Package ‘Morpho’

February 19, 2015

Type Package

Title Calculations and visualisations related to Geometric Morphometrics

Version 2.2

Date 2014-12-16

Author Stefan Schlager

Maintainer Stefan Schlager <stefan.schlager@uniklinik-freiburg.de>

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, MASS, shapes

Depends R (>= 3.0.2)

Imports Rvcg (>= 0.7), rgl (>= 0.93.963), colorRamps, foreach (>= 1.4.0), Matrix (>= 1.0-1), parallel, yaImpute, doParallel (>= 1.0.6), Rcpp

LinkingTo Rcpp, RcppArmadillo (>= 0.4)

Copyright see COPYRIGHTS file for details

License GPL-2

LazyLoad yes

URL http://sourceforge.net/projects/morpho-rpackage/, https://github.com/zarquon42b/Morpho

NeedsCompilation yes

Repository CRAN

Date/Publication 2014-12-16 13:19:39
R topics documented:

- Morpho-package: 4
- angle.calc: 5
- anonymize: 6
- applyTransform: 7
- arrMean3: 7
- asymPermuate: 8
- barycenter: 9
- bindArr: 10
- boneData: 11
- CAC: 11
- cExtract: 12
- checkLM: 13
- classify: 14
- closemeshKD: 15
- colors: 17
- computeTransform: 17
- covDist: 18
- covW: 20
- createAtlas: 21
- CreateL: 22
- createMissingList: 23
- crossProduct: 24
- cSize: 25
- cutMeshPlane: 26
- cutSpace: 26
- CVA: 27
- deformGrid2d: 31
- deformGrid3d: 32
- exVar: 33
- file2mesh: 34
- findOutliers: 35
- fixLMmirror: 36
- fixLMtps: 38
- getFaces: 39
- getTrafo4x4: 40
- getTrafoRotaxis: 40
- groupPCA: 41
- histGroup: 43
- icpMat: 44
- invertFaces: 45
- kendalldist: 46
- lineplot: 47
- mcNNindex: 48
- meanMat: 49
- mergeMeshes: 49
- mesh2grey: 50
<table>
<thead>
<tr>
<th>R topics documented:</th>
</tr>
</thead>
<tbody>
<tr>
<td>mesh2obj</td>
</tr>
<tr>
<td>meshcube</td>
</tr>
<tr>
<td>meshDist.matrix</td>
</tr>
<tr>
<td>meshPlaneIntersect</td>
</tr>
<tr>
<td>meshres</td>
</tr>
<tr>
<td>mirror</td>
</tr>
<tr>
<td>name2factor</td>
</tr>
<tr>
<td>NNshapeReg</td>
</tr>
<tr>
<td>nose</td>
</tr>
<tr>
<td>pcAlign</td>
</tr>
<tr>
<td>pcaplot3d</td>
</tr>
<tr>
<td>PCdist</td>
</tr>
<tr>
<td>permudist</td>
</tr>
<tr>
<td>permuvvec</td>
</tr>
<tr>
<td>placePatch</td>
</tr>
<tr>
<td>plotAtlas</td>
</tr>
<tr>
<td>plotNormals</td>
</tr>
<tr>
<td>pls2B</td>
</tr>
<tr>
<td>points2plane</td>
</tr>
<tr>
<td>predictShape.lm</td>
</tr>
<tr>
<td>proc.weight</td>
</tr>
<tr>
<td>procAOVsym</td>
</tr>
<tr>
<td>ProcGPA</td>
</tr>
<tr>
<td>procSym</td>
</tr>
<tr>
<td>projRead</td>
</tr>
<tr>
<td>qqmat</td>
</tr>
<tr>
<td>quad2trimesh</td>
</tr>
<tr>
<td>r2morphoj</td>
</tr>
<tr>
<td>ray2mesh</td>
</tr>
<tr>
<td>read.csv.folder</td>
</tr>
<tr>
<td>read.lmdta</td>
</tr>
<tr>
<td>read.mpp</td>
</tr>
<tr>
<td>read.pts</td>
</tr>
<tr>
<td>readallTPS</td>
</tr>
<tr>
<td>readLandmarks.csv</td>
</tr>
<tr>
<td>regdist</td>
</tr>
<tr>
<td>RegScore</td>
</tr>
<tr>
<td>relaxLM</td>
</tr>
<tr>
<td>relWarp</td>
</tr>
<tr>
<td>render.matrixDist</td>
</tr>
<tr>
<td>retroDeform3d</td>
</tr>
<tr>
<td>retroDeformMesh</td>
</tr>
<tr>
<td>rotaxis3d</td>
</tr>
<tr>
<td>rotaxisMat</td>
</tr>
<tr>
<td>rotmesh.onto</td>
</tr>
<tr>
<td>rotonmat</td>
</tr>
<tr>
<td>rotonto</td>
</tr>
<tr>
<td>scalesmesh</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Package:</th>
<th>Morpho</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>2.2</td>
</tr>
<tr>
<td>Date:</td>
<td>2014-12-16</td>
</tr>
<tr>
<td>License:</td>
<td>GPL</td>
</tr>
<tr>
<td>LazyLoad:</td>
<td>yes</td>
</tr>
</tbody>
</table>

Note

The pdf-version of Morpho-help can be obtained from CRAN on http://cran.r-project.org/web/packages/Morpho/Morpho.pdf

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

Author(s)

Stefan Schlager <stefan.schlager@uniklinik-freiburg.de>
Maintainer: Stefan Schlager <stefan.schlager@uniklinik-freiburg.de>

References


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)
anonymize  

*Replace ID-strings of data and associated files.*

**Description**

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

**Usage**

```r
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**

```r
anonymize(iris, remove=1)
```
**applyTransform**  
*apply affine transformation to data*

### Description
apply affine transformation to data

### Usage
applyTransform(x, trafo, inverse)

```r
## S3 method for class 'matrix'
applyTransform(x, trafo, inverse = FALSE)
```

```r
## S3 method for class 'mesh3d'
applyTransform(x, trafo, inverse = FALSE)
```

### Arguments
- **x**: matrix or mesh3d
- **trafo**: 4x4 transformation matrix (for mesh3d the matrix will be transformed to a 4x4 matrix)
- **inverse**: logical: if TRUE, the inverse of the transformation is applied

### Value
the transformed object

### Examples
```r
data(boneData)
rot <- rotonto(boneLM[,1],boneLM[,2])
trafo <- getTrafo4x4(rot)
boneM2trafo <- applyTransform(boneLM[,2],trafo)
```

**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**

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

---

**asymPermute**

Assess differences in amount and direction of asymmetric variation

**Description**

Assess differences in amount and direction of asymmetric variation

**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 lengths
barycenter

permangle 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. If there is no or only very little directional asymmetry present, the angles will only be significant when larger than 90 degrees (pi/2). So careful interpretation is advised.

See Also

procSym

definition

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

require(rgl)
data(nose)
bary <- barycenter(shortnose.mesh)
## Not run:
##visualize mesh
wire3d(shortnose.mesh)
# visualize barycenters
bindArr

**concatenate multiple arrays/matrices**

### Description

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

### Usage

```r
bindArr(..., along = 1)
```

### Arguments

- `along` dimension along which to concatenate.
- `...` matrices and/or arrays with appropriate dimensionality to combine to one array, or a single list containing suitable matrices, or arrays).

### 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

```r
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


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 <- showPC(max(cac$CACscores), cac$CAC, proc$mshape)
small <- showPC(min(cac$CACscores), cac$CAC, proc$mshape)
deformGrid3d(small, large, ngrid=8)

## End(Not run)

cExtract

extract information about fixed landmarks, curves and patches from an 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**

```r
checkLM(dat.array, path = NULL, prefix = "", suffix = ".ply",
        col = "white", pt.size = NULL, alpha = 0.7, begin = 1,
        render = c("w", "s"), point = c("s", "p"), add = FALSE, Rdata = FALSE,
        atlas = NULL, text.lm = FALSE)
```

**Arguments**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dat.array</td>
<td>array or list containing landmark coordinates.</td>
</tr>
<tr>
<td>path</td>
<td>optional character: path to files where surface meshes are stored locally. If not specified only landmarks are displayed.</td>
</tr>
<tr>
<td>prefix</td>
<td>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)</td>
</tr>
<tr>
<td>suffix</td>
<td>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)</td>
</tr>
<tr>
<td>col</td>
<td>mesh color</td>
</tr>
<tr>
<td>pt.size</td>
<td>size of plotted points/spheres. If point=&quot;s&quot;, pt.size defines the radius of the spheres. If point=&quot;p&quot; it sets the variable size used in point3d.</td>
</tr>
<tr>
<td>alpha</td>
<td>value between 0 and 1. Sets transparency of mesh 1=opaque 0= fully transparent.</td>
</tr>
<tr>
<td>begin</td>
<td>integer: select a specimen to start with.</td>
</tr>
<tr>
<td>render</td>
<td>if render=&quot;w&quot;, a wireframe will be drawn, else the meshes will be shaded.</td>
</tr>
<tr>
<td>point</td>
<td>how to render landmarks. &quot;s&quot;=spheres, &quot;p&quot;=points.</td>
</tr>
<tr>
<td>add</td>
<td>logical: add to existing rgl window.</td>
</tr>
<tr>
<td>Rdata</td>
<td>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 <code>file2mesh</code>.</td>
</tr>
<tr>
<td>atlas</td>
<td>provide object generated by <code>createAtlas</code> to specify coloring of surface patches, curves and landmarks</td>
</tr>
<tr>
<td>text.lm</td>
<td>logical: number landmarks. Only applicable when atlas=NULL.</td>
</tr>
</tbody>
</table>
classify specimen based on between-group PCA or CVA

declare

classify specimen based on between-group PCA or CVA
closemeshKD

Usage

classify(x, cv = TRUE)

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

## S3 method for class 'CVA'
classify(x, cv = T)

Arguments

- `x`: result of groupPCA or CVA
- `cv`: logical: use cross-validated scores if available

Value

- `class`: classification result
- `groups`: original grouping variable
  for object of CVA, also the posterior probabilities are returned.

---

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 closest 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 homologous 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


See Also

`ply2mesh`

Examples

```r
require(rgl)
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

**computeTransform**

calculate an affine transformation matrix

**Description**

calculate an affine transformation matrix

**Usage**

```r
computeTransform(x, y, type = c("rigid", "similarity", "affine"),
                 reflection = FALSE)
```

**Arguments**

- `x`: fix landmarks
- `y`: moving landmarks
- `type`: set type of affine transformation: options are "rigid", "similarity" (rigid + scale) and "affine".
- `reflection`: logical: if TRUE "rigid" and "similarity" allow reflections.

**Value**

returns a 4x4 (3x3 in 2D case) transformation matrix
Examples

```r
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

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


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$p.matrix # differences between covariance matrices
# are significant
## visualize covariance ellipses of first 5 PCs of shape
covW(proc$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$boot.data[,1:2]), xlim=range(cpca$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$boot.data[i,1:2,]), col=i)
  points(t(cpca$PCscores[i,1,]), col=1, pch=8, cex=1.5)
}

---

**covW**

*calculate the pooled within groups covariance matrix*

### Description

calculate the pooled within groups covariance matrix

### Usage

covW(data, groups)

### Arguments

data a matrix containing data
groups grouping variables

### Value

Returns the pooled within group covariance matrix

### Author(s)

Stefan Schlager

### See Also

cov, typprobClass
createAtlas

**Examples**

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

---

**createAtlas**

*Create an atlas needed in placePatch*

---

**Description**

Create an atlas needed in placePatch

**Usage**

```r
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 specifying 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 specify `corrCurves = list(c(2:4), c(5:10))`.
- `patchCurves` a vector or a list containing vectors specifying 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 specify `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`
CreateL

Create Matrices necessary for Thin-Plate Spline

Description

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

Usage

CreateL(matrix, lambda = 0, output = c("L", "Linv", "Lsubk", "Lsubk3"))

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: "L", "Linv", "Lsubk", "Lsubk3". sliding of semilandmarks.

Value

depending on the choices in output:

- **L**: Matrix L as specified in Bookstein (1989)
- **Linv**: Inverse of matrix L as specified in Bookstein (1989)
- **Lsubk**: upper left k x k submatrix of Linv
- **Lsubk3**: Matrix used for sliding in slider3d and relaxLM. Only available if blockdiag = TRUE

Note

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

References


CreateMissingList

create MissingList

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

require(rgl)
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))
Examples

```r
## 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

```r
crossProduct(x, y)
tangentPlane(x)
```

Arguments

- `x`: vector of length 3.
- `y`: vector of length 3.

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
cSize

calculate Centroid Size for a landmark configuration

Description

calculate Centroid Size for a landmark configuration

Usage

cSize(x)

Arguments

x matrix where each row contains coordinates for landmarks

Value

returns Centroid size

Examples

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

*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

*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 position 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

```r
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)

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 number of observations and m the number of variables (this can be PC scores for example)
groups a character/factor vector containing 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 withingroup covariance matrix.
plot Logical: determines whether in the two-sample case a histogram is 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.

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 posterior 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


See Also

groupPCA

Examples

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

library(shapes)
# perform procrustes fit on raw data
alldat<-procSym(abind(gorf.dat,gorm.dat))
# create factors
groups<-as.factor(c(repl("female",30),repl("male",29)))
# perform CVA and test Mahalanobis distance
# between groups with permutation test by 100 rounds
cvall<-CVA(alldat$orddata,groups,rounds=10000)
## visualize a shape change from score 1 to 5:
cvvis5 <- 5*matrix(cvall$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)
# 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$CVscores[facto==ii,1:2])
    lines(rbind(kk,cbind(kk[1,],kk[2,][kk<1])))
    }
```
lines(cva.1$CVscores[fac==ii,1][c(kk, kk[1])]),
cva.1$CVscores[fac==ii,2][c(kk, kk[1]), col=jj]
)

# add 80% ellipses
require(car)
for(ii in 1:length(levels(facto))){
  dataEllipse(cva.1$CVscores[fac==levels(facto)[ii],1],
  cva.1$CVscores[fac==levels(facto)[ii],2],
  add=TRUE,levels=.80, col=c(1:7)[ii])
}

# histogram per group
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
dendoS=hclust(cva.1$Dist$GroupdistMaha)
dendoS$labels=levels(facto)
par(mar=c(4,4.5,1,1))
dendoS=as.dendrogram(dendoS)
plot(dendoS, 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$Grandm
cvvis5 <- showPC(cvvis5,proc$PCs[,1:5],proc$mshape)
### deformGrid2d

*visualise differences between two superimposed sets of 2D landmarks*

---

**Description**

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

**Usage**

```r
deformGrid2d(matrix, tarmatrix, ngrid = 0, lwd = 1, show = c(1:2),
              lines = TRUE, lcol = 1, col1 = 2, col2 = 3, pcaxis = FALSE, ...)
```

**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
- `col1`: color of "matrix"
- `col2`: color of "tarmat"
- `pcaxis`: logical: align grid by shape’s principal axes.
- `...`: additional parameters passed to plot

**Author(s)**

Stefan Schlager

**See Also**

tps3d
Examples

```r
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**

```r
deformGrid3d(matrix, tarmatrix, ngrid = 0, 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)
```

**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.
- `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.

**Author(s)**

Stefan Schlager
exVar

See Also
tps3d

Examples

```r
## Not run:
data(nose)
deformGrid3d(shortnose.lm,longnose.lm,ngrid=10)

## End(Not run)
```

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

Description

calculates a quotient of the overall variance 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")
- `ncomp`: How many latent variables to use (only for mvr models)
- `val`: use cross-validated predictions (only for mvr models)
- `...`: currently unused additional arguments.

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 performed by the R-package "ffmanova", which reports this value factor-wise.
Author(s)

Stefan Schlager

References


Examples

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

---

**file2mesh**

*Import 3D surface mesh files*

**Description**

Import 3D surface mesh files

**Usage**

```r
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**

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

mesh2obj(shortnose.mesh)
mesh2 <- obj2mesh("shortnose.mesh.obj")
```

**find.outliers**

*Graphical interface to find outliers and/or to switch mislabeled landmarks*

**Description**

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

**Usage**

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

**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

**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.clean` 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
## Not run:
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

## End(Not run)
fixLMmirror

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.

Details

The configurations are mirrored and the relabeled version is matched onto the original using a thin-plate spline deformation. The missing landmark is now estimated using its bilateral counterpart.

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)
fixLMtps

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)

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.

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
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
getTrafoRotaxis

Value

vector of face indices

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

---

**groupPCA**

*Perform PCA based of the group means’ covariance matrix*

Description

Calculate covariance matrix of the group means 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

```r
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 number of observations and m the number of variables (this can be PC scores for example)
groups a character/factor vector containing 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.
groupPCA

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 variance explained by each eigenvalue
- **Scores**: Scores of all observations in the PC-space
- **probs**: P-values of pairwise group differences - based on permutation testing
- **groupdists**: Euclidean distances between groups' averages
- **groupmeans**: Group means
- **Grandmean**: Grand mean
- **CV**: Cross-validated scores
- **groups**: Grouping Variable

Author(s)

Stefan Schlager

References


See Also

- **CVA**

Examples

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

### plot scores
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)
```
histGroup

```r
## calculate matrix containing landmarks of grandmean
grandmean <- matrix(gpca$Grandmean, dims[1], dims[2])
## calculate landmarks from first between-group PC
# (+2 and -2 standard deviations)
gpcavis2sd<- showPC(2*sd(gpca$Scores[,1]), gpca$groupPCs, grandmean)
gpcavis2sd.neg<- showPC(-2*sd(gpca$Scores[,1]), gpca$groupPCs, grandmean)
deformGrid3d(gpcavis2sd, gpcavis2sd.neg, ngrid = 0)
require(rgl)
## visualize grandmean mesh

grandm.mesh <- warp.mesh(skull_0144_ch_fe.mesh, boneLM[,1], grandmean)
wire3d(grandm.mesh, col="white")
spheres3d(grandmean, radius=0.005)

## End(Not run)
```

### histGroup

**plot histogram for multiple groups.**

**Description**

plot a histogram for multiple groups, each group colored individually

**Usage**

```r
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, scale = 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
scale: logical: if TRUE, scaling is allowed
invertFaces

Value

returns the rotated landmarks

Examples

data(nose)
icp <- icpmat(shortnose.lm, longnose.lm, iterations=10, subsample = 20)

## 2D example using icpmat to determine point correspondences
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
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

library(shapes)
OPA <- rotonto(gorf.dat[,1],gorf.dat[,2])
kendalldist(OPA$X,OPA$Y)
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, 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
add logical: add to existing plot

Note

works with 2D and 3D configurations

Author(s)

Stefan Schlager

See Also

pcaplot3d

Examples

require(rgl)
library(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:
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)
mcNNindex  

*find nearest neighbours for point clouds*

**Description**

find nearest neighbours for point clouds by using algorithms from the ANN library. This is just a wrapper for the function ann from the package yaImpute, enabling parallel processing.

**Usage**

```r
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. Speed benefits are only relevant for `k > 20`
- `k`: integer: how many closest points are sought.
- `...`: additional arguments - currently unused.

**Details**

wraps the function `ann` from package `yaImpute` to allow multicore processing

**Value**

1 x k matrix containing indices of closest points.

**See Also**

closemeshKD

**Examples**

```r
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)
```
meanMat

meanMat  fast calculation of a Matrix' per row/ per column mean - useful for very large matrices

Description

fast calculation of a Matrix’ per row/ per column mean - equivalent to apply(X,2,mean) or apply(X,1,mean)-useful for very large matrices

Usage

meanMat(A, usedim = 2)

Arguments

A numeric matrix
usedim integer: select over which dimension to average

Value

vector containing row/column mean

Examples

A <- matrix(rnorm(1e6),1000,1000)
b <- meanMat(A)
# same as apply(A,2,mean)
b1 <- meanMat(A,1)
# same as apply(A,1,mean)
## Not run:
#compare timing
system.time(meanMat(A))
system.time(apply(A,2,mean))
## 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

```r
require(rgl)
data(boneData)
data(nose)
mergedMesh <- mergeMeshes(shortnose.mesh, skull_0144_ch_fe.mesh)
## Not run:
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
Description

export mesh objects to disk.

Usage

```r
mesh2obj(x, filename = dataname)

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 automatically. If not specified, the file will be named as the exported object.
- **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.
- **writeNormals**: logical: if TRUE, existing normals of a mesh are written to file - can slow things down for very large meshes.

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.

Author(s)

Stefan Schlager

See Also

- `ply2mesh`, `quad2trimesh`
Examples

```r
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,-1.8,1.0,1.8,
      -1.8,1.8,1.0,-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
```

### meshcube

**calculate the corners of a mesh’s bounding box**

calculate the corners of a mesh’s bounding box

Usage

```r
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

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

## End(Not run)
```
meshDist.matrix
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

## S3 method for class 'matrix'

```r
meshDist(x, mesh2 = NULL, distvec = NULL, from = NULL,
to = NULL, steps = 20, ceiling = FALSE, uprange = 1, plot = TRUE,
sign = TRUE, tol = NULL, type = c("s", "p"), radius = NULL,
displace = FALSE, add = FALSE, ...)
```

```r
meshDist(x, ...)
```

## S3 method for class 'mesh3d'

```r
meshDist(x, mesh2 = NULL, distvec = NULL, from = NULL,
to = NULL, steps = 20, ceiling = FALSE, file = "default",
imagedim = "100x100", uprange = 1, ray = FALSE, raytol = 50,
save = FALSE, plot = TRUE, sign = TRUE, tol = NULL,
displace = FALSE, shade = TRUE, method = c("vcglib", "morpho"),
add = FALSE, ...)
```

Arguments

- **x**: reference mesh; object of class "mesh3d" or a n x 3 matrix containing 3D coordinates.
- **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 of mesh1, if distvec != NULL, x 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
- **uprange**: numeric between 0 and 1: restricts "to" to a quantile of "to", if to is NULL.
- **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.
meshDist.matrix

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.
displace logical: if TRUE, displacement vectors between original and closest points are drawn colored according to the distance.
add logical: if TRUE, visualization will be added to the rgl window currently in focus
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.
ray logical: if TRUE, the search is along vertex normals.
raytol maximum distance to follow a normal.
save logical: save a colored mesh.
shade logical: if FALSE, the rendering of the colored surface will be suppressed.
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.
... additional arguments passed to shade3d. See rgl.material for details.

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
this function needs the command line tools from the Auxiliaries section in http://sourceforge.net/projects/morpho-rpackage/files/Auxiliaries installed.

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:
meshPlaneIntersect

See Also

render.meshDist, export.meshDist, shade3d

Examples

 require(rgl)
 data(nose)# load data
 ## warp a mesh onto another landmark configuration:
 warpnose.long <- warp.mesh(shortnose.mesh, shortnose.lm, longnose.lm)
 ## Not run:
 meshDist(warpnose.long, shortnose.mesh, method="m")

 ## End(Not run)
 # use signed distances and
 # color distances < 0.01 green:
 # Not run:
 meshDist(warpnose.long, shortnose.mesh, sign=TRUE, tol=0.01, method="m")
 
 ## End(Not run)

meshPlaneIntersect  get intersections between mesh and a plane

Description

get intersections between mesh and a plane

Usage

meshPlaneIntersect(mesh, v1, v2, v3)

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

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)
Description

mirror landmarks or triangular mesh in place

Usage

mirror(x, icpiter = 50, subsample = NULL)

## S3 method for class 'matrix'
mirror(x, icpiter = 50, subsample = NULL)

## S3 method for class 'mesh3d'
mirror(x, icpiter = 50, subsample = NULL)

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

Details

reflect a mesh configuration at the plane spanned by its first 2 principal axis, then try to rigidly register the reflected configuration onto the original one using iterative closest point search to establish correspondences.

Value

returns the reflected object

Examples

data(boneData)
boneMir <- mirror(boneLM[,1],icpiter=50)
## 2D Example:
require(shapes)
gorfMir <- mirror(gorf.dat[,1])
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)
## compare result to original
wire3d(skull_0144_ch_fe.mesh,col=3)
name2factor

Description

extract data from array names

Usage

name2factor(x, sep = ",", which, collapse = sep)

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

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
NNshapeReg

Examples

```r
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 from Mahalanobis or Procrustes distances. This can be useful for data with missing landmarks.

**Usage**

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

**Arguments**

- **x**: an array or matrix (one row per specimen) with data used for estimating weights.
- **y**: an array or matrix (one row per specimen) 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

```r
library(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

```r
pcAlign(x, y, optim = TRUE, subsample = NULL)

## S3 method for class 'matrix'
pcAlign(x, y, optim = TRUE, subsample = NULL)

## S3 method for class 'mesh3d'
pcAlign(x, y, optim = TRUE, subsample = NULL)
```
pcaplot3d

**Arguments**

- **x**: matrix or mesh3d
- **y**: matrix or mesh3d, if missing, x will be centered by its centroid and aligned by its principal axis.
- **optim**: logical, if TRUE, all possible PC-axis are tested and the rotation with the smallest RMSE between configs will be used.
- **subsample**: integer, use subsampled points to decrease computation time

**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**

```r
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**

```r
pcaplot3d(x, ...)
```

## S3 method for class 'symproc'
```r
cpacplot3d(x, pcshow = c(1, 2, 3), mag = 3, color = 4,
        lwd = 1, sym = TRUE, ...)
```

## S3 method for class 'nosymproc'
```r
cpacplot3d(x, pcshow = c(1, 2, 3), mag = 3, color = 4,
        lwd = 1, ...)
```
Arguments

- **x**: a object derived from the function procSym calculated on 3D coordinates.
- **pcscores**: a vector containing the PC scores 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.
- **...**: Additional parameters which will be passed to the methods.

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

```r
cC not run:
data(nose)
# make a tiny sample
nosearr <- bindArr(longnose.lm, shortnose.lm, along=3)
proc <- procSym(nosearr)
pcaplot3d(proc, pcscores=1, mag=3)# only one PC available

cC end not run
```

### Description

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

### Usage

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

Arguments

- **PCs**: \( m \times k \) matrix of Principal Components where \( m \) is the number of PCs.
- **PCscores**: \( n \times m \) 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

```r
library(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

```r
permudist(data, groups, rounds = 1000, which = NULL)
```

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 determines which factor-levels to use
permuvec

**Value**

- `dist`: distance matrix with distances between actual group means
- `p.value`: distance matrix containing pairwise p-values obtained by comparing the actual distance to randomly acquired distances

**Examples**

```r
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)
```

---

**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**

```r
permuvec(data, groups, subgroups = NULL, rounds = 10000, 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 covariance 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

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>angle</td>
<td>angle between the vectors of the subgroups means</td>
</tr>
<tr>
<td>dist</td>
<td>distances between subgroups</td>
</tr>
<tr>
<td>meanvec</td>
<td>matrix containing the means of all four subgroups</td>
</tr>
<tr>
<td>permutangles</td>
<td>vector containing angles (in radians) from random permutation</td>
</tr>
<tr>
<td>permudists</td>
<td>vector containing distances from random permutation</td>
</tr>
<tr>
<td>p.angle</td>
<td>p-value of angle between residual vectors</td>
</tr>
<tr>
<td>p.dist</td>
<td>p-value of length difference between residual vectors</td>
</tr>
<tr>
<td>subdist</td>
<td>length of residual vectors connecting the subgroups</td>
</tr>
</tbody>
</table>

Examples

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

## visualize if the amount of sexual dimorphism differs between
## (lengths 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 erroneous 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.ores 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


See Also

createAtlas, relaxLM, checkLM, slider3d, warp.mesh

Examples

```r
## Not run:
data(nose)
require(rgl)
### create mesh for longnose
longnose.mesh <- warp.mesh(shortnose.mesh, shortnose.lm, longnose.lm)
## create atlas
fix <- c(1:5, 20:21)
```
plotAtlas <- 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)

---

**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.
plotNormals

   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,

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 x, long = 1, lwd = 1, col = 1

Arguments

   x           object of class "mesh3d"
long         length of the normals (default is 1)
lwd          width of the normals
col          color of the normals

Description

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

Usage

plotNormals(x, long = 1, lwd = 1, col = 1)

plots the normals of a triangular surface mesh.
Author(s)

Stefan Schlager

Examples

```r
## Not run:
require(rgl)
data(nose)
plotNormals(shortnose.mesh,col=4,long=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

```r
pls2B(x, y, tol = 1e-12, same.config = FALSE, rounds = 0,
    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.
- **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

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 landmarks of block x are replaced by corresponding landmarks of other specimen.

Value

svd  
PLS-scores of x
PLS-scores of y
Dataframe containing singular values, explained covariation, correlation coefficient between PLS-scores and p-values

Author(s)

Stefan Schlager

References


See Also

svd

Examples

library(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$rotated[,4,,],proc$rotated[,5:8,,],
  same.config=TRUE,rounds=50,mc.cores=2)

## End(Not run)
pls1 <- pls2B(proc$rotated[,4,,],proc$rotated[,5:8,,],
  same.config=FALSE,rounds=50,mc.cores=1)
pls1$CoVar
layout(matrix(1:4,2,2,byrow=TRUE))
for(i in 1:4)
  plot(pls1$scores[,i]-pls1$Yscores[,i])
## Description
projects a 3D coordinate orthogonally onto a plane

## Usage
```r
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
```r
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])
```

```r
## 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:
predictShape.lm

spheres3d(points2plane(boneLM[9,1],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)

---

**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 = 5)

## 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 male for 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)

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.

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

library(shapes)
proc <- procSym(gorf.dat)
## get weights for 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
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


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)
procAOVs(y(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 = FALSE, 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 speci-
mens, 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

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)

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, tol = 1e-05, pairedLM = NULL, sizeshape = FALSE,  
use.lm = NULL, center.part = 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
reflect logical: allow reflections.
CSinit logical: if TRUE, all configurations are initially scaled to Unit Centroid Size.
procSym

orp logical: if TRUE, an orthogonal projection at the meanshape into tangent space is performed.
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.
pcaAlign 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 there 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**: $k \times m \times n$ array of the rotated configurations
- **Sym**: $k \times m \times n$ array of the Symmetrical component - only available for the "Symmetry"-Option (when pairedLM is defined)
- **Asym**: $k \times m \times n$ array of the Asymmetrical component - only available for the "Symmetry"-Option (when pairedLM is defined)
- **asymmean**: $k \times m$ matrix of mean asymmetric deviation from symmetric mean
- **msshape**: sample meanshape
- **symmean**: meanshape of symmetrized configurations
- **tan**: if orp=TRUE: Residuals in tangentspace else, Procrustes residuals - only available without the "Symmetry"-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
- **eigenasymp**: 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**: $k \times m \times n$ 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

**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


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
capsym3d(symproc, sym=FALSE)

## visualize distribution of symmetric PCScores population
pop <- name2factor(boneLM, which=3)
require(car)
spm(~symproc$PCscore_sym[,1:5], groups=pop)
## visualize distribution of asymmetric PCScores population
spm(~symproc$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: return 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:


**See Also**

closeshellKD
QQMAT

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 Mahalanobis distances of a given sample against those expected from a Gaussian distribution.

Usage

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

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

```r
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

```r
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

```r
Sigma <- diag(3:1)  # create a 3D-covariance matrix
require(rgl)
quadmesh <- ellipse3d(Sigma)# create quadmesh
trimesh <- quad2trimesh(quadmesh)# convert to trimesh
```
r2morphoj

---

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

---

**Description**

Export data to MorphoJ and Morphologika

**Usage**

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

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

**Arguments**

- `x`  
  3-dimensional 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**

```r
library(shapes)
r2morphoj(gorf.dat, file="gorf.dat")
```

```r
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")
```
**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**

```r
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**

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

**Arguments**

- `folder` character: path to folder
- `x` either a vector specifying which rows are to be imported, or character vector containing variable names to be sought for.
- `y` a vector specifying which columns of the spreadsheet is to be imported.
- `rownames` integer: specifies columns where variable names are stored.
- `header` logical: if spreadsheet contains header-row.
- `dec` character: defines decimal separator.
- `sep` character: defines column separator.
- `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.lmdta  

**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  

**Description**
Read saved pick-points from meshlab

**Usage**
read.mpp(file, info = FALSE)

**Arguments**
- **file**: file to import
- **info**: logical: if TRUE, additional infos are returned
read.pts

Value

if info=FALSE:
a matrix containing picked-points coordinates
if info=TRUE: a list containing

data matrix containing coordinates
info additional info contained in file

Author(s)

Stefan Schlager

See Also

read.pts

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.pst(shortnose.lm, filename="shortnose")
data <- read.pst("shortnose.pst")
readallTPS  
*Import landmarks and outlines from TPS files*

**Description**

Imports outlines and landmarks from files generated by tpsdig2

**Usage**

```r
readallTPS(file)
```

**Arguments**

- `file`  
  A TPS-file generated by tpsdig2

**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

**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`
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 specifying which rows are to be imported, or character vector
    containing variable names to be sought for.

y a vector specifying, which columns of the spreadsheet 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 sepearator.

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.

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

Author(s)

Stefan Schlager

See Also

regdist

Examples

library(shapes)
regdist(gorf.dat)
RegScore

calculate regression scores for linear model

Description

calculate 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


Examples

```r
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:

```r
# generate a factor with 4 levels
m.lm <- lm(proc$PCscores ~ pop.size)
# scores associated with proc$size
rs <- RegScore(m.lm)

## validate by using a subsample for fitting
```n
```r
rand <- sample(dim(m.lm)[3])
```n
```r
m.lm0 <- lm(proc$PCscores[rand[1:50],] ~ proc$size[rand[1:50]])
```n
```r
rs0 <- RegScore(m.lm0[, proc$PCscores[rand[-c(1:50),] ] ]
plot(rs0[, proc$size[rand[-c(1:50)]]])
```n
## 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**

```r
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))
```

**Arguments**

- **lm**
  - k x 3 or k x 2 matrix containing landmark data to be slidden.

- **reference**
  - k x 3 or k x 2 matrix containing landmark of the reference

- **SMvector**
  - A vector containing the row indices of (semi-) landmarks on the curve(s) that are allowed to slide

- **outlines**
  - A vector (or if there 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.

- **mesh**
  - triangular mesh of class "mesh3d" loaded into the R workspace, when specified, "sur.name" has to be NULL. The function `closeshell3D` will be used for reprojection onto the surface.
tol numeric: Threshold for convergence in the sliding process. 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 $\mathbf{Y} = \mathbf{Y}^0 + \text{stepsize} \times \mathbf{U}^T$. Default is set to 1 for bending=TRUE and 0.5 for bending=FALSE.

Value
returns kx3 matrix of slidden landmarks

Author(s)
Stefan Schlager

References

See Also
slider3d

Examples
```r
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
```
relWarp <- 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)

rot2ref <- 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")

## End(Not run)

---

relWarp  

*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.
relWarps

Usage

relWarps(data, scale = TRUE, CSinit = TRUE, alpha = 1, tol = 1e-10, orp = TRUE)

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 emphasised, 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.

Value

bescores: relative warp scores
uniscores: uniform scores
Var: non-affine variation explained by each relative warp
mshape: sample’s consensus shape
rotated: Procrustes superimposed data
bePCs: vector basis of nonaffine shape variation - relative warps
uniPCs: vector basis of affine shape variation - uniform component

Author(s)

Stefan Schlager

References


Examples

data(boneData)
pop <- name2factor(boneLM,which=3)
rW <- relWarps(boneLM, alpha = -1)
## Not run:
require(car)
# plot first 5 relative warps scores grouped by population
render.matrixDist

plot or save the results of meshDist

Description

plot or save the results of meshDist

Usage

## S3 method for class 'matrixDist'
render(x, from = NULL, to = NULL, steps = NULL,
       ceiling = NULL, uprange = NULL, tol = NULL, type = c("s", "p"),
       radius = NULL, displace = FALSE, sign = NULL, add = FALSE, ...)

render(x, ...)

## S3 method for class 'meshDist'
render(x, from = NULL, to = NULL, steps = NULL,
       ceiling = NULL, uprange = NULL, tol = NULL, displace = FALSE,
       shade = TRUE, sign = NULL, add = FALSE, ...)

export(x, ...)

## S3 method for class 'meshDist'
export(x, file = "default", imagedim = "100x800", ...)
Arguments

- **x**
  - object of class meshDist
- **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 green.
- **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.
- **displace**
  - logical: if TRUE, displacement vectors between original and closest points are drawn colored according to the distance.
- **sign**
  - logical: request signed distances to be visualised.
- **add**
  - logical: if TRUE, visualization will be added to the rgl window currently in focus
- **shade**
  - logical: if FALSE, the rendering of the colored surface will be supressed.
- **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.
- **...**
  - for render.meshDist: additional arguments passed to shade3d. See rgl.material for details.

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
**Description**

symmetrize a bilateral landmark configuration by removing bending and stretching

**Usage**

```r
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`: damping factor for calculating local weights
- `alpha`: factor controlling spacing along x-axis

**Value**

- `deformed`: matrix containing deformed landmarks
- `orig`: matrix containing original landmarks

**References**


---

**Description**

symmetrize a triangular mesh

**Usage**

```r
gretroDeformMesh(mesh, mat, pairedLM, hmult = 5, alpha = 0.01, rot = TRUE, lambda = 0)
```
rotaxis3d

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
- **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

Details

This function performs retroDeform3d and deforms the mesh accordingly using the function warp.mesh.

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

```r
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)
```

<table>
<thead>
<tr>
<th>rotaxisMat</th>
<th>calculate a rotation matrix around an arbitrary axis through the origin in 3D</th>
</tr>
</thead>
</table>

Description

calculate a rotation matrix around an arbitrary axis in 3D

Usage

```
rotaxisMat(u, theta, homogeneous = FALSE)
```
rotmesh.onto

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.

Value

- **mesh** rotated mesh
- **yrot** rotated refmat
- **trafo** 4x4 transformation matrix
Author(s)

Stefan Schlager

See Also

file2mesh, warp.mesh, rotonto, mesh2ply

Examples

```r
require(rgl)
data(boneData)
## rotate, translate and scale the mesh belonging to the first specimen
## onto the landmark configuration of the 10th specimen
rotmesh <- rotonmesh(to(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)
```

Description

rotate matrix of landmarks by using a rotation determined by two matrices.

Usage

```r
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 centerweights=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 useful, 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 another using Procrustes fitting

Description
rotates, translates and scales one matrix onto another 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` \( k \times m \) matrix to be rotated onto (target matrix)
- `y` \( k \times m \) 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: if weights are defined and centerweights=TRUE, the matrix will be centered according to these weights instead of the barycenter.
- `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 \( \text{reflect} = 1 \), reflections are involved in the superimposition. Else, \( \text{reflect} = 0 \)

## Author(s)

Stefan Schlager

## References

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.oneto
Examples

data(nose)
  # inflate mesh by factor 4
  largenose <- scalemesh(shortnose.mesh, 4)

showPC(scores, PC, mshape)

Arguments

scores vector of 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)

Details

Rotates and translates PC-scores derived from shape data back into configuration space.

Value

returns matrix containing landmarks

Author(s)

Stefan Schlager

See Also

prcomp, procSym
Examples

```r
library(shapes)
## generate landmarks using
## the first PC-score of the first specimen

proc <- procSym(gorf.dat)
lm <- showPC(proc$PCscores[1,1],proc$PCs[,1],proc$mshape)
plot(lm,asp=1)

## now the first 3 scores
lm2 <- showPC(proc$PCscores[1,1:3],proc$PCs[,1:3],proc$mshape)
points(lm2,col=2)
```

slider3d slides Semilandmarks along curves and surfaces in 3D by minimizing 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

```r
slider3d(dat.array, SMvector, outlines = NULL, surp = NULL,
    sur.path = "sur", 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)
```

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 there 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 \( \gamma = \gamma^0 + \text{stepsize} \times U^T \). 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. In Windows, parallel processing is disabled.

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).

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 surface 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 your computer with this function with inappropriate data.

Author(s)

Stefan Schlager

References


See Also

relaxLM, createMissingList

Examples

```r
## Not run:
data(nose)
### create mesh for longnose
longnose.mesh <- warp.mesh(shortnose.mesh, shortnose.lm, longnose.lm)
### 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,
```
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.
symmetrize

Value

<table>
<thead>
<tr>
<th>basis</th>
<th>matrix containing the basis of the solution space</th>
</tr>
</thead>
<tbody>
<tr>
<td>translate</td>
<td>translation vector</td>
</tr>
</tbody>
</table>

Examples

```r
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
```

Description

create a perfectly symmetric version of landmarks

Usage

`symmetrize(x, pairedLM)`

Arguments

- `x`  
  k x m matrix with rows containing landmark coordinates
- `pairedLM`  
  A X x 2 matrix containing the indices (rownames) 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 relabeled 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

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

Description

maps a datamatrix via thin plate spline between calculated by a reference on a target configuration in 2D and 3D

Usage

tps3d(M, refmat, tarmat, lambda = 0)

Arguments

M              datamatrix - e.g. the matrix information of vertices of a given surface
refmat         reference matrix - e.g. landmark configuration on a surface
tarmat         target matrix - e.g. landmark configuration on a target surface
lambda         integer: regularisation parameter of the TPS.

Value

returns the warped datamatrix

Author(s)

Stefan Schlager

References


See Also

gasp, warp.mesh
typprob

desc

Examples

require(Morpho)
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)
## Not run:
## visualize results by applying a deformation grid
deformGrid3d(shortnose.lm, deformed, ngrid = 5)

## End(Not run)

typprob

calculate typicality probabilities

desc

Usage

typprob(x, data, small = FALSE, method = c("chisquare", "wilson"),
center = NULL, cova = NULL)

typprobClass(x, data, groups, small = FALSE, method = c("chisquare",
"wilson"), outlier = 0.01, sep = FALSE)

Arguments

x vector or matrix of data the probability is to be calculated.
data Reference data set.
small adjustment 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 mahalanobis distance, 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.
typprob

cova
covariance matrix to calculate mahalanobis-distance: specify custom covariance matrix to calculate distance.

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.

Details
get the probability for an observation belonging to a given multivariate normal 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"

Author(s)
Stefan Schlager

References


Examples
library(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, apply(data, 2, mean), 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, data2, fac, method = "w", small = TRUE)
## only 59 specimen is rather small.
typClass2 <- typprobClass(data2, data2, fac, method = "c")## use default settings

### check results for first method:
ct <- table(fac, typClass$groupaffin)
### 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`.

rmVertex removes specified vertices from triangular meshes.

vert2points extracts vertex coordinates from triangular meshes.

rmUnrefVertex removes unreferenced vertices from triangular meshes.
updateNormals

**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**

```r
code
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:
rgl.pop()
points3d(vert2points(cleanmesh),col=2) ### now the holes are empty!!

## End(Not run)
```

**updateNormals**

*Compute face or vertex normals of a triangular mesh*

**Description**

Compute face or vertex normals of a triangular mesh of class "mesh3d"

**Usage**

```r
code
updateNormals(x, angle = TRUE)
facenormals(x)
```
**updateNormals**

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


See Also

- `ply2mesh`

Examples

```R
require(rgl)
require(Morpho)
data(nose)
### calculate vertex normals
shortnose.mesh$normals <- NULL  ##remove normals
## Not run:
shade3d(shortnose.mesh,col=3)  ##render

## Not run
shortnose.mesh <- updateNormals(shortnose.mesh)
## Not run:
rgl.clear()
shade3d(shortnose.mesh,col=3)  ##smoothly rendered now

## Not run
## calculate facenormals
facemesh <- facenormals(shortnose.mesh)
## Not run:
plotNormals(facemesh,long=0.01)
```
vecx

**Description**

converts a 3D-array (e.g. containing landmark coordinates) into a matrix, one row per specimen or reverse this.

**Usage**

```r
vecx(x, byrow = FALSE, revert = FALSE, lmdim)
```

**Arguments**

- `x`: array or matrix
- `byrow`: logical: if TRUE, the resulting vector for each specimen will be \(x_1, y_1, z_1, x_2, y_2, z_2, \ldots\), and \(x_1, x_2, \ldots, y_1, y_2, \ldots, z_1, z_2, \ldots\) otherwise (default). The same is for reverting the process: if the matrix contains the coordinates as rows like: \(x_1, y_1, z_1, x_2, y_2, z_2, \ldots\) 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**

```r
library(shapes)
data <- vecx(gorf.dat)
# revert the procedure
gdat.restored <- vecx(data, revert=TRUE, lmdim=2)
range(gdat.restored-gorf.dat)
```
warp.mesh

warp.mesh  

warping a mesh onto another configuration

Description

warp a mesh onto another configuration via reference and target landmark configuration by using a thin-plate spline interpolation.

Usage

warp.mesh(mesh, matr, matt, lambda = 0, updateNormals = TRUE, silent = FALSE)

Arguments

mesh  
object of class "mesh3d"

matr  
matrix of landmarks on the reference surface

matt  
matrix of corresponding landmarks on the target surface

lambda  
integer: regularisation parameter of the TPS.

updateNormals  
Logical: requests the (re)calculation of vertex normals.

silent  
logical: suppress messages.

Details

the surface is mapped via the tps3d function onto the target shape.

Value

object of class "mesh3d"

Author(s)

Stefan Schlager

See Also

ply2mesh, file2mesh, mesh2ply, warpmovie3d, rotmesh.onto

Examples

require(rgl)
data(nose)# load data
## warp a mesh onto another landmark configuration:
warpnose.long <- warp.mesh(shortnose.mesh, shortnose.lm, longnose.lm)
## Not run:
shade3d(warpnose.long, col=skin1)
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 = "green", 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
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 (overrides automatic 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 graphical 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
...  additional arguments passed to shade3d (3D) or points (2D).

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: animate).

Author(s)

Stefan Schlager

See Also

ply2mesh, file2mesh, mesh2ply, warp.mesh
Examples

### 3D example

data(nose)# load data
## Not run:
## warp a mesh onto another landmark configuration:
warpnose.long <- warp.mesh(shortnose.mesh,shortnose.lm,longnose.lm)

warpmovie3d(shortnose.mesh,warpnose.long,n=15)# create 15 images.

## ad some landmarks
warpmovie3d(shortnose.mesh,warpnose.long,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.

## End(Not run)
## 2D example
library(shapes)
bb <- procSym(gorf.dat)
## morph superimposed first specimen onto sample mean
warpmovie2d(bb$rotated[,1],bb$shape,n=20,links=c(1,5,4:2,8:6,1),imagedim="600x400")

---

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)

Arguments

x k x m matrix containing landmark configuration
filename character: Path/name of the requested output - extension will be added automatically. If not specified, the file will be named as the exported object.

Details

you can import the information into the program landmarks available at http://graphics.idav.ucdavis.edu/research/EvoMorph
**write.pts**

**Author(s)**
Stefan Schlager

**See Also**
read.pts

**Examples**

data(nose)
write.pts(shortnose.lm, filename="shortnose")
Index

*Topic datasets
  boneData, 11
colors, 17
nose, 60
*Topic package
  Morpho-package, 4

angleNcalc, 5
anonymize, 6
applyTransform, 7, 41
array, 10
arrMean3, 7
asymPermute, 8
barycenter, 9
bindArr, 10
bone1 (colors), 17
bone2 (colors), 17
bone3 (colors), 17
boneData, 11
boneLM (boneData), 11

CAC, 11
cbind, 10
cExtract, 12
checkLM, 13, 67
classify, 14
closemeshKD, 9, 15, 48, 82, 86, 94
colors, 17
computeTransform, 17
cov, 20
covDist, 18
covPCA (covDist), 18
covW, 20
createAtlas, 13, 14, 21, 66–69
CreateL, 22
createMissingList, 23, 111
crossProduct, 24
cSize, 25
cutMeshPlane, 26
cutSpace, 26, 26
CVA, 27, 42
defformGrid2d, 31
defformGrid3d, 32
export (render.matrixDist), 98
export.meshDist, 55
exVar, 33
facenormals (updateNormals), 118
file2mesh, 13, 14, 34, 50, 104, 110, 118, 121, 123
find.outliers, 35
fixLMMirror, 23, 36
fixLMtps, 23, 38, 59
formula, 73
getFaces, 39
getTrafo4x4, 40
getTrafoRotaxis, 40
groupPCA, 29, 41

hist, 44
histGroup, 43
icpmat, 44
invertFaces, 45
kendalldist, 46
lineplot, 47
lm, 73
longnose.lm (nose), 60
mcNNindex, 48
meanMat, 49
mergeMeshes, 49
mesh2grey, 50
mesh2obj, 51
mesh2ply, 50, 104, 110, 121, 123
INDEX

mesh2ply (mesh2obj), 51
meshcube, 52
meshDist, 99
meshDist (meshDist.matrix), 53
meshDist.matrix, 53
meshPlaneIntersect, 55
meshres, 56
mirror, 57
model.matrix, 73
Morpho (Morpho-package), 4
Morpho-package, 4

name2factor, 58
name2num (name2factor), 58
NNshapereg, 59
nose, 60

obj2mesh (file2mesh), 34

par, 123
pcAlign, 60
pcaplot3d, 47, 61
PCdist, 62
permudist, 63
permuvex, 64
placePatch, 14, 21, 66, 69
plotAtlas, 14, 21, 68
plotNormals, 69
pls2B, 70
ply2mesh, 16, 50, 51, 86, 118, 119, 121, 123
ply2mesh (file2mesh), 34
points, 44, 123
points2plane, 72
pcomp, 19, 108
predictShape.lm, 73
proc.weight, 38, 39, 59, 74
procAOVsym, 76
ProcGPA, 77
procSym, 8, 9, 62, 76, 78, 85, 108
projRead, 82

qmat, 83
qplot, 83
quad2trimesh, 51, 84

r2morphoj, 85
r2morphologika (r2morphoj), 85
ray2mesh, 86
rbind, 10

read.csv.folder, 87
read.lmdata, 12, 88, 90
read.mpp, 88
read.pts, 12, 89, 90, 125
read.table, 87, 91
readallTPS, 90
readLandmarks.csv, 91
regrid, 92
RegScore, 93
relLM, 22, 67, 94, 111
relWarp, 96
render (render.matrixDist), 98
render.matrixDist, 98
render.meshDist, 55
retroDeform3d, 100, 101
retroDeformMesh, 100
rgl.material, 54, 99
rmUnrefVertex (unrefVertex), 117
rmVertex (unrefVertex), 117
rotaxis3d, 101, 103
rotaxisMat, 102
rotmesh.onto, 102, 103, 105, 107, 121
rotonmat, 104
rotonto, 78, 102, 104, 105, 105
rotreverse (rotonto), 105

scalemesh, 107
shade3d, 54, 55, 99, 123
shorthnose.lm (nose), 60
shorthnose.mesh (nose), 60
showPC, 108
skin1 (colors), 17
skin2 (colors), 17
skin3 (colors), 17
skin4 (colors), 17
skull_0144_ch_fe.mesh (boneData), 11
slider3d, 22, 23, 67, 80, 81, 95, 109
solutionSpace, 112
svd, 71
symmetrize, 113

tangentPlane (crossProduct), 24
tps3d, 23, 31, 33, 39, 101, 114
typprob, 36, 115
typprobClass, 20, 36
typprobClass (typprob), 115

unrefVertex, 117
updateNormals, 45, 118
vecx, 120
vert2points (unrefVertex), 117

warp.mesh, 23, 67, 101, 104, 114, 121, 123
warpmovie2d (warpmovie3d), 122
warpmovie3d, 121, 122
write.pts, 124