- name2factor(boneLM,which=4)
## use non scaled distances by setting \code{scale = FALSE}
## and only use first 10 PCs
perm <- permuvec(proc$PCscores[,1:10], groups=pop, subgroups=sex,
                 scale=FALSE, rounds=100, mc.cores=2)


## visualize if the amount of sexual dimorphism differs between
# (lenghts of vectors connecting population specific sex's averages)
# differs between European and Chines
hist(perm$permudist, xlim=c(0,0.1),main="measured vs. random distances",
     xlab="distances")
points(perm$dist,10,col=2,pch=19)#actual distance
text(perm$dist,15,label=paste("actual distance\n
     (p=",perm$p.dist,")"))
## not significant!!

## visualize if the direction of sexual dimorphism
# (angle between vectors connecting population specific sex's averages)
# differs between European and Chines
hist(perm$permutangles, main="measured vs. random angles",
     xlab="angles")
points(perm$angle,10,col=2,pch=19)#actual distance
text(perm$angle,15,label=paste("actual distance\n
    (p=",perm$p.angle,")"))
## also non-significant
```

---

## placePatch — *Project semi-landmarks from a predefined atlas onto all specimen in a sample*

---

### Description

Project semi-landmarks from a predefined atlas onto all specimen in a sample. Various mechanisms are implemented to avoid errorneous placement on the wrong surface layer (e.g. inside the bone).

### Usage

```
placePatch(
  atlas,
  dat.array,
  path,
  prefix = NULL,
  fileext = ".ply",
  ray = TRUE,
  inflate = NULL,
  tol = inflate,
  relax.patch = TRUE,
  keep.fix = NULL,
  rhotol = NULL,
  silent = FALSE,
  mc.cores = 1
)
```

### Arguments

| | |
|---|---|
| `atlas` | object of class "atlas" created by `createAtlas` |
| `dat.array` | k x 3 x n array containing reference landmarks of the sample or a matrix in case of only one target specimen. |

| | |
|---|---|
| `path` | character: specify the directory where the surface meshes of the sample are stored. |
| `prefix` | character: prefix to the specimens names (stored in `dimnames(dat.array)[[3]]`) to match the corresponding file names. If `dat.array` has no dimnames (e.g. because it is a matrix - see example below), this can also be a character vector containing the filenames to which `fileext` will be appended. |
| `fileext` | character: file extension of the surface meshes. |
| `ray` | logical: projection will be along surface normals instead of simple closest point search. |
| `inflate` | inflate (or deflate - if negative sign) the semilandmarks along the normals of the deformed atlas to make sure that they stay on the outside (inside) of the target mesh. |
| `tol` | numeric: threshold to follow the ray back after inflation. See details below. If no surface is hit after `tol` mm, the simple closest point will be used. |
| `relax.patch` | logical: request relaxation minimising bending energy toward the atlas. |
| `keep.fix` | integer: rowindices of those landmarks that are not allowed to be relaxed in case `relax.patch=TRUE`. If not specified, all landmarks will be kept fix. This is preferably set during atlas creation with `createAtlas`: In case you specified corrCurves on the atlas, you should define explicitly which landmarks (also on the curves) are supposed to fix to prevent them from sliding. |
| `rhotol` | numeric: maximum amount of deviation a hit point's normal is allowed to deviate from the normal defined on the atlas. If `relax.patch=TRUE`, those points exceeding this value will be relaxed freely (i.e. not restricted to tangent plane). |
| `silent` | logical: suppress messages. |
| `mc.cores` | run in parallel (experimental stuff now even available on Windows). On windows this will only lead to a significant speed boost for many configurations, as all required packages (Morpho and Rvcg) need to be loaded by each newly spawned process. |

## Details

This function allows the (relatively) easy projection of surface points defined on an atlas onto all surface of a given sample by Thin-Plate Spline deformation and additional mechanisms to avoid distortions. The algorithm can be outlined as followed.

1. relax curves (if specified) against atlas.
2. deform atlas onto targets by TPS based on predefined landmarks (and curves).
3. project coordinates on deformed atlas onto target mesh
4. 'inflate' or 'deflate' configuration along their normals to make sure all coordinates are on the outside/inside
5. Project inflated points back onto surface along these normals.
6. Check if normals are roughly pointing into the same direction as those on the (deformed) atlas.
7. Relax all points against atlas.
8. the predefined coordinates will note change afterwards!

## Value

array containing the projected coordinates appended to the data.array specified in the input. In case dat.array is a matrix only a matrix is returned.

## Author(s)

Stefan Schlager

## References

Schlager S. 2013. Soft-tissue reconstruction of the human nose: population differences and sexual dimorphism. PhD thesis, Universitätsbibliothek Freiburg. URL: http://www.freidok.uni-freiburg.de/volltexte/9181/.

## See Also

createAtlas, relaxLM, checkLM,slider3d, tps3d

## Examples

```
## Not run:
data(nose)
require(rgl)
###create mesh for longnose
longnose.mesh <- tps3d(shortnose.mesh,shortnose.lm,longnose.lm,threads=1)
## create atlas
fix <- c(1:5,20:21)
atlas <- createAtlas(shortnose.mesh, landmarks =
           shortnose.lm[fix,], patch=shortnose.lm[-c(1:5,20:21),])
## view atlas

plotAtlas(atlas)

## create landmark array with only fix landmarks
data <- bindArr(shortnose.lm[fix,], longnose.lm[fix,], along=3)
dimnames(data)[[3]] <- c("shortnose", "longnose")

### write meshes to disk
mesh2ply(shortnose.mesh, filename="shortnose")
mesh2ply(longnose.mesh, filename="longnose")

patched <- placePatch(atlas, data, path="./", inflate=5)
## now browse through placed patches
checkLM(patched, path="./", atlas=atlas)

## same example with only one target specimen
data <- longnose.lm[fix, ]

patched <- placePatch(atlas, data, prefix="longnose", path="./", inflate=5)
wire3d(longnose.mesh,col=3)
spheres3d(patched)
```

```
## End(Not run)
```

---

`plot.slider3d` *plot the result of slider3d*

---

## Description

plot the result of slider3d

## Usage

```
## S3 method for class 'slider3d'
plot(
  x,
  cols = 2:4,
  pt.size = NULL,
  point = c("sphere", "point"),
  specimen = 1,
  add = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | result of `slider3d` call |
| `cols` | vector containing colors for each coordinate type cols[1]=landmarks, cols[2]=surface landmarks, cols[3]=outlines. |
| `pt.size` | size of plotted points/spheres. If `point="s"`. `pt.size` defines the radius of the spheres. If `point="p"` it sets the variable `size` used in `point3d`. |
| `point` | how to render landmarks. |
| `specimen` | integer: select the specimen to plot |
| `add` | logical: if TRUE, a new rgl window is opened. |
| `...` | additonal, currently unused parameters |

## plotAtlas *visualize an atlas defined by createAtlas*

### Description

visualize an atlas defined by createAtlas

### Usage

```
plotAtlas(
  atlas,
  pt.size = NULL,
  alpha = 1,
  render = c("w", "s"),
  point = c("s", "p"),
  meshcol = "white",
  add = TRUE,
  legend = TRUE,
  cols = 2:5
)
```

### Arguments

| | |
|---|---|
| `atlas` | object of class atlas created by `createAtlas`. |
| `pt.size` | size of plotted points/spheres. If `point="s"`. `pt.size` defines the radius of the spheres. If `point="p"` it sets the variable `size` used in `point3d`. |
| `alpha` | value between 0 and 1. Sets transparency of mesh 1=opaque 0= fully transparent. |
| `render` | if `render="w"`, a wireframe will be drawn, if `render="s"`, the mesh will be shaded. |
| `point` | how to render landmarks. "s"=spheres, "p"=points. |
| `meshcol` | color to render the atlas mesh |
| `add` | logical: if TRUE, a new rgl window is opened. |
| `legend` | logical: request plot of legend specifying landmark coloring. |
| `cols` | vector containing colors for each coordinate type cols[1]=landmarks, cols[2]=patch, cols[3]=corrCurves, cols[4]=patchCurves. |

### Details

If `legend=TRUE`, a plot with a legend will open where coloring of the 3D-spheres is specified.

### Value

returns invisible vector containing `rgl.id` of rendered objects.

### See Also

placePatch, createAtlas

### Examples

```
data(nose)
atlas <- createAtlas(shortnose.mesh, landmarks =
            shortnose.lm[c(1:5,20:21),], patch=shortnose.lm[-c(1:5,20:21),])
## Not run:
plotAtlas(atlas)

## End(Not run)
```

---

## plotNormals — *plots the normals of a triangular surface mesh.*

### Description

visualises the vertex normals of a triangular surface mesh of class mesh3d. If no normals are contained, they are computed.

### Usage

```
plotNormals(x, length = 1, lwd = 1, col = 1, ...)
```

### Arguments

| | |
|---|---|
| `x` | object of class "mesh3d" |
| `length` | either a single numeric value or a numeric vector defining per-normals lenght (default is 1) |
| `lwd` | width of the normals |
| `col` | color of the normals |
| `...` | addtional parameters, currently not in use. |

### Author(s)

Stefan Schlager

### Examples

```
## Not run:
require(rgl)
data(nose)
plotNormals(shortnose.mesh,col=4,length=0.01)
shade3d(shortnose.mesh,col=3)

## End(Not run)
```

pls2B *Two-Block partial least square regression.*

## Description

Performs a Two-Block PLS on two sets of data and assesses the significance of each score by permutation testing

## Usage

```
pls2B(
  x,
  y,
  tol = 1e-12,
  same.config = FALSE,
  rounds = 0,
  useCor = FALSE,
  cv = FALSE,
  cvlv = NULL,
  mc.cores = parallel::detectCores(),
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | array containing superimposed landmark data second block.Matrices are also allowed but the option 'same.config' will not work. |
| `y` | array containing superimposed landmark data of the first block. Matrices are also allowed but the option 'same.config' will not work. |
| `tol` | threshold for discarding singular values. |
| `same.config` | logical: if `TRUE` each permutation includes new superimposition of permuted landmarks. This is necessary if both blocks originate from landmarks that are superimposed together. |
| `rounds` | rounds of permutation testing. |
| `useCor` | if TRUE, the correlation matrix instead of the covariance matrix is used. |
| `cv` | logical: if TRUE, a leave-one-out cross-validation is performed |
| `cvlv` | integer: number of latent variables to test |
| `mc.cores` | integer: determines how many cores to use for the |
| `...` | arguments passed to `ProcGPA` computation. The default is autodetect. But in case, it doesn't work as expected cores can be set manually. Parallel processing is disabled on Windows due to occasional errors. |

## Details

The Two-Block PLS tries to find those linear combinations in each block maximising the covariance between blocks. The significance of each linear combination is assessed by comparing the singular value to those obtained from permuted blocks. If both blocks contain landmarks superimposed TOGETHER, the option `same.config=TRUE` requests superimposition of the permuted configurations (i.e. where the the landmarks of block `x` are replaced by corresponding landmarks of other specimen.

## Value

| | |
|---|---|
| `svd` | singular value decomposition (see `svd`) of the 'common' covariance block |
| `Xscores` | PLS-scores of x |
| `Yscores` | PLS-scores of y |
| `CoVar` | Dataframe containing singular values, explained covariation, correlation coeffictient between PLS-scores and p-values for singular values obtained from permutation testing |
| `xlm` | linear model: `lm(Xscores ~ Yscores - 1)` |
| `ylm` | linear model: `lm(Yscores ~ Xscores - 1)` |
| `predicted.x` | array containing matrices of cross-validated predictions for `x`(landmarks arrays will be vectorized using `vecx`) |
| `predicted.y` | array containing matrices of cross-validated predictions for `y` (landmarks arrays will be vectorized using `vecx`) |
| `rv` | RV-coefficient |
| `p.value.RV` | p-value for RV-coefficient determined by permutation testing |

## Author(s)

Stefan Schlager

## References

Rohlf FJ, Corti M. 2000. Use of two-block partial least-squares to study covariation in shape. Systematic Biology 49:740-753.

## See Also

`plsCoVar`, `getPLSfromScores`, `predictPLSfromScores`, `getPLSscores`, `predictPLSfromData`,`svd` , `plsCoVarCommonShape`, `getPLSCommonShape`

## Examples

```
if (require(shapes)) {
### very arbitrary test:
### check if first 4 landmarks covaries with the second 4
proc <- procSym(gorf.dat)
## we do only 50 rounds to minimize computation time
## Not run: #same.config takes too long for CRAN check
pls1 <- pls2B(proc$rotated[1:4,,],proc$rotated[5:8,,],
              same.config=TRUE,rounds=50,mc.cores=2)

## End(Not run)
pls1 <- pls2B(proc$rotated[1:4,,],proc$rotated[5:8,,],
              same.config=FALSE,rounds=50,mc.cores=1)
pls1
layout(matrix(1:4,2,2,byrow=TRUE))
for(i in 1:4)
 plot(pls1$Xscores[,i]~pls1$Yscores[,i])


## predict first 4 landmarks from second 4 for first config
layout(1)
predPLS <- predictPLSfromData(pls1,y=proc$rotated[5:8,,1])
## show differences between prediction and original
deformGrid2d(predPLS,proc$rotated[1:4,,1],pch=19)
##plot the complete first config
points(proc$rotated[,,1])

##show effects of first latent variable
plsEffects <- plsCoVar(pls1,i=1)
deformGrid2d(plsEffects$x[,,1],plsEffects$x[,,2])##show on x
deformGrid2d(plsEffects$y[,,1],plsEffects$y[,,2],add=TRUE,pch=19)##show on y

##show effects of 2nd latent variable
plsEffects2 <- plsCoVar(pls1,i=2)
deformGrid2d(plsEffects2$x[,,1],plsEffects2$x[,,2])##show on x
deformGrid2d(plsEffects2$y[,,1],plsEffects2$y[,,2],add=TRUE,pch=19)##show on y
}
```

## `plsCoVar` *Get the shape changes from pls2B associated with each latent variable*

### Description

Get the shape changes from pls2B associated with each latent variable

### Usage

```
plsCoVar(pls, i, sdx = 3, sdy = 3)
```

### Arguments

| | |
|---|---|
| `pls` | output of pls2B |
| `i` | integer: which latent variable to show. E.g. i=3 will show the changes associated with the 3rd latent variable. |
| `sdx` | standard deviation on the xscores. sdx=3 will show the effecs of -3sd vs +3sd |
| `sdy` | standard deviation on the yscores. sdy=3 will show the effecs of -3sd vs +3sd |

## Value

| | |
|---|---|
| `x` | matrix/array with reconstructed x |
| `y` | matrix/array with reconstructed y, with each prediction named accordingly: e.g. neg_x_sd_3 means the prediction of x at a score of `-3*sd(Xscores)` |

.

## See Also

`pls2B`, `getPLSfromScores`, `predictPLSfromScores`, `getPLSscores`, `predictPLSfromData`,`svd`, `plsCoVarCommonShape`

---

`plsCoVarCommonShape` *Compute the shape changes along the common axis of deformations*

---

## Description

Compute the shape changes between two blocks of 2D or 3D shape coordiantes along the common axis of deformations defined by each dimension of the latent space

## Usage

```
plsCoVarCommonShape(pls, i, sdcommon = 1)
```

## Arguments

| | |
|---|---|
| `pls` | object of class "pls2B" |
| `i` | integer: dimension of latent space to show shape changes for |
| `sdcommon` | standard deviations derived from scores scaled to a consensus scale |

## Value

returns an k x m x 2 array with the common shape changes associated with +-`sdcommon` SD of the `i-th` latent dimension

## Note

this give the same results as `plsCoVar`, however, using common shape vectors as suggested by Mitteroecker and Bookstein (2007)

## References

Mitteroecker P, Bookstein F. 2007. The conceptual and statistical relationship between modularity and morphological integration. Systematic Biology 56(5):818-836.

## See Also

pls2B, getPLSfromScores, predictPLSfromScores, getPLSscores, predictPLSfromData,svd, plsCoVar, getPLSCommonShape

## Examples

```
data(boneData)
proc <- procSym(boneLM)
pls <- pls2B(proc$orpdata[1:4,,],proc$orpdata[5:10,,])
commShape <- getPLSCommonShape(pls)
## get common shape for first latent dimension at +-2 sd of the scores
pred <- plsCoVarCommonShape(pls,1,2)
## Not run:
deformGrid3d(pred[,,1],pred[,,2])

## End(Not run)
```

---

`points2plane` *projects a 3D coordinate orthogonally onto a plane*

---

## Description

projects a 3D coordinate orthogonally onto a plane

## Usage

```
points2plane(x, v1, normal = NULL, v2 = NULL, v3 = NULL)
```

## Arguments

| | |
|---|---|
| `x` | 3D-vector or a k x 3 matrix with 3D vectors stored in rows |
| `v1` | point on plane |
| `normal` | plane normal (overrides specification by v2 and v3) |
| `v2` | if pNorm=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |
| `v3` | if pNorm=NULL, the plane will be defined by three points `v1`, `v2`, `v3` |

## Value

projected point

## Examples

```
data(boneData)
##project rhinion onto plane spanned by Nasion and both Nariales
rpro <- points2plane(boneLM[10,,1],v1=boneLM[9,,1],v2=boneLM[3,,1],v3=boneLM[4,,1])

## Not run:
require(rgl)
#visualize
wire3d(skull_0144_ch_fe.mesh,col="white")
##get plane normal
normal <- crossProduct(boneLM[3,,1]-boneLM[9,,1],boneLM[4,,1]-boneLM[9,,1])
#' ## get plane offset
d <- norm(points2plane(c(0,0,0),v1=boneLM[9,,1],normal=normal),"2")
spheres3d(boneLM[,,1],radius=0.5)
spheres3d(boneLM[c(3,4,9),,1],radius=0.6,col=3)
##original position of Rhinion
spheres3d(boneLM[10,,1],radius=0.6,col=2)
##projected onto plane
spheres3d(rpro,radius=0.9,col=6)
lines3d(rbind(rpro,boneLM[10,,1]),lwd=3)
##plot plane
planes3d(normal[1],normal[2],normal[3],d=d,col=2,alpha=0.5)

##now we project all points onto that plane:
spheres3d(points2plane(boneLM[,,1],v1=boneLM[9,,1],v2=boneLM[3,,1],v3=boneLM[4,,1]),col=3)

## and finally project the vertices of the mesh onto the plane
meshpro <- points2plane(vert2points(skull_0144_ch_fe.mesh),v1=boneLM[9,,1],normal=normal)
points3d(meshpro,col=2)

## End(Not run)
```

---

`prcompfast` *fast Principal Component Analysis (PCA)*

---

## Description

fast Principal Component Analysis (PCA)

## Usage

```
prcompfast(x, retx = TRUE, center = TRUE, scale. = FALSE, tol = NULL, ...)
```

## Arguments

| | |
|---|---|
| `x` | a numeric or complex matrix (or data frame) which provides the data for the principal components analysis. |
| `retx` | a logical value indicating whether the rotated variables should be returned |

| | |
|---|---|
| `center` | a logical value indicating whether the variables should be shifted to be zero centered. Alternately, a vector of length |
| `scale.` | a logical value indicating whether the variables should be scaled to have unit variance before the analysis takes place. The default is `FALSE` for consistency with S, but in general scaling is advisable. Alternatively, a vector of length equal the number of columns of `x` can be supplied. The value is passed to `scale`. equal the number of columns of `x` can be supplied. The value is passed to `scale`. |
| `tol` | a value indicating the magnitude below which components should be omitted. (Components are omitted if their standard deviations are less than or equal to `tol` times the standard deviation of the first component.) With the default null setting, no components are omitted. Other settings for tol could be `tol = 0` or `tol = sqrt(.Machine$double.eps)`, which would omit essentially constant components. |
| `...` | arguments passed to or from other methods. |

## Value

`prcomp` returns a list with class `prcomp` containing the followin components:

| | |
|---|---|
| `sdev` | the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix). |
| `rotation:` | the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). The function `princomp` returns this in the element `loadings`. |
| `x:` | if `retx` is true the value of the rotated data (the centred (and scaled if requested) data multiplied by the `rotation` matrix) is returned. Hence, `cov(x)` is the diagonal matrix `diag(sdev^2)`. For the formula method, `napredict()` is applied to handle the treatment of values omitted by the `na.action`. |
| `center`, `scale:` | the centering and scaling used, or `FALSE` |

. pcafast <- prcompfast(iris[,1:4]) pcadefault <- prcompfast(iris[,1:4]) ## check if both results are idential (ignoring the sign) all.equal(lapply(pcafast,abs),lapply(pcadefault,abs))

## Note

this function returns the same results as `prcomp` (apart from sign differences) but uses smarter matrix decompositions making it faster for nrow(x) » ncol(x) and nrow(x) « ncol(x).

---

## `predict.bgPCA` *Compute between-group-PC scores from new data*

---

### Description

Compute between-group-PC scores from new data

## Usage

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

## Arguments

| | |
|---|---|
| `object` | object of class bgPCA returned from `groupPCA` |
| `newdata` | matrix or 3D array containing data in the same format as originally used to compute groupPCA |
| `...` | currently not used. |

## Value

returns the between-group-PC scores for new data

## Examples

```
data(boneData)

boneLMPart <- boneLM[,,-(1:2)]
procPart <- procSym(boneLMPart)
pop_sex <- name2factor(boneLMPart, which=3:4)
## compute group PCA without first 2 specimens
gpcaPart <- groupPCA(procPart$orpdata, groups=pop_sex, rounds=0, mc.cores=2,cv=FALSE)
## align new data to Procrustes analysis
newdata <- align2procSym(procPart,boneLM[,,1:2])
## get scores for new data
newscores <- predict(gpcaPart,newdata)
```

---

`predict.CVA` *Compute CV-scores from new data*

---

## Description

Compute CV-scores from new data

## Usage

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

## Arguments

| | |
|---|---|
| `object` | object of class `CVA` |
| `newdata` | matrix or 3D array containing data in the same format as originally used to compute CVA |
| `...` | currently not used. |

## Value

returns the CV-scores for new data

---

## `predictPLSfromData` *predict 2 Block-PLS from new data*

---

### Description

predict 2 Block-PLS from new data

### Usage

```
predictPLSfromData(pls, x, y, ncomp = NULL)
```

### Arguments

| | |
|---|---|
| `pls` | output of pls2B |
| `x` | data in the same format as in original pls2B (for landmarks this can be an array or a matrix and for other data a matrix of a vector) |
| `y` | data in the same format as in original pls2B (for landmarks this can be an array or a matrix and for other data a matrix of a vector) |
| `ncomp` | number of (latent) components to use for prediction. |

### Value

returns an array/matrix/vector of predictions - depending on input for computing `pls`

### Note

either x or y must be missing

### See Also

`pls2B`, `getPLSscores`,`predictPLSfromScores`

### Examples

```
##see examples in pls2B
```

## `predictPLSfromScores` *predict data from 2-Block PLS-scores*

### Description

predict data from 2-Block PLS-scores

### Usage

```
predictPLSfromScores(pls, x, y)
```

### Arguments

| | |
|---|---|
| `pls` | output of pls2B |
| `x` | scores associated with dataset x in original pls2B |
| `y` | scores associated with dataset y in original pls2B |

### Value

returns an array/matrix of landmarks or original values, depending on input for computing `pls`

### Note

either x or y must be missing. If x-scores are provided, the yscores will be estimated and the predictions calculated.

### See Also

`pls2B`, `getPLSscores`,`predictPLSfromData`

## `predictRelWarps` *predict relative warps for data not included in the training data set*

### Description

predict relative warps for data not included in the training data set

### Usage

```
predictRelWarps(x, newdata, noalign = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | output from `relWarps` |
| `newdata` | k x m x n array holding new landmark data |
| `noalign` | logical: if TRUE, data is assumed to be already aligned to training data and alignment is skipped. |

### Details

This function aligns the new data to the mean from `x` and transforms it into the relative warp space computed from the training data.

### Value

returns a list containing

| | |
|---|---|
| `bescores` | relative warp scores (PC-scores if `alpha = 0`) |
| `uniscores` | uniform scores, NULL if `alpha = 0` |

### Examples

```
data(boneData)
set.seed(42)
training <- sample(1:80,size=60)
rW1 <- relWarps(boneLM[,,training], alpha = -1)
## predict scores for the entire sample
predAll <- predictRelWarps(rW1,boneLM)

## now compare the scores predicted scores to the original ones
layout(matrix(1:4,2,2))
for (i in 1:2) {
  plot(rW1$bescores[,i],predAll$bescores[training,i],main=paste("RW",i))
  plot(rW1$uniscores[,i],predAll$uniscores[training,i],main=paste("UC",i))
}
```

---

## `predictShape.lm` *Predict shapes based on linear models calculated from PCscores*

---

### Description

Predict shapes based on linear models calculated from PCscores.

### Usage

```
predictShape.lm(fit, datamod, PC, mshape)
```

### Arguments

| | |
|---|---|
| `fit` | model of class `lm` where the PCscores are fitted onto |
| `datamod` | a one-sided "model" formula, of the form `~ x1 + x2 + ... + xk`, corresponding to the right hand term in the model used in `fit`. If omitted, the predicted shapes of all specimen are calculated based on the fitted values. |
| `PC` | Matrix/vector containing Principal components (rotation matrix) corresponding to PC-scores used in `fit`. |
| `mshape` | matrix of the meanshape's landmarks by which the data was centered before rotation in covariance eigenspace. |

## Details

This function predicts the landmarks based on models calculated from PCscores.

## Value

| | |
|---|---|
| `predicted` | array or matrix containing predicted landmark coordinates |
| `predictedPC` | matrix containing predicted PC-scores |

## Warning

Make sure that the levels of the variables used in `datamod` correspond exactly to those used in `fit`. Otherwise model matrix will be calculated erroneous.

## See Also

`model.matrix`, `lm`, `formula`

## Examples

```
data(boneData)
proc <- procSym(boneLM)
pop <- name2factor(boneLM,which=3)
##easy model with only one factor based on the first four PCs
fit <- lm(proc$PCscores[,1:4] ~ pop)
## get shape for Europeans only
datamod <- ~as.factor(levels(pop))[2]
Eu <- predictShape.lm(fit,datamod, proc$PCs[,1:4],proc$mshape)

## get shape for Europeans and Chinese
datamod <- ~as.factor(levels(pop))
pred <- predictShape.lm(fit,datamod, proc$PCs[,1:4],proc$mshape)
## Not run:
deformGrid3d(pred$predicted[,,1], pred$predicted[,,2], ngrid = 0)

## End(Not run)

## more complicated model

sex <- name2factor(boneLM,which=4)
fit <- lm(proc$PCscores[,1:4] ~ pop*sex)
## predict female for chinese and European
datamod <- ~(as.factor(levels(pop))*rep(as.factor(levels(sex))[1],2))
pred <- predictShape.lm(fit,datamod, proc$PCs[,1:4],proc$mshape)

## predict female and malefor chinese and European
popmod <- factor(c(rep("eu",2),rep("ch",2)))
sexmod <- rep(as.factor(levels(sex)),2)
datamod <- ~(popmod*sexmod)
pred <- predictShape.lm(fit,datamod, proc$PCs[,1:4],proc$mshape)

## add some (randomly generated) numeric covariate
```

```
somevalue <- rnorm(80,sd=10)
fit <- lm(proc$PCscores[,1:4] ~ pop+somevalue)
probs <- quantile(somevalue, probs=c(0.05, 0.95))
## make model for European at 5% and 95% quantile
popmod <- rep(factor(levels(pop))[2],2)
datamod <- ~(popmod+probs)
pred <- predictShape.lm(fit,datamod, proc$PCs[,1:4],proc$mshape)
```

## proc.weight *calculate weights inverse to the distances from the specified observation.*

### Description

for calculation of a shape model by averaging the observations neighbouring the configuration in question, it is necessary to calculate weights by similarity.

### Usage

```
proc.weight(
  data,
  number,
  ref,
  report = TRUE,
  reg = 0,
  log = FALSE,
  mahalanobis = FALSE,
  weightfun = NULL
)
```

### Arguments

| | |
|---|---|
| `data` | array containing landmark configurations |
| `number` | integer: how many of the neighbours are to be involved. |
| `ref` | integer: position in the array that is used as reference. |
| `report` | logical: require report about name of the reference. |
| `reg` | numeric: regularise mahalanobis distance by adding reg to the diagonal of eigenvalues of the covariance matrix. |
| `log` | logical: use the logarithm of the distances. |
| `mahalanobis` | logical: use mahalanobis distance. |
| `weightfun` | custom function that operates on a vector of distances (see examples) and generates weights accordingly. |

## Details

distances of zero will get a weight of 1e12 (this is scaled to all weights summing to one), thus weights for observations further away are converging to zero.

## Value

| | |
|---|---|
| `data` | dataframe containing id, procrustes/mahalanobis distance and weight according to the reference |
| `reference` | returns observations' names if available |
| `rho.all` | dataframe containing distances to references of all observations |

## Examples

```
if (require(shapes)) {
proc <- procSym(gorf.dat)
##get weights for the the four specimen closest to the first observation.
weights <- proc.weight(proc$rotated,4,1)

##estimate the first specimen by weighted neighbour shapes.
estim <- proc$mshape*0;
for (i in 1:4)
{estim <-estim+proc$rotated[,,weights$data$nr[i]]*weights$data$weight[i]}

### visualise
plot(estim,asp=1)## show estimation
points(proc$rotated[,,1],col=3)##show original

## use a gaussian smoother to compute weights using a bandwidth of 0.05
gaussWeight <- function(r,sigma=0.05) {
   sigma <- 2*sigma^2
   return(exp(-r^2/ sigma))
}
weights <- proc.weight(proc$rotated,4,1,weightfun=gaussWeight)
}
```

---

`procAOVsym` *Procrustes ANOVA for structures with object symmetry*

---

## Description

Procrustes ANOVA for structures with object symmetry, currently only supporting the factors 'specimen', 'side' and the interaction term.

## Usage

```
procAOVsym(symproc, indnames = NULL)
```

## Arguments

| | |
|---|---|
| `symproc` | object returned by `procSym`, where `pairedLM` is specified |
| `indnames` | vector containing specimen identifiers. Only necessary, if data does not contain dimnames containing identifiers |

## Details

performs a Procrustes ANOVA for configurations with object symmetry (as described in Klingenberg et al. 2002).

## Value

returns a dataframe containing Sums of Squares for each factor.

## Note

In future releases the implementation of support for bilateral symmetry and more factors is intended.

## Author(s)

Stefan Schlager

## References

Klingenberg CP, Barluenga M, Meyer A. 2002. Shape analysis of symmetric structures: quantifying variation among individuals and asymmetry. Evolution 56:1909-20.

## See Also

`procSym`

## Examples

```
data(boneData)
left <- c(4,6,8)
## determine corresponding Landmarks on the right side:
# important: keep same order
right <- c(3,5,7)
pairedLM <- cbind(left,right)
symproc <- procSym(boneLM, pairedLM=pairedLM)
procAOVsym(symproc)
```

## ProcGPA *Workhorse function for procSym, responsible for Procrustes registration*

### Description

Workhorse function for procSym, responsible for Procrustes registration

### Usage

```
ProcGPA(
  dat.array,
  tol = 1e-05,
  scale = TRUE,
  CSinit = FALSE,
  silent = TRUE,
  weights = NULL,
  centerweight = FALSE,
  reflection = TRUE,
  pcAlign = TRUE
)
```

### Arguments

| | |
|---|---|
| `dat.array` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. |
| `tol` | numeric: Threshold for convergence during iterative superimpositioning. |
| `scale` | logical: indicating if scaling is requested |
| `CSinit` | logical: if TRUE, all configurations are initially scaled to Unit Centroid Size. |
| `silent` | logical: suppress output of elapsed time. |
| `weights` | numeric vector: assign per landmark weights. |
| `centerweight` | logical: if TRUE, the landmark configuration is scaled according to weights during the rotation process, instead of being scaled to the Centroid size. |
| `reflection` | logical: allow reflections. |
| `pcAlign` | logical: if TRUE, the shapes are aligned by the principal axis of the first specimen, otherwise the orientation of the first specimen is used. |

### Value

returns a list with

| | |
|---|---|
| `rotated` | k x m x n array of the rotated configurations |
| `mshape` | sample meanshape |

## Author(s)

Stefan Schlager

## References

Goodall C. 1991. Procrustes methods in the statistical analysis of shape. Journal of the Royal Statistical Society. Series B. Statistical Methodology 53:285-239.

Dryden IL, Mardia KV. 1998. Statistical shape analysis. John Wiley and sons, Chichester.

## See Also

`procSym`, `rotonto`

## Examples

```
data(boneData)
proc <- ProcGPA(boneLM, CSinit=TRUE, silent=TRUE)
#now we landmarks 5 - 9 double the weight as  the others
weights <- c(rep(1,4),rep(2,5),1)
proc.wt <- ProcGPA(boneLM, CSinit=TRUE, weights=weights, silent=TRUE)
```

---

## procSym *Procrustes registration*

---

## Description

`procSym` performs Procrustes superimposition including sliding of semi-landmarks on curves/outlines in 2D and 3D.

## Usage

```
procSym(
  dataarray,
  scale = TRUE,
  reflect = TRUE,
  CSinit = TRUE,
  orp = TRUE,
  proctol = 1e-05,
  tol = 1e-05,
  pairedLM = NULL,
  sizeshape = FALSE,
  use.lm = NULL,
  center.part = FALSE,
  weights = NULL,
  centerweight = FALSE,
  pcAlign = TRUE,
  distfun = c("angle", "riemann"),
  SMvector = NULL,
  outlines = NULL,
  deselect = FALSE,
  recursive = TRUE,
  iterations = 0,
  initproc = FALSE,
  bending = TRUE,
  stepsize = 1
)
```

## Arguments

| | |
|---|---|
| `dataarray` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. |
| `scale` | logical: indicating if scaling is requested to minimize the General Procrustes distance. To avoid all scaling, one has to set `CSinit=FALSE`, too. |
| `reflect` | logical: allow reflections. |
| `CSinit` | logical: if TRUE, all configurations are initially scaled to Unit Centroid Size. |
| `orp` | logical: if TRUE, an orthogonal projection at the meanshape into tangent space is performed. |
| `proctol` | numeric: Threshold for convergence in the alignment process |
| `tol` | numeric: Threshold for convergence in the sliding process |
| `pairedLM` | A X x 2 matrix containing the indices (rownumbers) of the paired LM. E.g. the left column contains the lefthand landmarks, while the right side contains the corresponding right hand landmarks. |
| `sizeshape` | Logical: if TRUE, a log transformed variable of Centroid Size will be added to the shapedata as first variable before performing the PCA. |
| `use.lm` | vector of integers to define a subset of landmarks to be used for Procrustes registration. |
| `center.part` | Logical: if TRUE, the data superimposed by the subset defined by use.lm will be centered according to the centroid of the complete configuration. Otherwise orp will be set to FALSE to avoid erroneous projection into tangent space. |
| `weights` | numeric vector: assign per landmark weights. |
| `centerweight` | logical: if TRUE, the landmark configuration is scaled according to weights during the rotation process, instead of being scaled to the Centroid size. |
| `pcAlign` | logical: if TRUE, the shapes are aligned by the principal axis of the first specimen |
| `distfun` | character: "riemann" requests a Riemannian distance for calculating distances to mean, while "angle" uses an approximation by calculating the angle between rotated shapes on the unit sphere. |
| `SMvector` | A vector containing the landmarks on the curve(s) that are allowed to slide |
| `outlines` | A vector (or if threre are several curves) a list of vectors (containing the rowindices) of the (Semi-)landmarks forming the curve(s) in the successive position on the curve - including the beginning and end points, that are not allowed to slide. |

| | |
|---|---|
| `deselect` | Logical: if TRUE, the SMvector is interpreted as those landmarks, that are not allowed to slide. |
| `recursive` | Logical: if TRUE, during the iterations of the sliding process, the outcome of the previous iteration will be used. Otherwise the original configuration will be used in all iterations. |
| `iterations` | integer: select manually how many iterations will be performed during the sliding process (usefull, when there is very slow convergence). 0 means iteration until convergence. |
| `initproc` | Logical: indicating if the first Relaxation step is performed against the mean of an initial Procrustes superimposition. Symmetric configurations will be relaxed against a perfectly symmetrical mean. |
| `bending` | if TRUE, bending energy will be minimized, Procrustes distance otherwise (not suggested with large shape differences) |
| `stepsize` | integer: dampening factor for the sliding. Useful to keep semi-landmarks from sliding too far off the surface. The displacement is calculated as `stepsize * displacement`. |

## Details

This function performs Procrustes registration, allowing a variety of options, including scaling, orthogonal projection into tangentspace and relaxation of semi-landmarks on curves (without reprojection onto the surface/actual outline). It also allows the superimpositioning to be performed using only a subset of the available landmark. For taking into account object symmetry, `pairedLM` needs to be set. This generates an object of class `"symproc"`. Otherwise an object of class `"nosymproc"`.

## Value

| | |
|---|---|
| `size` | a vector containing the Centroid Size of the configurations |
| `rotated` | k x m x n array of the rotated configurations |
| `Sym` | k x m x n array of the Symmetrical component - only available for the "Symmetry"-Option (when pairedLM is defined) |
| `Asym` | k x m x n array of the Asymmetrical component. It contains the per-landmark asymmetric displacement for each specimen. Only available for the "Symmetry"-Option (when pairedLM is defined) |
| `asymmean` | k x m matrix of mean asymmetric deviation from symmetric mean |
| `mshape` | sample meanshape |
| `symmean` | meanshape of symmetrized configurations |
| `tan` | if orp=TRUE: Residuals in tangentspace else, Procrustes residuals - only available without the "Symmetrie"-Option |
| `PCs` | Principal Components - if sizeshape=TRUE, the first variable of the PCs is size information (as log transformed Centroid Size) |
| `PCsym` | Principal Components of the Symmetrical Component |
| `PCasym` | Principal Components of the Asymmetrical Component |
| `PCscores` | PC scores |

| | |
|---|---|
| `PCscore_sym` | PC scores of the Symmetrical Component |
| `PCscore_asym` | PC scores of the Asymmetrical Component |
| `eigenvalues` | eigenvalues of the Covariance matrix |
| `eigensym` | eigenvalues of the "Symmetrical" Covariance matrix |
| `eigenasym` | eigenvalues of the "Asymmetrical" Covariance matrix |
| `Variance` | Table of the explained Variance by the PCs |
| `SymVar` | Table of the explained "Symmetrical" Variance by the PCs |
| `AsymVar` | Table of the explained "Asymmetrical" Variance by the PCs |
| `orpdata` | k x m x n array of the rotated configurations projected into tangent space |
| `rho` | vector of Riemannian distance from the mean |
| `dataslide` | array containing slidden Landmarks in the original space - not yet processed by a Procrustes analysis. Only available if a sliding process was requested |
| `meanlogCS` | mean log-transformed centroid size |

## Note

For processing of surface landmarks or including the reprojection of slid landmarks back onto 3D-surface representations, the usage of `slider3d` is recommended.

## Author(s)

Stefan Schlager

## References

Dryden IL, and Mardia KV. 1998. Statistical shape analysis. Chichester.

Klingenberg CP, Barluenga M, and Meyer A. 2002. Shape analysis of symmetric structures: quantifying variation among individuals and asymmetry. Evolution 56(10):1909-1920.

Gunz, P., P. Mitteroecker, and F. L. Bookstein. 2005. Semilandmarks in Three Dimensions, in Modern Morphometrics in Physical Anthropology. Edited by D. E. Slice, pp. 73-98. New York: Kluwer Academic/Plenum Publishers.

## See Also

`slider3d`

## Examples

```
require(rgl)
data(boneData)

### do an analysis of symmetric landmarks
## visualize landmarks on surface
## Not run:
 spheres3d(boneLM[,,1])
wire3d(skull_0144_ch_fe.mesh,col=3)
```

```
## get landmark numbers
text3d(boneLM[,,1],text=paste(1:10),adj = 1, cex=3)

## End(Not run)
## determine paired Landmarks left side:
left <- c(4,6,8)
## determine corresponding Landmarks on the right side:
# important: keep same order
right <- c(3,5,7)
pairedLM <- cbind(left,right)
symproc <- procSym(boneLM, pairedLM=pairedLM)
## Not run:
## visualize first 3 PCs of symmetric shape
pcaplot3d(symproc, sym=TRUE)
## visualize first 3 PCs of asymmetric shape
pcaplot3d(symproc, sym=FALSE)

## visualze distribution of symmetric PCscores population
pop <- name2factor(boneLM, which=3)
if (require(car)) {
spm(~symproc$PCscore_sym[,1:5], groups=pop)
## visualze distribution of asymmetric PCscores population
spm(~symproc$PCscore_asym[,1:5], groups=pop)
}

## End(Not run)
```

## projRead *Project points onto the closest point on a mesh*

### Description

project points onto a given surface and return projected points and normals.

### Usage

```
projRead(lm, mesh, readnormals = TRUE, smooth = FALSE, sign = TRUE, ...)
```

### Arguments

| | |
|---|---|
| `lm` | m x 3 matrix containing 3D coordinates. |
| `mesh` | character: specify path to mesh file. |
| `readnormals` | logical: return normals of projected points. |
| `smooth` | logical: rerturn smoothed normals. |
| `sign` | logical: request signed distances. |
| `...` | additional arguments currently not used. |

## Value

if readnormals = FALSE, a m x 3 matrix containing projected points is returned, otherwise a list, where

| | |
|---|---|
| `vb` | 3 x m matrix containing projected points |
| `normals` | 3 x m matrix containing normals |
| `quality` | vector containing distances |

## Author(s)

Stefan Schlager

## References

Detection of inside/outside uses the algorithm proposed in:

Baerentzen, Jakob Andreas. & Aanaes, H., 2002. Generating Signed Distance Fields From Triangle Meshes. Informatics and Mathematical Modelling.

## See Also

`closemeshKD`

## Examples

```
data(nose)
## Not run:
repro <- projRead(shortnose.lm,shortnose.mesh)

## End(Not run)
```

---

`qqmat` *Q-Q plot to assess normality of data*

---

## Description

qqmat plots Mahalanobisdistances of a given sample against those expected from a Gaussian distribution

## Usage

```
qqmat(x, output = FALSE, square = FALSE)
```

## Arguments

| | |
|---|---|
| `x` | sample data: matrix or vector |
| `output` | logical: if TRUE results are returned |
| `square` | plot in a square window - outliers might be cut off. |

## Value

if `output=TRUE`, the following values are returned

| | |
|---|---|
| `x` | distances from an expected Gaussian distribution |
| `y` | observed distances - sorted |
| `d` | observed distances - unsorted |

## Author(s)

Stefan Schlager

## See Also

`qqplot`

## Examples

```
require(MASS)
### create normally distributed data
data <- mvrnorm(100,mu=rep(0,5),Sigma = diag(5:1))
qqmat(data)

###create non normally distributed data
data1 <- rchisq(100,df=3)
qqmat(data1,square=FALSE)
```

---

## `quad2trimesh` *converts a mesh containing quadrangular faces into one only consisting of triangles*

---

## Description

converts a mesh containing quadrangular faces into one only consisting of triangles

## Usage

```
quad2trimesh(mesh, updateNormals = TRUE)
```

### Arguments

| | |
|---|---|
| `mesh` | object of class "mesh3d" |
| `updateNormals` | logical: request recalculation of (angle weighted) vertex normals. |

### Value

triangular mesh with updated normals

### Examples

```
Sigma <- diag(3:1) #create a 3D-covariance matrix
require(rgl)
quadmesh <- ellipse3d(Sigma)##create quadmesh
trimesh <- quad2trimesh(quadmesh)# convert to trimesh
```

---

## `r2morphoj` *Export data to MorphoJ and Morphologika*

---

### Description

Export data to MorphoJ and Morphologika

### Usage

```
r2morphoj(x, file, id.string = NULL)

r2morphologika(x, file = file, labels = NULL, labelname = NULL, ...)
```

### Arguments

| | |
|---|---|
| `x` | 3-dimensionla array containing landmark data. E.g. the input/output from `procSym`. |
| `file` | character: name the output file |
| `id.string` | a string with ids or factors to append |
| `labels` | character vector specify labels to create for Morphologika |
| `labelname` | character: name the labels for Morphologika. |
| `...` | unused at the moment |

### Details

Export data to MorphoJ and Morphologika

### Examples

```
if (require(shapes)) {
r2morphoj(gorf.dat,file="gorf.dat")

data <- bindArr(gorf.dat, gorm.dat, along=3)
datalabels <- c(rep("female",dim(gorf.dat)[3]),
rep("male",dim(gorm.dat)[3]))
labelname <- "sex"
r2morphologika(data, labels=datalabels, labelname= labelname, file="data.dat")
## cleanup
unlink(c("gorf.dat","data.dat"))
}
```

---

## `ray2mesh` *projects the vertices of a mesh along its normals onto the surface of another one.*

---

### Description

projects the vertices of a mesh onto the surface of another one by searching for the closest point along vertex normals on the target by for each vertex.

### Usage

```
ray2mesh(mesh1, tarmesh, tol = 1e+12, inbound = FALSE, mindist = FALSE, ...)
```

### Arguments

| | |
|---|---|
| `mesh1` | mesh to project. Can be an object of class "mesh3d" or path to an external mesh file (ply, obj, stl). |
| `tarmesh` | mesh to project onto. Can be an object of class "mesh3d" or path to an external mesh file (ply, obj, stl). |
| `tol` | numeric: maximum distance to search along ray, closest Euclidean distance will be used, if tol is exceeded. |
| `inbound` | inverse search direction along rays. |
| `mindist` | search both ways (ray and -ray) and select closest point. |
| `...` | additional arguments not used at the moment. |

### Value

returns projected mesh with additional list entries:

| | |
|---|---|
| `quality` | integer vector containing a value for each vertex of `x`: 1 indicates that a ray has intersected 'tarmesh' within the given threshold, while 0 means not |
| `distance` | numeric vector: distances to intersection |

### Author(s)

Stefan Schlager

### See Also

ply2mesh, closemeshKD

---

## read.csv.folder *batch import data from files*

---

### Description

imports all data files contained in a specified folder.

### Usage

```
read.csv.folder(
  folder,
  x,
  y = 2:4,
  rownames = NULL,
  header = TRUE,
  dec = ".",
  sep = ";",
  pattern = "csv",
  addSpec = NULL,
  back = TRUE
)
```

### Arguments

| | |
|---|---|
| folder | character: path to folder |
| x | either a vector specifiing which rows are to be imported, or character vector containing variable names to be sought for. |
| y | a vector specifiing, which columns of the speradsheet ist to be imported. |
| rownames | integer: specifies columns, where variable names are stored. |
| header | logical : if spreadsheet contains header-row. |
| dec | character: defines decimal sepearator. |
| sep | character: defines column seperator. |
| pattern | character: specify file format (e.g. csv). |
| addSpec | character: add a custom specifier to the dimnames of the array. |
| back | logical: where to place the specifier. |

## Value

|  |  |
|---|---|
| `arr` | array containing imported data |
| `NAs` | vector containing position of observations with NAs |
| `NA.list` | list: containing vectors containing information which LMs are missing in which observation |

## Author(s)

Stefan Schlager

## See Also

`read.table`

---

`read.fcsv` *read fiducials from slicer4*

---

## Description

read fiducials from slicer4

## Usage

```
read.fcsv(x, na = NULL, lps2ras = FALSE)
```

## Arguments

|  |  |
|---|---|
| `x` | filename |
| `na` | value to be replaced by NA |
| `lps2ras` | logical: if the coordinate system is LPS and `lps2ras=TRUE`, the data will be rotated into the RAS space by inverting the first two dimensions using `LPS2RAS`. |

## Value

a k x 3 matrix with landmarks

## `read.lmdta` *read dta files*

### Description

reads .dta files created by the software Landmark http://graphics.idav.ucdavis.edu/research/EvoMorph

### Usage

```
read.lmdta(file = "x", na = 9999)
```

### Arguments

| | |
|---|---|
| `file` | a dta file |
| `na` | specifies a value that indicates missing values |

### Value

| | |
|---|---|
| `arr` | array containing landmarks dimnames will be Information of ID and landmark names specified in Landmark |
| `info` | Information extracted from the header of the dta file |
| `idnames` | character vector containing the names of the individuals as specified in the dta file |

## `read.mpp` *Read saved pick-points from meshlab*

### Description

imports pick points selected with meshlab

### Usage

```
read.mpp(file, info = FALSE)
```

### Arguments

| | |
|---|---|
| `file` | file to import |
| `info` | logical: if TRUE, addtional infos are returned |

## Value

if `info=FALSE`:

a matrix containing picked-points coordinates (only those tagged as active).

if `info=TRUE`: a list containing

| | |
|---|---|
| `data` | matrix containing coordinates - including points tagged as inactive |
| `info` | additional info contained in file. |

## Author(s)

Stefan Schlager

## See Also

`read.pts`

---

`read.pts` *reads pts files*

---

## Description

reads Landmark data exported from the software Landmark from http://graphics.idav.ucdavis.edu/research/EvoMorph

## Usage

```
read.pts(file = "x", na = 9999)
```

## Arguments

| | |
|---|---|
| `file` | pts file |
| `na` | specifies a value that indicates missing values |

## Value

| | |
|---|---|
| `matrix` | matrix containing landmark information rownames will be the names given to the landmarks in Landmark |

## See Also

`read.pts`

## Examples

```
data(nose)
write.pts(shortnose.lm, filename="shortnose")
data <- read.pts("shortnose.pts")
```

## `read.slicerjson` *read Landmarks from Slicer in Json format*

### Description

read Landmarks from Slicer in Json format

### Usage

```
read.slicerjson(x, lps2ras = FALSE, na = NULL)
```

### Arguments

| | |
|---|---|
| `x` | path to json file |
| `lps2ras` | logical: if the coordinate system is LPS and `lps2ras=TRUE`, the data will be rotated into the RAS space by inverting the first two dimensions using `LPS2RAS`. |
| `na` | value to be replaced by NA |

### Value

returns matrix or list of matrices with imported landmark coordinates

## `readallTPS` *Import landmarks and outlines from TPS files*

### Description

Imports outlines and landmarks from files generated by tpsdig2

### Usage

```
readallTPS(file, scale = TRUE)
```

### Arguments

| | |
|---|---|
| `file` | A TPS-file generated by tpsdig2 |
| `scale` | logical: if TRUE the data will be scaled according to the SCALE entry. |

### Value

| | |
|---|---|
| `ID` | Specimen IDs read from TPS file |
| `LM` | list of landmarks contained in the TPS-file |
| `outlines` | a list containing sublists for each specimen with all the outlines read from TPS file |
| `SCALE` | vector containing the scale factors for each landmark config. |

## Note

currently only landmarks, ID and outlines are read from the TPS-file

## Author(s)

Stefan Schlager

## References

http://life.bio.sunysb.edu/ee/rohlf/software.html

## See Also

`read.lmdta`, `read.pts`

---

## `readLandmarks.csv` *import landmark data from csv files*

---

### Description

import landmark data from csv files

### Usage

```
readLandmarks.csv(
  file,
  x,
  y = 2:4,
  rownames = NULL,
  header = TRUE,
  dec = ".",
  sep = ";"
)
```

### Arguments

| | |
|---|---|
| `file` | character: path to file containing landmark data. |
| `x` | either a vector specifiing which rows are to be imported, or character vector containing variable names to be sought for. |
| `y` | a vector specifiing, which columns of the speradsheet ist to be imported. |
| `rownames` | integer: specifies columns, where variable names are stored. |
| `header` | logical : if spreadsheet contains header-row. |
| `dec` | character: defines decimal sepearator. |
| `sep` | character: defines column seperator. |

## Value

| | |
|---|---|
| `LM` | matrix containing imported data |
| `NAs` | vector containing rows containing NAs |

## Author(s)

Stefan Schlager

## See Also

`read.table`

---

`regdist` *correlation between shape space and tangent space*

---

## Description

performs a partial Procrustes superimposition of landmark data and calculates the correlation between tangent and shape space.

## Usage

```
regdist(
  dataarray,
  plot = TRUE,
  main = "",
  rho = "angle",
  dist.mat.out = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `dataarray` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. |
| `plot` | Logical: whether to plot the distances between observations. |
| `main` | character string: Title of the plot. |
| `rho` | chose how to calculate distances in shape space. Options: "riemdist"=Riemannian distance (function from the shapes package-takes along time to calculate), "angle"=calculates the angle between shape vectors, "sindist"=sinus of length of residual vector between shape vectors. |
| `dist.mat.out` | Logical: If TRUE, output will contain distance matrices. |
| `...` | additional parameters passed to `procSym` |

## Value

| | |
|---|---|
| `cor` | correlation coefficient between distances in shape space and tangent space |
| `procSS` | Procrustes Sums of Squares (of full procrustes distance) |
| `tanSS` | Tangent Sums of Squares |
| `rhoSS` | Procrustes Sums of Squares (of angle) |
| `euc.dist` | distance matrix of euclidean distance in Tangent space |
| `proc.dist` | distance matrix of Procrustes distance in Shape space |
| `lm` | linear model regressing tangent space distances onto Procrustes distances |

## Author(s)

Stefan Schlager

## See Also

`regdist`

## Examples

```
if (require(shapes)) {
regdist(gorf.dat)
}
```

---

## `RegScore` *calulate regression scores for linear model*

---

## Description

calulate regression scores for linear model as specified in Drake & Klingenberg(2008)

## Usage

```
RegScore(model, x = NULL)
```

## Arguments

| | |
|---|---|
| `model` | linear model |
| `x` | optional: matrix containing fitted data to be projected onto the regression lines. If omitted the model's fitted values will be used. |

## Details

the data are orthogonally projected onto the regression lines associated with each factor.

## Value

returns a n x m matrix containing the regression scores for each specimen.

## Warning

if `model` contains factors with more than 2 levels, R calculates one regression line per 2 factors. Check the `colnames` of the returned matrix to select the appropriate one. See examples for details.

## References

Drake, AG. & Klingenberg, CP. The pace of morphological change: historical transformation of skull shape in St Bernard dogs. Proceedings of the Royal Society B: Biological Sciences, The Royal Society, 2008, 275, 71-76.

## Examples

```
model <- lm(as.matrix(iris[,1:3]) ~ iris[,4])
rs <- RegScore(model)
plot(rs,iris[,4])

##now use a random subsample for model fitting
rand <- sample(nrow(iris))
x <- iris[rand[1:100],4]
newmod <- lm(as.matrix(iris[rand[1:100],1:3]) ~ x)
##predict the rest of data and get their regression scores
rsPred <- RegScore(newmod,as.matrix(iris[rand[101:150],1:3]))
plot(rsPred,iris[rand[101:150],4])
## Not run:
data(boneData)
proc <- procSym(boneLM)
pop.sex <- name2factor(boneLM,which=3:4) # generate a factor with 4 levels
lm.ps.size <- lm(proc$PCscores ~ pop.sex+proc$size)
rs <- RegScore(lm.ps.size)
colnames(rs) # in this case, the last column contains the regression
# scores associated with proc$size
## validate by using a subsample for fitting
rand <- sample(dim(boneLM)[3])
lm.ps.size0 <- lm(proc$PCscores[rand[1:50],] ~ proc$size[rand[1:50]])
rs0 <- RegScore(lm.ps.size0,proc$PCscores[rand[-c(1:50)],] )
plot(rs0,proc$size[rand[-c(1:50)]])

## End(Not run)
```

---

`relaxLM` *relax one specific landmark configuration against a reference*

---

## Description

relax one specific landmark configuration against a reference (e.g. a sample mean)

## Usage

```
relaxLM(lm, ...)

## S3 method for class 'matrix'
relaxLM(
  lm,
  reference,
  SMvector,
  outlines = NULL,
  surp = NULL,
  sur.name = NULL,
  mesh = NULL,
  tol = 1e-05,
  deselect = FALSE,
  inc.check = TRUE,
  iterations = 0,
  fixRepro = TRUE,
  missing = NULL,
  bending = TRUE,
  stepsize = ifelse(bending, 1, 0.5),
  use.lm = NULL,
  silent = FALSE,
  ...
)

## S3 method for class 'mesh3d'
relaxLM(
  lm,
  reference,
  tol = 1e-05,
  deselect = FALSE,
  inc.check = TRUE,
  iterations = 0,
  fixRepro = TRUE,
  missing = NULL,
  bending = FALSE,
  stepsize = ifelse(bending, 1, 0.5),
  use.lm = NULL,
  silent = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `lm` | k x 3 or k x 2 matrix containing landmark data to be slidden - or a triangular mesh of class "mesh3d". See details |
| `...` | additonal arguments - currently unused |

| | |
|---|---|
| `reference` | k x 3 or k x 2 matrix containing landmark of the reference, or a mesh with the same amount of vertices as there are landmarks in `lm`. |
| `SMvector` | A vector containing the row indices of (semi-) landmarks on the curve(s) that are allowed to slide |
| `outlines` | A vector (or if threre are several curves) a list of vectors (containing the rowindices) of the (Semi-)landmarks forming the curve(s) in the successive position on the curve - including the beginning and end points, that are not allowed to slide. |
| `surp` | integer vector containing the row indices of semi-landmarks positioned on surfaces. |
| `sur.name` | character: containing the filename of the corresponding surface.When specified, mesh has to be NULL. If `sur.name=NULL` and `mesh=NULL`, the tangent planes will be estimated from the point cloud. |
| `mesh` | triangular mesh of class "mesh3d" loaded into the R workspace, when specified, "sur.name" has to be NULL. |
| `tol` | numeric: Threshold for convergence in the sliding proces. Full Procrustes distance between actual result and previous iteration. |
| `deselect` | Logical: if TRUE, the SMvector is interpreted as those landmarks, that are not allowed to slide. |
| `inc.check` | Logical: if TRUE, the program stops when convergence criterion starts increasing and reports result from last iteration. |
| `iterations` | integer: maximum amounts the algorithm runs - even when 'tol' is not reached. When iterations=0, the algorithm runs until convergence. |
| `fixRepro` | logical: if `TRUE`, fix landmarks will also be projected onto the surface. If you have landmarks not on the surface, select `fixRepro=FALSE` |
| `missing` | vector of integers, specifying row indices of missing (semi-)landmarks. They will be relaxed freely in 3D and not projected onto the target (works only for 2D data). |
| `bending` | if TRUE, bending energy will be minimized, Procrustes distance otherwise (not suggested with large shape differences) |
| `stepsize` | integer: dampening factor for the amount of sliding. Useful to keep semi-landmarks from sliding too far off the surface. The displacement is calculated as $\Upsilon = \Upsilon^0 + stepsize * UT$. Default is set to 1 for bending=TRUE and 0.5 for bending=FALSE. |
| `use.lm` | indices specifying a subset of (semi-)landmarks to be used in the rotation step - only used if `bending=FALSE`. |
| `silent` | logical: if TRUE, console output is suppressed. |

## Details

if `lm` is a surface mesh, all vertices will be treated as semilandmarks and a allowed to freely slide along the surface.

## Value

returns kx3 matrix of slidden landmarks

## Author(s)

Stefan Schlager

## References

Gunz, P., P. Mitteroecker, and F. L. Bookstein. 2005. Semilandmarks in Three Dimensions, in Modern Morphometrics in Physical Anthropology. Edited by D. E. Slice, pp. 73-98. New York: Kluwer Academic/Plenum Publishers.

## See Also

slider3d

## Examples

```
require(rgl)
data(nose)
### relax shornose against longnose

# define fix landmarks
fix <- c(1:5,20:21)
# define surface patch by specifying row indices of matrices
# all except those defined as fix
surp <- c(1:dim(shortnose.lm)[1])[-fix]

relax <- relaxLM(shortnose.lm,
         longnose.lm, mesh=shortnose.mesh, iterations=1,
         SMvector=fix, deselect=TRUE, surp=surp)

## example minimizing Procrustes distance when displacement is not
## dampened by stepsize
relaxProcD <- relaxLM(shortnose.lm,
         longnose.lm, mesh=shortnose.mesh, iterations=1,
         SMvector=fix, deselect=TRUE, surp=c(1:623)[-fix],bending=FALSE,stepsize=1)

## Not run:
# visualize differences red=before and green=after sliding
deformGrid3d(shortnose.lm, relax, ngrid=0)


# visualize differences minimizing Procrusted distances red=before and green=after sliding

deformGrid3d(shortnose.lm, relaxProcD, ngrid=0)
## no smooth displacement, now let's check the distances:
rot2ref <- rotonto(relaxProcD,longnose.lm)
angle.calc(rot2ref$X,rot2ref$Y)
# 0.2492027 Procrustes distance between reference and slided shape
# (minimizing Procrustes distance)
rot2refBend <- rotonto(relax,longnose.lm)
angle.calc(rot2refBend$X,rot2refBend$Y)
# 0.2861322 Procrustes distance between reference and slided shape
# (minimizing bending energy)
```

```
rot2refOrig <- rotonto(shortnose.lm,longnose.lm)
angle.calc(rot2refOrig$X,rot2refOrig$Y)
# 0.3014957 Procrustes distance between reference and original shape
##result: while minimizing Procrustes distance, displacement is not
##guaranteed to be smooth

# add surface
wire3d(shortnose.mesh, col="white")


## finally relax two meshes with corresponding vertices:

mediumnose.mesh <- tps3d(shortnose.mesh,shortnose.lm, (shortnose.lm+longnose.lm)/2,threads=1)
## we use Procrustes distance as criterion as bending energy is pretty slow because
## of too many coordinates (more than 3000 is very unreasonable).
relaxMesh <- relaxLM(shortnose.mesh,mediumnose.mesh,iterations=2,bending=FALSE,stepsize=0.05)

## End(Not run)
```

## relWarps *calculate relative Warp analysis*

### Description

After Procrustes registration the data is scaled by the bending energy or its inverse to emphasize global/local differences when exploring a sample's shape.

### Usage

```
relWarps(
  data,
  scale = TRUE,
  CSinit = TRUE,
  alpha = 1,
  tol = 1e-10,
  orp = TRUE,
  pcAlign = TRUE,
  computeBasis = TRUE,
  noalign = FALSE
)
```

### Arguments

| | |
|---|---|
| `data` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. |
| `scale` | Logical: indicating if scaling is requested |
| `CSinit` | Logical: if TRUE, all configurations are initially scaled to Unit Centroid Size. |

| | |
|---|---|
| `alpha` | integer: power of the bending energy matrix. If alpha = 0 then standard Procrustes PCA is carried out. If alpha = 1 then large scale differences are emphasized, if alpha = -1 then small scale variations are emphasised. |
| `tol` | tolerance for the eigenvalues of the bending energy matrix to be zero |
| `orp` | logical: request orthogonal projection into tangent space. |
| `pcAlign` | logical: if TRUE, the shapes are aligned by the principal axis of the first specimen |
| `computeBasis` | logical: whether to compute the basis of the resulting vector space (takes a lot of memory and time for configurations with > 1000 coordinates. |
| `noalign` | logical: if TRUE, data is assumed to be already aligned and alignment and orthogonal projection are skipped. |

## Value

| | |
|---|---|
| `bescores` | relative warp scores (PC-scores if `alpha = 0`) |
| `uniscores` | uniform scores, NULL if `alpha = 0` |
| `Var` | non-affine variation explained by each relative warp |
| `mshape` | sample's conensus shape |
| `rotated` | Procrustes superimposed data |
| `bePCs` | vector basis of nonaffine shape variation- relative warps (plain PCs if `alpha = 0`) |
| `uniPCs` | vector basis of affine shape variation - uniform component. NULL if `alpha = 0` |

## Author(s)

Stefan Schlager

## References

Bookstein FL 1989. Principal Warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on pattern analysis and machine intelligence 11.

Bookstein FL, 1991. Morphometric tools for landmark data. Geometry and biology. Cambridge Univ. Press, Cambridge.

Rohlf FJ, Bookstein FL 2003. Computing the Uniform Component of Shape Variation. Systematic Biology 52:66-69.

## Examples

```
data(boneData)
pop <- name2factor(boneLM,which=3)
rW <- relWarps(boneLM, alpha = -1)
## Not run:
if (require(car)) {
# plot first 5 relative warps scores grouped by population
spm(rW$bescores[,1:5],group=pop)
# plot uniform component scores grouped by population
spm(rW$uniscores[,1:5],group=pop)
}
##plot non-affine variance associated with each relative warp
barplot(rW$Var[,2], xlab="relative Warps")
## visualize first relative warp +-3 sd of the scores
rw1 <- restoreShapes(as.matrix(c(-3,3)*sd(rW$bescores[,1])),rW$bePCs[,1,drop=FALSE],rW$mshape)
deformGrid3d(rw1[,,1],rw1[,,2],ngrid=5)

## 2D example:
if (require(shapes)) {
data <- bindArr(gorf.dat, gorm.dat, along=3)
sex <- factor(c(rep("fem", dim(gorf.dat)[3]), rep("male",dim(gorm.dat)[3])))
rW <- relWarps(data, alpha = -1)
if (require(car)) {
# plot first 3 relative warps scores grouped by population
spm(rW$bescores[,1:3],group=sex)
# plot uniform component scores grouped by population
spm(rW$uniscores[,1:2],group=sex)
}
##plot non-affine variance associated with each relative warp
barplot(rW$Var[,2], xlab="relative Warps")
## visualize first relative warp +-3 sd of the scores
rw1 <- restoreShapes(as.matrix(c(-3,3)*sd(rW$bescores[,1])),rW$bePCs[,1,drop=FALSE],rW$mshape)
deformGrid2d(rw1[,,1],rw1[,,2],ngrid=10)
}
## End(Not run)
```

## render *plot or save the results of meshDist*

### Description

plot or save the results of meshDist

### Usage

```
render(x, ...)

## S3 method for class 'meshDist'
render(
  x,
  from = NULL,
  to = NULL,
  steps = NULL,
  ceiling = NULL,
  uprange = NULL,
  tol = NULL,
  tolcol = NULL,
  rampcolors = NULL,
  NAcol = NULL,
  displace = FALSE,
  shade = TRUE,
  sign = NULL,
  add = FALSE,
  scaleramp = NULL,
  titleplot = "Distance in mm",
  ...
)

## S3 method for class 'matrixDist'
render(
  x,
  from = NULL,
  to = NULL,
  steps = NULL,
  ceiling = NULL,
  uprange = NULL,
  tol = NULL,
  tolcol = NULL,
  type = c("s", "p"),
  radius = NULL,
  rampcolors = NULL,
  NAcol = NULL,
  displace = FALSE,
  sign = NULL,
  add = FALSE,
  scaleramp = FALSE,
  titleplot = "Distance in mm",
  ...
)

export(x, ...)

## S3 method for class 'meshDist'
export(
  x,
  file = "default",
  imagedim = "100x800",
  titleplot = "Distance in mm",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | object of class meshDist |
| `...` | for render.meshDist: additional arguments passed to `shade3d`. See `material3d` |

| | |
|---|---|
| | for details. |
| `from` | numeric: minimum distance to color; default is set to 0 mm |
| `to` | numeric: maximum distance to color; default is set to the maximum distance |
| `steps` | integer: determines how many intermediate colors the color ramp has. |
| `ceiling` | logical: if TRUE, the next larger integer of "to" is used |
| `uprange` | numeric between 0 and 1: restricts "to" to a quantile of "to", if to is NULL. |
| `tol` | numeric: threshold to color distances within this threshold according to `tolcol`. |
| `tolcol` | a custom color to color vertices below a threshold defined by `tol`. Default is green. |
| `rampcolors` | character vector: specify the colors which are used to create a colorramp. |
| `NAcol` | character: specify color for values outside the range defined by `from` and `to`. |
| `displace` | logical: if TRUE, displacement vectors between original and closest points are drawn colored according to the distance. |
| `shade` | logical: if FALSE, the rendering of the colored surface will be supressed. |
| `sign` | logical: request signed distances to be visualised. |
| `add` | logical: if TRUE, visualization will be added to the rgl window currently in focus |
| `scaleramp` | if TRUE the ramp colors get scaled symmetrically into positive and negative direction. |
| `titleplot` | character: axis description of heatmap. |
| `type` | character: "s" shows coordinates as spheres, while "p" shows 3D dots. |
| `radius` | determines size of spheres; if not specified, optimal radius size will be estimated by centroid size of the configuration. |
| `file` | character: filename for mesh and image files produced. E.g. "mydist" will produce the files mydist.ply and mydist.png |
| `imagedim` | character of pattern "100x200" where 100 determines the width and 200 the height of the image. |

## Details

Visualise or save the results of meshDist to disk.

render.meshDist renders the colored mesh and displays the color ramp and returns an object of class "meshDist". export.meshDist exports the colored mesh as ply file and the color chart as png file.

## Author(s)

Stefan Schlager

## See Also

`meshDist`, `shade3d`

## `resampleCurve` *Resample a curve equidistantly*

### Description

Resample a curve equidistantly (optionally with smoothing)

### Usage

```
resampleCurve(x, n, smooth = FALSE, smoothn = n, open = TRUE)
```

### Arguments

| | |
|---|---|
| `x` | matrix containing coordinates |
| `n` | number of resulting points on the resampled curve |
| `smooth` | logical: if TRUE, the resulting curve will be smoothed by using bezier curves. |
| `smoothn` | integer: define the refinement of the bezier curve. The higher this value, the closer the final curve will be to the original. |
| `open` | logical: define whether it is a closed curve or not. |

### Value

returns a matrix containing the resampled curve

### Examples

```
data(nose)
x <- shortnose.lm[c(304:323),]
xsample <- resampleCurve(x,n=50)
```

## `restoreFromPCA` *restore original data from PCA*

### Description

restore original data from PCA by reverting rotation and centering

### Usage

```
restoreFromPCA(scores, rotation, center)
```

### Arguments

| | |
|---|---|
| `scores` | matrix containing the PC-scores |
| `rotation` | matrix containing the PCs |
| `center` | vector containing the center |

## Examples

```
myirispca <- prcomp(iris[,1:4])
myirisRecovered <- restoreFromPCA(myirispca$x,myirispca$rotation,myirispca$center)
all.equal(myirisRecovered,as.matrix(iris[,1:4]))
```

---

## restoreShapes *restore shapes from PC-Scores or similar projections*

---

### Description

restore shapes from PC-Scores or similar projections

### Usage

```
restoreShapes(
  scores,
  PC,
  mshape,
  sizeshape = FALSE,
  origsize = FALSE,
  meanlogCS
)
```

### Arguments

| | |
|---|---|
| `scores` | vector of PC-scores, or matrix with rows containing PC-scores |
| `PC` | Principal components (eigenvectors of the covariance matrix) associated with 'scores'. |
| `mshape` | matrix containing the meanshape's landmarks (used to center the data by the PCA) |
| `sizeshape` | logical: if TRUE, it is assumed that the data is the output of `procSym` run with `sizeshape=TRUE`. |
| `origsize` | logical: if `sizeshape = TRUE`, this will apply the scaling to the original size from the corresponding entry from the PC basis matrix. |
| `meanlogCS` | numeric: provide the average log Centroid Size of the original sample (see examples below). Only needed if `sizeshape = TRUE` and `origsize = TRUE` |

### Details

Rotates and translates PC-scores (or similar) derived from shape data back into configuration space.

### Value

returns matrix or array containing landmarks

### Author(s)

Stefan Schlager

### See Also

prcomp, procSym

getPCscores

### Examples

```
if (require(shapes)) {
## generate landmarks using
##the first PC-score of the first specimen

proc <- procSym(gorf.dat)
lm <- restoreShapes(proc$PCscores[1,1],proc$PCs[,1],proc$mshape)
plot(lm,asp=1)

##now the first 3 scores
lm2 <- restoreShapes(proc$PCscores[1,1:3],proc$PCs[,1:3],proc$mshape)
points(lm2,col=2)

## Now restore some sizeshape data
procSize <- procSym(gorf.dat,sizeshape=TRUE)
est1 <- restoreShapes(range(procSize$PCscores[,1]),procSize$PCs[,1],procSize$mshape,
                      sizeshape=TRUE,origsize=TRUE,meanlogCS=procSize$meanlogCS)
}
```

---

## retroDeform3d *symmetrize a bilateral landmark configuration*

---

### Description

symmetrize a bilateral landmark configuration by removing bending and stretching

### Usage

```
retroDeform3d(mat, pairedLM, hmult = 5, alpha = 0.01)
```

### Arguments

| | |
|---|---|
| mat | matrix with bilateral landmarks |
| pairedLM | 2-column integer matrix with the 1st columns containing row indices of left side landmarks and 2nd column the right hand landmarks |
| hmult | factor controlling the bandwidth for calculating local weights (which will be hmult * average distance between landmarks and their closest neighbour). |
| alpha | factor controlling spacing along x-axis |

### Value

| | |
|---|---|
| `deformed` | matrix containing deformed landmarks |
| `orig` | matrix containing original landmarks |

### References

Ghosh, D.; Amenta, N. & Kazhdan, M. Closed-form Blending of Local Symmetries. Computer Graphics Forum, Wiley-Blackwell, 2010, 29, 1681-1688

---

## `retroDeformMesh` *symmetrize a triangular mesh*

---

### Description

symmetrize a triangular mesh

### Usage

```
retroDeformMesh(
  mesh,
  mat,
  pairedLM,
  hmult = 5,
  alpha = 0.01,
  rot = TRUE,
  lambda = 1e-08,
  threads = 0
)
```

### Arguments

| | |
|---|---|
| `mesh` | triangular mesh of class mesh3d |
| `mat` | matrix with bilateral landmarks |
| `pairedLM` | 2-column integer matrix with the 1st columns containing row indices of left side landmarks and 2nd column the right hand landmarks |
| `hmult` | damping factor for calculating local weights which is calculated as `humult` times the average squared distance between a landmark and its closest neighbor (on each side). |
| `alpha` | factor controlling spacing along x-axis |
| `rot` | logical: if TRUE the deformed landmarks are rotated back onto the original ones |
| `lambda` | control parameter passed to `tps3d` |
| `threads` | integer: number of threads to use for TPS deform |

## Details

this function performs `retroDeform3d` and deforms the mesh accordingly using the function `tps3d`.

## Value

| | |
|---|---|
| `mesh` | symmetrized mesh |
| `landmarks` | a list containing the deformed and original landmarks |

---

## rotaxis3d — *Rotate an object (matrix or mesh) around an arbitrary axis in 3D*

---

## Description

Rotate an object around an arbitrary axis in 3D

## Usage

```
rotaxis3d(x, pt1, pt2 = c(0, 0, 0), theta)

## S3 method for class 'matrix'
rotaxis3d(x, pt1, pt2 = c(0, 0, 0), theta)

## S3 method for class 'mesh3d'
rotaxis3d(x, pt1, pt2 = c(0, 0, 0), theta)
```

## Arguments

| | |
|---|---|
| `x` | k x 3 matrix containing 3D-coordinates or a triangular mesh of class "mesh3d". |
| `pt1` | numeric vector of length 3, defining first point on the rotation axis. |
| `pt2` | numeric vector of length 3, defining second point on the rotation axis. |
| `theta` | angle to rotate in radians. With pt1 being the viewpoint, the rotation is counterclockwise. |

## Details

Rotate an object (matrix or triangular mesh) around an 3D-axis defined by two points.

## Value

returns rotated object (including updated normals for mesh3d objects)

## Author(s)

Stefan Schlager

### References

http://en.wikipedia.org/wiki/Rotation_matrix

### See Also

rotonto, rotmesh.onto

### Examples

```
require(rgl)
data(nose)
shrot.rot <- rotaxis3d(shortnose.mesh,pt1=c(1,1,1),theta=pi)
## Not run:
shade3d(shortnose.mesh,col=3,specular=1)
shade3d(shrot.rot,col=2)

###print rotation axis
#' lines3d(rbind(rep(-0.1,3),rep(0.1,3)))

## End(Not run)
```

---

## rotaxisMat *calculate a rotation matrix around an arbitrary axis through the origin in 3D*

---

### Description

calculate a rotation matrix around an arbitrary axis in 3D

### Usage

```
rotaxisMat(u, theta, homogeneous = FALSE)
```

### Arguments

| | |
|---|---|
| u | a vector around which to rotate |
| theta | angle in radians to rotate |
| homogeneous | logical: if TRUE a 4x4 rotation matrix is returned |

### Value

returns 3x3 rotation matrix

### References

http://en.wikipedia.org/wiki/Rotation_matrix

### See Also

rotaxis3d

| | |
|---|---|
| `rotmesh.onto` | *rotate ,scale and translate a mesh based on landmark information.* |

## Description

rotates and reflects a mesh onto by calculating the transformation from two sets of referenced landmarks.

## Usage

```
rotmesh.onto(
  mesh,
  refmat,
  tarmat,
  adnormals = FALSE,
  scale = FALSE,
  reflection = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| `mesh` | object of class mesh3d. |
| `refmat` | k x m matrix with landmarks on the mesh |
| `tarmat` | k x m matrix as target configuration |
| `adnormals` | logical - if TRUE, vertex normals will be recomputed after rotation. If `mesh` has normals and adnormals=FALSE, the existing normals are rotated by the same rotation matrix as the mesh's vertices. |
| `scale` | logical: if TRUE the mesh will be scaled according to the size of the target. |
| `reflection` | logical: allow reflection. |
| `...` | additional parameters passed on to `rotonto`. |

## Value

| | |
|---|---|
| `mesh` | rotated mesh |
| `yrot` | rotated refmat |
| `trafo` | 4x4 transformation matrix |

## Author(s)

Stefan Schlager

## See Also

`file2mesh`,`tps3d` ,`rotonto`,`mesh2ply`

## Examples

```
require(rgl)
data(boneData)
## rotate, translate and scale the mesh belonging to the first specimen
## onto the landmark configuration of the 10th specimen
rotmesh <- rotmesh.onto(skull_0144_ch_fe.mesh,boneLM[,,1],
                        boneLM[,,10], scale=TRUE)
## Not run:
## render rotated mesh and landmarks
shade3d(rotmesh$mesh, col=2, specular=1)
spheres3d(boneLM[,,1])
## render original mesh
shade3d(skull_0144_ch_fe.mesh, col=3, specular=1)
spheres3d(boneLM[,,10])

## End(Not run)
```

---

`rotonmat` *rotate matrix of landmarks*

---

## Description

rotate matrix of landmarks by using a rotation determined by two matrices.

## Usage

```
rotonmat(
  X,
  refmat,
  tarmat,
  scale = TRUE,
  reflection = FALSE,
  weights = NULL,
  centerweight = FALSE,
  getTrafo = FALSE
)
```

## Arguments

| | |
|---|---|
| `X` | Matrix to be rotated |
| `refmat` | reference matrix used to estimate rotation parameters |
| `tarmat` | target matrix used to estimate rotation parameters |
| `scale` | logical: requests scaling to minimize sums of squared distances |
| `reflection` | logical: if TRUE, reflections are allowed. |
| `weights` | vector of length k, containing weights for each landmark. |

| | |
|---|---|
| `centerweight` | logical: if weights are defined and centerweigths=TRUE, the matrix will be centered according to these weights instead of the barycenter. |
| `getTrafo` | logical: if TRUE, a 4x4 transformation matrix will also be returned. |

## Details

A matrix is rotated by rotation parameters determined by two different matrices. This is usefull, if the rotation parameters are to be estimated by a subset of landmark coordinates.

## Value

if `getTrafo=FALSE` the transformed X will be returned, else alist containing:

| | |
|---|---|
| `Xrot` | the transformed matrix X |
| `trafo` | a 4x4 transformation matrix |

## Author(s)

Stefan Schlager

## See Also

`rotonto`,`rotmesh.onto`

## Examples

```
data(nose)
shortnose.rot <-
rotonmat(shortnose.lm,shortnose.lm[1:9,],longnose.lm[1:9,])

##view result
## Not run:
deformGrid3d(shortnose.rot,shortnose.lm,ngrid=0)

## End(Not run)
```

---

`rotonto` *rotates, translates and scales one matrix onto an other using Procrustes fitting*

---

## Description

rotates, translates and scales one matrix onto an other using Procrustes fitting

## Usage

```
rotonto(
  x,
  y,
  scale = FALSE,
  signref = TRUE,
  reflection = TRUE,
  weights = NULL,
  centerweight = FALSE,
  ...
)

rotreverse(mat, rot)

## S3 method for class 'matrix'
rotreverse(mat, rot)

## S3 method for class 'mesh3d'
rotreverse(mat, rot)
```

## Arguments

| | |
|---|---|
| `x` | k x m matrix to be rotated onto (targetmatrix) |
| `y` | k x m matrix which will be rotated (reference matrix) |
| `scale` | logical: scale matrix to minimize sums of squares |
| `signref` | logical: report if reflections were involved in the rotation |
| `reflection` | allow reflections. |
| `weights` | vector of length k, containing weights for each landmark. |
| `centerweight` | logical or vector of weights: if weights are defined and centerweigths=TRUE, the matrix will be centered according to these weights instead of the barycenter. If centerweight is a vector of length `nrow(x)`, the barycenter will be weighted accordingly. |
| `...` | currently not used |
| `mat` | matrix on which the reverse transformations have to be applied |
| `rot` | an object resulting from the former application of rotonto |

## Details

rotate a matrix onto an other without loosing information about the location of the targetmatrix and reverse this transformations using rotreverse

## Value

| | |
|---|---|
| `yrot` | rotated and translated matrix |
| `Y` | centred and rotated reference matrix |

| | |
|---|---|
| `X` | centred target matrix |
| `trans` | vector between original position of target and centered reference (during rotation process) |
| `transy` | vector between original position of reference and centered reference (during rotation process) |
| `gamm` | rotation matrix |
| `bet` | scaling factor applied |
| `reflect` | if `reflect = 1`, reflections are involved in the superimposition. Else, reflect = 0 |

## Note

all lines containing NA, or NaN are ignored in computing the transformation.

## Author(s)

Stefan Schlager

## References

Lissitz, R. W., Schoenemann, P. H., & Lingoes, J. C. (1976). A solution to the weighted Procrustes problem in which the transformation is in agreement with the loss function. Psychometrika, 41,547-550.

## See Also

`rotmesh.onto`

## Examples

```
if (require(shapes)) {
lims <- c(min(gorf.dat[,,1:2]),max(gorf.dat[,,1:2]))
rot <- rotonto(gorf.dat[,,1],gorf.dat[,,2]) ### rotate the second onto the first config
plot(rot$yrot,pch=19,xlim=lims,ylim=lims) ## view result
points(gorf.dat [,,2],pch=19,col=2) ## view original config
rev1 <- rotreverse(rot$yrot,rot)
points(rev1,cex=2) ### show inversion by larger circles around original configuration
}
```

---

## `scalemesh` *scale a mesh of class "mesh3d"*

---

## Description

scales (the vertices of a mesh by a scalar

### Usage

```
scalemesh(mesh, size, center = c("bbox", "mean", "none"))
```

### Arguments

| | |
|---|---|
| `mesh` | object of class "mesh3d" |
| `size` | numeric: scale factor |
| `center` | character: method to position center of mesh after scaling: values are "bbox", and "mean". See Details for more info. |

### Details

The mesh's center is determined either as mean of the bounding box (center="bbox") or mean of vertex coordinates (center="mean") and then scaled according to the scaling factor. If center="none", vertex coordinates will simply be multiplied by "size".

### Value

returns a scaled mesh

### Author(s)

Stefan Schlager

### See Also

`rotmesh.onto`

### Examples

```
data(nose)
#inflate mesh by factor 4
largenose <- scalemesh(shortnose.mesh,4)
```

## slider2d *slides Semilandmarks along curves 2D by minimising bending energy of a thin-plate spline deformation.*

### Description

slides Semilandmarks along curves 2D. The positions are sought by minimising bending energy (of a thin-plate spline deformation) or Procrustes distance

## Usage

```
slider2d(
  dataframe,
  SMvector,
  outlines,
  tol = 1e-05,
  deselect = FALSE,
  recursive = TRUE,
  iterations = 0,
  initproc = FALSE,
  pairedLM = NULL,
  bending = TRUE,
  stepsize = 1,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `dataframe` | Input k x 2 x n real array, where k is the number of points and n is the sample size. Ideally the |
| `SMvector` | A vector containing the row indices of (semi-) landmarks on the curve(s) and surfaces that are allowed to slide |
| `outlines` | A vector (or if threre are several curves) a list of vectors (containing the rowindices) of the (Semi-)landmarks forming the curve(s) in the successive position on the curve - including the beginning and end points, that are not allowed to slide. |
| `tol` | numeric: Threshold for convergence in the sliding process |
| `deselect` | Logical: if TRUE, the SMvector is interpreted as those landmarks, that are not allowed to slide. |
| `recursive` | Logical: if TRUE, during the iterations of the sliding process, the outcome of the previous iteration will be used. Otherwise the original configuration will be used in all iterations. |
| `iterations` | integer: select manually the max. number of iterations that will be performed during the sliding process (usefull, when there is very slow convergence). 0 means iteration until convergence. |
| `initproc` | requests initial Procrustes fit before sliding. |
| `pairedLM` | A X x 2 numeric matrix with the indices of the rows containing paired Landmarks. E.g. the left column contains the lefthand landmarks, while the right side contains the corresponding right hand landmarks. - This will ideally create symmetric mean to get rid of assymetry. |
| `bending` | if TRUE, bending energy will be minimized, Procrustes distance otherwise. |
| `stepsize` | integer: dampening factor for the amount of sliding. Useful to keep semi-landmarks from sliding too far off the surface. The displacement is calculated as $\Upsilon = \Upsilon^0 + stepsize * UT$. Default is set to 1 for bending=TRUE and 0.5 for bending=FALSE. |
| `silent` | logical: if TRUE, console output is suppressed. |

### Value

returns an array containing slided coorndinates in the original space - not yet processed by a Procrustes analysis.

### Warning

Depending on the amount of landmarks this can use an extensive amount of your PC's resources, especially when running in parallel. As the computation time and RAM usage of matrix algebra involved is quadratic to the amount of landmarks used, doubling the amount of semi-landmarks will quadruple computation time and system resource usage. You can easily stall you computer with this function with inappropriate data.

### Author(s)

Stefan Schlager

### See Also

relaxLM, slider3d

---

## slider3d *slides Semilandmarks along curves and surfaces in 3D by minimising bending energy of a thin-plate spline deformation.*

---

### Description

slides Semilandmarks along curves and surfaces in 3D. The positions on the surface are sought which minimise bending energy (of a thin-plate spline deformation)

### Usage

```
slider3d(
  dat.array,
  SMvector,
  outlines = NULL,
  surp = NULL,
  sur.path = NULL,
  sur.name = NULL,
  meshlist = NULL,
  ignore = NULL,
  sur.type = "ply",
  tol = 1e-05,
  deselect = FALSE,
  inc.check = TRUE,
  recursive = TRUE,
  iterations = 0,
  initproc = TRUE,
  fullGPA = FALSE,
  pairedLM = 0,
  bending = TRUE,
  stepsize = ifelse(bending, 1, 0.5),
  mc.cores = parallel::detectCores(),
  fixRepro = TRUE,
  missingList = NULL,
  use.lm = NULL,
  smoothnormals = FALSE,
  silent = FALSE
)
```

## Arguments

| | |
|---|---|
| `dat.array` | Input k x m x n real array, where k is the number of points, m is the number of dimensions, and n is the sample size. Ideally the dimnames[[3]] vector contains the names of the surface model (without file extension) - e.g. if the model is named "surface.ply", the name of the corresponding matrix of the array would be "surface" |
| `SMvector` | A vector containing the row indices of (semi-) landmarks on the curve(s) and surfaces that are allowed to slide |
| `outlines` | A vector (or if threre are several curves) a list of vectors (containing the rowindices) of the (Semi-)landmarks forming the curve(s) in the successive position on the curve - including the beginning and end points, that are not allowed to slide. |
| `surp` | integer vector containing the row indices of semi-landmarks positioned on surfaces. |
| `sur.path` | Path to the surface models (e.g. ply, obj, stl files) |
| `sur.name` | character vector: containing the filenames of the corresponding surfaces - e.g. if the dat.array[,,i] belongs to surface_i.ply, sur.name[i] would be surface_i.ply. Only necessary if dat.array does not contain surface names. |
| `meshlist` | list containing triangular meshes of class 'mesh3d', for example imported with `mesh2ply` or `file2mesh` in the same order as the specimen in the array (see examples below). |
| `ignore` | vector containing indices of landmarks that are to be ignored. Indices of outlines/surfaces etc will be updated automatically. |
| `sur.type` | character:if all surfaces are of the same file format and the names stored in dat.array, the file format will be specified here. |
| `tol` | numeric: Threshold for convergence in the sliding process |
| `deselect` | Logical: if TRUE, the SMvector is interpreted as those landmarks, that are not allowed to slide. |
| `inc.check` | Logical: if TRUE, the program stops when convergence criterion starts increasing and reports result from last iteration. |
| `recursive` | Logical: if TRUE, during the iterations of the sliding process, the outcome of the previous iteration will be used. Otherwise the original configuration will be used in all iterations. |

| | |
|---|---|
| `iterations` | integer: select manually the max. number of iterations that will be performed during the sliding process (usefull, when there is very slow convergence). 0 means iteration until convergence. |
| `initproc` | requests initial Procrustes fit before sliding. |
| `fullGPA` | Logical: if FALSE, only a partial procrustes fit will be performed. |
| `pairedLM` | A X x 2 numeric matrix with the indices of the rows containing paired Landmarks. E.g. the left column contains the lefthand landmarks, while the right side contains the corresponding right hand landmarks. - This will ideally create symmetric mean to get rid of assymetry. |
| `bending` | if TRUE, bending energy will be minimized, Procrustes distance otherwise. |
| `stepsize` | integer: dampening factor for the amount of sliding. Useful to keep semi-landmarks from sliding too far off the surface. The displacement is calculated as $\Upsilon = \Upsilon^0 + stepsize * UT$. Default is set to 1 for bending=TRUE and 0.5 for bending=FALSE. |
| `mc.cores` | integer: determines how many cores to use for the computation. The default is autodetect. But in case, it doesn't work as expected cores can be set manually. |
| `fixRepro` | logical: if `TRUE`, fix landmarks will also be projected onto the surface. If you have landmarks not on the surface, select `fixRepro=FALSE` |
| `missingList` | a list of length samplesize containing integer vectors of row indices specifying missing landmars for each specimen. For specimens without missing landmarks enter `numeric(0)`. |
| `use.lm` | indices specifying a subset of (semi-)landmarks to be used in the rotation step - only used if `bending=FALSE`. |
| `smoothnormals` | logical: if TRUE, tangent planes will be computed from locally smoothed normals |
| `silent` | logical: if TRUE, console output is suppressed. |

## Value

| | |
|---|---|
| `dataslide` | array containing slidden Landmarks in the original space - not yet processed by a Procrustes analysis |
| `vn.array` | array containing landmark normals |

## Warning

Depending on the size of the suface meshes and especially the amount of landmarks this can use an extensive amount of your PC's resources, especially when running in parallel. As the computation time and RAM usage of matrix algebra involved is quadratic to the amount of landmarks used, doubling the amount of semi-landmarks will quadruple computation time and system resource usage. You can easily stall you computer with this function with inappropriate data.

## Note

if `sur.path = NULL` and `meshlist = NULL`, surface landmarks are relaxed based on a surface normals approximated by the pointcloud, this can lead to bad results for sparse sets of semilandmarks. Obviously, no projection onto the surfaces will be occur and landmarks will likely be off the original surface.

## Author(s)

Stefan Schlager

## References

Klingenberg CP, Barluenga M, and Meyer A. 2002. Shape analysis of symmetric structures: quantifying variation among individuals and asymmetry. Evolution 56(10):1909-1920.

Gunz, P., P. Mitteroecker, and F. L. Bookstein. 2005. Semilandmarks in Three Dimensions, in Modern Morphometrics in Physical Anthropology. Edited by D. E. Slice, pp. 73-98. New York: Kluwer Academic/Plenum Publishers.

Schlager S. 2012. Sliding semi-landmarks on symmetric structures in three dimensions. American Journal of Physical Anthropology, 147(S52):261. URL: http://dx.doi.org/10.1002/ajpa.21502.

Schlager S. 2013. Soft-tissue reconstruction of the human nose: population differences and sexual dimorphism. PhD thesis, Universitätsbibliothek Freiburg. URL: http://www.freidok.uni-freiburg.de/volltexte/9181/.

## See Also

relaxLM, createMissingList

## Examples

```
## Not run:
data(nose)
###create mesh for longnose
longnose.mesh <- tps3d(shortnose.mesh,shortnose.lm,longnose.lm,threads=1)
### write meshes to disk
mesh2ply(shortnose.mesh, filename="shortnose")
mesh2ply(longnose.mesh, filename="longnose")

## create landmark array
data <- bindArr(shortnose.lm, longnose.lm, along=3)
dimnames(data)[[3]] <- c("shortnose", "longnose")

# define fix landmarks
fix <- c(1:5,20:21)
# define surface patch by specifying row indices of matrices
# all except those defined as fix
surp <- c(1:nrow(shortnose.lm))[-fix]

slide <- slider3d(data, SMvector=fix, deselect=TRUE, surp=surp,
                  sur.path=".",iterations=1,mc.cores=1)
                  # sur.path="." is the current working directory

# now one example with meshes in workspace

meshlist <- list(shortnose.mesh,longnose.mesh)

slide <- slider3d(data, SMvector=fix, deselect=TRUE, surp=surp,
                  iterations=1, meshlist=meshlist,
                  mc.cores=1,fixRepro=FALSE)
require(rgl)
## visualize sliding
deformGrid3d(slide$dataslide[,,1],shortnose.lm,ngrid = 0)
## these are fix
spheres3d(slide$dataslide[fix,,1],col=4,radius=0.7)

###finally an example with missing landmarks:
## we assume that coordinates 185:189, 205:209 and 225:229 are in the second config are missing
missingList <- createMissingList(2)
missingList[[2]] <- c(185:189,205:209,225:229)
slideMissing <- slider3d(data, SMvector=fix, deselect=TRUE, surp=surp,
                  iterations=1, meshlist=meshlist,
                  mc.cores=1,fixRepro=FALSE,missingList=missingList)

## example with two curves
## Example with surface semilandmarks and two curves
fix <- c(1:5,20:21)
outline1 <- c(304:323)
outline2 <- c(604:623)
outlines <- list(outline1,outline2)
surp <- c(1:623)[-c(fix,outline1,outline2)]
slideWithCurves <- slider3d(data, SMvector=fix, deselect=TRUE, surp=surp,
                            meshlist=meshlist,iterations=1,mc.cores=1,outlines=outlines)
deformGrid3d(slideWithCurves$dataslide[,,1],shortnose.lm,ngrid = 0)
plot(slideWithCurves)

## finally an example with sliding without meshes by estimating the surface from the
## semi-landmarks

slideWithCurvesNoMeshes <- slider3d(data, SMvector=fix, deselect=TRUE, surp=surp,
                            iterations=1,mc.cores=1,outlines=outlines)
## compare it to the data with surfaces
deformGrid3d(slideWithCurves$dataslide[,,1],slideWithCurvesNoMeshes$dataslide[,,1],ngrid = 0)
## not too bad, only lonely surface semi-landmarks are a bit off


## End(Not run)
```

## solutionSpace *returns the solution space (basis and translation vector) for an equation system*

### Description

returns the solution space (basis and translation vector) for an equation system

### Usage

```
solutionSpace(A, b)
```

## Arguments

| | |
|---|---|
| `A` | numeric matrix |
| `b` | numeric vector |

## Details

For a linear equationsystem, $Ax = b$, the solution space then is

$$x = A^*b + (I - A^*A)y$$

where $A^*$ is the Moore-Penrose pseudoinverse of $A$. The QR decomposition of $I - A^*A$ determines the dimension of and basis of the solution space.

## Value

| | |
|---|---|
| `basis` | matrix containing the basis of the solution space |
| `translate` | translation vector |

## Examples

```
A <- matrix(rnorm(21),3,7)
b <- c(1,2,3)
subspace <- solutionSpace(A,b)
dims <- ncol(subspace$basis) # we now have a 4D solution space
## now pick any vector from this space. E.g
y <- 1:dims
solution <- subspace$basis%*%y+subspace$translate # this is one solution for the equation above
A%*%solution ## pretty close
```

---

`sortCurve` *sort curvepoints by using the subsequent neighbours*

---

## Description

sort curvepoints by using the subsequent neighbours

## Usage

```
sortCurve(x, k = 5, start = NULL)
```

## Arguments

| | |
|---|---|
| `x` | k x m matrix containing the 2D or 3D coordinates |
| `k` | number of nearest neighbours to look at. Set high for very irregularly clustered curves. |
| `start` | integer: which row of x to use as a starting point. If NULL, it is assumed that the curve is open and the point where the angle between the two nearest neighbours is closest will be chosen. |

## Value

| | |
|---|---|
| `xsorted` | matrix with coordinates sorted along a curve |
| `index` | vector containing the sorting indices |

## Examples

```
## generate a curve from a polynome
x <- c(32,64,96,118,126,144,152.5,158)
y <- c(99.5,104.8,108.5,100,86,64,35.3,15)
fit <- lm(y~poly(x,2,raw=TRUE))
xx <- seq(30,160, length=50)
layout(matrix(1:3,3,1))
curve <- cbind(xx,predict(fit, data.frame(x=xx)))
## permute order
set.seed(42)
plot(curve);lines(curve)
curveunsort <- curve[sample(1:50),]
## now the curve is scrambled
plot(curveunsort);lines(curveunsort,col=2)
curvesort <- sortCurve(curveunsort)
## after sorting lines are nice again
plot(curvesort$xsorted);lines(curvesort$xsorted,col=3)
```

---

`symmetrize` *create a perfectly symmetric version of landmarks*

---

## Description

create a perfectly symmetric version of landmarks

## Usage

```
symmetrize(x, pairedLM)
```

## Arguments

| | |
|---|---|
| `x` | k x m matrix or k x m x n array, with rows containing landmark coordinates |
| `pairedLM` | A X x 2 matrix containing the indices (rownumbers) of the paired LM. E.g. the left column contains the lefthand landmarks, while the right side contains the corresponding right hand landmarks. |

## Details

the landmarks are reflected and relabled according to `pairedLM` and then rotated and translated onto `x`. Both configurations are then averaged to obtain a perfectly symmetric one.

## Value

a symmetrized version of `x`

### References

Klingenberg CP, Barluenga M, and Meyer A. 2002. Shape analysis of symmetric structures: quantifying variation among individuals and asymmetry. Evolution 56(10):1909-1920.

### Examples

```
data(boneData)
left <- c(4,6,8)
right <- c(3,5,7)
pairedLM <- cbind(left,right)
symx <- symmetrize(boneLM[,,2],pairedLM)
## Not run:
deformGrid3d(symx,boneLM[,,2])

## End(Not run)
```

---

## tps3d *thin plate spline mapping (2D and 3D) for coordinates and triangular meshes*

---

### Description

maps landmarks or a triangular mesh via thin plate spline based on a reference and a target configuration in 2D and 3D

### Usage

```
tps3d(x, refmat, tarmat, lambda = 1e-08, threads = 0, ...)

tps2d(x, refmat, tarmat, lambda = 1e-08, threads = 0, ...)
```

### Arguments

| | |
|---|---|
| x | matrix - e.g. the matrix information of vertices of a given surface or a triangular mesh of class "mesh3d" |
| refmat | reference matrix - e.g. landmark configuration on a surface |
| tarmat | target matrix - e.g. landmark configuration on a target surface |
| lambda | numeric: regularisation parameter of the TPS. |
| threads | threads to be used for parallel execution in tps deformation. |
| ... | additional arguments, currently not used. |

### Value

returns the deformed input

## Note

`tps2d` is simply an alias for `tps3d` that can handle both cases.

## Author(s)

Stefan Schlager

## References

Bookstein FL. 1989. Principal Warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on pattern analysis and machine intelligence 11(6).

## See Also

`computeTransform`, `applyTransform`

## Examples

```
data(nose)
## define some landmarks
refind <- c(1:3,4,19:20)
## use a subset of shortnose.lm as anchor points for a TPS-deformation
reflm <- shortnose.lm[refind,]
tarlm <- reflm
##replace the landmark at the tip of the nose with that of longnose.lm
tarlm[4,] <- longnose.lm[4,]
##  deform a set of semilandmarks by applying a TPS-deformation
##  based on 5 reference points
deformed <- tps3d(shortnose.lm, reflm, tarlm,threads=1)
## Not run:
##visualize results by applying a deformation grid
deformGrid3d(shortnose.lm,deformed,ngrid = 5)


data(nose)##load data
##warp a mesh onto another landmark configuration:
longnose.mesh <- tps3d(shortnose.mesh,shortnose.lm,longnose.lm,threads=1)


require(rgl)
shade3d(longnose.mesh,col=skin1)

## End(Not run)

data(boneData)
## deform mesh belonging to the first specimen
## onto the landmark configuration of the 10th specimen

## Not run:
warpskull <- tps3d(skull_0144_ch_fe.mesh,boneLM[,,1],
                     boneLM[,,10], threads=1)
## render deformed mesh and landmarks
```

```
shade3d(warpskull, col=2, specular=1)
spheres3d(boneLM[,,1])
## render original mesh
shade3d(skull_0144_ch_fe.mesh, col=3, specular=1)
spheres3d(boneLM[,,10])


## End(Not run)
```

## typprob *calculate typicality probabilities*

### Description

calculate typicality probabilities

### Usage

```
typprob(
  x,
  data,
  small = FALSE,
  method = c("chisquare", "wilson"),
  center = NULL,
  cova = NULL,
  robust = c("classical", "mve", "mcd"),
  ...
)

typprobClass(
  x,
  data,
  groups,
  small = FALSE,
  method = c("chisquare", "wilson"),
  outlier = 0.01,
  sep = FALSE,
  cv = TRUE,
  robust = c("classical", "mve", "mcd"),
  ...
)
```

### Arguments

| | |
|---|---|
| `x` | vector or matrix of data the probability is to be calculated. |
| `data` | Reference data set. If missing x will be used. |
| `small` | adjustion of Mahalanobis D^2 for small sample sizes as suggested by Wilson (1981), only takes effect when method="wilson". |

| | |
|---|---|
| `method` | select method for probability estimation. Available options are "chisquare" (or any abbreviation) or "wilson". "chisquare" exploits simply the chisquare distribution of the mahalanobisdistance, while "wilson" uses the methods suggested by Wilson(1981). Results will be similar in general. |
| `center` | vector: specify custom vector to calculate distance to. If not defined, group mean will be used. |
| `cova` | covariance matrix to calculate mahalanobis-distance: specify custom covariance matrix to calculate distance. |
| `robust` | character: determines covariance estimation methods, allowing for robust estimations using `MASS::cov.rob`. Default is the standard product-moment covariance matrix. |
| `...` | additional parameters passed to `MASS::cov.rob` for robust covariance and mean estimations. |
| `groups` | vector containing grouping information. |
| `outlier` | probability threshold below which a specimen will not be assigned to any group- |
| `sep` | logical: if TRUE, probability will be calculated from the pooled within group covariance matrix. |
| `cv` | logical: if data is missing and `cv=TRUE`, the resulting classification will be validated by leaving-one-out crossvalidation. |

## Details

get the probability for an observation belonging to a given multivariate nromal distribution

## Value

typprob: returns a vector of probabilities.

typprobClass:

| | |
|---|---|
| `probs` | matrix of probabilities for each group |
| `groupaffin` | vector of groups each specimen has been assigned to. Outliers are classified "none" |
| `probsCV` | cross-validated matrix of probabilities for each group |
| `groupaffinCV` | cross-validated vector of groups each specimen has been assigned to. Outliers are classified "none" |
| `self` | logical: if TRUE, the data has been classified by self-inference. |

## Author(s)

Stefan Schlager

## References

Albrecht G. 1992. Assessing the affinities of fossils using canonical variates and generalized distances Human Evolution 7:49-69.

Wilson S. 1981. On comparing fossil specimens with population samples Journal of Human Evolution 10:207 - 214.

## Examples

```
if (require(shapes)) {
data <- procSym(gorf.dat)$PCscores[,1:3]
probas <- typprob(data,data,small=TRUE)### get probability for each specimen

### now we check how this behaves compared to the mahalanobis distance
maha <- mahalanobis(data,colMeans(data),cov(data))
plot(probas,maha,xlab="Probability",ylab="Mahalanobis D^2")

data2 <- procSym(abind(gorf.dat,gorm.dat))$PCscores[,1:3]
fac <- as.factor(c(rep("female",dim(gorf.dat)[3]),rep("male",dim(gorm.dat)[3])))
typClass <- typprobClass(data2,groups=fac,method="w",small=TRUE,cv=TRUE)
## only 59 specimen is rather small.
typClass2 <- typprobClass(data2,groups=fac,method="c",cv=TRUE)## use default settings

### check results for first method:
typClass


### check results for second method:
typClass2
}
```

---

unrefVertex *some little helpers for vertex operations on triangular meshes*

---

## Description

some little helpers for vertex operations on triangular meshes

## Usage

```
unrefVertex(mesh)

rmVertex(mesh, index, keep = FALSE)

vert2points(mesh)

rmUnrefVertex(mesh, silent = FALSE)
```

## Arguments

| | |
|---|---|
| mesh | triangular mesh of class mesh3d. |
| index | vector containing indices of vertices to be removed. |
| keep | logical: if TRUE, the vertices specified by index are kept and the rest is removed. |
| silent | logical: suppress output about info on removed vertices. |

## Details

extract vertex coordinates from meshes, find and/or remove (unreferenced) vertices from triangular meshes

`unrefVertex` finds unreferenced vertices in triangular meshes of class `mesh3d` or `tmesh3d`.

`rmVertex` removes specified vertices from triangular meshes.

`vert2points` extacts vertex coordinates from triangular meshes.

`rmUnrefVertex` removes unreferenced vertices from triangular meshes.

## Value

`unrefVertex`: vector with indices of unreferenced vertices.

`rmVertex`: returns mesh with specified vertices removed and faces and normals updated.

`vert2points`: k x 3 matrix containing vertex coordinates.

`rmUnrefVertex`: mesh with unreferenced vertices removed.

## Author(s)

Stefan Schlager

## See Also

`ply2mesh`, `file2mesh`

## Examples

```
require(rgl)
data(nose)
testmesh <- rmVertex(shortnose.mesh,1:50) ## remove first 50 vertices
## Not run:
shade3d(testmesh,col=3)### view result

## End(Not run)
testmesh$vb <- cbind(testmesh$vb,shortnose.mesh$vb[,1:50]) ## add some unreferenced vertices
## Not run:
points3d(vert2points(testmesh),col=2)## see the vertices in the holes?

## End(Not run)
cleanmesh <- rmUnrefVertex(testmesh)## remove those lonely vertices!
## Not run:
pop3d()
points3d(vert2points(cleanmesh),col=2) ### now the holes are empty!!

## End(Not run)
```

## updateIndices — *update a vector of indices after removal of some referenced items*

### Description

update a vector of indices after removal of some referenced items

### Usage

```
updateIndices(x, ignore, indexrange)
```

### Arguments

| | |
|---|---|
| `x` | vector containing indices (e.g. to matrix rows) |
| `ignore` | integer vector: remove those items from the original structure |
| `indexrange` | maximum range of the index in the referenced item structure |

### Examples

```
refItem <- matrix(1:10,5,2)
index <- c(1,3,5) # this indexes some rows of the matrix we are interested in
## now we want to ignore row 2 and 5 and want to update the index so it will still fit
indexNew <- updateIndices(index,c(2,5),indexrange=5)

## Here a more useful example:
data(boneData)
left <- c(4,6,8)
  ## determine corresponding Landmarks on the right side:
    # important: keep same order
    right <- c(3,5,7)
    pairedLM <- cbind(left,right)
## now we want to remove some landmarks and need to updated the pairedLM indices
ignore <- c(5,6)
mynewboneLM <- boneLM[-ignore,,]
pairedLMnew <- apply(pairedLM,2,updateIndices,ignore=ignore,indexrange=dim(boneLM)[1])
```

## updateNormals — *Compute face or vertex normals of a triangular mesh*

### Description

Compute face or vertex normals of a triangular mesh of class "mesh3d"

### Usage

```
updateNormals(x, angle = TRUE)

facenormals(x)
```

## Arguments

| | |
|---|---|
| x | triangular mesh of class "mesh3d" |
| angle | logical: if TRUE, angle weighted normals are used. |

## Value

`updateNormals` returns mesh with updated vertex normals.

`facenormals` returns an object of class "mesh3d" with

| | |
|---|---|
| vb | faces' barycenters |
| normals | faces' normals |

## Note

only supports triangular meshes

## Author(s)

Stefan Schlager

## References

Baerentzen, Jakob Andreas. & Aanaes, H., 2002. Generating Signed Distance Fields From Triangle Meshes. Informatics and Mathematical Modelling, .

## See Also

ply2mesh

## Examples

```
require(rgl)
require(Morpho)
data(nose)
### calculate vertex normals
shortnose.mesh$normals <- NULL ##remove normals
## Not run:
shade3d(shortnose.mesh,col=3)##render

## End(Not run)
shortnose.mesh <- updateNormals(shortnose.mesh)
## Not run:
clear3d()
shade3d(shortnose.mesh,col=3)##smoothly rendered now

## End(Not run)
## calculate facenormals
facemesh <- facenormals(shortnose.mesh)
## Not run:
plotNormals(facemesh,long=0.01)
```

```
points3d(vert2points(facemesh),col=2)
wire3d(shortnose.mesh)

## End(Not run)
```

---

`vecx` *convert an 3D array into a matrix and back*

---

## Description

converts a 3D-array (e.g. containing landmark coordinates) into a matrix, one row per specimen or reverse this.

## Usage

```
vecx(x, byrow = FALSE, revert = FALSE, lmdim)
```

## Arguments

| | |
|---|---|
| `x` | array or matrix |
| `byrow` | logical: if TRUE, the resulting vector for each specimen will be `x1,y1,z1,x2,y2,z2,...`, and `x1,x2,...,y1,y2,...,z1,z2,...` otherwise (default). The same is for reverting the process: if the matrix contains the coordinates as rows like: `x1,y1,z1,x2,y2,z2,...` set `byrow=TRUE` |
| `revert` | revert the process and convert a matrix with vectorized landmarks back into an array. |
| `lmdim` | number of columns for reverting |

## Value

returns a matrix with one row per specimen

## Author(s)

Stefan Schlager

## Examples

```
if (require(shapes)) {
data <- vecx(gorf.dat)
#revert the procedure
gdat.restored <- vecx(data,revert=TRUE,lmdim=2)
range(gdat.restored-gorf.dat)
}
```

## virtualMeshScan *remove all parts of a triangular mesh, not visible from a set of viewpoints*

### Description

remove all parts of a triangular mesh, not visible from a set of viewpoints

### Usage

```
virtualMeshScan(x, viewpoints, offset = 0.001, cores = 1)
```

### Arguments

| | |
|---|---|
| `x` | triangular mesh of class 'mesh3d' |
| `viewpoints` | vector or k x 3 matrix containing a set of viewpoints |
| `offset` | value to generate an offset at the meshes surface (see notes) |
| `cores` | integer: number of cores to use (not working on windows) |

### Value

returns a list containing subsets of the original mesh

| | |
|---|---|
| `visible` | the parts visible from at least one of the viewpoints |
| `invisible` | the parts not visible from the viewpoints |

### Note

The function tries to filter out all vertices where the line connecting each vertex with the viewpoints intersects with the mesh itself. As, technically speaking this always occurs at a distance of value=0, a mesh with a tiny offset is generated to avoid these false hits.

### Examples

```
SCP1 <- file2mesh(system.file("extdata","SCP1.ply",package="Morpho"))
viewpoints <- read.fcsv(system.file("extdata","SCP1_Endo.fcsv",package="Morpho"))
## Create a quick endocast
quickEndo <- virtualMeshScan(SCP1,viewpoints)
## Not run:
rgl::shade3d(quickEndo$visible,col="orange")
rgl::shade3d(SCP1,col="white",alpha=0.5)

## End(Not run)
```

## warpmovie3d

*Creates a sequence of images showing predefined steps of warping two meshes or landmark configurations (2D and 3D) into each other*

### Description

Creates a sequence of images showing predefined steps of warping two meshes or landmark configurations (2D and 3D) into each other

### Usage

```
warpmovie3d(
  x,
  y,
  n,
  col = "green",
  palindrome = FALSE,
  folder = NULL,
  movie = "warpmovie",
  ...
)

## S3 method for class 'matrix'
warpmovie3d(
  x,
  y,
  n,
  col = "green",
  palindrome = FALSE,
  folder = NULL,
  movie = "warpmovie",
  add = FALSE,
  close = TRUE,
  countbegin = 0,
  ask = TRUE,
  radius = NULL,
  links = NULL,
  lwd = 1,
  ...
)

warpmovie2d(
  x,
  y,
  n,
  col = "green",
  palindrome = FALSE,
  folder = NULL,
  movie = "warpmovie",
  links = NULL,
  lwd = 1,
  imagedim = "800x800",
  par = list(xaxt = "n", yaxt = "n", bty = "n"),
  ...
)

## S3 method for class 'mesh3d'
warpmovie3d(
  x,
  y,
  n,
  col = NULL,
  palindrome = FALSE,
  folder = NULL,
  movie = "warpmovie",
  add = FALSE,
  close = TRUE,
  countbegin = 0,
  ask = TRUE,
  radius = NULL,
  xland = NULL,
  yland = NULL,
  lmcol = "black",
  ...
)
```

## Arguments

| | |
|---|---|
| `x` | mesh to start with (object of class mesh3d) |
| `y` | resulting mesh (object of class mesh3d), having the same amount of vertices and faces than the starting mesh |
| `n` | integer: amount of intermediate steps. |
| `col` | color of the mesh |
| `palindrome` | logical: if TRUE, the procedure will go forth and back. |
| `folder` | character: output folder for created images (optional) |
| `movie` | character: name of the output files |
| `...` | additional arguments passed to `shade3d` (3D) or `points` (2D). |
| `add` | logical: if TRUE, the movie will be added to the focussed rgl-windows. |
| `close` | logical: if TRUE, the rgl window will be closed when finished. width and 200 the height of the image. |
| `countbegin` | integer: number to start image sequence. |
| `ask` | logical: if TRUE, the viewpoint can be selected manually. |

| | |
|---|---|
| `radius` | numeric: define size of spheres (overides atuomatic size estimation). |
| `links` | vector or list of vectors containing wireframe information to connect landmarks (optional). |
| `lwd` | numeric: controls width of lines defined by "links". |
| `imagedim` | character of pattern "100x200" where 100 determines the width and 200 the height of the image. |
| `par` | list of graphial parameters: details can be found here: `par`. |
| `xland` | optional argument: add landmarks on mesh x |
| `yland` | optional argument: add landmarks on mesh y |
| `lmcol` | optional argument: color of landmarks xland and yland |

## Details

given two landmark configurations or two meshes with the same amount of vertices and faces (e.g a mesh and its warped counterpart), the starting configuration/mesh will be subsequently transformed into the final configuration/mesh by splitting the differences into a predefined set of steps.

A series of png files will be saved to disk. These can be joined to animated gifs by external programs such as imagemagick or used to create animations in PDFs in a latex environment (e.g. latex package: aninmate).

## Author(s)

Stefan Schlager

## See Also

`ply2mesh`, `file2mesh`, `mesh2ply`, `tps3d`

## Examples

```
###3D example
 data(nose)##load data
if (interactive()){
##warp a mesh onto another landmark configuration:
longnose.mesh <- tps3d(shortnose.mesh,shortnose.lm,longnose.lm,threads=1)

warpmovie3d(shortnose.mesh,longnose.mesh,n=15)## create 15 images.

### ad some landmarks
warpmovie3d(shortnose.mesh,longnose.mesh,n=15,xland=shortnose.lm,
            yland=longnose.lm)## create 15 images.


### restrict to landmarks
warpmovie3d(shortnose.lm,longnose.lm,n=15,movie="matrixmovie")## create 15 images.

### the images are now stored in your current working directory and can
### be concatenated to a gif using an external program such as
### imagemagick.
}
### 2D example
if (require(shapes)) {
bb <- procSym(gorf.dat)
### morph superimposed first specimen onto sample mean
warpmovie2d(bb$rotated[,,1],bb$mshape,n=20,links=c(1,5,4:2,8:6,1),imagedim="600x400")
## remove files
unlink("warpmovie00*")
}
```

## write.fcsv *write fiducials in slicer4 format*

### Description

write fiducials in slicer4 format

### Usage

```
write.fcsv(x, filename = dataname, description = NULL, slicer4.11 = FALSE)
```

### Arguments

| | |
|---|---|
| `x` | matrix with row containing 2D or 3D coordinates |
| `filename` | will be substituted with ".fcsv" |
| `description` | optional: character vector containing a description for each landmark |
| `slicer4.11` | logical: Slicer changed their fiducial format in version >= 4.11. Set TRUE if you use the latest Slicer version |

### Examples

```
require(Rvcg)
data(dummyhead)
write.fcsv(dummyhead.lm)
## remove file
unlink("dummyhead.lm.fcsv")
```

## write.pts *exports a matrix containing landmarks into .pts format*

### Description

exports a matrix containing landmarks into .pts format that can be read by IDAV Landmark.

### Usage

```
write.pts(x, filename = dataname, rownames = NULL, NA.string = 9999)
```

### Arguments

| | |
|---|---|
| `x` | k x m matrix containing landmark configuration |
| `filename` | character: Path/name of the requested output - extension will be added atuomatically. If not specified, the file will be named as the exported object. |
| `rownames` | provide an optional character vector with rownames |
| `NA.string` | specify the string to use for encoding missing values |

### Details

you can import the information into the program landmarks available at http://graphics.idav.ucdavis.edu/research/EvoMorph

### Author(s)

Stefan Schlager

### See Also

`read.pts`

### Examples

```
data(nose)
write.pts(shortnose.lm, filename="shortnose")
unlink("shortnose.pts")
```

`write.slicerjson` *Export landmarks (or any 3D coordinates) to the new slicer json format*

## Description

Export landmarks (or any 3D coordinates) to the new slicer json format

## Usage

```
write.slicerjson(
  x,
  filename = dataname,
  type = c("Fiducial", "Curve", "ClosedCurve"),
  coordinateSystem = c("LPS", "RAS"),
  labels = dataname
)
```

## Arguments

| | |
|---|---|
| `x` | k x 3 matrix containing 3D coordinates |
| `filename` | will be substituted with ".mrk.json" |
| `type` | character: specify type of coordinates. Can be any of "Fiducial", "Curve", "ClosedCurve". |
| `coordinateSystem` | character: specify coordinate system the data are in. Can be "LPS" or "RAS". |
| `labels` | character or character vector containing landmark labels. |

# Index