Package 'GeometricMorphometricsMix'

January 27, 2026

Title Heterogeneous Methods for Shape and Other Multidimensional Data

Version 0.6.0.1

Description Tools for geometric morphometric analyses and multidimensional data. Implements methods for morphological disparity analysis using bootstrap and rarefaction, as reviewed in Foote (1997) <doi:10.1146/annurev.ecolsys.28.1.129>. Includes integration and modularity testing, following Fruciano et al. (2013) <doi:10.1371/journal.pone.0069376>, using Escoufier's RV coefficient as test statistic as well as two-block partial least squares - PLS, Rohlf and Corti (2000) <doi:10.1080/106351500750049806>. Also includes vector angle comparisons, orthogonal projection for data correction (Burnaby (1966) <doi:10.2307/2528217>; Fruciano (2016) <doi:10.1007/s00427-016-0537-4>), and parallel analysis for dimensionality reduction (Buja and Eyuboglu (1992) <doi:10.1207/s15327906mbr2704_2>).

Depends R (>= 3.5.0), ape, stats, corpcor

Imports methods, mclust

Suggests Morpho, utils, geometry, nlshrink, lmf, MASS, clusterGeneration, ggplot2, Rmpfr, knitr, rmarkdown, phytools, mvMORPH, testthat (>= 3.0.0), future, future.apply

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.3.3

VignetteBuilder knitr, rmarkdown

Config/testthat/edition 3

NeedsCompilation no

Author Carmelo Fruciano [aut, cre]

Maintainer Carmelo Fruciano <carmelo.fruciano@unict.it>

Repository CRAN

Date/Publication 2026-01-27 21:00:24 UTC

Contents

`adjRand_test` *Test the significance of the adjusted Rand index*

Description

Permutation test of the adjusted Rand index, which quantifies the level of agreement between two partitions (e.g., two schemes of classification of the same individuals obtained with two methods)

Usage

```
adjRand_test(A, B, perm = 999)
```

Arguments

A, B	numerical or character vectors reflecting the assignment of individual observations to groups
perm	number of permutations

Details

The adjusted Rand index (Hubert and Arabie 1985), is an adjusted for chance version of the Rand index (Rand 1971). The adjusted Rand index has an expected value of zero in the case of random partitions, and values approaching one as the two partitions become more similar to each other (with one being perfect match of the classification). This function implements the permutation test proposed by Qannari et al. (2014) to obtain a p value against the null hypothesis of independence of the two partitions.

This function is useful in various contexts, such as in integrative taxonomy when comparing the classification of individual specimens obtained using different data (e.g., sequence data and morphometric data). For an example of the application of this technique with the classification obtained with genetic data and morphometric data for multiple traits, see Fruciano et al. 2016.

Value

The function outputs a vector with the adjusted Rand index and the p value obtained from the permutation test

Notice

The function requires internally the package mclust.

Citation

If you use this function in the context of integrative taxonomy or similar (comparison of classification/unsupervised clustering with biological data), please cite all the papers in the references (otherwise, please use the relevant citations for the context).

References

Fruciano C, Franchini P, Raffini F, Fan S, Meyer A. 2016. Are sympatrically speciating Midas cichlid fish special? Patterns of morphological and genetic variation in the closely related species Archocentrus centrarchus. Ecology and Evolution 6:4102-4114.

Hubert L, Arabie P. 1985. Comparing partitions. Journal of Classification 2:193-218.

Qannari EM, Courcoux P, Faye P. 2014. Significance test of the adjusted Rand index. Application to the free sorting task. Food Quality and Preference 32, Part A:93-97.

Rand WM. 1971. Objective Criteria for the Evaluation of Clustering Methods. Journal of the American Statistical Association 66:846-850.

Examples

```
library(mclust)
set.seed(123)

irisBIC = mclustBIC(iris[,-5])
mclustBIC_classification = summary(irisBIC,iris[,-5])$classification
original_classification = iris[,5]
# This is one of the examples in the package mclust
# Here a classification algorithm is used on the iris dataset

adjustedRandIndex(mclustBIC_classification, original_classification)
# The mclust package allows computing the adjusted Rand index
# which quantifies the agreement between the original (correct)
# classification and the one obtained with the algorithm.
# However, it is not clear whether the adjusted Rand index is
# "large enough" compared to the null hypothesis of independence
# between the two classification schemes

adjRand_test(mclustBIC_classification, original_classification,
             perm = 999)
# For that, we use the function adjRand_test, which performs
# the permutation test of Qannari et al. 2014 (in this case
# p<0.001, as 1000 permutations have been used).

adjRand_test(original_classification, original_classification,
             perm = 999)
# As it can be seen, in the ideal case of the exact same grouping,
# the adjusted Rand index takes a value of 1 (which is obviously
# significant)
```

---

arching_vector *Body arching vector from brown trout study*

Description

A list containing the body arching vector obtained using common principal components following the procedure described in Fruciano et al. 2020, and the GPA consensus used to align individual models when the vector was computed. The GPA consensus is provided to align new data using the same consensus), but using the consensus for downstream work is not recommended.

Usage

```
data(arching_vector)
```

Format

A list with two components as described above.

Details

The object is a list with the following elements:

GPA_consensus_used A matrix of landmark coordinates for the GPA consensus to which the models have been aligned (brown trout dataset in this package)

arching_vector_CPCA A numeric vector of shape change obtained using common principal component analysis as detailed in Fruciano et al. 2020.

References

Fruciano C., Schmidt, I., Ramirez Sanchez, M.M., Morek, W., Avila Valle, Z.A., Talijancic, I., Pecoraro, C., Schermann Legionnet, A. 2020. Tissue preservation can affect geometric morphometric analyses: a case study using fish body shape. Zoological Journal of the Linnean Society 188:148-162.

See Also

`ProjectOrthogonal` for projection/removal of an effect

Examples

```
data(arching_vector)
a = arching_vector
names(a)
dim(a$GPA_consensus_used)
length(a$arching_vector_CPCA)
```

---

brown_trout *Brown trout landmark data*

Description

A list containing landmark coordinates for a subset of brown trout specimens used in Fruciano et al. 2014 (Biological Journal of the Linnean Society) and Fruciano et al. 2020 (Zoological Journal of the Linnean Society). The subset originates from two rivers and was digitised by a single operator.

Usage

```
data(brown_trout)
```

Format

A list with components described above.

Details

Notice that the dataset has been realigned using Generalized Procrustes Analysis (GPA) and corrected for body arching using the arching vector in `data(arching_vector)`. Because this is a small subset of the fish in the original study (Fruciano et al. 2014), and contain only data from a single operator and treatment from Fruciano et al. 2020, the dataset should be strictly considered as a "toy" dataset and not used for formal statistical analyses. For conclusions about biological variation and measurement error due to preservation, please refer to the original studies (Fruciano et al. 2014, Fruciano et al. 2020, respectively).

The object is a list with the following components:

with_arching Landmark coordinates for specimens without correction for body arching.

without_arching Landmark coordinates for the same specimens after correction for body arching.

Factors A data.frame with metadata for each specimen (e.g., centroid size, sex and river).

References

Fruciano C, Pappalardo AM, Tigano C, Ferrito V. 2014. Phylogeographical relationships of Sicilian brown trout and the effects of genetic introgression on morphospace occupation. Biological Journal of the Linnean Society 112:387-398.

Fruciano C., Schmidt, I., Ramirez Sanchez, M.M., Morek, W., Avila Valle, Z.A., Talijancic, I., Pecoraro, C., Schermann Legionnet, A. 2020. Tissue preservation can affect geometric morphometric analyses: a case study using fish body shape. Zoological Journal of the Linnean Society 188:148-162.

Examples

```
data(brown_trout)
d = brown_trout
names(d)
str(d$Factors)
```

BTailTest *BTailTest for difference in disparity/morphospace occupation*

Description

Performs the BTailTest, in the same spirit as implemented in the Matlab package MDA (Navarro 2003) and used in various empirical papers (e.g., Fruciano et al. 2014, 2016).

Usage

```
BTailTest(Reference, Test, boot = 1000)
```

Arguments

`Reference`	Matrix or data frame containing data for the reference group (observations in rows, variables in columns).
`Test`	Matrix or data frame containing data for the test group (observations in rows, variables in columns).
`boot`	number of bootstrap replicates

Details

This is a test of the difference in disparity between two groups: a reference and a test group. The function proceeds by computing a bootstrapped distribution of the test statistics (multivariate variance and mean pairwise Euclidean distances in this implementation) in the reference sample and then comparing the statistics observed in the test sample to this distribution to obtain p-values.

Value

The function outputs a list with the following elements:

BootstrappedSamplesEstimates Estimates of both multivariate variance and mean pairwise Euclidean distance for each of the bootstrapped samples

pvalues p values obtained for the test

Citation

If you use this function, in addition to this package, please cite Navarro (2003)

References

Navarro N. 2003. MDA: a MATLAB-based program for morphospace-disparity analysis. Computers & Geosciences 29:655-664.

Fruciano C, Franchini P, Raffini F, Fan S, Meyer A. 2016. Are sympatrically speciating Midas cichlid fish special? Patterns of morphological and genetic variation in the closely related species Archocentrus centrarchus. Ecology and Evolution 6:4102-4114.

Fruciano C, Pappalardo AM, Tigano C, Ferrito V. 2014. Phylogeographical relationships of Sicilian brown trout and the effects of genetic introgression on morphospace occupation. Biological Journal of the Linnean Society 112:387-398.

See Also

`disparity_resample`, `disparity_test`

`critical_angle` *Compute the critical angle for the test of the angle between two multivariate vectors*

Description

This function computes the angle below which two vectors would be "significantly similar" using Li 2011 test.

Usage

```
critical_angle(dimensions, alpha = 0.05, prec = "normal")
```

Arguments

`dimensions`	number of dimensions to use
`alpha`	significance level (by default 0.05)
`prec`	whether one has to use the internal R functions ("normal", default), or mpfr precision ("mpfr")

Details

This function considers the formulas in Li 2011 which have been used to perform a test of the angle between multivariate vectors. This is the test implemented in MorphoJ (Klingenberg 2011), in the R package Morpho (Schlager 2016), and in the function TestOfAngle of this package. The test produces a p value for the angle between two vectors (with significant values being "significantly similar").

This function reverts the logic of the formula/test and, given a number of dimensions (e.g., morphometric variables) and a level of significance (by default 0.05), it computes the angle below which the test would be significant.

Value

The function outputs the critical angle (in degrees)

Notice

In case of very many dimensions, there are numerical problems in performing the test. If prec="normal" the function uses internally the package Morpho (which should be installed) and these problems will start occurring above about 340 variables. If prec="mpfr" the function uses internally a custom function which requires the package Rmpfr (which should be installed). This allows the computation of extremely large or small numbers, it is currently set to a 120 bit precision and allows the computation using up to about 1200 variables (over this number, there will be problems even using the option "mpfr")

References

Klingenberg CP. 2011. MorphoJ: an integrated software package for geometric morphometrics. Mol Ecol Resour 11:353-357.

Li S. 2011. Concise formulas for the area and volume of a hyperspherical cap. Asian Journal of Mathematics and Statistics 4:66-70.

Schlager S. 2016. Morpho: Calculations and Visualisations Related to Geometric Morphometrics.

See Also

TestOfAngle

Examples

```
critical_angle(50)
# This is the angle below which two vectors (e.g., PCA axes)
# of 50 variables would be would be "significantly similar"
```

---

disparity_resample *Resampling-based estimates (bootstrap or rarefaction) of disparity / morphospace occupation*

Description

Provides a unified interface to obtain resampled (bootstrap or rarefied) estimates of several disparity / morphospace occupation statistics.

Usage

```
disparity_resample(
  Data,
  group = NULL,
  n_resamples = 1000,
  statistic = "multivariate_variance",
  CI = 0.95,
  bootstrap_rarefaction = "bootstrap",
  sample_size = NULL
)
```

Arguments

Data	A data frame, matrix, vector, or 3D array. Observations (specimens) must be in rows (if a 3D array is supplied, the third dimension is assumed to index specimens).
group	A factor or a vector indicating group membership (will be coerced to factor). If 'NULL' (default) a single analysis is performed on the full dataset.

`n_resamples` Number of resampling replicates (default 1000).

`statistic` Character string identifying the statistic to compute. One of ‘"multivariate_variance"‘, ‘"mean_pairwise_euclidean_distance"‘, ‘"convex_hull_volume"‘, ‘"claramunt_proper_variance"‘. Default is ‘"multivariate_variance"‘. Ignored for univariate data (vector input).

`CI` Desired two-sided confidence interval level (default 0.95) used for percentile confidence intervals. Use CI=1 to display the full range (minimum to maximum) of resampled values.

`bootstrap_rarefaction`
Either ‘"bootstrap"‘ (default) for resampling with replacement or ‘"rarefaction"‘ for resampling without replacement.

`sample_size` Either ‘NULL‘ (default), a positive integer indicating the number of rows to sample, or the character ‘"smallest"‘ to use the size of the smallest group (all groups resampled to that size). For ‘bootstrap_rarefaction=="rarefaction"‘, sampling is without replacement and this parameter is required (not ‘NULL‘). For ‘bootstrap_rarefaction=="bootstrap"‘, sampling is with replacement; if ‘NULL‘, uses original group sizes, otherwise uses the specified sample size. If ‘"smallest"‘ is supplied but no groups are defined, an error is returned.

Details

The function allows choosing among the following multivariate statistics:

- Multivariate variance (trace of the covariance matrix; sum of variances)
- Mean pairwise Euclidean distance
- Convex hull volume (n-dimensional)
- Claramunt proper variance (Claramunt 2010) based on a linear shrinkage covariance matrix

If the input ‘Data‘ is univariate (i.e., a vector), the analysis defaults to computing the (univariate) variance within each group (the attribute ‘statistic‘ is ignored).

If ‘bootstrap_rarefaction=="bootstrap"‘, the function performs resampling with replacement (i.e., classical bootstrap) by sampling rows of the data matrix / data frame. Optionally, the user can specify a custom sample size via the ‘sample_size‘ argument. This allows comparison of bootstrap confidence intervals at the same sample size (essentially, this is rarefaction sampling with replacement), which can be useful to compare bootstrapped confidence intervals across different groups for statistics which are sensitive to sample size (at the expense of broader than necessary confidence intervals for groups that are larger).

If ‘bootstrap_rarefaction=="rarefaction"‘, the function performs resampling without replacement at the sample size indicated in ‘sample_size‘ (numeric) or, if ‘sample_size=="smallest"‘, at the size of the smallest group (all groups are resampled to that size). Rarefaction requires specifying a valute to the attribute ‘sample_size‘; an error is returned otherwise.

This function uses the future framework for parallel processing, requiring packages future and future.apply. Users should set up their preferred parallelization strategy using `future::plan()` before calling this function. For example:

- `future::plan(future::sequential)` for sequential processing
- `future::plan(future::multisession, workers = 4)` for parallel processing with 4 workers (works on all platforms including Windows)

- `future::plan(future::multicore, workers = 4)` for forked processes (Unix-like systems)
- `future::plan(future::cluster, workers = c("host1", "host2"))` for cluster computing

If no plan is set or the future packages are not available, the function will use sequential processing.

Value

A list containing:

chosen_statistic Character vector of length 1 with the human-readable name of the statistic used.

results A data frame with columns 'group', 'average', 'CI_min', 'CI_max'. One row per group. When CI=1, 'CI_min' and 'CI_max' represent the minimum and maximum of resampled values rather than confidence intervals.

resampled_values If a single group: numeric vector of length 'n_resamples' with the resampled values. If multiple groups: a named list with one numeric vector (length 'n_resamples') per group.

CI_level The CI level used (between 0 and 1). When CI=1, ranges are computed instead of confidence intervals.

The returned object has class "disparity_resample" and comes with associated S3 methods for convenient display and visualization:

- `print.disparity_resample`: Prints a formatted summary of results including confidence interval overlap assessment for multiple groups
- `plot.disparity_resample`: Creates a confidence interval plot using ggplot2

Parallelization

This function automatically uses parallel processing via the future framework, when the packages future and future.apply are installed. This is particularly useful for large datasets, large number of resamples or computationally intensive statistics (e.g., convex hull volume). The parallelization strategy is determined by the user's choice of future plan, providing flexibility across different computing environments (local multicore, cluster, etc.). The function performs parallelization at the level of individual bootstrap/rarefaction replicates within each group. The future plan should be set up by the user before calling this function using `future::plan()` (see examples). If no plan is set or the future packages are not available, the function will use sequential processing.

Average estimate

For bootstrap resampling, the average estimate reported is the mean of the bootstrap resampled values (for consistency across all bootstrap scenarios). For rarefaction, because the purpose is to make groups comparable at a common (smaller) sample size, the average estimate reported is the mean of the rarefied resampled values for that group (i.e., the mean across all rarefaction replicates).

Input data types

'Data' can be a data frame, a matrix, a vector, or a 3D array of landmark coordinates (e.g., p landmarks x k dimensions x n specimens). In the latter case, the array is converted internally to a 2D matrix with specimens in rows and (landmark * dimension) variables in columns.

Note

Because of how the computation works, convex hull volume computation requires the number of observations (specimens) to be (substantially) greater than the number of variables (dimensions). In case of shape or similar, consider using the scores along the first (few/several) principal components. Sometimes errors are thrown due to near-zero components, in this case try reducing the number of principal components used. Examples of use of this statistic with geometric morphometric data include Drake & Klingenberg 2010 (American Naturalist), Fruciano et al. 2012 (Environmental Biology of Fishes) and Fruciano et al. 2014 (Biological Journal of the Linnean Society). Because of the sensitivity of this statistic to outliers, usually rarefaction is preferred to bootstrapping.

"Multivariate variance" is also called "total variance", "Procrustes variance" (in geometric morphometrics) and "sum of univariate variances". Note how the computation here does not divide variance by sample size (other than the normal division performed in the computation of variances).

References

Drake AG, Klingenberg CP. 2010. Large-scale diversification of skull shape in domestic dogs: disparity and modularity. American Naturalist 175:289-301.

Claramunt S. 2010. Discovering exceptional diversifications at continental scales: the case of the endemic families of Neotropical Suboscine passerines. Evolution 64:2004-2019.

Fruciano C, Tigano C, Ferrito V. 2012. Body shape variation and colour change during growth in a protogynous fish. Environmental Biology of Fishes 94:615-622.

Fruciano C, Pappalardo AM, Tigano C, Ferrito V. 2014. Phylogeographical relationships of Sicilian brown trout and the effects of genetic introgression on morphospace occupation. Biological Journal of the Linnean Society 112:387-398.

Fruciano C, Franchini P, Raffini F, Fan S, Meyer A. 2016. Are sympatrically speciating Midas cichlid fish special? Patterns of morphological and genetic variation in the closely related species Archocentrus centrarchus. Ecology and Evolution 6:4102-4114.

See Also

disparity_test, print.disparity_resample, plot.disparity_resample

Examples

```
set.seed(123)
# Simulate two groups with different means but same covariance
if (requireNamespace("MASS", quietly = TRUE)) {
  X1 = MASS::mvrnorm(20, mu=rep(0, 10), Sigma=diag(10))
  X2 = MASS::mvrnorm(35, mu=rep(2, 10), Sigma=diag(10))
  Data = rbind(X1, X2)
  grp = factor(c(rep("A", nrow(X1)), rep("B", nrow(X2))))

  # Sequential processing
  # future::plan(future::sequential)  # Default sequential processing

  # Parallel processing (uncomment to use)
  # future::plan(future::multisession, workers = 2)  # Use 2 workers
```

```
  # Bootstrap multivariate variance
  boot_res = disparity_resample(
    Data, group=grp, n_resamples=200,
    statistic="multivariate_variance",
    bootstrap_rarefaction="bootstrap"
  )
  # Direct access to results table
  boot_res$results

  # Using the print method for formatted output
  print(boot_res)

  # Using the plot method to visualize results
  # plot(boot_res)  # Uncomment to create confidence interval plot

  # Rarefaction (to the smallest group size) of mean pairwise
  # Euclidean distance
  rar_res = disparity_resample(
    Data, group=grp, n_resamples=200,
    statistic="mean_pairwise_euclidean_distance",
    bootstrap_rarefaction="rarefaction", sample_size="smallest"
  )
# Now simulate a third group with larger variance
 X3 = MASS::mvrnorm(15, mu=rep(0, 10), Sigma=diag(10)*1.5)
 grp2 = factor(
   c(rep("A", nrow(X1)), rep("B", nrow(X2)), rep("C", nrow(X3)))
 )
 boot_res2 = disparity_resample(
   Data=rbind(X1, X2, X3), group=grp2, n_resamples=1000,
   statistic="multivariate_variance",
   bootstrap_rarefaction="bootstrap"
 )
 print(boot_res2)
 # plot(boot_res2)
 # Plot of the obtained (95%) confidence intervals (uncomment to plot)

 # Reset to sequential processing when done (optional)
 # future::plan(future::sequential)
}
```

`disparity_test` *Permutation test of difference in disparity/morphospace occupation*

Description

Performs a permutation test of difference in disparity between two groups.

Usage

```
disparity_test(X1, X2, perm = 999)
```

Arguments

X1, X2	Matrices or data frames containing data for each group (observations in rows, variables in columns).
perm	number of permutations

Details

The function employs commonly used test statistics to quantify disparity/morphospace occupation/variation in each group. The two statistics currently implemented are multivariate variance (also known as sum of variances, trace of the covariance matrix, Procrustes variance), and mean pairwise Euclidean distances. These two metrics have a long history in the quantification of disparity both in geometric morphometrics (e.g., Zelditch et al. 2004; Fruciano et al., 2014, 2016) and more in general in evolution (e.g., Foote, 1996; Willis 2001) The observed statistics are then compared to their empirical distributions obtained through permutations, to obtain a p-value.

Value

The function outputs a dataframe containing: the observed values of the tests statistics for each group, their absolute differences, and The p values obtained through the permutational procedure

Notice

The values of the test statistics in the output are the observed in the sample. If they are of interest, and the two groups have different sample size, consider computing their rarefied versions (for instance with the function disparity_resample) for reporting in papers and the like.

References

Foote M. 1997. The evolution of morphological diversity. Annual Review of Ecology and Systematics 28:129.

Wills MA. 2001. Morphological disparity: a primer. In. Fossils, phylogeny, and form: Springer. p. 55-144.

Zelditch ML, Swiderski DL, Sheets HD. 2004. Geometric morphometrics for biologists: a primer: Academic Press.

Fruciano C, Franchini P, Raffini F, Fan S, Meyer A. 2016. Are sympatrically speciating Midas cichlid fish special? Patterns of morphological and genetic variation in the closely related species Archocentrus centrarchus. Ecology and Evolution 6:4102-4114.

Fruciano C, Pappalardo AM, Tigano C, Ferrito V. 2014. Phylogeographical relationships of Sicilian brown trout and the effects of genetic introgression on morphospace occupation. Biological Journal of the Linnean Society 112:387-398.

See Also

disparity_resample, BTailTest

Examples

```
library(MASS)
set.seed(123)

X1=mvrnorm(20, mu=rep(0, 40), Sigma=diag(40))
X2=mvrnorm(100, mu=rep(5, 40), Sigma=diag(40))
# create two groups of random observations
# with different means and sample sizes,
# but the same covariance matrix

# We expect that the two groups will have the same
# variance (disparity/morphospace occupation)
# and therefore the test will be non-significant

disparity_test(X1, X2, perm=999)
# This is, indeed, the case
```

---

dist_mean_boot *Bootstrapped distance between two arrays*

Description

Computes bootstrapped estimates of the mean distance between two groups and their confidence intervals.

Usage

```
dist_mean_boot(A, B, boot = 1000, ci = 0.95, nA = nrow(A), nB = nrow(B))
```

Arguments

A, B	Matrices or data frames containing data (observations in rows, variables in columns).
boot	number of bootstrap resamples
ci	width of the confidence interval
nA, nB	sample sizes for each bootstrapped group (defaults to original sample size)

Details

This may be useful to compare whether the differences between two groups are larger or smaller than differences between two other groups.

For instance, if we wanted to quantify shape sexual dimorphism in two populations, we could run this analysis separately for the two populations and then check the confidence intervals. If the two confidence intervals are disjunct there is evidence for the two populations having different levels of sexual dimorphism.

The computation performs bootstrap by resampling with replacement within each of the two groups and at each round computing the Euclidean distance between the two groups. It is also possible to resample at a different sample size than the one in the data using the attributes nA and nB.

Notice that the confidence interval is expressed on a scale between 0 and 1 and not as a percentage (e.g., 0.95 means 95

Value

The function outputs a named vector with the mean, median, upper and lower confidence interval bounds obtained from the bootstrapped samples

---

`EscoufierRV` *Escoufier RV coefficient*

Description

Computes the Escoufier RV coefficient

Usage

```
EscoufierRV(Block1, Block2)
```

Arguments

`Block1`, `Block2`	Matrices or data frames containing each block of variables (observations in rows, variables in columns).

Details

This function computes the usual version of the Escoufier RV coefficient (Escoufier, 1973), which quantifies the level of association between two multivariate blocks of variables. The function accepts two blocks of variables, either two data frames or two matrices each of n observations (specimens) as rows. The two blocks must have the same number of rows (specimens), but can have different number of columns (variables, such as landmark coordinates). The Escoufier RV has been shown (Fruciano et al. 2013) to be affected by sample size so comparisons of groups (e.g., species, populations) with different sample size should be avoided, unless steps are taken to account for this problem

Value

The function returns a number, corresponding to the Escoufier RV coefficient

References

Escoufier Y. 1973. Le Traitement des Variables Vectorielles. Biometrics 29:751-760.

Fruciano C, Franchini P, Meyer A. 2013. Resampling-Based Approaches to Study Variation in Morphological Modularity. PLoS ONE 8:e69376.

See Also

RVrarefied

Examples

```
library(MASS)
set.seed(123)
A=mvrnorm(100,mu=rep(0,100), Sigma=diag(100))
# Create a sample of 100 'individuals'
# as multivariate normal random data
# We will consider the first 20 columns as the first
# block of variables, and the following one as the second block

EscoufierRV(A[,1:20],A[,21:ncol(A)])
# Compute the EscoufierRV using the two blocks of variables
```

`Kmultparallel` *Parallel implementation of Adams' Kmult with additional support for multiple datasets and tree sets*

Description

Parallel implementation of Kmult, a measure of phylogenetic signal which is a multivariate equivalent of Blomberg's K. This version supports multiple datasets and tree sets, computing Kmult for all combinations.

Usage

```
Kmultparallel(data, trees, burninpercent = 0, iter = 0, verbose = TRUE)
```

Arguments

`data`	Either a data.frame/matrix with continuous (multivariate) phenotypes, or a list where each element is a data.frame/matrix representing a separate dataset. Row names should match species names in the phylogenetic trees.
`trees`	Either a multiPhylo object containing a collection of trees (single tree set), or a list where each element is a multiPhylo object representing a separate tree set.
`burninpercent`	percentage of trees in each tree set to discard as burn-in (by default no tree is discarded)
`iter`	number of permutations to be used in the permutation test (this should normally be left at the default value of 0 as permutations slow down computation and are of doubtful utility when analyzing tree distributions)
`verbose`	logical, whether to print progress information (default TRUE)

Details

This is an updated and improved version of the function included in Fruciano et al. 2017. It performs the computation of Adams' Kmult (Adams 2014) in parallel with the aim of facilitating computation on a distribution of trees rather than a single tree. This version uses cross-platform parallel processing that works on Windows, Mac, and Linux systems. If one wanted to perform a computation of Kmult on a single tree, he/she would be advised to use the version implemented in the package geomorph, which receives regular updates.

This function uses the future framework for parallel processing. Users should set up their preferred parallelization strategy using `future::plan()` before calling this function. For example:

- `future::plan(future::sequential)` for sequential processing
- `future::plan(future::multisession, workers = 4)` for parallel processing with 4 workers (works in most platforms including Windows)
- `future::plan(future::multicore, workers = 4)` for forked processes (Unix-like systems)
- `future::plan(future::cluster, workers = c("host1", "host2"))` for cluster computing

If no plan is set, the function will use the default sequential processing.

Value

The function outputs a data.frame with classes "parallel_Kmult" and "data.frame" containing columns:

Kmult Value of Kmult for each tree-dataset combination

p value p value for the significance of the test (only if iter > 0)

treeset Identifier for the tree set (name from list or number)

dataset Identifier for the dataset (name from list or number)

tree_index Index of the tree within its tree set

Parallelization

This function automatically uses parallel processing via the future framework when beneficial. The parallelization strategy is determined by the user's choice of future plan, providing flexibility across different computing environments (local multicore, cluster, etc.). The function performs parallelization at the level of individual trees within each treeset, which is optimal for analyzing distributions of many trees. The future plan should be set up by the user before calling this function using `future::plan()` (see also examples).

Citation

If you use this function please kindly cite both Fruciano et al. 2017 (because you're using this parallelized function) and Adams 2014 (because the function computes Adams' Kmult)

S3 Methods

The returned object has specialized S3 methods:

- print.parallel_Kmult: Provides a summary of Kmult ranges for each dataset-treeset combination
- plot.parallel_Kmult: Creates density plots of Kmult values grouped by dataset-treeset combinations
- summary.parallel_Kmult: Provides detailed summary statistics for the analysis results

References

Adams DC. 2014. A Generalized K Statistic for Estimating Phylogenetic Signal from Shape and Other High-Dimensional Multivariate Data. Systematic Biology 63:685-697.

Fruciano C, Celik MA, Butler K, Dooley T, Weisbecker V, Phillips MJ. 2017. Sharing is caring? Measurement error and the issues arising from combining 3D morphometric datasets. Ecology and Evolution 7:7034-7046.

Examples

```
# Load required packages for data simulation
library(phytools)
library(MASS)
library(mvMORPH)
library(ape)  # for drop.tip function
library(future)
library(future.apply)

# Generate 20 random phylogenetic trees with 100 tips each
all_trees = replicate(20, pbtree(n = 100), simplify = FALSE)
class(all_trees) = "multiPhylo"
# Create a collection of 20 random trees

# Split trees into 2 tree sets
treeset1 = all_trees[1:5]
treeset2 = all_trees[6:20]
class(treeset1) = class(treeset2) = "multiPhylo"
# Split the 20 trees into 2 separate tree sets

# Get tip names from the first tree for consistent naming
tip_names = all_trees[[1]]$tip.label[1:40]
# Use first 40 tip names for consistent data generation

# Generate 1 random dataset using multivariate normal distribution
dataset_random = mvrnorm(n = 40, mu = rep(0, 5), Sigma = diag(5))
rownames(dataset_random) = tip_names
# Create one random dataset which should not display phylogenetic signal

# Generate 1 dataset using Brownian motion evolution on the first tree
tree_temp = treeset1[[1]]
# Get only the first 40 tips to match our data size
```

```
tips_to_keep = tree_temp$tip.label[1:40]
tree_pruned = ape::drop.tip(tree_temp,
                            setdiff(tree_temp$tip.label, tips_to_keep))

# Simulate data under Brownian motion
sim_data = mvSIM(tree = tree_pruned, nsim = 1, model = "BM1",
                 param = list(sigma = diag(5), theta = rep(0, 5)))
# Convert to matrix and ensure proper row names
if (is.list(sim_data)) sim_data = sim_data[[1]]
dataset_bm = as.matrix(sim_data)
rownames(dataset_bm) = tree_pruned$tip.label
# Generate 1 dataset evolving under Brownian motion
# This dataset should display strong phylogenetic signal when combined
# with treeset1

# Example 1: Single dataset and single treeset analysis (sequential
# processing)
future::plan(future::sequential)  # Use sequential processing
result_single = Kmultparallel(dataset_bm, treeset1)
# Analyze BM dataset with first treeset (sequential processing)

# Use S3 methods to examine results
print(result_single)
# Display summary of Kmult values
# Notice how the range is very broad because we have high
# phylogenetic signal for the case in which the dataset has been
# simulated under Brownian motion with the first tree, but low
# phylogenetic signal when we use the other trees in the treeset.

plot(result_single)
# Create density plot of Kmult distribution
# Notice the bimodal distribution with low phylogenetic signal
# corresponding to a mismatch between the tree used and the true
# evolutionary history of the traits, and the high phylogenetic
# signal when the correct tree is used.

# Example 2: Multiple datasets and multiple treesets analysis with
# parallel processing
# Set up parallel processing with future
future::plan(future::multisession, workers = 4)

# Combine datasets into a list
all_datasets = list(random = dataset_random, brownian = dataset_bm)
# Combine random and BM datasets

# Combine treesets into a list
all_treesets = list(treeset1 = treeset1, treeset2 = treeset2)
# Create list of both tree sets

# Run comprehensive analysis on all combinations
result_multiple = Kmultparallel(all_datasets, all_treesets)
# Analyze all dataset-treeset combinations with parallel processing
```

```
# Examine results using S3 methods
print(result_multiple)
# Display summary showing ranges for each combination

plot(result_multiple)
# Create grouped density plots by combination
# Notice how the distribution of Kmult when we use the random dataset
# has a strong peak at small values (no phylogenetic signal, as
# expected)

# Custom plotting with different transparency
plot(result_multiple, alpha = 0.5,
     title = "Kmult Distribution Across All Combinations")
# Customize the plot appearance

# Example 3: Setting up parallel processing with future
future::plan(future::multisession, workers = 4)
result_parallel = Kmultparallel(dataset_bm, treeset1)
# Use 4 worker processes for parallel processing

# Clean up: Reset to sequential processing to close parallel workers
future::plan(future::sequential)
```

`LM_relativepos_check` *Check the relative positions for a set of landmarks, compared to a reference specimen*

Description

The function compares the relative position of a set of landmarks to the one observed in a reference specimen. It also outputs which coordinates do not much the pattern observed in the reference specimen.

Usage

```
LM_relativepos_check(
  Dataset,
  LM_to_check,
  reference_specimen = 1,
  only_by_landmark_order = FALSE,
  use_specimen_names = TRUE
)
```

Arguments

`Dataset`	k x p x n array of k landmarks and p dimensions for n observations (specimens).
`LM_to_check`	Vector with the indices of which landmarks of Dataset should be tested

`reference_specimen`
: The index of the specimen to use as reference (by default, the first).

`only_by_landmark_order`
: If TRUE, each landmark in LM_to_check will be tested relative to the next in LM_to_check; otherwise all the pairwise relative positions between landmarks will be run

`use_specimen_names`
: whether the names of specimens in Dataset should be used for the output

Details

Compare relative positions of landmarks to the one observed in a reference specimen. This function is useful to identify specimens with switched landmark positions and similar problems when a dataset has been collected using consistent criteria (e.g., a set of fish body pictures, with the mouth-tail axis approximately horizontal for all specimens).

For instance, if we want to check that landmarks 1, 2, and 3 are all aligned along the y coordinate in 2D, we will use a specimen (usually the first), checking that it has been landmarked correctly. Then, we will run the function on landmarks 1, 2, and 3 (by setting LM_to_check=c(1, 2, 3)). The function will output a list of which specimens seem to be problematic and which landmarks and coordinates for these specimens seem problematic. The putatively problematic coordinates will be indicated with "0". Clearly, if we are only interested in 1, 2, and 3 being along y, it will not matter if in some case we will find "0" under Coord1, but only if we find a "0" under Coord2.

Value

The function outputs a list with the following elements:

potentially_problematic_idx Indices of potentially problematic specimens

potentially_problematic_LMs List of potentially problematic landmarks and coordinates for the specimens in potentially_problematic_idx

Notice

Generally, it is better to use this function with a small number of carefully selected landmarks, instead of running it on all landmarks at the same time.

The parameter only_by_landmark_order allows to set whether all combinations of landmarks should be tested (default), or not.

If only_by_landmark_order is set to TRUE, each landmark in LM_to_check will be only tested with the next one. For instance if LM_to_check=c(1,2,3) the function will only compare the first landmark with the second, and the second with the third.

If the function cannot find potentially problematic specimens, a message to this effect will be presented.

`parallel_analysis` *Perform parallel analysis*

Description

Parallel analysis based on permutations

Usage

```
parallel_analysis(X, perm = 999, fun = c("prcomp", "fastSVD", "shrink"))
```

Arguments

`X`	Matrix or data frame containing the original data (observations in rows, variables in columns).
`perm`	number of permutations
`fun`	function to use internally to obtain eigenvalues (see Details)

Details

The function allows performing parallel analysis, which is a way to test for the number of significant eigenvalues/axes in a PCA. In this implementation, a null distribution of eigenvalues is obtained by randomly permuting observations independently for each of the starting variables. To compute p values, the observed eigenvalues are compared to the corresponding eigenvalues from this null distribution.

Parallel analysis may be used for dimensionality reduction, retaining only the first block of consecutive significant axes. That is, if for example the first 3 axes were significant, then the fourth not significant, one would keep only the first 3 axes (regardless of significance of the axes from the fifth on). Similarly, if the first axis is not significant, this may suggest lack of a clear structure in the data.

The function internally employs three possible strategies to obtain eigenvalues (argument of fun):

- "prcomp" - the function prcomp (default)
- "fastSVD" - an approach based on the function fast.svd (requires the package corpcor)
- "shrink" - a decomposition of the covariance matrix estimated using linear shrinkage (much slower, requires the package nlshrink; Ledoit & Wolf 2004)

This choice should not make much difference in terms of the final result. However, for consistency, it is a good idea to use for parallel analysis the same function used for the actual PCA (this is why these three options are provided).

Value

Vector of class "parallel_analysis" containing the p values for each of the axes of a PCA on the data provided

The object of class parallel_analysis returned by the function has a summary() method associated to it. This means that using summary() on an object created by this function, a suggestion on the number of significant axes (if any) is provided (see examples).

Citation

The most appropriate citation for this approach to parallel analysis (using permutations) is Buja & Eyuboglu (1992).

References

Ledoit O, Wolf M. 2004. A well-conditioned estimator for large-dimensional covariance matrices. Journal of Multivariate Analysis 88:365-411.

Buja A, Eyuboglu N. 1992. Remarks on parallel analysis. Multivariate Behavioral Research 27(4):509-540.

Examples

```
set.seed(666)
X=MASS::mvrnorm(100, mu=rep(0, 50), Sigma=diag(50))
# Simulate a multivariate random normal dataset
# with 100 observations and 50 indipendent variables

PA=parallel_analysis(X, perm = 999, fun = "fastSVD")
# Perform parallel analysis

summary(PA)
# Look at a summary of the results from parallel analysis
# Notice that no axis is significant
# This is correct, as we had simulated data with no structure

print(PA)
# Look at the p values for each axis
```

---

plot.disparity_resample

Plot method for disparity_resample objects

Description

Creates a confidence interval plot for disparity resample results

Usage

```
## S3 method for class 'disparity_resample'
plot(
  x,
  point_color = "darkblue",
  errorbar_color = "darkred",
  title = NULL,
  ...
)
```

Arguments

x	An object of class "disparity_resample"
point_color	A single color or a vector of colors for point estimates. If length 1, the same color is used for all points. If length equals the number of groups, colors are assigned per group. (default "darkblue")
errorbar_color	A single color or a vector of colors for error bars. Follows the same recycling rules as 'point_color'. (default "darkred")
title	Optional character string for plot title (default NULL)
...	Additional arguments (currently unused)

Value

A ggplot object

---

plot.EscoufierRVrarefy

Plot method for EscoufierRVrarefy objects

Description

Creates a confidence interval plot for RV rarefaction results

Usage

```
## S3 method for class 'EscoufierRVrarefy'
plot(x, point_color = "darkblue", errorbar_color = "darkred", ...)
```

Arguments

x	An object of class "EscoufierRVrarefy"
point_color	A single color or a vector of colors for point estimates. If length 1, the same color is used for all points. If length equals the number of groups, colors are assigned per group. (default "darkblue")
errorbar_color	A single color or a vector of colors for error bars. Follows the same recycling rules as 'point_color'. (default "darkred")
...	Additional arguments passed to the underlying plotting function

Value

A ggplot object

`plot.parallel_Kmult` *Plot method for parallel_Kmult objects*

Description

Creates density plots of Kmult values, with separate densities for each combination of dataset and treeset (if multiple combinations are present).

Usage

```
## S3 method for class 'parallel_Kmult'
plot(x, alpha = 0.25, title = NULL, x_lab = "Kmult", ...)
```

Arguments

`x`	An object of class 'parallel_Kmult' produced by `Kmultparallel`
`alpha`	Transparency level for density plots (0-1, default = 0.25)
`title`	Character string for plot title (default NULL for automatic title)
`x_lab`	Character string for x-axis label (default "Kmult")
`...`	Additional arguments passed to the plotting function

Value

A ggplot object

Examples

```
# Create simple example data
library(phytools)
trees = replicate(5, pbtree(n = 20), simplify = FALSE)
class(trees) = "multiPhylo"
data = matrix(rnorm(20 * 4), nrow = 20, ncol = 4)
rownames(data) = trees[[1]]$tip.label

# Run analysis and plot results
result = Kmultparallel(data, trees)
plot(result)

# With custom settings
plot(result, alpha = 0.5, title = "Kmult Distribution")
```

`pls` *Partial least squares (PLS) analysis*

Description

Performs a two-block PLS analysis, optionally allowing for tests of significance using permutations

Usage

```
pls(X, Y, perm = 999, global_RV_test = TRUE)
```

Arguments

`X, Y`	Matrices or data frames containing each block of variables (observations in rows, variables in columns).
`perm`	number of permutations to use for hypothesis testing
`global_RV_test`	logical - whether global significance of the association should be tested

Details

This function performs a PLS analysis (sensu Rohlf & Corti 2000). Given two blocks of variables (shape or other variables) scored on the same observations (specimens), this analysis finds a series of pairs of axis accounting for maximal covariance between the two blocks. If tests of significance with permutations are selected, three different significance tests are performed:

Global significance Tested using Escoufier RV

Axis-specific significance (singular value) Same test described in Rohlf & Corti 2000

Axis-specific significance (correlation of PLS scores) For each pair of PLS (singular) axes, tests the correlation between scores of the first and second block

The object of class pls_fit returned by the function has print() and summary() methods associated to it. This means that using these generic functions on an object created by this function (see examples), it is possible to obtain information on the results. In particular, print() returns a more basic set of results on the global association, whereas summary() returns (only if permutation tests are used) results for each pair of singular axes.

Value

The function outputs an object of class "pls_fit" and "list" with the following elements:

XScores Scores along each singular (PLS) axis for the first block of variables (X)

YScores Scores along each singular (PLS) axis for the second block of variables (Y)

U Left singular axes

V Right singular axes

D Singular values

percentage_squared_covariance Percented squared covariance accounted by each pair of axes

global_significance_RV (only if perm>0 and global_RV_test is TRUE) Observed value of Escoufier RV coefficient and p value obtained from the permutation test

singular_axis_significance (only if perm>0) For each pair of singular (PLS) axis, the singular value, the correlation between scores, their significance level based on permutation, and the proportion of squared covariance accounted are reported

OriginalData Data used in the analysis

x_center Values used to center data in the X block

y_center Values used to center data in the Y block

Notice

- The function does NOT perform GPA when applied to separate configurations of points.
- When using the Escoufier RV, notice that the value reported is the observed value without rarefaction. For a description of the problem, please see Fruciano et al 2013. To obtain rarefied estimates of Escoufier RV and their confidence interval, use the function RVrarefied.
- In the permutation test, rows of Y are permuted, so using the block with fewer variables as Y may speed up computations and substantially reduce memory usage.
- When using the print() and summary() on the pls_fit objects obtained with this function, some of the values are rounded for ease of interpretation. The non-rounded values can be obtained accessing individual elements of the object (see examples).

Citation

If you use this function to perform the PLS analysis and test for significance, cite Rohlf & Corti 2000 (or earlier references outside of geometric morphometrics). If you report the test of significance based on the Escoufier RV coefficient, also cite Escoufier 1975. If you also use the major axis approach to obtain estimates of the shape (or other variable) predicted by each pair of axes, please cite Fruciano et al. 2020

References

Escoufier Y. 1973. Le Traitement des Variables Vectorielles. Biometrics 29:751-760.

Fruciano C, Colangelo P, Castiglia R, Franchini P. 2020. Does divergence from normal patterns of integration increase as chromosomal fusions increase in number? A test on a house mouse hybrid zone. Current Zoology 66:527–538.

Rohlf FJ, Corti M. 2000. Use of Two-Block Partial Least-Squares to Study Covariation in Shape. Systematic Biology 49:740-753.

See Also

RVrarefied

Examples

```
##############################
### Example 1: random data ###
##############################

library(MASS)
set.seed(123)
A=as.data.frame(mvrnorm(100,mu=rep(0,20), Sigma=diag(20)))
B=as.data.frame(mvrnorm(100,mu=rep(0,10), Sigma=diag(10)))
# Create two blocks of, respectively, 20 and 10 variables
# for 100 observations.
# This simulates two different blocks of data (shape or otherwise)
# measured on the same individuals.
# Note that, as we are simulating them independently,
# we don't expect substantial covariation between blocks

PLS_AB=pls(A, B, perm=99)
# Perform PLS analysis and use 99 permutations for testing
# (notice that in a real analysis, normally one uses more permutations)
print(PLS_AB)
# As expected, we do not find significant covariation between the
# two blocks

summary(PLS_AB)
# The same happens when we look at the results for each of the axes

# Notice that both for print() and summary() some values are
# rounded for ease of visualization
# However, the correct values can be always obtained from the
# object created by the function
# e.g.,
PLS_AB$singular_axis_significance


#####################################
### Example 2: using the classical ###
### iris data set as a toy example ###
#####################################

data(iris)
# Import the iris dataset
set.seed(123)

versicolor_data=iris[iris$Species=="versicolor",]
# Select only the specimens belonging to the species Iris versicolor
versicolor_sepal=versicolor_data[,grep(
 "Sepal", colnames(versicolor_data)
)]
versicolor_petal=versicolor_data[,grep(
 "Petal", colnames(versicolor_data)
)]
# Separate sepal and petal data
```

```
PLS_sepal_petal=pls(versicolor_sepal, versicolor_petal, perm=99)
# Perform PLS with permutation test
# (again, chosen few permutations)

print(PLS_sepal_petal)
summary(PLS_sepal_petal)
# Global results and results for each axis
# (suggesting significant association)
```

`pls_major_axis` *Major axis predictions for partial least squares (PLS) analysis*

Description

Project data on the major axis of PLS scores and obtain associated predictions

Usage

```
pls_major_axis(
  pls_object,
  new_data_x = NULL,
  new_data_y = NULL,
  axes_to_use = 1,
  scale_PLS = TRUE
)
```

Arguments

`pls_object`	object of class "pls_fit" obtained from the function pls
`new_data_x`, `new_data_y`	(optional) matrices or data frames containing new data
`axes_to_use`	number of pairs of PLS axes to use in the computation (by default, this is performed only on the first axis)
`scale_PLS`	logical indicating whether PLS scores for different blocks should be scaled prior to computing the major axis

Details

This function acts on a pls_fit object obtained from the function pls. More in detail, the function:

- Projects the original data onto the major axis for each pair of PLS axes (obtaining for each observation of the original data a score along this axis).
- For each observation (specimen) of the original data, obtains the shape predicted by its score along the major axis.

- (Optionally) if new data is provided, these data are first projected in the original PLS space and then the two operations above are performed on the new data.

A more in-depth explanation with a figure which allows for a more intuitive understanding is provided in Fruciano et al 2020 The idea is to obtain individual-level estimates of the shape predicted by a PLS model. This can be useful, for instance, to quantify to which extent the shape of a given individual from one group resembles the shape that individual would have according to the model computed in another group. This can be done by obtaining predictions with this function and then computing the distance between the actual shape observed for each individual and its prediction obtained from this function. This is, indeed, how this approach has been used in Fruciano et al 2020.

The function returns a list with two or three main elements which are themselves lists. The most useful elements for the final user are highlighted in boldface.

original_major_axis_projection is a list containing as many elements as specified in axes_to_use (default 1). Each of this elements contains the details of the computation of the major axis (as a PCA of PLS scores for a pair of axes), and in particular:

major_axis_rotation Eigenvector

mean_pls_scores Mean scores for that axis pair used in the computation

pls_scale Scaling factor used

original_data_PLS_projection Scores of the original data on the major axis

original_major_axis_predictions_reversed contains the predictions of the PLS model for the original data, back-transformed to the original space (i.e., if the original data was shape, this will be shape). If axes_to_use > 1, these predictions will be based on the major axis computed for all pairs of axes considered. This element has two sub-elements:

Block1 Prediction for block 1

Block2 Prediction for block 2

new_data_results is only returned when new data is provided and contains the results of the analyses obtained using a previous PLS model on new data and, in particular:

new_data_Xscores PLS scores of the new data using the old model for the first block

new_data_Yscores PLS scores of the new data using the old model for the second block

new_data_major_axis_proj Scores of the new data on the major axis computed using the PLS model provided in pls_object. If axes_to_use > 1, each column corresponds to a separate major axis

new_data_Block1_proj_prediction_revert Predictions for Block 1 of the new data obtained by first computing the major axis projections (element new_data_major_axis_proj) and then back-transforming these projections to the original space (e.g., shape)

new_data_Block2_proj_prediction_revert Predictions for Block 2 of the new data obtained by first computing the major axis projections (element new_data_major_axis_proj) and then back-transforming these projections to the original space (e.g., shape)

Value

The function outputs a list with the following elements (please, see the Details section for explanations on their sub-elements):

original_major_axis_projection For each PLS axis pair, results of the computation of major axis and projection of the original data on each axis

original_major_axis_predictions_reversed Data obtained back-transforming the scores on the major axis into the original space (e.g., shape)

new_data_results (only if new data has been provided) PLS scores for the new data, scores of the new data on the major axis, preditions for the new data back-transformed into the original space (e.g., shape)

Citation

If you use this function, please cite Fruciano et al. 2020.

Notice

- If new data is provided, this is first centered to the same average as in the original analysis, then it is translated back to the original scale.

References

Fruciano C, Colangelo P, Castiglia R, Franchini P. 2020. Does divergence from normal patterns of integration increase as chromosomal fusions increase in number? A test on a house mouse hybrid zone. Current Zoology 66:527–538.

See Also

pls

Examples

```
####################################
### Example using the classical    ###
### iris data set as a toy example ###
####################################

data(iris)
# Import the iris dataset
versicolor_data=iris[iris$Species=="versicolor",]
# Select only the specimens belonging to the species Iris versicolor
versicolor_sepal=versicolor_data[,grep("Sepal",
                                       colnames(versicolor_data))]
versicolor_petal=versicolor_data[,grep("Petal",
                                       colnames(versicolor_data))]
# Separate sepal and petal data for I. versicolor
```

```
PLS_sepal_petal_versicolor=pls(versicolor_sepal,
                               versicolor_petal,
                               perm=99)
summary(PLS_sepal_petal_versicolor)
# Compute the PLS for I. versicolor


plot(PLS_sepal_petal_versicolor$XScores[,1],
     PLS_sepal_petal_versicolor$YScores[,1],
     asp = 1,
     xlab = "PLS 1 Block 1 scores",
     ylab = "PLS 1 Block 2 scores")
# Plot the scores for the original data on the first pair of PLS
# axes (one axis per block)
# This is the data based on which we will compute the major axis
# direction
# Imagine fitting a line through those point, that is the major axis

Pred_major_axis_versicolor=pls_major_axis(PLS_sepal_petal_versicolor,
                                          axes_to_use=1)
# Compute for I. versicolor the projections to the major axis
# using only the first pair of PLS axes (and scaling the scores
# prior to the computation)

hist(Pred_major_axis_versicolor$original_major_axis_projection[[1]]$original_data_PLS_projection,
    main="Original data - projections on the major axis - based on the first pair of PLS axes",
      xlab="Major axis score")
# Plot distribution of PLS scores for each individual in the
# original data (I. versicolor)
# projected on the major axis for the first pair of PLS axis

Pred_major_axis_versicolor$original_major_axis_predictions_reversed$Block1
Pred_major_axis_versicolor$original_major_axis_predictions_reversed$Block2
# Shape for each individual of the original data (I. versicolor)
# predicted by its position along the major axis

# Now we will use the data from new species (I. setosa and I
# virginica) and obtain predictions from the PLS model obtained for
# I. versicolor

# The easiest is to use the data for all three species
# as if they were both new data
# (using versicolor as new data is not going to affect the model)


all_sepal=iris[,grep("Sepal", colnames(iris))]
all_petal=iris[,grep("Petal", colnames(iris))]
# Separate sepals and petals (they are the two blocks)

Pred_major_axis_versicolor_newdata=pls_major_axis(
  pls_object=PLS_sepal_petal_versicolor,
  new_data_x = all_sepal,
```

```
  new_data_y = all_petal,
  axes_to_use=1)
# Perform the major axis computation using new data
# Notice that:
# - we are using the old PLS model (computed on versicolor only)
# - we are adding the new data in the same order as in the original
#   model (i.e., new_data_x is sepal data, new_data_y is petal data)


plot(Pred_major_axis_versicolor_newdata$new_data_results$new_data_Xscores[,1],
     Pred_major_axis_versicolor_newdata$new_data_results$new_data_Yscores[,1],
     col=iris$Species, asp=1,
     xlab = "Old PLS, Axis 1, Block 1",
     ylab = "Old PLS, Axis 1, Block 2")
# Plot the new data (both versicolor and setosa)
# in the space of the first pair of PLS axes computed only on
# versicolor
# The three species follow a quite similar trajectories
# but they have different average value on the major axis

# To visualize this better, we can plot the scores along the major
# axis for the three species
boxplot(Pred_major_axis_versicolor_newdata$new_data_results$new_data_major_axis_proj[,1]~
        iris$Species,
        xlab="Species",
        ylab="Score on the major axis")

# We can also visualize the deviations from the major axis
# For instance by putting the predictions of the two blocks together
# Computing differences and then performing a PCA
predictions_all_data=cbind(
 Pred_major_axis_versicolor_newdata$new_data_results$new_data_Block1_proj_prediction_revert,
 Pred_major_axis_versicolor_newdata$new_data_results$new_data_Block2_proj_prediction_revert)
# Get the predictions for the two blocks (sepals and petals)
# and put them back together

Euc_dist_from_predictions=unlist(lapply(seq(nrow(iris)),
                                        function(i)
  dist(rbind(iris[i,1:4],predictions_all_data[i,]))))
# for each flower, compute the Euclidean distance between
# the original values and what is predicted by the model

boxplot(Euc_dist_from_predictions~iris$Species,
        xlab="Species",
        ylab="Euclidean distance from prediction")
# I. setosa is the one which deviates the most from the prediction
```

print.disparity_resample *Print method for disparity_resample objects*

Description

Prints results table and checks for CI overlap among groups

Usage

```
## S3 method for class 'disparity_resample'
print(x, ...)
```

Arguments

x	An object of class "disparity_resample"
...	Additional arguments (not used)

Value

Invisibly returns the input object

---

print.EscoufierRVrarefy *Print method for EscoufierRVrarefy objects*

Description

Prints results table and checks for CI overlap among groups

Usage

```
## S3 method for class 'EscoufierRVrarefy'
print(x, ...)
```

Arguments

x	An object of class "EscoufierRVrarefy"
...	Additional arguments (not used)

Value

Invisibly returns the input object

print.parallel_Kmult *Print method for parallel_Kmult objects*

Description

Provides a summary of Kmult analysis results showing the range of Kmult values for each combination of dataset and treeset.

Usage

```
## S3 method for class 'parallel_Kmult'
print(x, ...)
```

Arguments

x	An object of class 'parallel_Kmult' produced by Kmultparallel
...	Additional arguments (currently not used)

Value

Invisibly returns the input object

Examples

```
# Create simple example data
library(phytools)
trees = replicate(3, pbtree(n = 20), simplify = FALSE)
class(trees) = "multiPhylo"
data = matrix(rnorm(20 * 4), nrow = 20, ncol = 4)
rownames(data) = trees[[1]]$tip.label

# Run analysis and print results
result = Kmultparallel(data, trees)
print(result)  # or simply: result
```

ProjectOrthogonal *Project to subspace orthogonal to a vector*

Description

This function projects data to the subspace orthogonal to a multivariate column vector

Usage

```
ProjectOrthogonal(Data, vector)
```

Arguments

Data	matrix n x p of n observation for p variables,
vector	column vector (matrix p x 1 of p variables)

Details

This function is useful to remove from a dataset the variation along a specific direction (e.g., a principal component). It has been used extensively for many applications, such as

- 'Size Correction' (removal of an allometric vector), also called 'Burnaby's method (Burnaby 1966; see also Rohlf & Bookstein 1987)
- Remove body arching from fish (Valentin et al. 2008; see also Fruciano 2016 for a discussion and other examples of usage in the context of measurement error in geometric morphometrics)
- Removing variation due to sexual dimorphism on a set of individuals with unknown sex (Fruciano et al. 2014)

Optionally, vector can also be a matrix with more than one column (in this case the Data is projected to the subspace orthogonal to the space spanned by all dimensions in vector)

Value

The function outputs a matrix n x p of the original data projected to the subspace orthogonal to the vector

References

Burnaby T. 1966. Growth-invariant discriminant functions and generalized distances. Biometrics:96-110.

Fruciano C. 2016. Measurement error in geometric morphometrics. Development Genes and Evolution 226:139-158.

Fruciano C, Pappalardo AM, Tigano C, Ferrito V. 2014. Phylogeographical relationships of Sicilian brown trout and the effects of genetic introgression on morphospace occupation. Biological Journal of the Linnean Society 112:387-398.

Rohlf FJ, Bookstein FL. 1987. A Comment on Shearing as a Method for' Size Correction'. Systematic Zoology:356-367.

Valentin AE, Penin X, Chanut JP, Sévigny JM, Rohlf FJ. 2008. Arching effect on fish body shape in geometric morphometric studies. Journal of Fish Biology 73:623-638.

Examples

```
library(MASS)
A=mvrnorm(50,mu=rep(1,50),Sigma=diag(50))
B=mvrnorm(50,mu=rep(0,50),Sigma=diag(50))
AB=rbind(A,B)
Group=as.factor(c(rep(1,50),rep(2,50)))
# Create two groups of observations (e.g., specimens)
# one centered at 0 and the other at 1
# and combine them in a single sample
```

```
PCA=prcomp(AB)
# Combine the two groups and perform a PCA

plot(PCA$x[,1],PCA$x[,2], asp=1, col=Group)
# Plot the scores along the first two principal components
# The two groups are clearly distinct (red and black)

ABproj=ProjectOrthogonal(AB,cbind(PCA$rotation[,1]))
# Project the original data (both groups)
# to the subspace orthogonal to the first principal component
# (which is the direction along which there is most of variation
# among groups)

PCAproj=prcomp(ABproj)
# Perform a new PCA on the 'corrected' dataset

plot(PCAproj$x[,1], PCAproj$x[,2], asp=1, col=Group)
# Plot the scores along the first two principal components
# of the 'corrected' data
# Notice how the two groups are now pretty much
# indistinguishable
```

`repeated_measures_test` *Perform test on two repeated measures*

Description

Test based on Hotelling's T squared for the null hypothesis of no effect between two repeated measures (e.g., treatment/control)

Usage

```
repeated_measures_test(T1, T2, rnames = TRUE, shrink = FALSE)
```

Arguments

T1, T2	matrices (n x p of n observation for p variables) or arrays (t x p x n of n observations, t landmarks in p dimensions),
rnames	if TRUE (default) the rownames of the matrix or the names along the 3rd dimension (for arrays) will be used to match the order
shrink	if TRUE, a shrinkage estimator of covariance is used internally

Details

The function assumes that each individual observation (e.g., specimen) has been measured two times (e.g., at two time points, or between two treatments).

If rnames is TRUE (default), the rownames of the matrix or the names along the 3rd dimension (for arrays) will be used to match the order of observations (e.g., specimens) between the two datasets. Otherwise, the function will assume that the observations in T1 and T2 are in the same order.

This function is useful in various contexts, such as:

- testing the effect of preservation (Fruciano et al. 2020)
- testing for variation through time

For instance, in the context of the effect of preservation on geometric morphometrics, it has been argued (Fruciano, 2016) that various studies have improperly used on repeated measures data methods developed for independent observations, and this can lead to incorrect inference.

Value

The function outputs a matrix n x p of the original data projected to the subspace orthogonal to the vector

Notice

The function requires internally non-singular matrices (for instance, number of cases should be larger than the number of variables). One solution can be to perform a principal component analysis and use the scores for all the axes with non-zero and non-near-zero eigenvalues. To overcome some situations where a singular matrix can occurr, the function can use internally a shrinkage estimator of the covariance matrix (Ledoit & Wolf 2004). This is called setting shrink = TRUE. However, in this case, the package nlshrink should have been installed. Also, notice that if the matrices T1 and T2 are provided as arrays, this requires the package Morpho to be installed.

Citation

If you use this function please cite Fruciano et al. 2020

References

Fruciano C. 2016. Measurement error in geometric morphometrics. Development Genes and Evolution 226:139-158.

Fruciano C., Schmidt, I., Ramirez Sanchez, M.M., Morek, W., Avila Valle, Z.A., Talijancic, I., Pecoraro, C., Schermann Legionnet, A. 2020. Tissue preservation can affect geometric morphometric analyses: a case study using fish body shape. Zoological Journal of the Linnean Society 188:148-162.

Ledoit O, Wolf M. 2004. A well-conditioned estimator for large-dimensional covariance matrices. Journal of Multivariate Analysis 88:365-411.

rescale_by_landmark_distance *Rescale landmark data based on interlandmark distances*

Description

Convenience function which rescales a dataset of landmarks based on a vector of distances between two landmarks

Usage

```
rescale_by_landmark_distance(Data, lm1, lm2, lengths)
```

Arguments

Data	array k x p x n of k landmarks and p dimensions for n observations (specimens)
lm1, lm2	index of the two landmarks whose inter-landmark distance is known
lengths	vector of n lengths (distances between lm1 and lm2 in the appropriate scale)

Details

This function can be useful when one has the distance between two landmarks (e.g., obtained with a caliper), but a scale has not been set when acquiring the data (for instance, a scale bar was missing on photos, so configuration of landmarks are scaled in pixels).

Value

The function outputs an array k x p x n of rescaled landmark coordinates

reversePCA *'Reverse' PCA*

Description

Obtain data in the original variables starting from PCA scores

Usage

```
reversePCA(Scores, Eigenvectors, Mean)
```

Arguments

Scores	matrix n x s of n observation for s scores. Can be a single column (for instance, if one is interested in PC1 only)
Eigenvectors	matrix p x s of eigenvectors (as column eigenvectors); notice that s (number of column eigenvectors) must be the same as the number of columns of PC scores in Scores
Mean	vector of length p with the multivariate mean computed on the original dataset (the dataset on which the PCA was carried out)

Details

Given a set of PCA scores, a set of eigenvectors and the mean of the data on which the PCA was originally computed, this function returns a set of observations in the original data space. This simple function can have many applications (not only in morphometrics) such as

- Producing predicted shapes for extreme, unobserved, values of PC scores (e.g., the shape, or other set of variables, corresponding to 3 times the most extreme positive PC1 score), or their combinations (e.g., a value of 0.4 on PC1 and 0.3 on PC2)
- Interpret results of analyses carried out on PC scores by converting these back to shapes

Value

The function outputs a matrix n x p of the scores 'back-transformed' into the original variables

Examples

```
library(MASS)
set.seed(123)
A=mvrnorm(10,mu=rep(1,2),Sigma=diag(2))
# Create a small dataset of 10 observations and 2 variables

Mean=colMeans(A)
PCA=prcomp(A)
# Compute mean and perform PCA

B=reversePCA(Scores=PCA$x, Eigenvectors=PCA$rotation, Mean=Mean)
# 'Revert' the PCA (in this case using all scores and all axes
# to get the same results as the starting data)

round(A,10)==round(B,10)
# Check that all the results are the same
# (rounding to the 10th decimal because of numerical precision)
```

`rotate_landmarks` *User-defined rotation of a landmark configuration*

Description

Rotates a single 2D or 3D landmark configuration about the origin of the coordinate system.

Usage

```
rotate_landmarks(LMdata, rotation, radians = TRUE, center = TRUE)
```

Arguments

`LMdata`	matrix k x p of k landmarks and p dimensions
`rotation`	vector containing the rotation angle(s) (a single rotation for 2D data, three rotations for 3D data)
`radians`	a logical on whether the angle(s) are provided in radians (default) or not
`center`	a logical on whether the rotation should be on the centered configuration

Details

This function can be useful to change the orientation of data for display purposes It could also be used to empirically check the rotation-invariance of an analysis. Notice that this function works on a single configuration of landmarks provided as a k x p matrix of k landmarks in p dimensions (i.e., p is 2 for 2D data and 3 for 3D data) As user-supplied rotation, the function expects a single number in the case of 2D data (rotation on the plane), a vector with three values (corresponding to rotation relative to the three axes) in 3D. If center=TRUE (default), first the configuration of landmarks is centered, then the rotation is performed, and finally the landmark coordinated are translated back to the original position. This accomplishes rotating the landmark configuration around its center

Value

The function outputs a matrix k x p of the original configuration of landmarks rotated according to the user-supplied rotation

Examples

```
library(ggplot2)

Poly1=scale(t(matrix(c(4,1,3,1,1,2,2,3),nrow=2,ncol=4)),
center=TRUE, scale=FALSE)
# Create a polygon centered at the origin
Poly2=rotate_landmarks(LMdata=Poly1, rotation=10, radians=FALSE)
# Create a new polygon which is the rotated version of the first
# with respect to the origin (rotation of 10 degrees)

BothPolys4Plotting=as.data.frame(rbind(Poly1,Poly2))
```

```
BothPolys4Plotting[,3]=c(rep("Original",4),rep("Rotated",4))
BothPolys4Plotting[,4]=rep(1:4,2)
colnames(BothPolys4Plotting)=c("X","Y","Polygon","Landmark")
# Put them together in a way that is easy to plot in ggplot2
GraphLims=range(BothPolys4Plotting[,1:2])
# limits of the plot

ggplot() +
geom_polygon(data=BothPolys4Plotting,
             mapping=aes(x=X, y=Y, group=Polygon, fill=Polygon),
             alpha=0.5) +
geom_point(data=BothPolys4Plotting, aes(x=X, y=Y, color=Polygon)) +
geom_text(data=BothPolys4Plotting, aes(x=X, y=Y, label=Landmark),
          hjust=1, vjust=1, size=4)+
coord_fixed(ratio=1, xlim=GraphLims, ylim=GraphLims)+
theme_classic()
# Plot the two polygons (landmarks are numbered for ease of
# visualization)
```

`RVcomparison` *Compare Escoufier RV coefficient between groups*

Description

Performs permutation tests for the difference in Escoufier RV between groups. For multiple groups, performs pairwise comparisons between all pairs of groups.

Usage

```
RVcomparison(Block1, Block2, group, perm = 999, center = TRUE)
```

Arguments

`Block1, Block2`	Matrices or data frames containing each block of variables (observations in rows, variables in columns).
`group`	factor or vector indicating group membership for observations.
`perm`	number of permutations for the test
`center`	whether the groups should be mean-centered prior to the test

Details

This function is one of the solutions proposed by Fruciano et al. 2013 to deal with the fact that values of Escoufier RV coefficient (Escoufier 1973), which is routinely used to quantify the levels of association between multivariate blocks of variables (landmark coordinates in the case of morphometric data, but it might be any multivariate dataset) are not comparable across groups with different number of observations/individuals (Smilde et al. 2009; Fruciano et al. 2013). The solution is a permutation test. This test was originally implemented by Adriano Franchini in the Java program RVcomparator, of which this function is an updated and improved version

Value

A data frame with one row per pairwise comparison and the following columns:

group1 Name of the first group in the comparison

group2 Name of the second group in the comparison

Observed_RV_group1 Observed Escoufier RV for the first group in the comparison

Observed_RV_group2 Observed Escoufier RV for the second group in the comparison

Absolute_difference_in_RV Absolute difference in the observed Escoufier RV between the two groups

p_value p value of the permutation test

For multiple groups, the data frame includes additional columns identifying the groups being compared.

Notice

The values of RV for each of the groups the function outputs are the observed ones, and can be reported but their value should not be compared. If one wants to obtain comparable values one solution (Fruciano et al 2013) is to obtain rarefied estimates, which can be done with the function RVrarefied in this package

Citation

If you use this function please cite both Fruciano et al. 2013 (for using the rarefaction procedure) and Escoufier 1973 (because the procedure is based on Escoufier RV)

References

Escoufier Y. 1973. Le Traitement des Variables Vectorielles. Biometrics 29:751-760.

Fruciano C, Franchini P, Meyer A. 2013. Resampling-Based Approaches to Study Variation in Morphological Modularity. PLoS ONE 8:e69376.

Smilde AK, Kiers HA, Bijlsma S, Rubingh CM, van Erk MJ. 2009. Matrix correlations for high-dimensional data: the modified RV-coefficient. Bioinformatics 25:401-405.

See Also

EscoufierRV

RVrarefied

Examples

```
library(MASS)
set.seed(123)

# Create sample data with different correlation structures
S1 = diag(50)  # Uncorrelated variables for group 1
S2 = diag(50)
S2[11:50, 11:50] = 0.3  # Some correlation in second block for group 2
```

```
S2 = (S2 + t(S2)) / 2  # Ensure symmetry
diag(S2) = 1

# Generate data for two groups
A = mvrnorm(30, mu = rep(0, 50), Sigma = S1)
B = mvrnorm(40, mu = rep(0, 50), Sigma = S2)

# Combine data and create group labels
combined_data1 = A[, 1:20]
combined_data2 = A[, 21:50]
combined_data1 = rbind(combined_data1, B[, 1:20])
combined_data2 = rbind(combined_data2, B[, 21:50])
groups = c(rep("GroupA", 30), rep("GroupB", 40))

# Perform RV comparison
result = RVcomparison(combined_data1, combined_data2,
                      group = groups, perm = 99)
print(result)

# Example with three groups for pairwise comparisons
C = mvrnorm(25, mu = rep(0, 50), Sigma = diag(50))
combined_data1_3grp = rbind(combined_data1, C[, 1:20])
combined_data2_3grp = rbind(combined_data2, C[, 21:50])
groups_3 = c(groups, rep("GroupC", 25))

result_3grp = RVcomparison(combined_data1_3grp, combined_data2_3grp,
                           group = groups_3, perm = 99)
print(result_3grp)
```

`RVrarefied` *Rarefied version of Escoufier RV coefficient*

Description

Computes a rarefied estimate of the Escoufier RV coefficient to account for the dependence on sample size of RV, so that RV can be compared among groups with different number of observations (sample sizes)

Usage

```
RVrarefied(Block1, Block2, reps = 1000, samplesize, group = NULL, CI = 0.95)
```

Arguments

`Block1, Block2`	Matrices or data frames containing each block of variables (observations in rows, variables in columns).
`reps`	number of resamplings to obtain the rarefied estimate
`samplesize`	sample size to which the rarefaction procedure is carried out

group	factor or vector indicating group membership for observations. If NULL (default), a single-group analysis is performed. If provided, the analysis is performed separately for each group.
CI	confidence interval level (default 0.95). Must be between 0 and 1.

Details

This function computes a rarefied estimate of Escoufier RV coefficient as suggested by Fruciano et al 2013 - Plos One This can be useful to compare RV among groups with the same variables but different sample sizes (as RV depends on sample size, see Fruciano et al 2013, where this procedure is described). The idea is the one rarefies the two groups at the same sample size. Following the approach in Fruciano et al. 2013 - Plos One, "rarefaction" is meant resampling to a smaller sample size with replacement (like in bootstrap).

Value

The function outputs a list with the following elements:

results A data frame with columns ‘group‘, ‘Mean‘, ‘Median‘, ‘CI_min‘, ‘CI_max‘. One row per group. For single-group analysis, group will be "All".

AllRarefiedSamples A list with all RV values obtained using the rarefaction procedure for each group. For single-group analysis, this will be a list with one element named "All".

The returned object has class "EscoufierRVrarefy" and comes with associated S3 methods for convenient display and visualization:

- print.EscoufierRVrarefy: Prints a formatted summary of results including confidence interval overlap assessment for multiple groups
- plot.EscoufierRVrarefy: Creates a confidence interval plot using ggplot2

Notice

the function does NOT perform GPA on each rarefied sample this may or may not make a difference in estimates. In many cases, it will probably not make much difference (e.g., Fig. 2 in Fruciano et al 2013)

Citation

If you use this function please cite both Fruciano et al. 2013 (for using the rarefaction procedure) and Escoufier 1973 (because the procedure is based on Escoufier RV)

References

Escoufier Y. 1973. Le Traitement des Variables Vectorielles. Biometrics 29:751-760.

Fruciano C, Franchini P, Meyer A. 2013. Resampling-Based Approaches to Study Variation in Morphological Modularity. PLoS ONE 8:e69376.

See Also

EscoufierRV

Examples

```
library(MASS)
set.seed(123)
Pop=mvrnorm(100000,mu=rep(0,100), Sigma=diag(100))
# Create a population of 100,000 'individuals'
# as multivariate normal random data
# We will consider the first 20 columns as the first
# block of variables, and the following one as the second block

A=Pop[1:50,]
B=Pop[501:700,]
# Take two groups (A and B)
# from the same population (there should be no difference
# between them)

EscoufierRV(A[,1:20],A[,21:ncol(A)])
EscoufierRV(B[,1:20],B[,21:ncol(B)])
# Notice how we obtain very different values of Escoufier RV
# (this is because they two groups have very different
# sample sizes, one 50 observations, the other 200)

RarA=RVrarefied(A[,1:20],A[,21:ncol(A)],reps=1000,samplesize=30)
RarB=RVrarefied(B[,1:20],B[,21:ncol(A)],reps=1000,samplesize=30)
RarA$results  # Data frame with Mean, Median, CI_min, CI_max
RarB$results  # Data frame with Mean, Median, CI_min, CI_max
# Rarefying both groups at the same sample size
# (in this case 30)
# it is clear that the two groups have very similar levels
# of association between blocks

# Multi-group analysis with custom CI
combined_data = rbind(A, B)
group_labels = c(rep("GroupA", nrow(A)), rep("GroupB", nrow(B)))
multi_result = RVrarefied(combined_data[,1:20], combined_data[,21:ncol(combined_data)],
                          reps=1000, samplesize=30, group=group_labels, CI=0.90)
print(multi_result$results)  # Data frame with results for each group
# Columns: group, Mean, Median, CI_min, CI_max
```

---

scaled_variance_of_eigenvalues

Compute scaled variance of eigenvalues

Description

Compute estimates of the scaled variance of eigenvalues using only the positive eigenvalues

Usage

```
scaled_variance_of_eigenvalues(
  data_matrix,
  boot = 999,
  rarefy = FALSE,
  shrinkage = FALSE
)
```

Arguments

data_matrix	Matrix or data frame containing the original data (observations in rows, variables in columns).
boot	number of bootstrap resamples (no bootstrap if 0)
rarefy	either a logical to determine whether rarefaction will be performed or a number indicating the sample size at which the samples will be rarefied
shrinkage	logical on whether the analysis should be based on a covariance matrix obtained through linear shrinkage

Details

The function allows computing the scaled variance of eigenvalues (Pavlicev et al. 2009) under a variety of settings. The scaled variance of eigenvalues is a commonly used index of morphological integration. Its value is comprised between 0 and 1, with larger values suggesting stronger integration.

Only positive eigenvalues are used in the computations used in this function.

The function employes two possible strategies to obtain eigenvalues:

- a singular value decomposition of the data matrix (default)
- an eigenvalue decomposition of the covariance matrix estimated using linear shrinkage (option shrinkage=TRUE; Ledoit & Wolf 2004)

Further, the function allows obtaining bootstrapped estimates by setting boot to the number of bootstrap replicates (resampling with replacement)

It is also possible to obtain rarefied estimates by setting rarefy to the desired sample size. This is useful when comparing the scaled variance of eigenvalues across multiple groups with different sample sizes. In this case, the suggestion is to use either the smallest sample size or less

Using a bootstrap estimate with the singular value decomposition approach represents a good compromise between computation time and accuracy. This should be complemented by rarefaction to the smallest sample size (or lower) in case one wants to compare the value obtained across different groups.

Value

If boot=0 the function outputs a vector containing the scaled variance of eigenvalues and the number of dimensions used in the computations. If, instead, boot>0 (recommended) the function outputs a list containing

- the mean scaled variance of eigenvalues across all bootstrap samples
- the median number of dimensions used across all bootstrap samples
- the 95 eigenvalues and dimensions
- the scaled variance of eigenvalues and dimensions used for each of the bootstrap replicates

Notice

When boot>0 the rarefied estimates are based on sampling with replacement (bootstrap). However, if boot=0, then a **single** rarefied estimate is obtained by sampling without replacement. In this case, the user should repeat the same operation multiple times (e.g., 100) and take the average of the scaled variance of eigenvalues obtained.

Also notice that using the shrinkage-based estimation of the covariance matrix requires longer computational time and memory. This option requires the package *nlshrink*

Computational details

For the computation, this function uses only positive eigenvalues (which are also used to identify data dimensionality). The eigenvalues are first scaled by dividing them by their sum (Young 2006), then their variance is computed as population variance (rather than sample variance; see Haber 2011). Finally, this value is standardized to a scale between 0 and 1 by dividing it by its maximum theoretical value of (p-1)/p^2 (where p is the number of dimensions) - this is the same scaling used in the software MorphoJ (Klingenberg 2011).

References

Ledoit O, Wolf M. 2004. A well-conditioned estimator for large-dimensional covariance matrices. Journal of Multivariate Analysis 88:365-411.

Young NM. 2006. Function, ontogeny and canalization of shape variance in the primate scapula. Journal of Anatomy 209:623-636.

Pavlicev M, Cheverud JM, Wagner GP. Measuring Morphological Integration Using Eigenvalue Variance. Evolutionary Biology 36(1):157–170.

Haber A. 2011. A Comparative Analysis of Integration Indices. Evolutionary Biology 38:476-488.

Klingenberg CP. 2011. MorphoJ: an integrated software package for geometric morphometrics. Molecolar Ecology Resources 11:353-357.

`summary.parallel_Kmult`

Summary method for parallel_Kmult objects

Description

Provides detailed summary statistics for Kmult analysis results.

Usage

```
## S3 method for class 'parallel_Kmult'
summary(object, ...)
```

Arguments

object	An object of class 'parallel_Kmult' produced by Kmultparallel
...	Additional arguments (currently not used)

Value

Invisibly returns the input object

Examples

```
# Create simple example data
library(phytools)
trees = replicate(3, pbtree(n = 20), simplify = FALSE)
class(trees) = "multiPhylo"
data = matrix(rnorm(20 * 4), nrow = 20, ncol = 4)
rownames(data) = trees[[1]]$tip.label

# Run analysis and get summary
result = Kmultparallel(data, trees)
summary(result)
```

TestOfAngle *Perform a test of the angle between two multivariate vectors*

Description

This function performs a test for the angle between two multivariate vector, optionally allowing for "flipping" one of the axes

Usage

```
TestOfAngle(x, y, flip = TRUE)
```

Arguments

x, y	numerical vectors
flip	logical stating whether (TRUE, default) axes should be "flipped" in case the angle is larger than 90 degrees (see Details)

Details

This function is useful to perform a test of the angle between two multivariate vectors (e.g., principal component eigenvectors computed from two covariance matrices, such as covariance matrices of two different species). The function uses internally angleTest from the package Morpho by Stefan Schlager. That function, in turn, uses the formulas in Li 2011 to compute significance, rather than using permutations. This is the same approach also implemented in MorphoJ (Klingenberg 2011). There is no special advantage in using this function relative to the original in the package Morpho, except that this function outputs angles both in radians and degrees and that it optionally allow to "flip" one of the two axes. This is useful in the cases where the direction of one axis is arbitrary, such as in PCA (where positive and negative values are interchangable and recomputing PCA might result in positive scores for the observations which were negative (and vice versa). In this situation, angles larger than 90 degrees are not meaningful and one way of dealing with this is, by choosing the option flip=TRUE (default), to change the sign of one of the two vectors ("flip").

Value

The function outputs a list with

`angle`	angle between vectors in radians
`angle_deg`	angle between vectors in degrees
`p.value`	p value
`critical_angle`	angle below which two vectors of the same number of dimensions as the ones being tested would be significant

Notice

The p value is for the probability that the angle between two random vectors is smaller or equal to the one computed from the vectors provided (x, y). This means that significant values indicate that the two provided vectors are "significantly similar", whereas non-significant values means that the two vectors are substantially different

References

Klingenberg CP. 2011. MorphoJ: an integrated software package for geometric morphometrics. Mol Ecol Resour 11:353-357.

Li S. 2011. Concise formulas for the area and volume of a hyperspherical cap. Asian Journal of Mathematics and Statistics 4:66-70.

Schlager S. 2016. Morpho: Calculations and Visualisations Related to Geometric Morphometrics.

See Also

`critical_angle`

Examples

```
library(MASS)
library(clusterGeneration)
set.seed(123)
```

```
Data=mvrnorm(500,mu=rep(1,50),Sigma=genPositiveDefMat(dim=50)$Sigma)
A=Data[1:250,]
B=Data[251:500,]
# Create two groups of observations (e.g., specimens)
# from the same multivariate normal distribution
# with the same starting covariance matrix

PCA_A=prcomp(A)
PCA_B=prcomp(B)
# Perform separate PCAs for each of the two groups of observations

TestOfAngle(PCA_A$rotation[,1],PCA_B$rotation[,1], flip=TRUE)
# Test for the angle between the two first eigenvectors
```

Index