

Quick Guide to geomorph 3.0.5

Emma Sherratt

7 August 2017

Contents

1	Getting Started	3
1.1	What is <i>geomorph</i> ?	3
1.2	How to use this manual	3
1.3	Getting help beyond this manual	6
1.4	Installing <i>R</i> and <i>geomorph</i>	6
1.5	Workflows for common analyses	9
1.6	Example datasets in <i>geomorph</i>	9
1.7	Permutation tests	13
1.8	Statistical Designs	14
2	Data Input: Importing landmark data	15
2.1	NTS files (<code>readland.nts</code>)	16
2.2	Multiple NTS files of single specimens (<code>readmulti.nts</code>)	17
2.3	Morphologika files (<code>read.morphologika</code>)	18
2.4	Other text files	20
3	Data Preparation: Manipulating landmark data and classifiers	22
3.1	Data arrays	22
3.2	Converting a 2D array into a 3D array (<code>arrayspecs</code>)	22
3.3	Converting a 3D array into a 2D array (<code>two.d.array</code>)	22
3.4	Making a factor: group variables	23
3.5	Subest 3D array of landmark coordinates by a factor (<code>coords.subset</code>)	24
3.6	Averaging 3D array of landmark coordinates by a factor	24
3.7	Estimating missing landmarks (<code>estimate.missing</code>)	25
3.8	Rotate a subset of 2D landmarks to common articulation angle (<code>fixed.angle</code>)	26
4	Generalized Procrustes Analysis	28
4.1	Generalized Procrustes Analysis (<code>gpgen</code>)	28
4.2	Generalized Procrustes Analysis with Bilateral Symmetry Analysis (<code>bilat.symmetry</code>)	33
4.3	Helper functions	36
5	DATA ANALYSIS	40
6	Covariation methods	41
6.1	Procrustes ANOVA/regression for shape data (<code>procD.lm</code>)	41
6.2	Advanced Procrustes ANOVA and pairwise tests for shape data, using complex linear models (<code>advanced.procD.lm</code>)	50
6.3	Procrustes ANOVA/regression, specifically for shape-size covariation (allometry) (<code>procD.allometry</code>)	55
6.4	Two-block partial least squares analysis for shape data (<code>two.b.pls</code>)	58
6.5	Helper function: <code>compare.pls</code>	60
7	Morphological Integration methods	63
7.1	Quantify morphological integration between two modules (<code>integration.test</code>)	63
7.2	Compare modular signal to alternative landmark subsets (<code>modularity.test</code>)	64

8	Phylogenetic Comparative methods	66
8.1	Assessing phylogenetic signal in morphometric data (<code>physignal</code>)	68
8.2	Quantify phylogenetic morphological integration between two sets of variables (<code>phylo.integration</code>)	69
8.3	Comparing rates of shape evolution on phylogenies (<code>compare.evol.rates</code>)	71
8.4	Comparing rates of shape evolution among traits on phylogenies (<code>compare.multi.evol.rates</code>)	72
8.5	Phylogenetic ANOVA/regression for shape data (<code>procD.pgls</code>)	74
9	Other methods	75
9.1	Calculate morphological disparity for one or more groups (<code>morphol.disparity</code>)	75
9.2	Quantify and compare shape change trajectories (<code>trajectory.analysis</code>)	77
10	DATA VISUALIZATION	82
11	Ordination graphs	82
11.1	Principal Components Analysis (<code>plotTangentSpace</code>)	82
11.2	Plot phylogenetic tree and specimens in tangent space (<code>plotGMPPhyloMorphoSpace</code>)	86
12	Shape change graphs	88
12.1	Shape prediction from numeric predictors (<code>shape.predictor</code>)	88
12.2	Plot shape differences between a reference and target specimen (<code>plotRefToTarget</code>)	99
13	Data inspection	107
13.1	Plot landmark coordinates for all specimens (<code>plotAllSpecimens</code>)	107
13.2	Find potential outliers (<code>plotOutliers</code>)	108
13.3	Plot 3D specimen, fixed landmarks and surface semilandmarks (<code>plotspec</code>)	110
14	Plots for analytical functions	111
14.1	plot on objects from <code>gpagen</code>	111
14.2	plot on objects from <code>bilat.symmetry</code>	111
14.3	plot on objects from <code>procD.allometry</code>	112
14.4	plot on objects from <code>advanced.procD.lm</code> , <code>procD.lm</code> , <code>procD.pgls</code> and <code>procD.allometry</code> . .	114
14.5	plot on objects from <code>phylo.modularity</code> and <code>modularity.test</code>	117
14.6	plot on objects from <code>compare.evol.rates</code> and <code>compare.multi.evol.rates</code>	117
14.7	plot on objects from <code>physignal</code>	117
14.8	plot on objects from <code>two.b.pls</code> , <code>integration.test</code> and <code>phylo.integration</code>	117
14.9	plot on objects from <code>trajectory.analysis</code>	118
14.10	Helper functions	120
15	Digitizing	125
15.1	2D data collection (<code>digitize2d</code>)	125
15.2	3D data collection: landmarks (<code>digit.fixed</code>)	127
15.3	Importing 3D surface files (<code>read.ply</code>)	129
15.4	Calculate semilandmarks along a curve (<code>digit.curves</code>)	135
15.5	Calculate linear distances between landmarks (<code>interlmkdlist</code>)	136
16	Frequently Asked Questions	137
17	References	138



Figure 1: geomorph logo (by E. Sherratt)

1 Getting Started

1.1 What is *geomorph*?

geomorph is a freely available software package for geometric morphometric analyses of two- and three-dimensional landmark (shape) data in the *R* statistical computing environment. It can be installed from the Comprehensive *R* Archive Network, CRAN <http://cran.r-project.org/web/packages/geomorph/>. Occasionally, we make updates between uploads to CRAN. Users can install via GitHub the current beta version from <https://github.com/geomorphR/geomorph>.

How to cite: When using *geomorph* in publications, please cite the software with version

```
citation(package="geomorph")
```

To cite package 'geomorph' in a publication use:

```
Adams, D. C., M. L. Collyer, A. Kaliontzopoulou, and E.
Sherratt. 2017. Geomorph: Software for geometric morphometric
analyses. R package version 3.0.5.
https://cran.r-project.org/package=geomorph.
```

A BibTeX entry for LaTeX users is

```
@Misc{,
  title = {Geomorph: Software for geometric morphometric analyses. R package version 3.0.5. https://c
  author = {D.C. Adams and M.L. Collyer and A. Kaliontzopoulou and E. Sherratt},
  year = {2017},
}
```

As *geomorph* is evolving quickly, you may want to cite also its version number (found with `'library(help = geomorph)'`).

1.2 How to use this manual

This manual is not meant to be exhaustive – the benefit of working within the *R* environment is its flexibility and infinite possibilities. Instead, the manual presents the functions in *geomorph* and how they can be used together to perform analyses to address a variety of questions in Biology, Anthropology, Paleontology, Archaeology, Medicine etc. This help guide is structured according to the pipeline outlined in **Figure 2**, which is based on a general workflow for morphometric analysis.

In **chapter 2**, we go over how to import data files of (raw) landmark coordinates digitized elsewhere, e.g., using software such as ImageJ or tpsDig for 2D data, or IDAV Landmark editor, AMIRA, Microscribe for 3D data (note that data collection – digitizing landmarks – can also be done in *geomorph*, and is outlined in **chapter 13**). Then **chapter 3** we demonstrate some techniques and functions for preparing and manipulating imported datasets, such as adding grouping variables and estimating missing data, and adjusting articulated datasets (2D only). Note that some functions described in this section can also be used on Procrustes coordinate data, but are presented here because they are important steps to learn familiarize the user with the *R* environment. In **chapter 4** the raw data are taken through the morphometric-specific step of alignment using a generalized Procrustes superimposition, which is imperative for raw coordinate data. In **chapters 5–8**, the statistical analysis functions are presented in order by type of analysis (**Table 1**). In **chapters 9–13**, we describe how to plot and visualize the data analysis results, including shape deformation graphs and ordination plots (e.g., PCA). In **chapter 14** the functions that can be used to generate coordinate data from 2D images and 3D surface files (i.e., an ASCII .ply) are discussed. In **chapter 15** there are some frequently asked questions and their solutions, and **chapter 15** gives references listed in this guide.

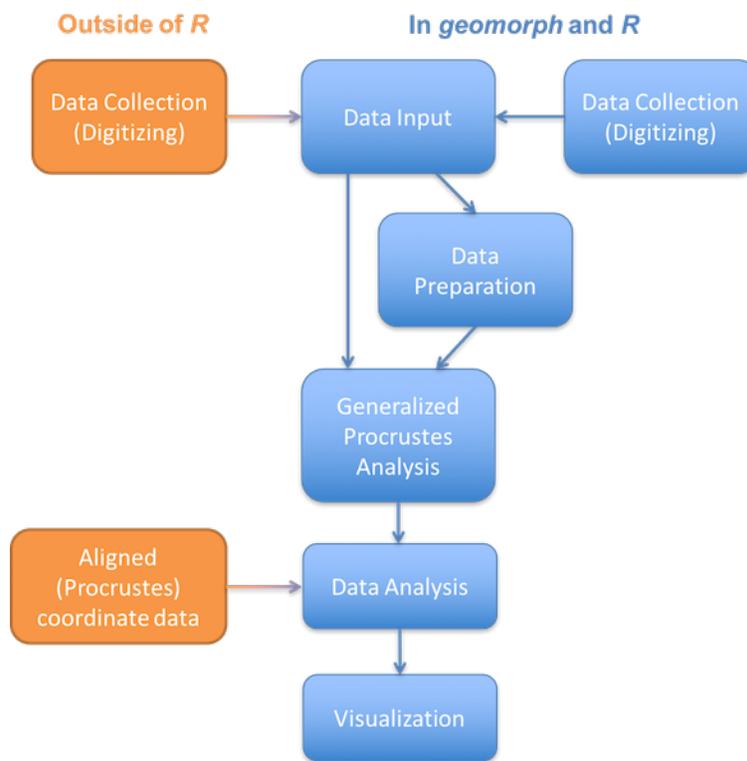


Figure 2: Overview of the morphometric analysis process. In blue are the steps performed in *R* and *geomorph*, and those in orange are done outside of *R* and imported into *R*

Table 1 Functions in *geomorph*.

Input	Preparation	Analysis
read.morphologika	arrayspecs	advanced.procD.lm
readland.nts	coords.subset	bilat.symmetry
readland.tps	define.links	compare.evol.rates
readmulti.nts	define.modules	compare.multi.evol.rates
	estimate.missing	compare.pls
	findMeanSpec	globalIntegration
	fixed.angle	gpagen
	geomorph.data.frame	integration.test
	mshape	modularity.test
	two.d.array	morphol.disparity
	writeland.tps	nested.update
		phylo.pls
		physignal
		procD.lm
		procD.PGLS
		trajectory.analysis
		two.b.pls

Visualization	Dataset	Digitizing
gridPar	hummingbirds	buildtemplate
plotAllSpecimens	larvalTails	define.sliders
plotGMPhyloMorphoSpace	larvalMorph	digit.curves
plotOutliers	mosquito	digit.fixed
plotRefToTarget	motionpaths	digitize2d
plotspec	plethodon	digitsurface
plotTangentSpace	plethspecies	editTemplate
shape.predictor	plethShapeFood	interlmkdist
warpRefMesh	pupfish	read.ply
warpRefOutline	ratland	
scallopPLY		
scallops		

Throughout this manual, we will use the following abbreviations as is conventional in morphometrics and *R*:

n number of specimens/individuals

p number of landmarks

k number of dimensions

a comment, in *R* this is text that is ignored (not run)

... data not shown if in *R* output; other options allowed if in a function code code to be written into the *R* console

[1] in a code example at the start of a line, a number in brackets denotes the first element of the output and is not intended to be typed

Briefly understanding functions; below is a *geomorph* function annotated by color:

```
procD.allometry(f1, f2 = NULL, f3 = NULL, logsz = TRUE, iter = 999,
  seed = NULL, alpha = 0.05, RRPP = TRUE, data = NULL)

readland.tps(file, specID = c("None", "ID", "imageID"), readcurves = FALSE,
  warnmsg = TRUE) # # marks a comment; it is not code.
```

In dark blue, the function name and options. In black, an object, usually data, a formula of data, or sometimes a file name. In green, a multipart option, requires choice of ONE of the presented values. In brown, a logical option that requires a TRUE or FALSE input, or an option that requires a value. In blue, an option that requires a numeric value. # in brown is a comment- this is ignored by R and used simply to tell you what the code before or after it does.

Usually only the objects are necessary to run a function, as it will use the defaults for the options (which are presented in the function as above, and under “usage” in the *R* help pages). Always read the help pages and check the examples for usage. Order does not matter as long as the option is written in full, e.g., `A= mydata`. But " " are important, e.g., `method = "RegScore"`.

Progress bars are implemented in all analytically functions (`print.progress = TRUE`) and also `warpRefMesh`, which is to help users for long-running analyses. In this manual, progress bars are not printed, for brevity.

1.3 Getting help beyond this manual

If you have questions or problems using *geomorph*, please post your questions to the *geomorph* forum: <https://groups.google.com/d/forum/geomorph-r-package>.

Finally, I occasionally write source code for very specific issues to complement *geomorph* functions. They can be found here: <http://emsherratt.github.io/MorphometricSupportCode/>.

1.4 Installing *R* and *geomorph*

To install *R*, download the latest version from <https://www.r-project.org/>. For Mac users: please also install XQuartz from <https://www.xquartz.org/>. This allows the library(rgl) to function.

We suggest also installing RStudio <https://www.rstudio.com/products/RStudio/#Desktop>.

To install *geomorph* from CRAN

```
install.packages("geomorph", dependencies = TRUE)
```

This will install the latest version of *geomorph* from CRAN <https://cran.rstudio.com/>.

Alternatively, if you prefer menus:

Rapp: Packages & Data > Package Installer > Choose CRAN (binaries) from drop down menu, type *geomorph* in box and click get list. Select *geomorph*, select install dependencies box, and click install selected.
or

Rstudio: Packages tab > Install: Install from CRAN repository > type *geomorph* in box and select install dependencies box, and click install.

1.4.1 Installing from GitHub

CRAN restricts the number of updates package maintainers can make in a year. Occasionally, bugs slip through that need to be fixed immediately. We maintain a “Stable” version of the current CRAN version of *geomorph* in our GitHub repository, which can be installed as source.

To install the source package from GitHub:

```
install.packages("devtools", dependencies = TRUE)  
devtools::install_github("geomorphR/geomorph",ref = "Stable")
```

1.4.2 Installing the beta version

We have a beta version for the upcoming version that contains the most current updates and new features. These have not been fully tested so users do so at their own risk! It is held on a GitHub repository:

To install the beta *geomorph* package:

```
devtools::install_github("geomorphR/geomorph",ref = "Develop")
```

1.4.2.1 Installing compilers for Mac users:

Previous versions of *geomorph* required users to have compilers installed in order to install packages from source. This is no longer necessary from *geomorph* version 3.0. However the information is provided here if any issues arise.

- 1) Go to the Mac App store and download Xcode Development Tools <https://itunes.apple.com/au/app/xcode/id497799835?mt=12> and follow install instructions. For OS10.6, follow the instructions here <http://kitcambridge.tumblr.com/post/17778742499/installing-the-xcode-command-line-tools-on-snow>.
- 2) Download the compilers. For OS10.8 and below: go to CRAN website here and download the GNU Fortran compiler (gfortran-4.2.3.pkg) and follow install instructions. For OS10.9 and above: Open Terminal (Applications/Utilities) and type in:

```
curl -O http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2
```

This will download the installer. Then type

```
sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /
```

This will install the compilers into the `/usr/local/lib/` folder. The command `sudo` will ask for your password. Type it in, but note it will not appear on the line. Press return. The terminal window will fill with all the files being written. (This information is thanks to the The Coatless Professor [HTTP://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2](http://r.research.att.com/libs/gfortran-4.8.2-darwin13.tar.bz2) `sudo tar fvxz gfortran-4.8.2-darwin13.tar.bz2 -C /`).

- 3) Download and install XQuartz (X11) <http://xquartz.macosforge.org/trac> if it is not already on your Mac. It will be installed in the Utilities folder. This program must be running every time you use *geomorph* (required by *rgl*).

This is a short version of information available here <http://cran.r-project.org/bin/macosx/tools/> and here <http://r.research.att.com/tools/>.

1.4.2.2 Installing compilers for Windows users:

Go to the *R* website and download RTools <http://cran.rstudio.com/bin/windows/Rtools>. Make sure to download the correct version! Follow the install instructions. You may need to modify the path (asked during the installing). Try first without ticking the box on the install window. If running `install_github` above does not work then re-install the RTools with the new path box ticked). Alternatively, use the function `find_rtools` in *devtools* package.

For more information for Windows users, see here <http://cran.r-project.org/doc/manuals/R-admin.html#The-Windows-toolset>.

1.4.3 Using *geomorph*

Regardless of how you install *geomorph*, in order to use it you must start every session by loading the package

```
library(geomorph)
```

Loading required package: *rgl*

Loading required package: ape

You'll notice that a black warning message is printed in the console saying the package rgl and ape are also loaded. All of the 3D plots of interactive functions of *geomorph* are run through rgl <https://cran.r-project.org/web/packages/rgl/index.html>. ape is called for several phylogenetic analyses.

1.5 Workflows for common analyses

Below are some pathways to perform common analyses in *geomorph*. This is not an exhaustive list, but provides a reference for users familiar with other morphometric software to navigate the functions. In red the type of question or analysis is presented, and in blue the specific *geomorph* functions in sequence.

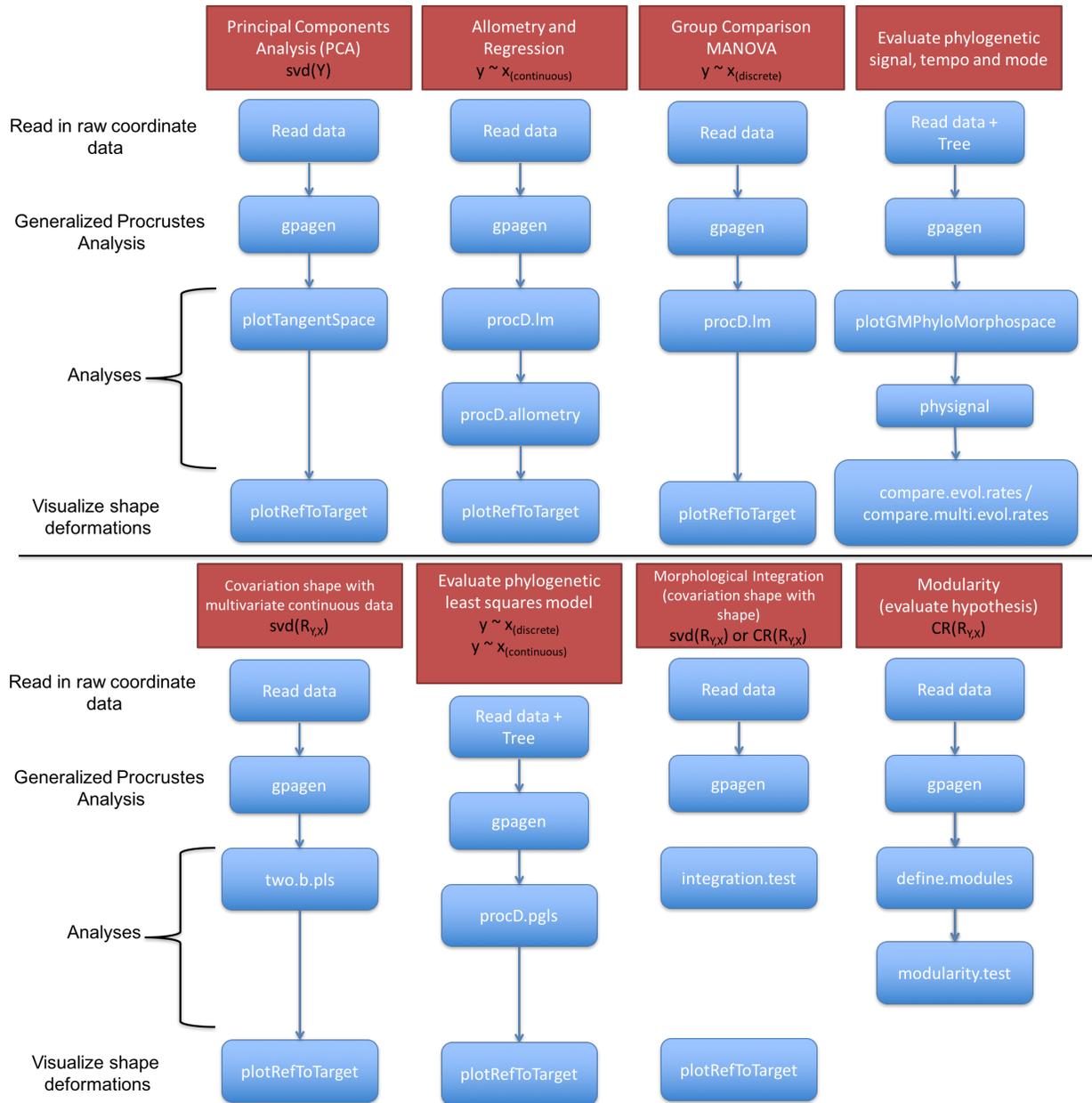


Figure 3: Example workflows in *geomorph* of morphometric analyses

1.6 Example datasets in *geomorph*

For chapters II through IV, many of the examples will be using data included with *geomorph*. There are 12 datasets: plethodon, scallops, hummingbirds, larvalTails, larvalMorph, mosquito, ratland, plethspecies, plethShapeFood, pupfish, motionpaths and scallopPLY. It is advised to run and examine

these example datasets before performing own analyses in order to understand how a function and its options work.

To load an example dataset:

```
data(plethodon)
attributes(plethodon)
```

```
$names
[1] "land"    "links"   "species" "site"    "outline"
```

```
head(plethodon$land[,1])
```

```
      [,1]      [,2]
[1,] 8.89372 53.77644
[2,] 9.26840 52.77072
[3,] 5.56104 54.21028
[4,] 1.87340 52.75100
[5,] 1.28180 53.18484
[6,] 1.24236 53.32288
```

```
head(plethodon$links)
```

```
      [,1] [,2]
[1,]    4    5
[2,]    3    5
[3,]    2    4
[4,]    1    2
[5,]    1    3
[6,]    6    7
```

```
plethodon$species
```

```
[1] Jord  Teyah
[12] Teyah Jord  Jord
[23] Jord  Jord  Jord  Jord  Jord  Jord  Jord  Jord  Jord  Teyah Teyah Teyah
[34] Teyah Teyah Teyah Teyah Teyah Teyah Teyah
Levels: Jord Teyah
```

```
plethodon$site
```

```
[1] Symp Symp
[15] Symp Symp Symp Symp Symp Symp Allo Allo Allo Allo Allo Allo Allo Allo
[29] Allo Allo
Levels: Allo Symp
```

```
head(plethodon$outline)
```

```
      [,1]      [,2]
[1,] 0.3986487 -0.2022766
[2,] 0.4000364 -0.2022312
[3,] 0.4014240 -0.2021857
[4,] 0.4028116 -0.2021400
[5,] 0.4041990 -0.2020943
[6,] 0.4055864 -0.2020484
```

The dataset above, `plethodon`, is a list containing several components: the coordinate data (`plethodon$land`), two sets of grouping variables as factors (`plethodon$species`, `plethodon$site`), the wirelink addresses (`plethodon$links`), and an matrix of outline coordinates for visualizations (`plethodon$outline`).

```
class(plethodon$site)
```

```
[1] "factor"
```

```
class(plethodon$land)
```

```
[1] "array"
```

```
class(plethodon$links)
```

```
[1] "matrix"
```

Note that your own data will not necessarily be a list - depending on how you read in each element of your data to *R*. However, learning how to manipulate the example datasets here will give you practice for working with lists later on.

Many of the novice user problems when using *geomorph* and *R* stem from having the object input in the wrong format. Here are some useful base functions in *R* to help understand formatting of one's data:

```
class() # Object Classes
attributes() # Object Attribute Lists
dim() # Dimensions of an Object
nrow() ; ncol() # The Number of Rows/Columns of a 2D array
dimnames() # Dimnames of an Object
names() # The Names of an Object
rownames() ; colnames() # Row and Column Names
is.numeric() # very useful to know if the data are numeric
```

geomorph primarily has data stored in a 2D array or 3D array (matrix and array respectively) (see below and section 2.1), grouping variables are vectors and factors, and outputs of functions may be lists. For more information about the object classes in *R* see <http://www.statmethods.net/input/datatypes.html>

1.6.1 Data arrays

Landmark data in *geomorph* can be found as objects in two formats: a 2D array (matrix; Figure 4A) or a 3D array (Figure 4B). These data formats follow the convention in other morphometric packages (e.g., *shapes*, *Morpho*) and in J.Claude's book *Morphometrics in R* (2008), and help to distinguish Shape Variables from other continuous morphometric data (linear measurements).

1.6.1.1 3D array (p x k x n)

An array with three dimensions, i.e., number of rows (*p*), number of columns (*k*) and number of "sheets" (*n*). Imagine a 3D array like a stack of filing cards. Data in this format are needed for most *geomorph* analysis functions. If one has inputted data using `readland.nts`, `readmulti.nts`, `readland.tps`, `read.morphologika`, then the data will be a 3D array object (unless the *morphologika* file contains other information, such as labels and groups, in which case the 3D array will be stored within the returned list as `$coords`). Check by typing

```
dim(plethodon$land)
```

```
[1] 12  2 40
```

If `dim` gives three numbers, it is a 3D array. Here *mydata* has *p*=12, *k*=2, *n*=40. If `dim` gives two numbers, it is a 2D array (a matrix). If `dim` returns NULL, they use `class` to determine what the format of the object is.

Understanding how to index the 3D array

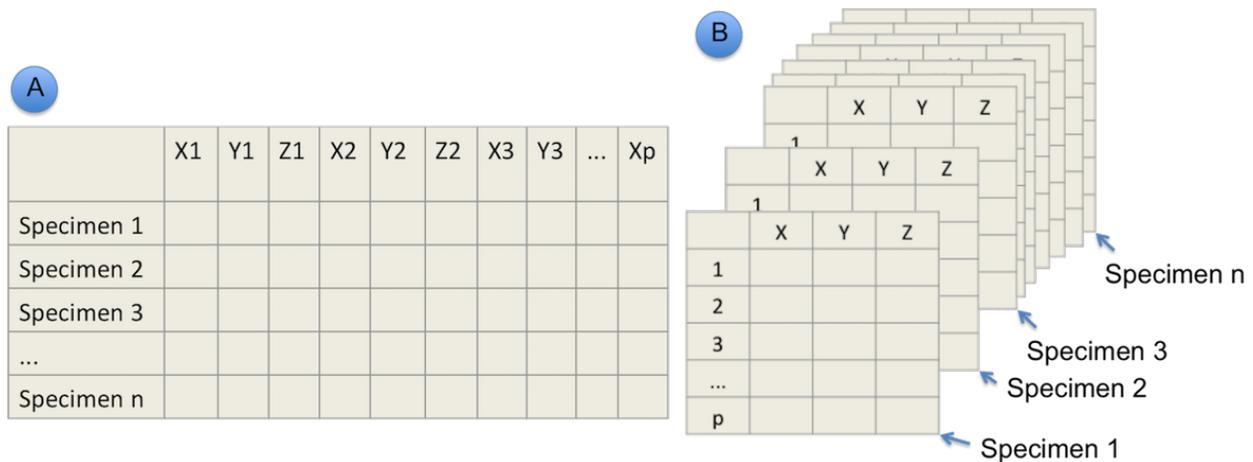


Figure 4: 2D array and 3D array of landmark coordinate data. NOTE: This example shows 3D landmark coordinate data, but the same format would be used for 2D coordinate data.

```
dim(plethodon$land)
```

```
[1] 12  2 40
```

```
# To access the first specimen (or any specimen by number)
```

```
plethodon$land[,1]
```

```
      [,1]      [,2]
[1,] 8.89372 53.77644
[2,] 9.26840 52.77072
[3,] 5.56104 54.21028
[4,] 1.87340 52.75100
[5,] 1.28180 53.18484
[6,] 1.24236 53.32288
[7,] 0.84796 54.70328
[8,] 3.35240 55.76816
[9,] 6.29068 55.70900
[10,] 8.87400 55.25544
[11,] 10.74740 55.43292
[12,] 14.39560 52.75100
```

```
# To look at coordinates of landmarks 2 & 3 of specimen 1
```

```
plethodon$land[c(2:3),1]
```

```
      [,1]      [,2]
[1,] 9.26840 52.77072
[2,] 5.56104 54.21028
```

```
# To look at the x coordinates of all landmarks of specimen 1
```

```
plethodon$land[,1,1]
```

```
[1] 8.89372 9.26840 5.56104 1.87340 1.28180 1.24236 0.84796
[8] 3.35240 6.29068 8.87400 10.74740 14.39560
```

1.6.1.2 2D array (n x [p x k])

An array (matrix) with two dimensions, i.e., number of rows (n) and number of columns (p*k).

```
dim(two.d.array(plethodon$land))
```

```
[1] 40 24
```

The 2D array is most commonly the format for other variables (n x variables), such as interlandmark distances.

1.6.2 *geomorph* data frame

A data frame in *R* is usually used for storing tables data tables. In *geomorph* a `geomorph.data.frame` is a special list that contains all data to be used in your analyses. The purpose is similar to the base *R* function, `data.frame`, but without the constraint that data must conform to an n (observations) x p (variables) matrix. Rather, the list produced is constrained only by n. List objects can be Procrustes residuals (coordinates) arrays, matrices, variables, distance matrices, and phylogenetic trees. Results from `gpagen` can be directly imported into a `geomorph.data.frame` to utilize the coordinates and centroid size as variables. The `geomorph.data.frame` is of particular importance when using the functions: `procD.lm`, `procD.pgls`, `advanced.procD.lm`, `morphol.disparity`, `procD.allometry`, `trajectory.analysis`.

```
data(plethodon)
Y.gpa <- gpagen(plethodon$land,PrinAxes=FALSE, print.progress = FALSE)
# make a geomorph dataframe from the output
gdf <- geomorph.data.frame(Y.gpa)
# here Y.gpa is a list containing coords and Csize,
# which is coreced into a geomorph dataframe
attributes(gdf)

$names
[1] "coords" "Csize"

$class
[1] "geomorph.data.frame"

gdf <- geomorph.data.frame(Y.gpa, species = plethodon$species, site = plethodon$site)
attributes(gdf)

$names
[1] "coords" "Csize" "species" "site"

$class
[1] "geomorph.data.frame"
```

As with a regular R dataframe used in functions like `lm`, the dataframe is used where a `geomorph` function has the option `data=`. And the parts of the `geomorph` dataframe are called by name, e.g.: `procD.lm(coords ~ Csize + species * site, data = gdf)`

1.7 Permutation tests

Many of the function in *geomorph* test for statistical significance using a permutation procedure (e.g., Good 2000). A randomization test takes the original data, shuffles and resamples, calculates the test statistic and compares this to the original. This is repeated for a number of iterations, creating a distribution of random tests statistics in which the original can be evaluated. The proportion of random samples that provide a better fit to the data than the original provides the P-value. Therefore the number of decimal places for the P-value is correlated to the number of iterations. When deciding how many iterations to use more is better. However there is a point where it is time consuming and not helpful (Adams and Anthony 1996). The default in *geomorph* is 999, up to 10,000 is reasonable. The examples in this manual and on the package help files

are usually very low to make them fast to run, and it is not recommended to run the function at these small iterations for one’s own data.

In several functions (e.g., `procD.lm`, `procD.gls`, `advanced.procD.lm`, `procD.allometry`), two possible resampling procedures are provided. First, if `RRPP=FALSE`, the rows of the matrix of shape variables are randomized relative to the design matrix. This is analogous to a ‘full’ randomization. Second, if `RRPP=TRUE`, a residual randomization permutation procedure is utilized (Collyer et al. 2014). Here, residual shape values from a reduced model are obtained, and are randomized with respect to the linear model under consideration. These are then added to predicted values from the remaining effects to obtain pseudo-values from which SS are calculated. NOTE: for single-factor designs, the two approaches are identical. However, when evaluating factorial models it has been shown that RRPP attains higher statistical power and thus has greater ability to identify patterns in data should they be present (see Anderson and terBraak 2003).

1.8 Statistical Designs

For functions requiring a linear model formula, `f1`, the following is a guide for different models:

An expression of the form `y ~ model` is interpreted as a specification that the response `y` is modelled by a linear predictor specified symbolically by `model` (examples in the table below; not an exhaustive list! `y` is used to denote a single dependant variable, and `Y` is a matrix for multiple dependant variables). In geometric morphometrics with shape data, the `x,y` (and `z`) coordinates of all the landmarks are treated together as `Y`.

Common designs	f1
Simple Linear Regression	<code>y ~ x</code>
Single-factor ANOVA	<code>y ~ a</code>
Multivariate Regression	<code>Y ~ x</code>
Single-factor MANOVA	<code>Y ~ a</code>
Single-factor MANCOVA	<code>Y ~ x * a</code>
Multiple-factor MANOVA	<code>Y ~ a + b</code>
Factorial MANOVA	<code>Y ~ a * b</code>
Nested MANOVA	<code>Y ~ a / b</code>

NOTE: * denotes interaction. Suggested reading: <http://www.statmethods.net/stats/anova.html>

Type I and II Sums of Squares (SS): For models that have 3 or more factors the option `int.first = TRUE/FALSE` is important. TRUE adds the interactions of first main effects before the subsequent main effects (Type I, a.k.a “sequential” SS). FALSE adds them in order (Type II SS), for example, the model: `shape ~ a*b*c`

`int.first = FALSE : shape ~ a + b + c + a:b + a:c + b:c + a:b:c`

`int.first = TRUE : shape ~ a + b + a:b + c + a:c + b:c + a:b:c`

For one or two factors, this is inconsequential and thus `int.first` can be left at default.

Finally, this manual only covers *geomorph* functions. It is recommended that users look to some “getting started with *R*” resources, such as **Quick-R**, and the **R Introduction manual**, and various Springer eBooks in the series ‘Use *R*!’. Also highly recommended is J. Claude’s book *Morphometric with R* (2008).

2 Data Input: Importing landmark data

Landmark data brought into R can be in a variety of formats. Our functions can deal with the most common: tps files, nts files, and morphologicka files. The following section describes how to use these functions on these files. At the end, we describe how data brought in as a simple excel-style matrix can be imported and manipulated into the correct format.

All of these functions return a 3D array (see section Data Preparation: Data arrays for details) which is the preferred data format for landmark data. ## TPS files (`readland.tps`) **Function**

```
readland.tps(file, specID = c("None", "ID", "imageID"), readcurves = FALSE, warnmsg = TRUE)
```

Arguments

- * *file* A .tps file containing two- or three-dimensional landmark data
- * *specID* a character specifying whether to extract the specimen ID names from the ID or IMAGE lines (default is "None")
- * *readcurves* A logical value stating whether CURVES= field and associated coordinate data will be read as semilandmarks (TRUE) or ignored (FALSE)
- * *warnmsg* A logical value stating whether warnings should be printed

This function reads a .tps file containing two- or three-dimensional landmark coordinates for a set of specimens. Tps files are text files in one of the standard formats for geometric morphometrics (see Rohlf 2010). Two-dimensional landmarks coordinates are designated by the identifier "LM=", while three-dimensional data are designated by "LM3=". Landmark coordinates are multiplied by their scale factor if this is provided for all specimens. If one or more specimens are missing the scale factor (there is no line "SCALE="), landmarks are treated in their original units.

The name of the specimen can be given in the tps file by "ID=" (use `specID="ID"`) or "IMAGE=" (use `specID="imageID"`), otherwise the function defaults to `specID="None"`.

If there are curves defined in the file (i.e., CURVES= fields), the option `readcurves` should be used. When `readcurves = TRUE`, the coordinate data for the curves will be returned as semilandmarks and will be appended to the fixed landmark data. Then the user needs to use `define.sliders` to create a matrix designating how the curve points will slide (used by 'curves=' in `gpagen`). When `readcurves = FALSE`, only the landmark data are returned (the curves are ignored).

At present, all other information that can be contained in tps files (comments, variables, radii, etc.) is ignored. E.g. a text file called "ratland.tps" in the .tps format:

```
ratland.tps
LM=8
-0.45 -0.475
-0.59 -0.28
-0.515 -0.12
-0.33 0
0 0
0.145 -0.395
-0.045 -0.42
-0.26 -0.465
ID = specimen103N

LM=8
...
```

To read into *R*:

```
mydata <- readland.tps("ratland.tps", specID = "ID")
[1] "Not all specimens have scale. Using scale = 1.0"
```

```
mydata[, ,1]
      [,1] [,2]
[1,] -0.450 -0.475
[2,] -0.590 -0.280
[3,] -0.515 -0.120
[4,] -0.330  0.000
[5,]  0.000  0.000
[6,]  0.145 -0.395
[7,] -0.045 -0.420
[8,] -0.260 -0.465
# This is the coordinates for first specimen in file
```

In this case, because there is no scale given in the tps file, the command simply warns that the data are treated in their original units. The function returns a 3D array containing the coordinate data, and if provided in the file, the names of the specimens (`dimnames(mydata)[[3]]`).

2.1 NTS files (`readland.nts`)

Function

```
readland.nts(file)
```

Arguments

- *file* A .nts file containing two- or three-dimensional landmark data for a set of specimens

Function reads a single .nts file containing a matrix of two- or three-dimensional landmark coordinates for a set of specimens. NTS files are text files in one of the standard formats for geometric morphometrics (see Rohlf 2012). The parameter line contains 5 or 6 elements, and must begin with a “1” to designate a rectangular matrix. The second and third values designate how many specimens (*n*) and how many total variables (*p* × *k*) are in the data matrix. The fourth value is a “0” if the data matrix is complete and a “1” if there are missing values. If missing values are present, the ‘1’ is followed by the arbitrary numeric code used to represent missing values (e.g., -999). These values will be replaced with “NA” in the output array. The final value of the parameter line denotes the dimensionality of the landmarks (2,3) and begins with “DIM=”. If specimen and variable labels are included, these are designated placing an “L” immediately following the specimen or variable values in the parameter file. The labels then precede the data matrix. Here there are *n* = 44 and *p***k* = 50 (25 2D landmarks). E.g. For a file that looks like:

```
rats.nts

" rats data, 164 rats, 8 landmarks in 2 dimensions

1 164 16 0 dim=2
-0.450 -0.475 -0.590 -0.280 -0.515 -0.120 -0.330 0 0 0 0.145 -0.395 -0.045 -0.420 -0.260 -0.465
-0.530 -0.555 -0.685 -0.320 -0.625 -0.120 -0.400 0 0 0 0.230 -0.425 -0.005 -0.480 -0.265 -0.525
-0.560 -0.570 -0.700 -0.335 -0.670 -0.120 -0.425 0 0 0 0.300 -0.440 0.015 -0.495 -0.270 -0.540
-0.590 -0.580 -0.745 -0.355 -0.700 -0.100 -0.435 0 0 0 0.330 -0.445 0.030 -0.505 -0.285 -0.565
-0.650 -0.580 -0.800 -0.340 -0.715 -0.090 -0.450 0 0 0 0.360 -0.445 0.040 -0.515 -0.300 -0.580
...
```

Here, the 2D coordinate data are given for each specimen (each line)

To read into *R*:

```
mydata <- readland.nts("rats.nts")
mydata[, ,1]
      [,1] [,2]
```

```

[1,] -0.450 -0.475
[2,] -0.590 -0.280
[3,] -0.515 -0.120
[4,] -0.330  0.000
...
mydata[, ,2]
      [,1] [,2]
[1,] -0.530 -0.555
[2,] -0.685 -0.320
[3,] -0.625 -0.120
[4,] -0.400  0.000
...

```

The function returns a 3D array containing the coordinate data, and if provided in the file, the names of the specimens, which can be accessed by: `dimnames(mydata)[[3]]`

Function is for *.nts file containing landmark coordinates for multiple specimens*. Note that *.dta* files in the *nts* format written by Landmark Editor <http://graphics.idav.ucdavis.edu/research/projects/EvoMorph>, and **.nts* files written by Stratovan Checkpoint <http://www.stratovan.com/> have incorrect header notation; every header is `1 n p-x-k 1 9999 Dim=3`, rather than `1 n p-x-k 0 Dim=3`, which denotes that missing data is in the file even when it is not. NAs will be introduced unless the header is manually altered.

2.2 Multiple NTS files of single specimens (`readmulti.nts`)

Function

```
readmulti.nts(filelist)
```

Arguments

- *filelist* A vector of names of *.nts* files containing two- or three-dimensional landmark data

This function reads a list containing the names of multiple *.nts files*, where each *.nts* file contains the landmark coordinates for a single specimen, e.g. made by `digit.fixed` or `digitsurface`.

For these files, the number of variables (columns) of the data matrix will equal the number of dimensions of the landmark data ($k = 2$ or 3). The parameter line contains 5 or 6 elements, and must begin with a “1” to designate a rectangular matrix. The second and third values designate the number of landmarks (p) and the dimensionality of the data (k) in the data matrix. The fourth value is a “0” if the data matrix is complete and a “1” if there are missing values. If missing values are present, the ‘1’ is followed by the arbitrary numeric code used to represent missing values (e.g., -999). These values will be replaced with “NA” in the output array. The final value of the parameter line denotes the dimensionality of the landmarks (2,3) and begins with “DIM=”. The specimen label is extracted from the file name, not the header. Here is an example of 3 *.nts* files, each $p = 166$, $k = 3$. These are then read and concatenated into a single 3D array for all specimens.

```

filelist <- list.files(pattern = ".nts")
filelist
[1] "ball01L.nts" "ball02L.nts" "ball03L.nts" ...

```

What the *nts* file looks like:

```
ball01L.nts
```

```

1 166 3 0 dim=3
37.366242091765 -19.7782715772904 -1.45757328357893
45.336342091765 -15.9657715772904 -4.28288328357893
45.562042091765 -1.20527157729041 -5.17616328357893

```

```

47.607342091765 13.3546284227096 -6.41333328357893
39.940142091765 18.5793284227096 -4.27434328357893
-44.504657908235 0.138928422709595 -5.40957328357893
-2.61418790823501 47.4933284227096 -3.09240328357893
-8.00038890823501 -45.0109715772904 2.45829671642107
30.500142091765 35.9411284227096 -2.80191328357893
...

```

To read into *R*:

```

mydata <- readmulti.nts(filelist)
mydata
, , ball01L

      [,1]      [,2]      [,3]
[1,] 37.3662421 -19.77827158 -1.45757328
[2,] 45.3363421 -15.96577158 -4.28288328
[3,] 45.5620421 -1.20527158 -5.17616328
[4,] 47.6073421 13.35462842 -6.41333328
[5,] 39.9401421 18.57932842 -4.27434328
...

```

The function returns a 3D array containing the coordinate data, and the names of the specimens (dimnames(mydata)[[3]]) extracted from the file names.

2.3 Morphologika files (read.morphologika)

Function

```
read.morphologika(file)
```

Arguments

- *file* A .txt file containing two- or three-dimensional landmark data for a set of specimens

This function reads a .txt file in the Morphologika format containing two- or three-dimensional landmark coordinates. Morphologika files are text files in one of the standard formats for geometric morphometrics (see O’Higgins and Jones 1998), see <http://sites.google.com/site/hymsfme/resources>. If the headers “[labels]”, “[labelvalues]” and “[groups]” are present in the file, then a data matrix containing all individual specimen information is returned. If the header “[wireframe]” is present, then a matrix of the landmark addresses for the wireframe is returned. If the header “[polygon]” is present, then a matrix of the landmark addresses for the polygon wireframe is returned.

The file looks like:

```

morphologikaexample.txt
[individuals]
15
[landmarks]
31
[Dimensions]
3
[names]
Specimen 1
Specimen 2
Specimen 3
...

```

```

Specimen 15
[labels]
Sex
[labelvalues]
Female
Female
...
Female
[rawpoints]
'#1
16.01 24.17 11.18
15 24.86 11.16
14.96 25.54 11.52
16.26 24.36 11.48
15.89 26.61 11.83
17.16 25.33 12.35
18.22 23.65 11.12
...

```

To read into *R*:

```

mydata <- read.morphologika("morphologikaexample.txt")
mydata$coords[, , 1]
      [,1] [,2] [,3]
[1,] 16.01 24.17 11.18
[2,] 15.00 24.86 11.16
[3,] 14.96 25.54 11.52
[4,] 16.26 24.36 11.48
[5,] 15.89 26.61 11.83
[6,] 17.16 25.33 12.35
...
mydata$labels
      Sex
Specimen 1 "Female"
Specimen 2 "Female"
Specimen 3 "Female"
Specimen 4 "Female"
Specimen 5 "Male"
Specimen 6 "Male"
...

```

The function returns a 3D array containing the coordinate data, and if provided, the names of the specimens (`dimnames(mydata)[[3]]`). If other optional headers are present in the file (e.g., “[labels]” or “[wireframe]”) function returns a list containing the 3D array of coordinates (`coords`), *and a data matrix of the data from labels* (`labels`) and/or the landmark addresses denoting the wireframe (`$wireframe`) – which can be passed to `plotRefToTarget` option ‘links’.

To read multiple Morphologika files that each contain a single specimen, download this file (<https://github.com/EmSherratt/MorphometricSupportCode/blob/master/read.multi.morphologika.R>) put in the working directory then to use:

```

source("read.multi.morphologika.R")
filelist <- list.files(pattern = "*.txt") ## list all morpholgika files
mydata <- read.multi.morphologika("morphologikaexample.txt")

```

2.4 Other text files

Base Functions: `read.table(file)` `read.csv(file)`

Using base read functions in R, one can read in data by many other ways. Here are two examples. These examples use data arrangement function `arrayspecs` (see section 2.1 for details) and creates an object in the same way as the previous functions. e.g. For a set of files (`file1.txt`, `file2.txt`, `file3.txt...`) each containing the landmark coordinates of a single specimen like:

```
file1.txt
16.01 24.17 11.18
15 24.86 11.16
14.96 25.54 11.52
16.26 24.36 11.48
15.89 26.61 11.83
17.16 25.33 12.35
18.22 23.65 11.12
...
```

To read into *R*:

```
filelist <- list.files(pattern = ".txt") # makes a list of all .txt files in working directory
names <- gsub(".txt", "", filelist) # extracts names of specimens from the file name
coords = NULL # make an empty object
for (i in 1:length(filelist)){
  tmp <- as.matrix(read.table(filelist[i]))
  coords <- rbind(coords, tmp)
}
coords <- arrayspecs(coords, p, k)
dimnames(coords)[[3]] <- names
```

e.g. For a single file containing the landmark coordinates of a set of specimens, where each row is a specimen, and coordinate data arranged in columns `x1`, `y1`, `x2`, `y2...` etc., and the first column is the ID of the specimens, e.g., from a data file exported from MorphoJ.

```
coordinatedata.txt
ID X1 Y1 X2 Y2 X3 Y3 ...
specimen1 0.595 0.1679 0.2232 0.5028 1.292 0.4237 0.51 ...
specimen2 0.0038 1.3925 0.7966 0.4132 0.1006 0.8483 ...
specimen3 0.6249 0.4515 0.3576 1.3262 0.9114 0.3611 ...
...
```

```
mydata <- read.table("coordinatedata.txt",header=TRUE,row.names=1,
stringsAsFactors = FALSE)
# The stringsAsFactors = FALSE is VERY important here
# Here row.names = 1 means "set the row names of the object to be the values in column 1".

is.numeric(mydata)
[1] FALSE
```

Here *R* tells us the data are not numeric, even though we can see they are if we use `View(mydata)`. Why? Because if there are characters in the file, all elements are automatically read as characters not numerical data

The solution is to force those to be numeric with: `as.matrix`. For example, say we know the shape coordinates are present in the file after two columns of non-shape coordinates (these could be centroid size, or a classifier), then:

```
coords <- as.matrix(mydata[,-(1:2)]) # here we say, use all columns except the first two.  
  
is.numeric(shape)  
[1] TRUE # now it's numeric 2D array  
coords <- arrayspecs(coords, p, k) # makes the matrix a 3D array
```

If the data file contains classifier variables, these can be extracted by subsetting columns. For example, if the classifiers are in the first two columns, then:

```
classifiers <- mydata[,1:2] # and if they are classifiers, they will probably need to be factors so:  
classifiers <- factor(classifiers)
```

3 Data Preparation: Manipulating landmark data and classifiers

3.1 Data arrays

Landmark data in *geomorph* can be found as objects in two formats: a 2D array (matrix) or a 3D array (see chapter 1.5.1). These data formats follow the convention in other morphometric packages (e.g., *shapes*, *Morpho*) and in J.Claude's book *Morphometrics in R* (2008), and help to distinguish Shape Variables from other continuous morphometric data (linear measurements).

3.2 Converting a 2D array into a 3D array (`arrayspecs`)

Function

```
arrayspecs(A, p, k)
```

Arguments

- *A* A 2D array (matrix) containing landmark coordinates for a set of specimens
- *p* Number of landmarks
- *k* Number of dimensions (2 or 3)

This function converts a matrix of landmark coordinates into a 3D array ($p \times k \times n$), which is the required input format for many functions in *geomorph*. The input matrix can be arranged such that the coordinates of each landmark are found on a separate row, or that each row contains all landmark coordinates for a single specimen.

```
A <- arrayspecs(mydata, p, k) # where mydata is a 2D array
A[, , 1] # look at just the first specimen
, , 1
      [,1] [,2]
[1,] 8.89372 53.77644
[2,] 9.26840 52.77072
[3,] 5.56104 54.21028
[4,] 1.87340 52.75100
[5,] 1.28180 53.18484
[6,] 1.24236 53.32288
[7,] 0.84796 54.70328
[8,] 3.35240 55.76816
[9,] 6.29068 55.70900
[10,] 8.87400 55.25544
[11,] 10.74740 55.43292
[12,] 14.39560 52.75100
```

3.3 Converting a 3D array into a 2D array (`two.d.array`)

Function

```
two.d.array(A)
```

Arguments

- *A* A 3D array containing landmark coordinates for a set of specimens

This function converts a 3D array ($p \times k \times n$) of landmark coordinates into a 2D array ($n \times [p \times k]$). The latter format of the shape data is useful for performing subsequent statistical analyses in *R* (e.g., PCA, MANOVA, PLS, etc.). Row labels are preserved if included in the original array.

```
a <- two.d.array(mydata) # where mydata is a 3D array
a
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,] 8.893720 53.77644 9.268400 52.77072 5.561040 54.21028 1.873400 52.75100 1.281800 53.18484
[2,] 8.679762 54.57819 8.935628 53.83027 5.451914 54.65691 1.987882 52.68871 1.515514 53.02331
[3,] 9.805328 56.06903 10.137712 55.27961 6.647680 55.73664 3.448484 53.86698 3.012230 54.34478
[4,] 9.637164 58.03294 9.952104 56.77318 6.109836 57.94896 2.645496 55.89135 2.015616 56.62621
[5,] 11.035692 58.75009 11.335110 57.85184 8.255382 58.92119 4.555431 57.46687 3.956595 58.08709
...
```

3.4 Making a factor: group variables

Many analyses will require a grouping variable (a classifier) for the data. For small datasets, this can be made easily within *R*:

```
group <- factor(c(0,0,1,0,0,1,1,0,0)) # specimens assigned in order to group 0 or 1
# assign specimen names from 3D array of data to the group classifier
names(group) <- dimnames(mydata)[[3]]
group
[1] 0 0 1 0 0 1 1 0 0
Levels: 0 1
```

If the data have many specimens or many different groups, it may be easier to make a table in excel, save as a .csv file and import using `read.csv`. classifier.csv

ID	Species	Habitat
specimen1	A.species	dry
specimen2	A.notherspecies	wet
specimen3	A.species	dry
specimen4	A.species	dry
specimen5	A.species	wet
specimen6	A.notherspecies	wet
specimen7	A.notherspecies	wet
specimen8	A.notherspecies	wet
specimen9	A.notherspecies	dry
...

Figure 5: Example of a classifier matrix

```
classifier <- read.csv("classifier.csv", header=T, row.names=1)
is.factor(classifier$Habitat) # check that it is a factor
[1] TRUE
classifier$Habitat
[1] dry wet dry dry wet wet wet wet dry ...
Levels: dry wet
```

Note: When reading in a data file, *R* usually treats character variables as factors, but numeric variables are not, and therefore must be coerced into factors. See `?factor` for more information. `##` Lists Another common data structure in *R* is a list. In essence, a list is a generic vector containing other objects. In *geomorph*, the example data are all lists. e.g. `plethodon`:

```
library(geomorph)
data(plethodon)
```

```
attributes(plethodon)
```

```
$names  
[1] "land"    "links"   "species" "site"    "outline"
```

Here, plethodon is made up of 5 named objects (which are each vectors, matrices/2D arrays or 3D arrays). To access named objects within a list, use `$`, such as `plethodon$links`. The returned output of many *geomorph* functions is also in named list form. Parts of a list can also be accessed using numbers with double square brackets `[[]]`. So in the above example, `links` is the 2nd object in the plethodon list, and so can also be accessed by `plethodon[[2]]`. For more details on this, a good resource is the website *R* tutorial (<http://www.r-tutor.com/r-introduction/list>).

3.5 Subest 3D array of landmark coordinates by a factor (`coords.subset`)

This function takes a factor and splits a set of landmark coordinates into subsets, as described by the factor. The results is a list of separate sets of landmarks. See above for more information on lists.

Function

```
coords.subset(A, group)
```

Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of specimens
- *group* A grouping factor of length n, for splitting the array into sub-arrays

To use the function, let's use an example using pupfish dataset:

```
data(pupfish)  
group <- factor(paste(pupfish$Pop, pupfish$Sex)) # make a 4 level factor  
levels(group) # see the levels  
  
## [1] "Marsh F"    "Marsh M"    "Sinkhole F" "Sinkhole M"  
  
new.coords <- coords.subset(A = pupfish$coords, group = group)  
names(new.coords) # see the list levels  
  
## [1] "Marsh F"    "Marsh M"    "Sinkhole F" "Sinkhole M"  
  
# access any element by:  
# new.coords$`Marsh F` # Can be used in any analysis, just like pupfish$coords  
# note `` surrounds level name because it has a space in it  
  
# group shape means  
group.means <- lapply(new.coords, mshape)
```

3.6 Averaging 3D array of landmark coordinates by a factor

There is no *geomorph* function to do this, because one can easily do this with `aggregate`. See <https://github.com/geomorphR/geomorph/wiki/Averaging-data-by-group> for more details.

To use `aggregate` function, let's use an example using pupfish dataset:

```
data(pupfish)  
group <- factor(paste(pupfish$Pop, pupfish$Sex)) # make a 4 level factor  
levels(group) # see the levels  
  
## [1] "Marsh F"    "Marsh M"    "Sinkhole F" "Sinkhole M"
```

```

new.coords <- coords.subset(A = pupfish$coords, group = group)
means <- (aggregate(two.d.array(pupfish$coords) ~ group, FUN=mean))[, -1]
# function requires a 2D array format,
# also returns the group names in the first column so we omit that
rownames(means) <- levels(group)
means <- arrayspecs(means, 2, 56) # make a 3D array again

```

3.7 Estimating missing landmarks (estimate.missing)

All analysis and plotting functions in *geomorph* require a full complement of landmark coordinates. Either the missing values are estimated, or subsequent analyses are performed on a subset dataset excluding specimens with missing values. Below is the function to estimate missing data, followed by steps of how to just exclude specimens with missing values.

Function

```
estimate.missing(A, method = c("TPS", "Reg"))
```

Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of specimens
- *method* Method for estimating missing landmark locations

The function estimates the locations of missing landmarks for incomplete specimens in a set of landmark configurations, where missing landmarks in the incomplete specimens are designated by NA in place of the x,y,z coordinates. Two distinct approaches are implemented.

1. The first approach (`method="TPS"`) uses the thin-plate spline to interpolate landmarks on a reference specimen to estimate the locations of missing landmarks on a target specimen. Here, a reference specimen is obtained from the set of specimens for which all landmarks are present, Next, each incomplete specimen is aligned to the reference using the set of landmarks common to both. Finally, the thin-plate spline is used to estimate the locations of the missing landmarks in the target specimen (Gunz et al. 2009).
2. The second approach (`method="Reg"`) is multivariate regression. Here each landmark with missing values is regressed on all other landmarks for the set of complete specimens, and the missing landmark values are then predicted by this linear regression model. Because the number of variables can exceed the number of specimens, the regression is implemented on scores along the first set of PLS axes for the complete and incomplete blocks of landmarks (see Gunz et al. 2009).

One can also exploit bilateral symmetry to estimate the locations of missing landmarks. Several possibilities exist for implementing this approach (see Gunz et al. 2009). Example *R* code for one implementation is found in Claude (2008).

Missing landmarks in a target specimen are designated by NA in place of the x,y,z coordinates. To make this so:

```

any(is.na(mydata)) # check if there are NAs in the data
FALSE # if false then,
mydata[which(mydata == -999)] <- NA # change missing values from "-999" to NAs

```

.nts files give a value in the header that is used to designate missing data (often 9999, -999 etc.). The `which(mydata == -999)` searches for these values and replaces with NA.

To use the function, let's use an example using *Plethodon* dataset:

```

data(plethodon)
plethland<-plethodon$land

```

```

plethland[2,,2]<-plethland[6,,2]<-NA # create missing landmarks
plethland[2,,5]<-plethland[6,,5]<-NA # create missing landmarks
plethland[2,,10]<-NA # create missing landmarks
new.plethland <- estimate.missing(plethland,method="TPS")
new.plethland <- estimate.missing(plethland,method="Reg")

```

The function returns a 3D array with the missing landmarks estimated.

Instead of estimating missing, an alternative is to proceed with the specimens for which data are missing excluded. For example to make a dataset of only the complete specimens (starting with the dataset as 2D array), two ways are possible:

mydata

```

      [,1]      [,2]      [,3]      [,4]
[1,] 8.893720 53.77644  9.268400 52.77072
[2,] 8.679762 54.57819  8.935628 53.83027
[3,] 9.805328 56.06903  NA      NA
[4,] 9.637164 58.03294  9.952104 56.77318
[5,] NA      NA      11.335110 57.85184
[6,] 7.946625 55.71114  8.476400 54.82112
[7,] 8.849841 58.66961  9.396387 57.82877
[8,] 9.331504 56.36904 10.154872 55.31344

```

```

newdata <- mydata[complete.cases(mydata),] # keep only specimens with complete data
# OR
newdata <- na.omit(mydata) # use only specimens without NAs

```

newdata

```

      [,1]      [,2]      [,3]      [,4]
[1,] 8.893720 53.77644  9.268400 52.77072
[2,] 8.679762 54.57819  8.935628 53.83027
[3,] 9.637164 58.03294  9.952104 56.77318
[4,] 7.946625 55.71114  8.476400 54.82112
[5,] 8.849841 58.66961  9.396387 57.82877
[6,] 9.331504 56.36904 10.154872 55.31344

```

These functions can be used to make a dataset of only the landmarks in all specimens, by inputting the matrix `mydata` in transpose, e.g., `t(mydata)`. Note that these methods will re-label the specimen or landmark numbers.

3.8 Rotate a subset of 2D landmarks to common articulation angle (`fixed.angle`)

A function for rotating a subset of landmarks so that the articulation angle between subsets is constant. Presently, the function is only implemented for two-dimensional landmark data. **Function**

```

fixed.angle(A, art.pt = NULL, angle.pts = NULL, rot.pts = NULL,
            angle = 0, degrees = FALSE)

```

Arguments

- `A` A 3D array ($p \times k \times n$) containing landmark coordinates for a set of specimens
- `art.pt` A number specifying which landmark is the articulation point between the two landmark subsets
- `angle.pts` A vector containing numbers specifying which two points used to define the angle (one per subset)

- *rot.pts* A vector containing numbers specifying which landmarks are in the subset to be rotated
- *angle* An optional value specifying the additional amount by which the rotation should be augmented (in radians)
- *degrees* A logical value specifying whether the additional rotation angle is expressed in degrees or radians (radians is default)

This function standardizes the angle between two subsets of landmarks for a set of specimens. The approach assumes a simple hinge-point articulation between the two subsets, and rotates all specimens such that the angle between landmark subsets is equal across specimens (see Adams 1999). As a default, the mean angle is used, though the user may specify an additional amount by which this may be augmented.

Example using *Plethodon*. Articulation point is landmark 1, rotate mandibular landmarks (2-5) relative to cranium

```
data(plethspecies)
new.plethdata <- fixed.angle(plethspecies$land,
                             art.pt=1,
                             angle.pts=c(5,6),
                             rot.pts=c(2,3,4,5))
```

Function returns a 3D array containing the newly rotated data.

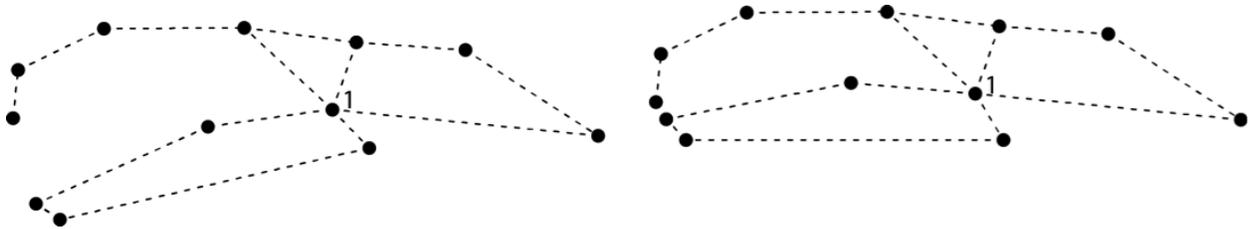


Figure 6: The position of the mandible before running `fixed.angle` (left) and after (right)

4 Generalized Procrustes Analysis

4.1 Generalized Procrustes Analysis (gpagen)

Generalized Procrustes Analysis (GPA: Gower 1975; Rohlf and Slice 1990) is the primary means by which shape variables are obtained from landmark data (for a general overview of geometric morphometrics see Bookstein 1991; Rohlf and Marcus 1993; Adams et al. 2004; Mitteroecker and Gunz 2009; Zelditch et al. 2012; Adams et al. 2013). GPA translates all specimens to the origin, scales them to unit-centroid size, and optimally rotates them (using a least-squares criterion) until the coordinates of corresponding points align as closely as possible. The resulting aligned Procrustes coordinates represent the shape of each specimen, and are found in a curved space related to Kendall’s shape space (Kendall 1984). Typically, these are projected into a linear tangent space yielding Kendall’s tangent space coordinates (Dryden and Mardia 1993; Rohlf 1999), which are used for subsequent multivariate analyses. Additionally, any semilandmarks on curves and are slid along their tangent directions or tangent planes during the superimposition (see Bookstein 1997; Gunz et al. 2005). Presently, two implementations are possible: 1) the locations of semilandmarks can be optimized by minimizing the bending energy between the reference and target specimen (Bookstein 1997), or by minimizing the Procrustes distance between the two (Rohlf 2010).

The first step in any geometric morphometric analysis is to perform a Procrustes superimposition of the raw coordinate data

Function

```
gpagen(A, curves = NULL, surfaces = NULL, PrinAxes = TRUE,  
       max.iter = NULL, ProcD = TRUE, Proj = TRUE, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of specimens
- *curves* An optional matrix defining which landmarks should be treated as semilandmarks on boundary curves, and which landmarks specify the tangent directions for their sliding (see `define.sliders`)
- *surfaces* An optional vector defining which landmarks should be treated as semilandmarks on surfaces
- *PrinAxes* A logical value indicating whether or not to align the shape data by principal axes
- *max.iter* The maximum number of GPA iterations to perform before superimposition is halted. The final number of iterations could be larger than this, if curves or surface semilandmarks are involved
- *ProcD* A logical value indicating whether or not Procrustes distance should be used as the criterion for optimizing the positions of semilandmarks
- *Proj* A logical value indicating whether or not the aligned Procrustes residuals should be projected into tangent space
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen

The function performs a Generalized Procrustes Analysis (GPA) on two-dimensional or three-dimensional landmark coordinates. The analysis can be performed on fixed landmark points, semilandmarks on curves, semilandmarks on surfaces, or any combination. To include semilandmarks on curves, one must specify a matrix defining which landmarks are to be treated as semilandmarks using the “`curves=`” option (this matrix can be made using `define.sliders`). Likewise, to include semilandmarks on surfaces, one must specify a vector listing which landmarks are to be treated as surface semilandmarks using the “`surfaces=`” option. The `ProcD=TRUE` option will slide the semilandmarks along their tangent directions using the Procrustes distance criterion, while `ProcD=FALSE` will slide the semilandmarks based on minimizing bending energy. The aligned Procrustes residuals can be projected into tangent space using the `Proj=TRUE` option. NOTE: Large datasets may exceed the memory limitations of *R*.

4.1.1 Notes for *geomorph* 3.0

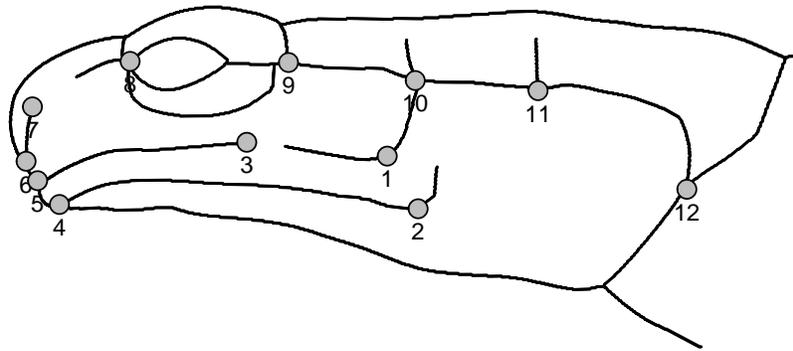
Compared to older versions of *geomorph*, users might notice subtle differences in Procrustes residuals when using semilandmarks (curves or surfaces). This difference is a result of using recursive updates of the consensus configuration with the sliding algorithms (minimized bending energy or Procrustes distances). (Previous versions used a single consensus through the sliding algorithms.) Shape differences using the recursive updates of the consensus configuration should be highly correlated with shape differences using a single consensus during the sliding algorithm, but rotational “flutter” can be expected. This should have no qualitative effect on inferential analyses using Procrustes residuals.

4.1.1.1 Using `plot`, `print/summary` on a `gpagen` object

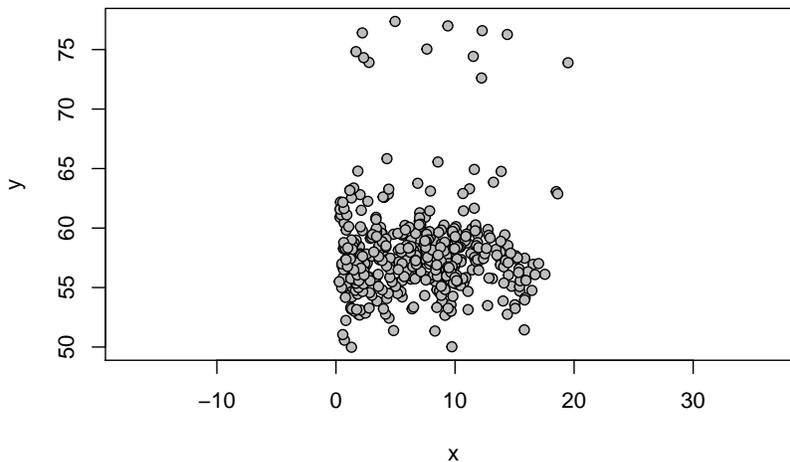
The generic functions, `print/summary`, and `plot` all work with `gpagen`. The generic function, `plot` calls `plotAllSpecimens` and plots the aligned landmarks of all specimens with the mean. `print/summary` provide details of the GPA, including how many fixed and sliding semilandmarks are in the dataset, as well as how many GPA iterations it took to converge. Use `attributes` to see the elements contained in the returned list from `gpagen`.

4.1.2 Procrustes Superimposition of Fixed Landmarks only

As an example, we will use the 2D landmarks of the salamander head data set saved within *geomorph*.



```
data(plethodon) # Load the data
# See that all the specimens are in different coordinate systems
plotAllSpecimens(plethodon$land, mean=FALSE)
```



Since all of the specimens were digitized from different images, when we plot the raw data we see the specimens are all in different coordinate systems and thus are all over the place.

To perform GPA-alignment

```
plethodon.gpa <- gpagen(plethodon$land, print.progress = FALSE)
summary(plethodon.gpa)
```

Call:

```
gpagen(A = plethodon$land, print.progress = FALSE)
```

Generalized Procrustes Analysis
with Partial Procrustes Superimposition

12 fixed landmarks
0 semilandmarks (sliders)
2-dimensional landmarks
2 GPA iterations to converge

Consensus (mean) Configuration

	X	Y
[1,]	0.15233235	-0.025236626
[2,]	0.19328978	-0.095041326
[3,]	-0.03370053	-0.006929886
[4,]	-0.28182427	-0.089370884
[5,]	-0.31072667	-0.057833073
[6,]	-0.32600020	-0.032082163
[7,]	-0.31757271	0.040056683
[8,]	-0.18824427	0.100347120
[9,]	0.02159274	0.098853350
[10,]	0.18946790	0.074940129
[11,]	0.35213100	0.061515224
[12,]	0.54925486	-0.069218549

Function returns a list containing the shape coordinates and centroid sizes:

```
plethodon.gpa$coords # a 3D array of Procrustes coordinates
plethodon.gpa$Csize # a vector of centroid sizes
```

Other important information about the GPA is stored in the `plethodon.gpa` list. Use `attributes` to see, including: *iter* The number of GPA iterations until convergence was found (or GPA halted); *points.VCV* Variance-covariance matrix among landmark coordinates; *points.var** Variances of landmark points; *consensus* The consensus (mean) configuration; akin to using `mshape`; *data* Data frame with an $n \times (pk)$ matrix of Procrustes residuals and centroid size; *Q* Final convergence criterion value; *slide.method* Method used to slide semilandmarks.

Stored in the example dataset `plethodon` are the wireframe links to aid visualizations. The function `plotAllSpecimens` plots landmark coordinates for a set of specimens:

```
plotAllSpecimens(plethodon.gpa$coords, links=plethodon$links)
```

To view the options for this function

4.1.3 Procrustes Superimposition of datasets containing semilandmarks along a curve

The first example contained landmarks that are fixed, representing homologous points on the object. Oftentimes, researchers need to also use semilandmarks, which are points on a geometric feature (curve, edge, surface) defined mathematically in terms of its position on the feature (most commonly equally spaced). Semilandmarks provide information about curvature, and should be used alongside fixed landmarks.

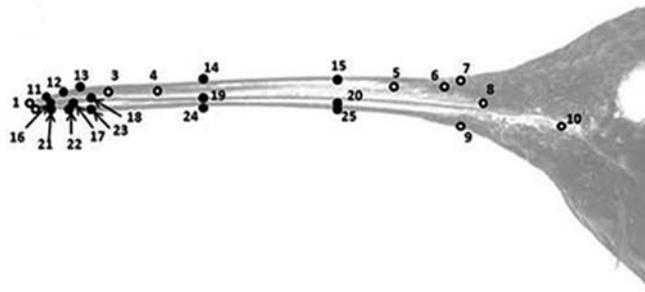


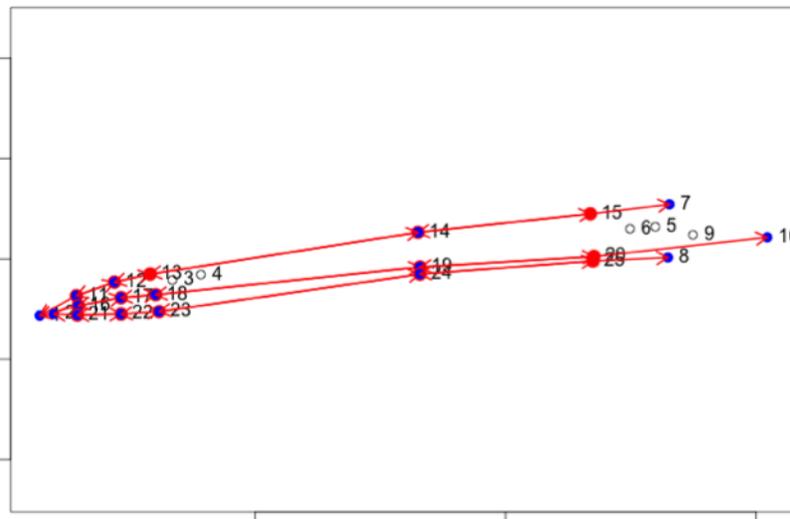
Figure 7: Hummingbird dataset from Berns & Adams 2010

The `hummingbird` dataset, from Berns & Adams 2010, contains fixed landmarks (open circles) and semilandmarks (closed circles) that define the curvature of the beak. When semilandmarks are included in the dataset, a few extra parameters should be specified in the `gpgen` function.

First a curve sliding matrix should be made, to tell the function which landmarks are semilandmarks and how they will slide during the superimposition.

```
data(hummingbirds) # example dataset
hummingbirds$curvepts # an example curve sliding matrix
  before slide after
[1,]      1     11     12
[2,]     11     12     13
[3,]     13     14     15
[4,]      7     15     14
[5,]     12     13     14
[6,]      1     16     17
[7,]     16     17     18
[8,]     17     18     19
[9,]     18     19     20
[10,]    10     20     19
[11,]     2     21     22
[12,]    21     22     23
[13,]    22     23     24
[14,]    23     24     25
[15,]     8     25     24
```

The landmarks listed in the middle column are the semilandmarks.



This image is a visual version of the matrix, made using the interactive function `define.sliders` (see helper functions below).

This curve sliding matrix is used in the `gpagen` option `curves`. There are two options for how the semilandmarks will slide, using Procrustes distance or bending energy:

```
# Using Procrustes Distance for sliding
A <- gpagen(hummingbirds$land,
            curves=hummingbirds$curvepts,
            ProcD=TRUE, print.progress = FALSE)
# Using bending energy for sliding
B <- gpagen(hummingbirds$land,
            curves=hummingbirds$curvepts,
            ProcD=FALSE, print.progress = FALSE)
```

For more information of how these two methods differ, and when to use which, see Gunz and Mitteroecker 2013 (Hystrix, <http://www.italian-journal-of-mammalogy.it/article/view/6292>).

4.1.4 Procrustes Superimposition of datasets containing semilandmarks over a 3D surface

As stated above, semilandmarks can also be points on a geometric feature such as a 3D surface. These data can be collected using the `buildtemplate` and `digitsurface` functions in *geomorph*.

The example data we shall now use is `scallops`. The first specimen is plotted below, showing that there are 5 fixed landmarks (red), 11 semilandmarks along the shell edge (blue) and 30 semilandmarks over the shell surface.

```
data(scallops) # dataset of scallop shells with semilandmarks
attributes(scallops)
scallops$curvslide # the curve sliding matrix
head(scallops$surfslide) # the surfaces sliding matrix
scallop.gpa = gpagen(A=scallops$coorddata,
                    curves=scallops$curvslide,
                    surfaces=scallops$surfslide) # GPA-alignment
scallop.gpa$coords # 3D array of Procrustes coordinates
scallop.gpa$Csize # Vector of centroid sizes
```

With 3D data, the function plots the aligned specimens in an `rgl` window.

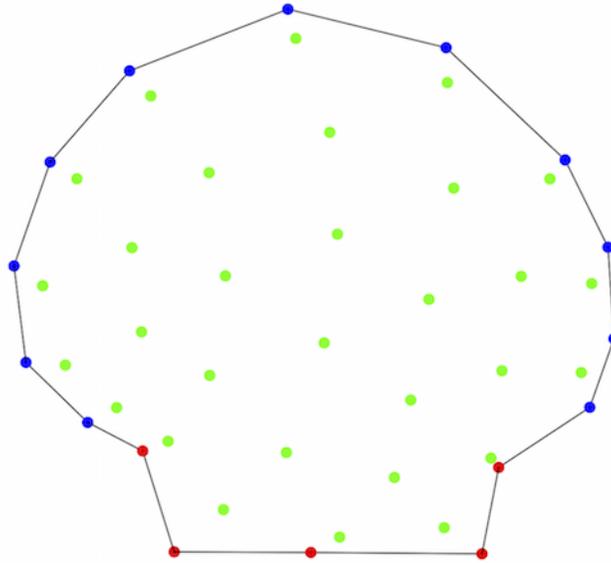


Figure 8: Scallop data

To make one these curve sliding matrices, you can use the interactive function `define.sliders` (see helper functions below).

For all subsequent analyses, the Procrustes coordinates (i.e., `Y$coords`) should be used

4.2 Generalized Procrustes Analysis with Bilateral Symmetry Analysis (`bilat.symmetry`)

If the data has bilateral symmetry, the first step is to perform a superimposition of the raw coordinate data taking into account the symmetry. This function also assesses the statistical differences in the symmetric data.

Function

```
bilat.symmetry(A, ind = NULL, side = NULL, replicate = NULL, object.sym = FALSE,
              land.pairs = NULL, data = NULL, iter = 999, seed = NULL, RRPP = TRUE,
              print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates for a set of specimens [for “object.sym=FALSE, A is of dimension (n x k x 2n)]
- *ind* A vector containing labels for each individual. For matching symmetry, the matched pairs receive the same label (replicates also receive the same label)
- *side* An optional vector (for matching symmetry) designating which object belongs to which ‘side-group’
- *replicate* An optional vector designating which objects belong to which group of replicates
- *object.sym* A logical value specifying whether the analysis should proceed based on object `symmetry =TRUE` or matching `symmetry =FALSE`

- *land.pairs* An optional matrix (for object symmetry) containing numbers for matched pairs of landmarks across the line of symmetry
- *data* A data frame for the function environment, see `geomorph.data.frame`. It is imperative that the variables “ind”, “side”, and “replicate” in the data frame match these names exactly
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If `seed = “random”`, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *RRPP* A logical value indicating whether residual randomization should be used for significance testing
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses.

The function quantifies components of shape variation for a set of specimens as described by their patterns of symmetry and asymmetry. Here, shape variation is decomposed into variation among individuals, variation among sides (directional asymmetry), and variation due to an individual x side interaction (fluctuating symmetry). These components are then statistically evaluated using Procrustes ANOVA and Goodall’s F tests (i.e., an isotropic model of shape variation). Methods for both matching symmetry and object symmetry can be implemented. Matching symmetry is when each object contains mirrored pairs of structures (e.g., right and left hands) while object symmetry is when a single object is symmetric about a midline (e.g., right and left sides of human faces). Analytical and computational details concerning the analysis of symmetry in geometric morphometrics can be found in Mardia et al. (2000) and Klingenberg et al. (2002).

Analyses of symmetry for matched pairs of objects is implemented when `object.sym=FALSE`. Here, a 3D array [p x k x 2n] contains the landmark coordinates for all pairs of structures (2 structures for each of n specimens). Because the two sets of structures are on opposite sides, they represent mirror images, and one set must be reflected prior to the analysis to allow landmark correspondence. It is assumed that the user has done this prior to performing the symmetry analysis. Reflecting a set of specimens may be accomplished by multiplying one coordinate dimension by ‘-1’ for these structures (either the x-, the y-, or the z-dimension). A vector containing information on individuals and sides must also be supplied. Replicates of each specimen may also be included in the dataset, and when specified will be used as measurement error (see Klingenberg and McIntyre 1998).

Analyses of object symmetry is implemented when `object.sym=TRUE`. Here, a 3D array [p x k x n] contains the landmark coordinates for all n specimens. To obtain information about asymmetry, the function generates a second set of objects by reflecting them about one of their coordinate axes. The landmarks across the line of symmetry are then relabeled to obtain landmark correspondence. The user must supply a list of landmark pairs. A vector containing information on individuals must also be supplied. Replicates of each specimen may also be included in the dataset, and when specified will be used as measurement error.

Notes for `geomorph 3.0`

Compared to older versions of *geomorph*, some results can be expected to be slightly different. Starting with *geomorph 3.0*, results use only type I sums of squares (SS) with either full randomization of raw shape values or RRPP (preferred with nested terms) for analysis of variance (ANOVA). Older versions used a combination of parametric and non-parametric results, as well as a combination of type I and type III SS. While analytical conclusions should be consistent (i.e., “significance” of effects is the same), these updates maintain consistency in analytical philosophy. This change will require longer computation time for large datasets, but the trade-off allows users to have more flexibility and eliminates combining disparate analytical philosophies.

Note also that significance of terms in the model are found by comparing F-values for each term to those obtained via permutation. F-ratios and df are not strictly necessary (a ratio of SS would suffice), but they are reported as is standard for anova tables. Additionally, users will notice that the df reported are based on the number of observations rather than a combination of objects * coordinates * dimensions, as is sometimes found in morphometric studies of symmetry. However, this change has no effect on hypothesis testing, as

only SS vary among permutations (df, coordinates, and dimensions are constants).

4.2.1 Example of matching symmetry

```
data(mosquito)
gdf <- geomorph.data.frame(wingshape = mosquito$wingshape,
  ind=mosquito$ind, side=mosquito$side,
  replicate=mosquito$replicate) # make geomorph.data.frame
mosquito.sym <- bilat.symmetry(A = wingshape, ind = ind, side = side,
  replicate = replicate, object.sym = FALSE, RRPP = TRUE,
  iter = 499, data = gdf, print.progress = FALSE) # perform matching symmetry GPA
summary(mosquito.sym)
```

Call:

```
bilat.symmetry(A = wingshape, ind = ind, side = side, replicate = replicate,
  object.sym = FALSE, data = gdf, iter = 499, RRPP = TRUE,
  print.progress = FALSE)
```

Symmetry (data) type: Matching

Type I (Sequential) Sums of Squares and Cross-products
Randomized Residual Permutation Procedure Used
500 Permutations

Shape ANOVA

	Df	SS	MS	Rsqr	F	Z	Pr(>F)
ind	9	0.104888	0.0116542	0.45533	2.6901	1.74089	0.062 .
side	1	0.003221	0.0032209	0.01398	0.7435	-0.58381	0.674
ind:side	9	0.038990	0.0043323	0.16926	1.0407	-1.40680	0.954
ind:side:replicate	20	0.083259	0.0041629	0.36143			

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Centroid Size ANOVA

	Df	SS	MS	Rsqr	F	Z	Pr(>F)
ind	9	4.1497e-09	4.6107e-10	0.18555	0.5965	-0.81888	0.898
side	1	3.4740e-10	3.4738e-10	0.01553	0.4494	-0.41443	0.530
ind:side	9	6.9569e-09	7.7299e-10	0.31108	1.4170	-0.32481	0.566
ind:side:replicate	20	1.0910e-08	5.4549e-10	0.48784			

Function returns the ANOVA table for analysis of symmetry and a graph showing the shape deformations relating to the symmetric and asymmetric components of shape. The function returns also the symmetric component of shape variation (`$symm.shape`) and the asymmetric component of shape variation (`$asymm.shape`), to be used in subsequent analyses, just as the Procrustes coordinates. NOTE: that the `dimnames()` of `$symm.shape` have a prefix `ind`. To remove, simply: `dimnames(mosquito.sym$symm.shape)[[3]] <- sub("ind", "", dimnames(mosquito.sym$symm.shape)[[3]])`

4.2.2 Example of object symmetry

```
data(scallops)
gdf <- geomorph.data.frame(shape = scallops$coorddata, ind=scallops$ind)
scallop.sym <- bilat.symmetry(A = shape, ind = ind, object.sym = TRUE,
                             land.pairs=scallops$land.pairs, data = gdf, RRPP = TRUE,
                             iter = 499, print.progress = FALSE) # perform object symmetry GPA
summary(scallop.sym)
```

Call:

```
bilat.symmetry(A = shape, ind = ind, object.sym = TRUE, land.pairs = scallops$land.pairs,
               data = gdf, iter = 499, RRPP = TRUE, print.progress = FALSE)
```

Symmetry (data) type: Object

Type I (Sequential) Sums of Squares and Cross-products
Randomized Residual Permutation Procedure Used
500 Permutations

Shape ANOVA

	Df	SS	MS	Rsqr	F	Z	Pr(>F)
ind	4	0.063030	0.0157574	0.64135	9.8316	-1.2844	0.978
side	1	0.028835	0.0288354	0.29341	17.9914	1.9004	0.064
ind:side	4	0.006411	0.0016027	0.06523			

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

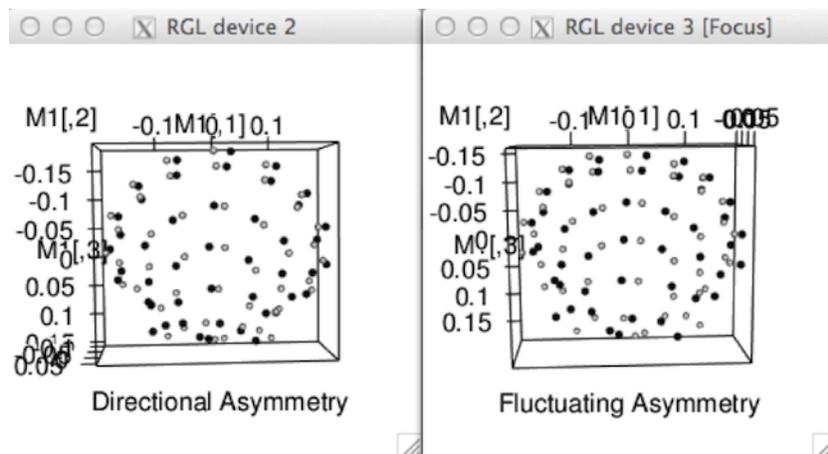


Figure 9: Scallop data

4.3 Helper functions

4.3.1 Define sliding semilandmarks (define.sliders)

An interactive function to define which landmarks will “slide” along curves.

Function

```
define.sliders(spec, nsliders, surfsliders = FALSE)
```

Arguments

- *spec* Name of specimen, as an object matrix containing 2D or 3D landmark coordinates
- *nsliders* Number of landmarks to be semilandmarks that slide along curves
- *surfsliders* Logical (3D only) If ‘spec’ contains landmarks that are “surface sliders”, made by `buildtemplate`, “surfslide.csv” should be in working directory

Function takes a matrix of digitized landmark coordinates and helps user choose which landmarks will be treated as “curve sliders” in Generalized Procrustes analysis `gpagen`. This type of semilandmark “slides” along curves lacking known landmarks (see Bookstein 1997 for algorithm details). Each sliding semilandmark (“sliders”) will slide between two designated points, along a line tangent to the specified curvature.

4.3.1.1 Selection in 2D

Choosing which landmarks will be sliders involves landmark selection using a mouse in the plot window. To define the sliders, for each sliding landmark along the curve in the format ‘before-slider-after’, using the LEFT mouse button (or regular button for Mac users), click on the hollow circle to choose the landmark in the following order: 1) Click to choose the first landmark between which semi-landmark will “slide”, 2) Click to choose sliding landmark, 3) Click to choose the last landmark between which semi-landmark will “slide”. Selected landmarks will be filled in and lines are drawn connecting the three landmarks, and will highlight the sliding semilandmark in red and the flanking landmarks in blue.

```
data(hummingbirds)
define.sliders(hummingbirds$land[, ,1], nsliders=10)
```

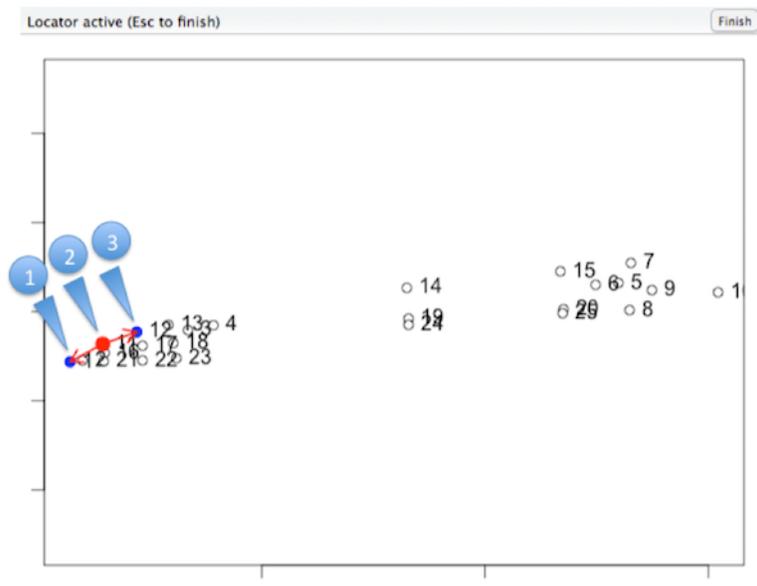


Figure 10: Interactive plot window where the landmarks are plotted (open circles). Left-click in the order shown to define how a landmark should slide

When complete,

4.3.1.2 Selection in 3D

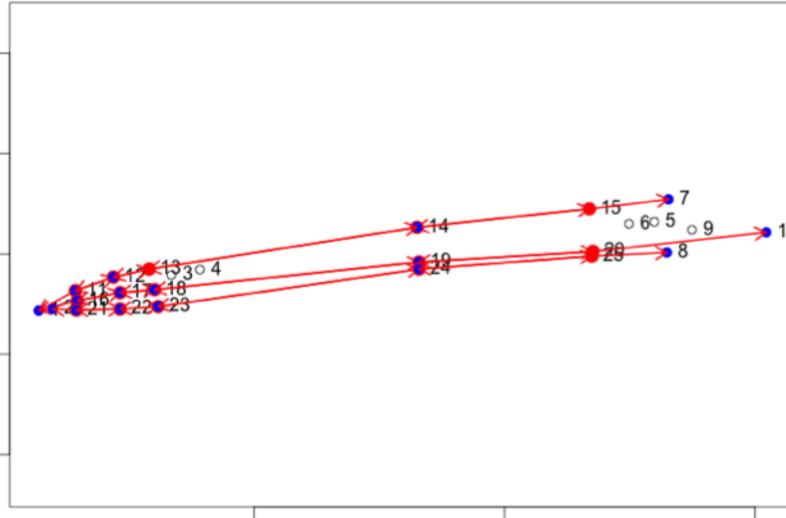


Figure 11: Completed window, where each landmark that will slide (blue) are shown, and red arrows indicate direction of sliding permitted

To define the sliders, for each sliding landmark along the curve in the format ‘before-slider-after’, using RIGHT mouse button to select: 1) Click on landmark to choose the first landmark between which semi-landmark will “slide”, 2) Click box to choose sliding landmark, 3) Click box to choose the last landmark between which semi-landmark will “slide”, Screen will show lines connecting the three landmarks, and will highlight the sliding semilandmark in red.

Here we will run it using the first specimen of the scallop example dataset

```
data(scallops)
define.sliders(scallops$coorddata[, ,1],
              nsliders=11,
              surfsliders = scallops$surfslide) # Interactive function in rgl window
```

The way this function works is to select the landmarks in a “before” “slide” “after” pattern, so defining between which landmarks the semilandmark will slide:

Left: In action. Right: Finished. And in the console will be printed:

```
semi-landmark 16 slides between landmarks 1 and 15
semi-landmark 15 slides between landmarks 16 and 14
semi-landmark 14 slides between landmarks 15 and 13
semi-landmark 13 slides between landmarks 14 and 12
semi-landmark 12 slides between landmarks 13 and 11
semi-landmark 11 slides between landmarks 12 and 10
semi-landmark 10 slides between landmarks 11 and 9
semi-landmark 9 slides between landmarks 10 and 8
semi-landmark 8 slides between landmarks 9 and 7
semi-landmark 7 slides between landmarks 8 and 6
semi-landmark 6 slides between landmarks 7 and 5
```

This procedure is overlapping, so for example a curve defined by a sequence of semilandmarks, the user must select the 2nd point of the first three to be the 1st for the next e.g., 1 2 3 then 2 3 4, etc. Function returns a ‘curves x 3’ matrix containing the landmark address of the curve sliders, indicating the points between which the selected point will “slide”, written to the working directory as “curveslide.csv”.

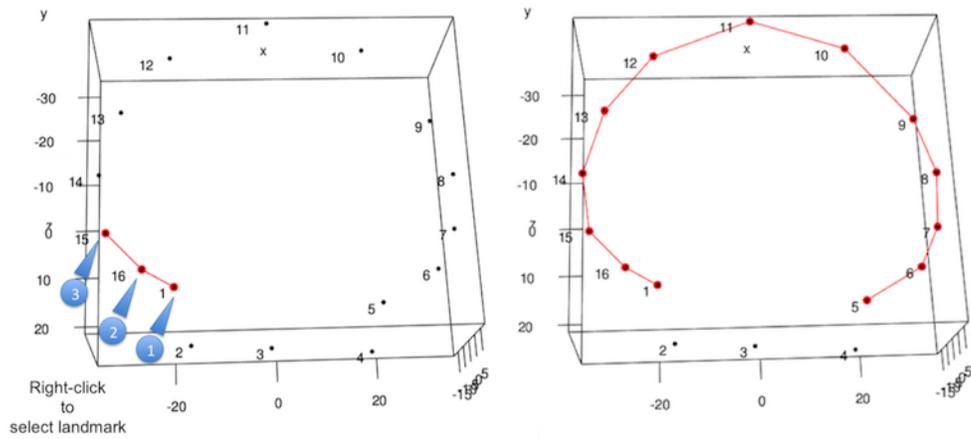


Figure 12: During (left) and completed (right) rgl windows showing `define.sliders` for 3D landmarks

curveslide.csv

```
before slide after
15 16 1
16 15 14
15 14 13
14 13 12
13 12 11
12 11 10
11 10 9
10 9 8
9 8 7
8 7 6
7 6 5
```

which can be read in for use with `gpagen`:

```
curves <- as.matrix(read.csv("curveslide.csv", header=T))
# as.matrix is necessary here to ascertain numeric
```

5 DATA ANALYSIS

After the data have been superimposed with `gpagen` or `bilat.symmetry`, the Procrustes coordinates (e.g., `$coords` in the `gpagen` out list) can be used in many ordination methods and visualization methods.

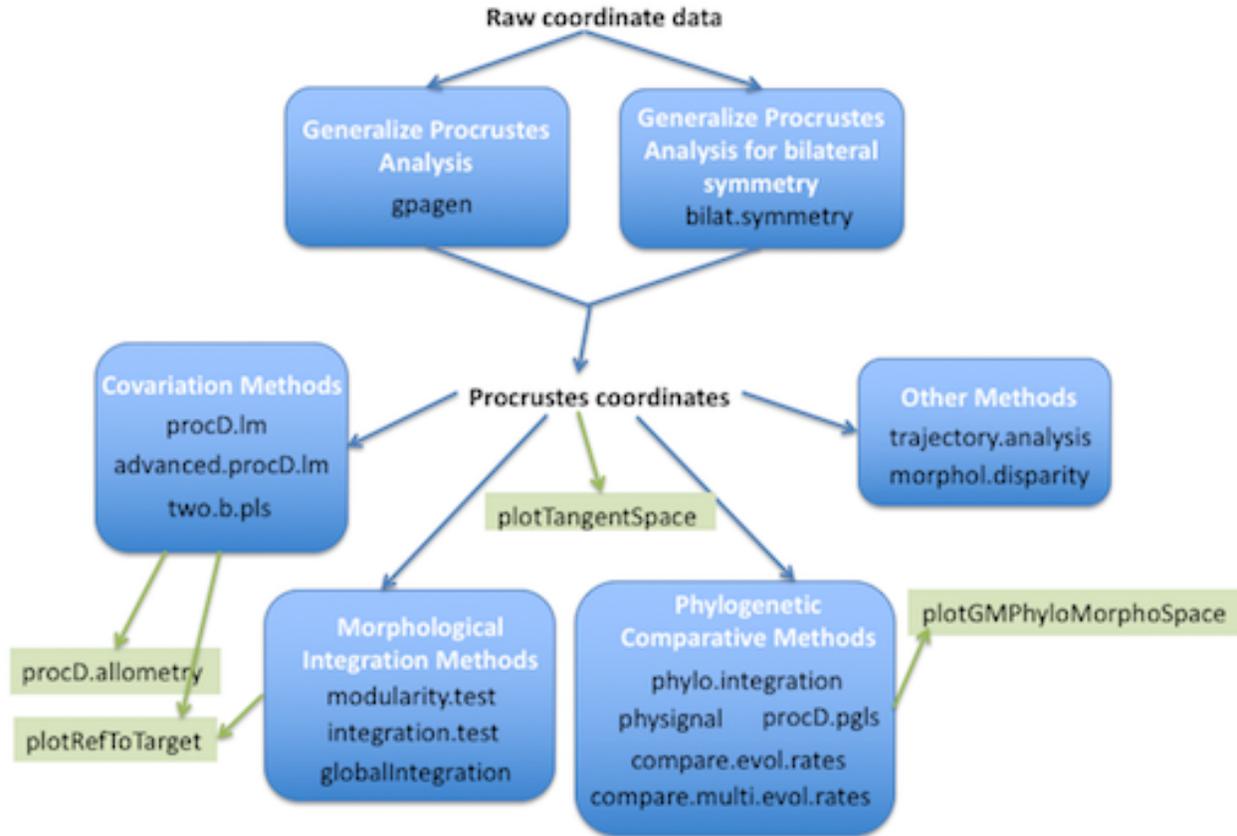


Figure 13: Overview of some of the analysis (blue) and visualization (green) functions in *geomorph*

The next 4 chapters will cover *geomorph*'s analytical functions, organized as follows (hyperlinks): covariation methods, morphological integration methods, phylogenetic comparative methods, other specialist methods. Ordination methods and visualization methods accompanying these analyses are in chapters 9-13.

For all of the following functions it is assumed that the landmarks have previously been aligned using Generalized Procrustes Analysis (GPA).

NOTE: The analytical functions presented here are for the analysis of multivariate data; most commonly this is Procrustes shape variables, but can also be used with non-shape, morphometric data (e.g. a set of interlandmark distances, a.k.a. linear measurements) or any set of continuous variables.

Presented here is an extended version what is provided in the manual pages - it is recommend to read the manual page for the function by typing `?function.name` in the console to see the most up-to-date info about the function. Many functions return information in the form of a list, therefore we recommend users save the results to an object as follows:

```
res <- procD.lm(shape ~ size)
```

From version 3.0, almost all analytical functions work with the generic functions, `print/summary`, and

`plot`. Thus following from the above example, using `plot(res)` will return a plot for that function (see `?plot.function.name` for more options), and `print(res)` or `summary(res)` will print into the console the results of the analytical function. Other objects returned in the `res` list might be useful for further analyses; see the specific function’s manual page for more details.

6 Covariation methods

The following methods allow the user to test for relationship between shape and other factors.

6.1 Procrustes ANOVA/regression for shape data (`procD.lm`)

Function performs Procrustes ANOVA with permutation procedures to assess statistical hypotheses describing patterns of shape variation and covariation for a set of Procrustes-aligned coordinates. To see the manual page for this function, type `?geomorph::procD.lm`.

Function

```
procD.lm(f1, iter = 999, seed = NULL, RRPP = TRUE,
        effect.type = c("F", "SS", "cohen"), int.first = FALSE,
        data = NULL, print.progress = TRUE, ...)
```

Arguments

- *f1* A formula for the linear model (e.g., $y \sim x_1 + x_2$)
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left `NULL` (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If `seed = "random"`, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *RRPP* A logical value indicating whether residual randomization should be used for significance testing (see Introduction VI. Permutation tests)
- *effect.type* One of “F”, “SS”, or “cohen”, to choose from which random distribution to estimate effect size. (The option, “cohen”, is for Cohen’s f-squared values. The default is “F”. Values are log-transformed before z-score calculation to assure normally distributed data.)
- *int.first* A logical value to indicate if interactions of first main effects should precede subsequent main effects}
- *data* A data frame for the function environment, see `geomorph.data.frame`
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses
- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)

6.1.1 Common uses of this function

This function requires an expression of the form $y \sim \text{model}$, interpreted as a specification that the response y is modeled by a linear predictor specified symbolically by `model`.

y is usually the shape coordinates (from `gpagen`). The other terms (x) can be continuous (e.g. centroid size) or discrete (categorical variable, e.g. sex). Some common models and questions:

- Simple Multivariate Regression: $y \sim x$, e.g., Does shape correlate with size (x)?
- Single-factor MANOVA: $y \sim a$, e.g., Does shape differ between sexes (a)?

- Single-factor MANCOVA: $y \sim x * a$ [means $a + x + a:x$, where $:$ denotes interaction], e.g., Does shape differ between sexes (a), while accounting for shape covarying with size (x)?
- Multiple-factor MANOVA: $y \sim a + b$, e.g., Does shape differ between sexes as well as localities (b)?
- Factorial MANOVA: $y \sim a * b$ [means $a + b + a:b$, where $:$ denotes interaction], e.g., Does shape differ between sexes and localities, accounting for the possibility that the two factors may interact (one sex more likely found in a particular locality)
- Nested MANOVA: $y \sim a / b$, a hierarchical model, where b is nested within a. e.g., replicates (b) of treatments (a).

NOTE: This function can also be used with non-shape, multivariate data in y (e.g. a set of iterlandmark distances, a.k.a. linear measurements).

Returned information The function returns a lot of information. In addition to using plot or summary on the returned object, the following may be useful for further analysis: * *residuals* The residuals (observed responses - fitted responses).

6.1.2 Helper function: `nested.update`

If one has performed a nested model with `procD.lm`, then the function `nested.update` is needed to adjust models. Type `?nested.update` for details.

Example MANOVA example for Goodall's F test (multivariate shape vs. factors)

```
data(plethodon) # example dataset
Y.gpa <- gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
gdf <- geomorph.data.frame(shape = Y.gpa$coords,
site = plethodon$site, species = plethodon$species) # make geomorph data frame

# permutation option 1: randomize raw values
procD.lm(shape ~ species * site, data = gdf, iter = 999,
RRPP = FALSE, print.progress = FALSE)
```

Call:

```
procD.lm(f1 = shape ~ species * site, iter = 999, RRPP = FALSE,
data = gdf, print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products

Randomization of Raw Values used

1000 Permutations

	Df	SS	MS	Rsqr	F	Z	Pr(>F)
species	1	0.029258	0.029258	0.14856	14.544	4.4828	0.001 **
site	1	0.064375	0.064375	0.32688	32.000	5.5588	0.001 **
species:site	1	0.030885	0.030885	0.15682	15.352	4.5841	0.001 **
Residuals	36	0.072422	0.002012				
Total	39	0.196940					

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
# permutation option 1: randomize residuals
procD.lm(shape ~ species * site, data = gdf, iter = 999,
RRPP = TRUE, print.progress = FALSE)
```

Call:

```
procD.lm(f1 = shape ~ species * site, iter = 999, RRPP = TRUE,
        data = gdf, print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products
 Randomized Residual Permutation Procedure Used
 1000 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
species	1	0.029258	0.029258	0.14856	14.544	4.4828	0.001 **
site	1	0.064375	0.064375	0.32688	32.000	5.9466	0.001 **
species:site	1	0.030885	0.030885	0.15682	15.352	6.8745	0.001 **
Residuals	36	0.072422	0.002012				
Total	39	0.196940					

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Example Multivariate Regression

```
data(ratland)
rat.gpa<-gpagen(ratland, print.progress = FALSE) #GPA-alignment
gdf <- geomorph.data.frame(rat.gpa)
# geomorph data frame is easy without additional input
# uses elements of the rat.gpa list

procD.lm(coords ~ Csize, data = gdf, iter = 999, RRPP = FALSE,
        print.progress = FALSE) # randomize raw values
```

Call:

```
procD.lm(f1 = coords ~ Csize, iter = 999, RRPP = FALSE, data = gdf,
        print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products
 Randomization of Raw Values used
 1000 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
Csize	1	0.64035	0.64035	0.76016	513.46	6.3446	0.001 **
Residuals	162	0.20203	0.00125				
Total	163	0.84239					

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```
procD.lm(coords ~ Csize, data = gdf, iter = 999, RRPP = TRUE,
        print.progress = FALSE) # randomize raw values
```

Call:

```
procD.lm(f1 = coords ~ Csize, iter = 999, RRPP = TRUE, data = gdf,
        print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products
 Randomized Residual Permutation Procedure Used
 1000 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
--	----	----	----	-----	---	---	--------

```

Csize      1 0.64035 0.64035 0.76016 513.46 6.3446 0.001 **
Residuals 162 0.20203 0.00125
Total      163 0.84239
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```
# Outcomes should be exactly the same
```

```

### Extracting objects and plotting options
rat.anova <- procD.lm(coords ~ Csize, data = gdf, iter = 999, RRPP = TRUE,
  print.progress = FALSE)
summary(rat.anova)

```

```

Call:
procD.lm(f1 = coords ~ Csize, iter = 999, RRPP = TRUE, data = gdf,
  print.progress = FALSE)

```

```

Type I (Sequential) Sums of Squares and Cross-products
Randomized Residual Permutation Procedure Used
1000 Permutations

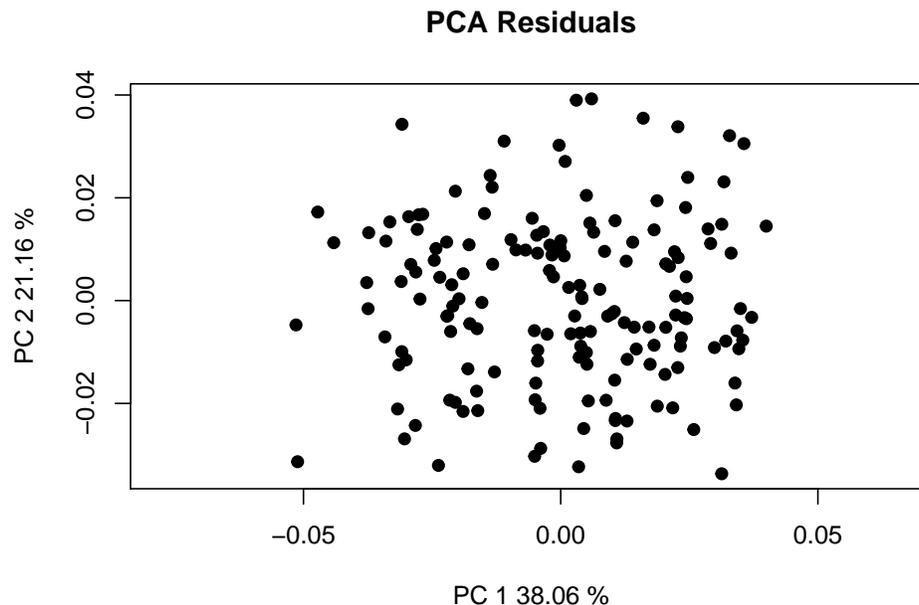
```

```

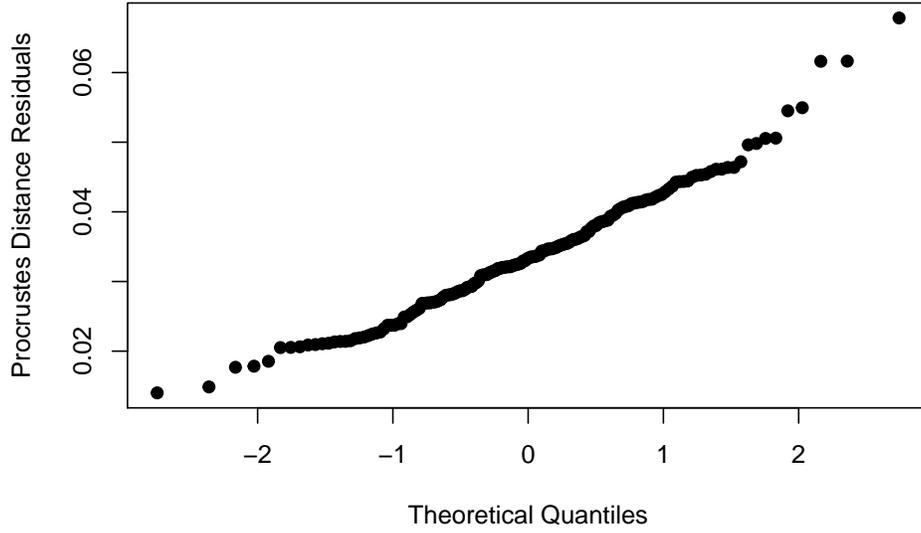
      Df      SS      MS      Rsq      F      Z Pr(>F)
Csize   1 0.64035 0.64035 0.76016 513.46 6.3446 0.001 **
Residuals 162 0.20203 0.00125
Total   163 0.84239
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

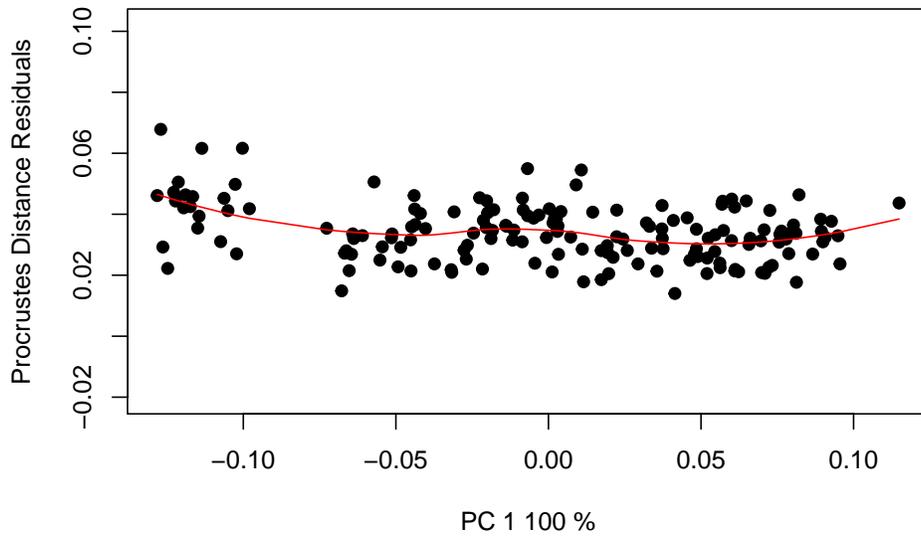
```
# diagnostic plots
plot(rat.anova, type = "diagnostics")
```



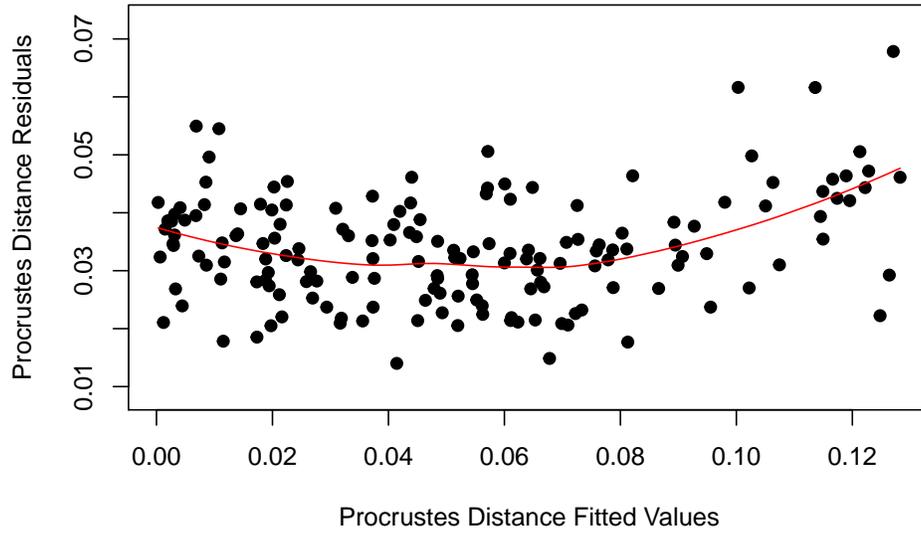
Q-Q plot



Residuals vs. PC 1 fitted

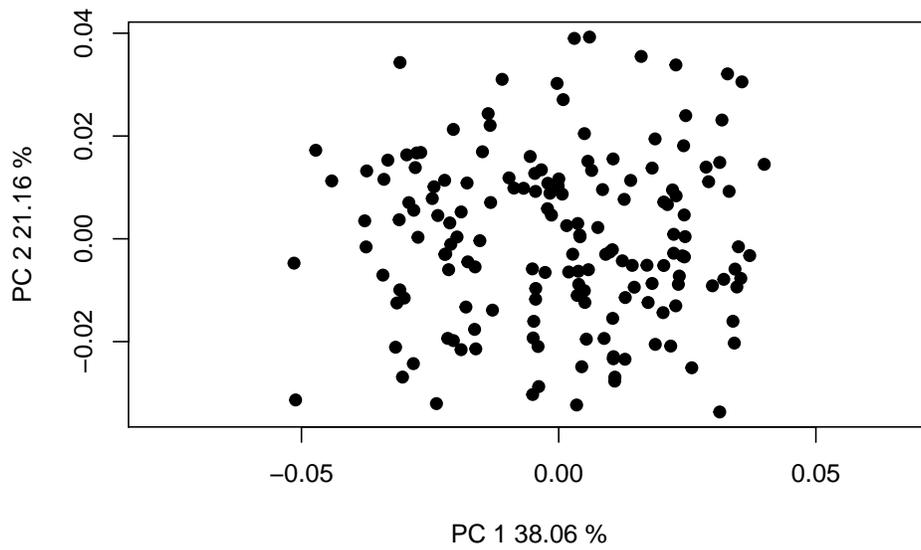


Residuals vs. Fitted

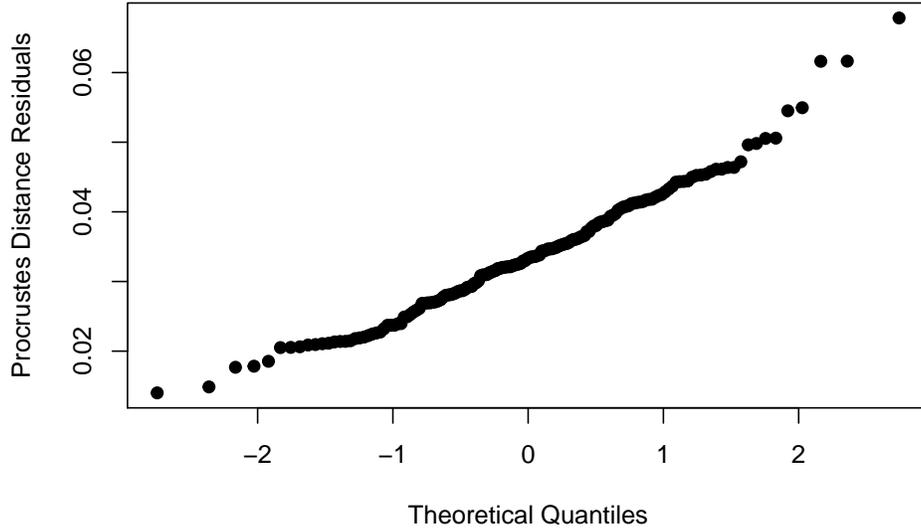


```
# diagnostic plots, including plotOutliers  
plot(rat.anova, type = "diagnostics", outliers = TRUE)
```

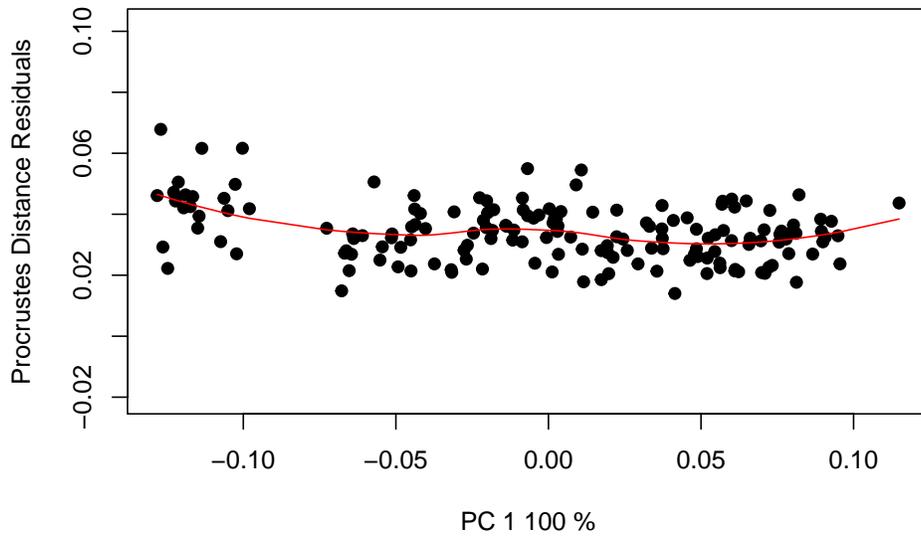
PCA Residuals



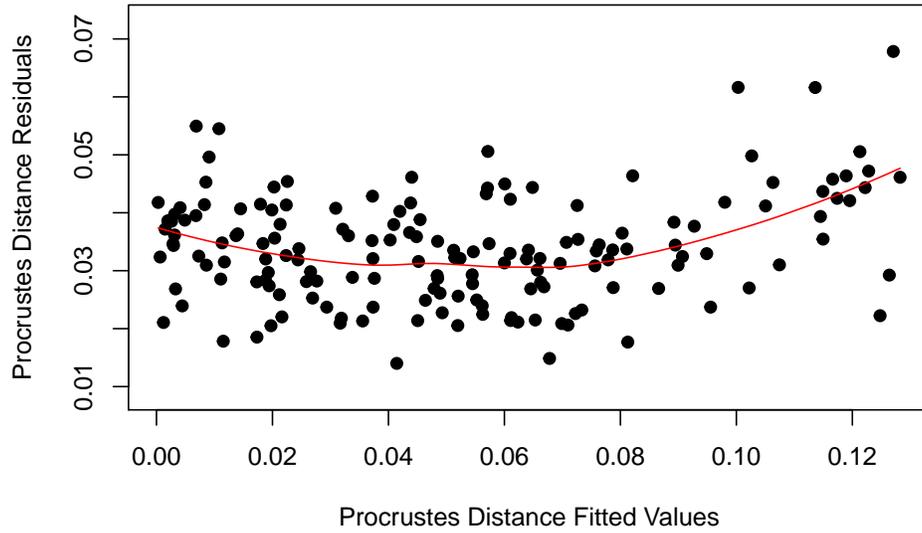
Q-Q plot



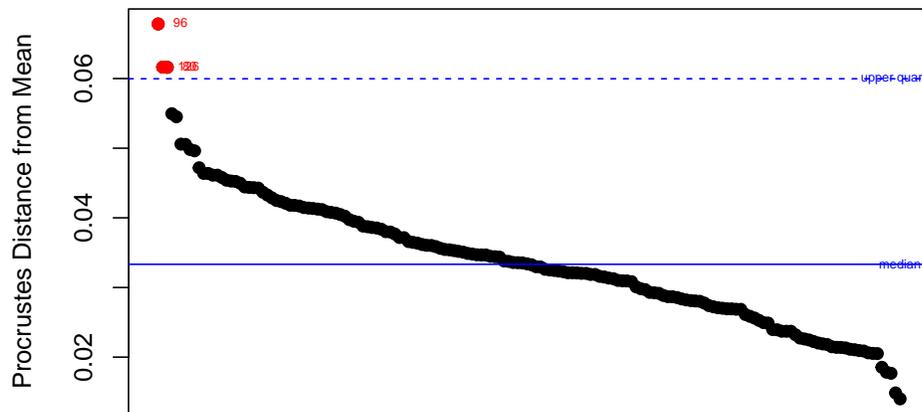
Residuals vs. PC 1 fitted



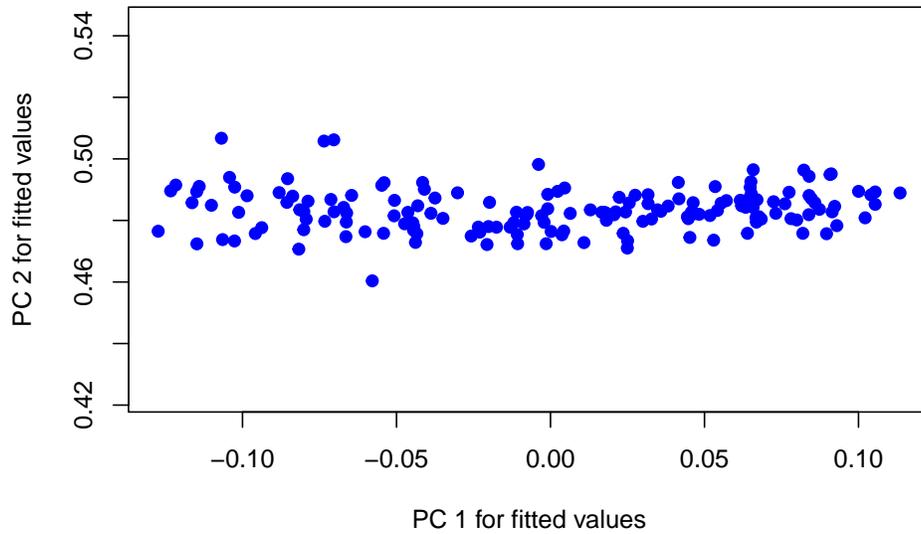
Residuals vs. Fitted



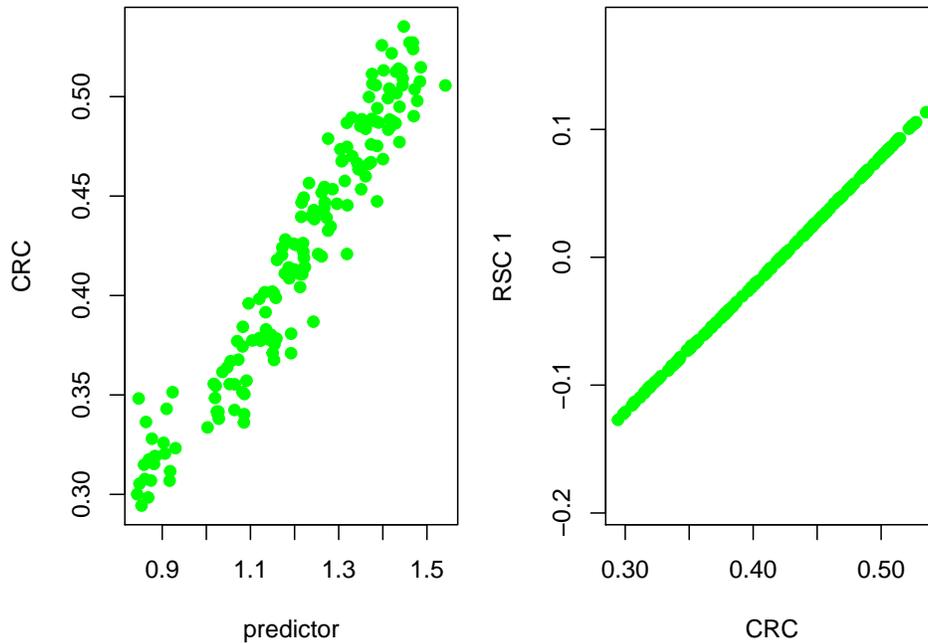
All Specimens



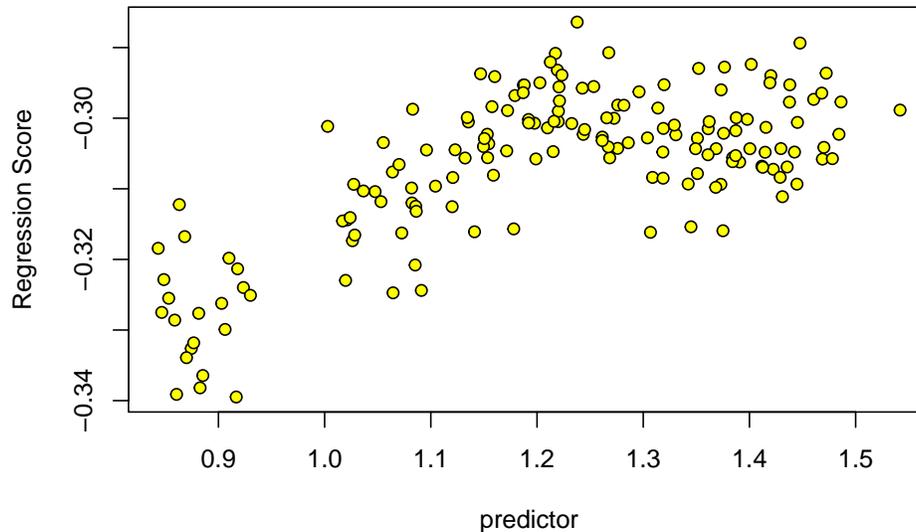
```
# PC plot rotated to major axis of fitted values  
plot(rat.anova, type = "PC", pch = 19, col = "blue")
```



```
# Uses residuals from model to find the common regression component
# for a predictor from the model
plot(rat.anova, type = "regression", predictor = gdf$Csize, reg.type = "CRC",
pch = 19, col = "green")
```



```
# Uses residuals from model to find the projected regression scores
rat.plot <- plot(rat.anova, type = "regression", predictor = gdf$Csize, reg.type = "RegScore",
pch = 21, bg = "yellow")
```



6.2 Advanced Procrustes ANOVA and pairwise tests for shape data, using complex linear models (`advanced.procD.lm`)

The function quantifies the relative amount of shape variation explained by a suite of factors and covariates in a “full” model, after accounting for variation in a “reduced” model. Inputs are formulae for full and reduced models (order is not important, but it is better to list the model with the most terms first or use a `geomorph` data frame), plus indication if means or slopes are to be compared among groups, with appropriate formulae to define how they should be compared. To see the manual page for this function, type `?geomorph::advanced.procD.lm`. (This function replaces the now defunct `pairwise.test` and `pairwise.slope.test`).

Function

```
advanced.procD.lm(f1, f2, groups = NULL, slope = NULL,
  angle.type = c("r", "deg", "rad"), phy = NULL, pc.shape = FALSE,
  iter = 999, seed = NULL, print.progress = TRUE, data = NULL,...)
```

Arguments

- *f1* A formula for the linear model (e.g., $y \sim x_1 + x_2$)
- *f2* A formula for another linear model (e.g., $\sim x_1 + x_2 + x_3 + a*b$). *f1* and *f2* should be nested.
- *groups* A formula for grouping factors (e.g., $\sim a$, or $\sim a*b$)
- *slope* A formula with one covariate (e.g., $\sim x_3$)
- *angle.type* A value specifying whether differences between slopes should be represented by vector correlations (`r`), radians (`rad`) or degrees (`deg`)
- *phy* A phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional)
- *pc.shape* An argument for whether analysis should be performed on the principal component scores of shape. This is a useful option if the data are high-dimensional (many more variables than observations) but will not affect results
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left `NULL` (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If `seed = "random"`, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *data* A data frame for the function environment, see `geomorph.data.frame`
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)

NOTE: This function can also be used with non-shape, multivariate data in `y` (e.g. a set of iterlandmark distances, a.k.a. linear measurements).

For more information on how to use this function see: <https://github.com/geomorphR/geomorph/wiki/advanced.procD.lm-for-pairwise-tests-and-model-comparisons>

Example

Example of a nested model comparison (as with ANOVA with RRPP)

```
data(plethodon) # example dataset
Y.gpa <- gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
gdf <- geomorph.data.frame(Y.gpa, species = plethodon$species,
site = plethodon$site) # make geomorph data frame

ANOVA <- advanced.procD.lm(f1= coords ~ log(Csize) + species,
f2= ~ log(Csize)*species*site, iter=249, data = gdf,
print.progress = FALSE)
# summary(ANOVA, formula = FALSE) # formulas too long to print
```

Example of a test of a factor interaction, plus pairwise comparisons

```
PW.means.test <- advanced.procD.lm(f1= coords ~ site*species, f2= ~ site + species,
groups = ~site*species, iter=249, data = gdf,
print.progress = FALSE)
# summary(PW.means.test, formula = TRUE)
```

Example of a test of a factor interaction, plus pairwise comparisons, accounting for a common allometry

```
PW.ls.means.test <- advanced.procD.lm(f1= coords ~ Csize + site*species,
f2= ~ log(Csize) + site + species,
groups = ~ site*species, slope = ~log(Csize), iter = 249, data = gdf,
print.progress = FALSE)
# summary(PW.ls.means.test, formula = TRUE)
```

Example of a test of homogeneity of slopes, plus pairwise slopes comparisons

```
gdf$group <- factor(paste(gdf$species, gdf$site, sep="."))
HOS <- advanced.procD.lm(f1= coords ~ log(Csize) + group,
f2= ~ log(Csize) * group, groups = ~ group,
slope = ~ log(Csize), angle.type = "deg", iter = 249, data = gdf,
print.progress = FALSE)
# summary(HOS, formula = FALSE) # formulas too long to print
```

Example of partial pairwise comparisons, given greater model complexity. Plus, working with class `advanced.procD.lm` objects.

```
aov.pleth <- advanced.procD.lm(f1= coords ~ log(Csize)*site*species,
f2= ~ log(Csize) + site*species, groups = ~ species,
slope = ~ log(Csize), angle.type = "deg", iter = 249, data = gdf,
print.progress = FALSE)
summary(aov.pleth, formula = FALSE) # formulas too long to print
```

Call:

```
advanced.procD.lm(f1 = coords ~ log(Csize) * site * species,
f2 = ~log(Csize) + site * species, groups = ~species, slope = ~log(Csize),
```

```
angle.type = "deg", iter = 249, print.progress = FALSE, data = gdf)
```

Randomized Residual Permutation Procedure Used

250 Permutations

ANOVA Table

	Df	SSE	SS	R2	F	Z	Pr(>F)
Reduced Model	35	0.068671					
Full Model	32	0.061718	0.0069531	0.035306	1.2017	5.036	0.004 **

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Slopes

	[,1]	[,2]	[,3]	[,4]	[,5]
Jord	-0.081484968	-0.03761677	-0.0648701016	-0.06178754	0.003922133
Teyah	0.007244613	-0.01070130	0.0006410929	-0.01848134	0.017360168
	[,6]	[,7]	[,8]	[,9]	[,10]
Jord	0.003676141	0.003351926	-0.01677718	0.001597638	-0.003321746
Teyah	0.021465694	-0.008738975	0.01889856	-0.009999672	0.014551254
	[,11]	[,12]	[,13]	[,14]	[,15]
Jord	0.007569908	0.018788107	0.02469451	0.008206906	0.03193667
Teyah	-0.015958614	-0.006784027	-0.02050871	-0.016958249	0.03575095
	[,16]	[,17]	[,18]	[,19]	[,20]
Jord	0.01536670	-0.001856277	0.0143840140	-0.02302729	-0.005194829
Teyah	-0.01869515	-0.011425740	-0.0003617453	0.01608166	-0.008239490
	[,21]	[,22]	[,23]	[,24]	
Jord	0.06433678	-0.002487285	0.03382909	0.06676348	
Teyah	0.01617331	0.002470573	-0.02662008	0.02283522	

Effect sizes (Z)

	Jord	Teyah
Jord	0.0000000	0.2415322
Teyah	0.2415322	0.0000000

P-values

	Jord	Teyah
Jord	1.000	0.412
Teyah	0.412	1.000

Effects sizes (Z)

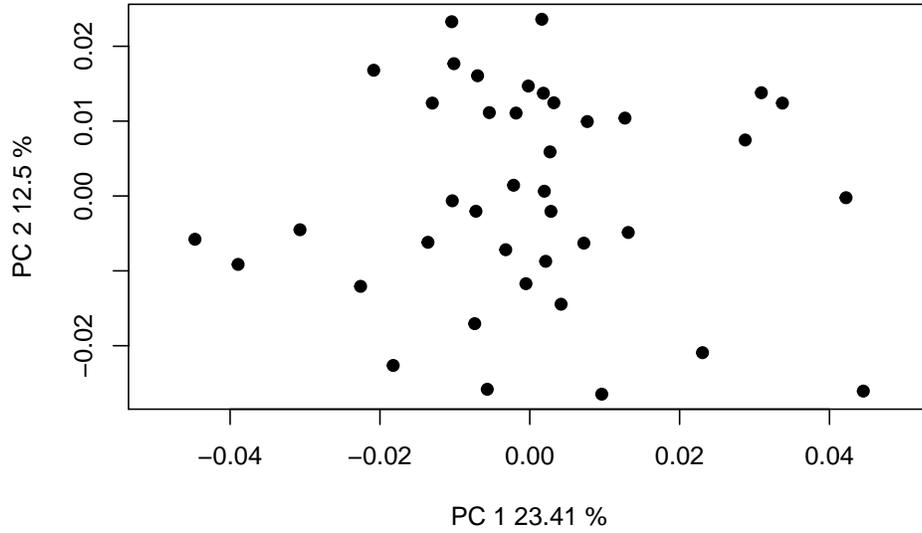
	Jord	Teyah
Jord	0.0000000	2.126904
Teyah	2.126904	0.0000000

P-values

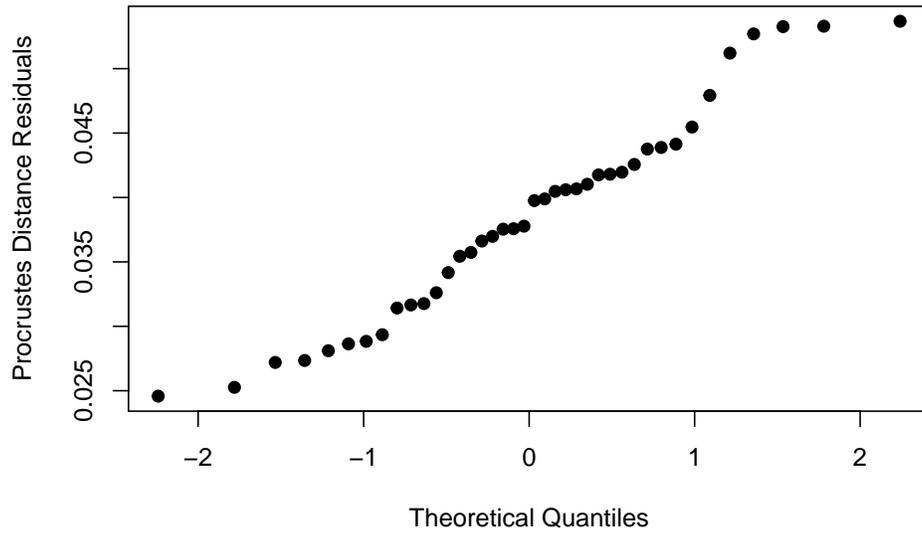
	Jord	Teyah
Jord	1.00	0.02
Teyah	0.02	1.00

```
# Diagnostic plots  
plot(aov.pleth)
```

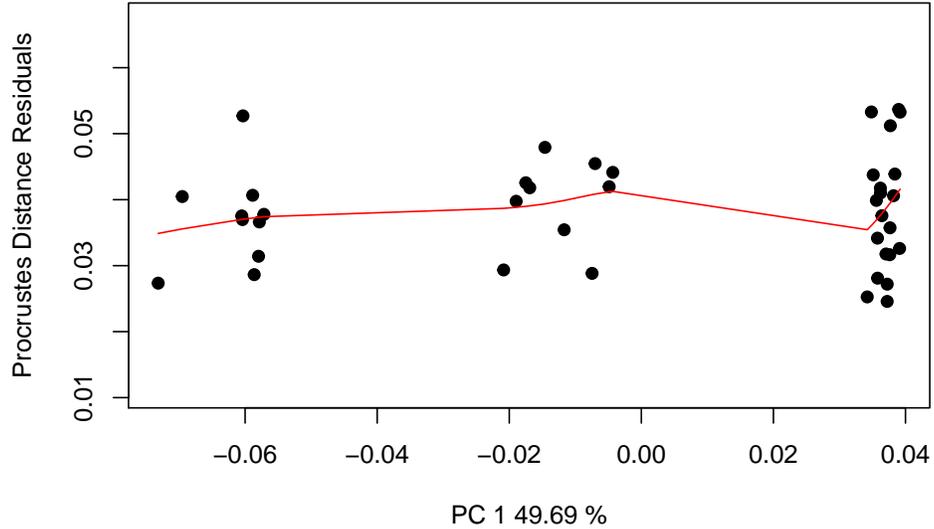
PCA Residuals



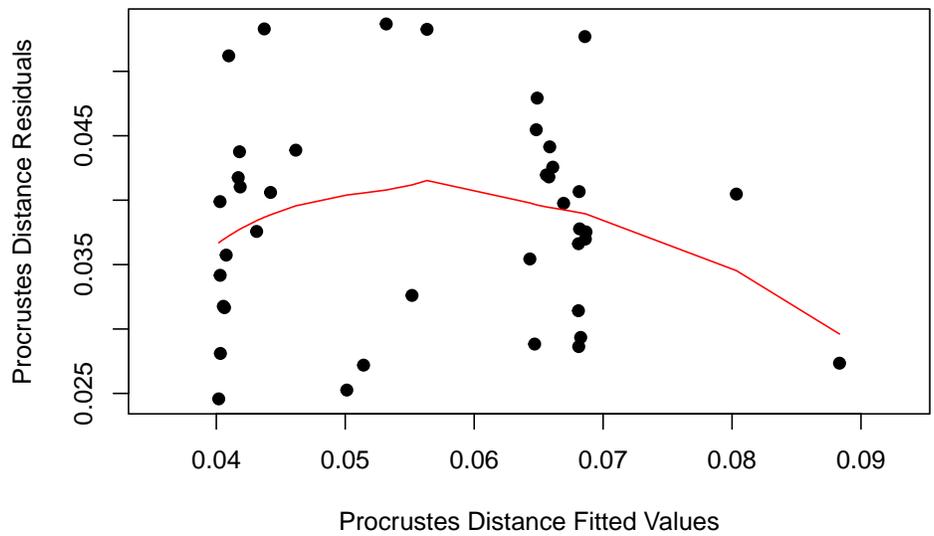
Q-Q plot



Residuals vs. PC 1 fitted



Residuals vs. Fitted



```
# Extracting objects from results
aov.pleth$slopes # extract the slope vectors
```

	[,1]	[,2]	[,3]	[,4]	[,5]
Jord	-0.081484968	-0.03761677	-0.0648701016	-0.06178754	0.003922133
Teyah	0.007244613	-0.01070130	0.0006410929	-0.01848134	0.017360168
	[,6]	[,7]	[,8]	[,9]	[,10]
Jord	0.003676141	0.003351926	-0.01677718	0.001597638	-0.003321746
Teyah	0.021465694	-0.008738975	0.01889856	-0.009999672	0.014551254
	[,11]	[,12]	[,13]	[,14]	[,15]
Jord	0.007569908	0.018788107	0.02469451	0.008206906	0.03193667
Teyah	-0.015958614	-0.006784027	-0.02050871	-0.016958249	0.03575095
	[,16]	[,17]	[,18]	[,19]	[,20]
Jord	0.01536670	-0.001856277	0.0143840140	-0.02302729	-0.005194829
Teyah	-0.01869515	-0.011425740	-0.0003617453	0.01608166	-0.008239490

	[,21]	[,22]	[,23]	[,24]
Jord	0.06433678	-0.002487285	0.03382909	0.06676348
Teyah	0.01617331	0.002470573	-0.02662008	0.02283522

6.3 Procrustes ANOVA/regression, specifically for shape-size covariation (allometry) (`procD.allometry`)

Function performs Procrustes ANOVA with permutation procedures to assess statistical hypotheses describing patterns of shape covariation with size for a set of Procrustes-aligned coordinates. Other factors or covariates can also be included in the analysis. This function also provides results for plotting allometric curves.

Function

```
procD.allometry(f1, f2 = NULL, logsz = TRUE, iter = 999,
  seed = NULL, alpha = 0.05, RRPP = TRUE, effect.type = c("F", "SS", "cohen"),
  print.progress = TRUE, data = NULL, ...)
```

Arguments

- *f1* A formula for the relationship of shape and size; e.g., $Y \sim X$
- *f2* An optional right-hand formula for the inclusion of groups; e.g., $\sim \text{groups}$
- *f3* A optional right-hand formula for the inclusion of additional variables; e.g., $\sim a + b + c + \dots$
- *logsz* A logical argument to indicate if the variable for size should be log-transformed
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = "random", a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *alpha* The significance level for the homogeneity of slopes test
- *RRPP* A logical value indicating whether residual randomization should be used for significance testing (see Introduction VI. Permutation tests)
- *effect.type* One of "F", "SS", or "cohen", to choose from which random distribution to estimate effect size. (The default is "F")
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses
- *data* A data frame for the function environment, see `geomorph.data.frame`
- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)

Details

The function quantifies the relative amount of shape variation attributable to covariation with organism size (allometry) plus (potentially) another grouping factor in a linear model, so as to provide initial visualizations of patterns of shape allometry. Data input is specified by formulae (e.g., $Y \sim X$), where 'Y' specifies the response variables (shape data), and 'X' contains A SINGLE independent continuous variable representing size. The response matrix 'Y' can be either in the form of a two-dimensional data matrix of dimension (n x [p x k]), or a 3D array (p x n x k). It is assumed that -if the data are based on landmark coordinates - the landmarks have previously been aligned using Generalized Procrustes Analysis (GPA) [e.g., with `gpagen`]. Additionally, one has the option of providing a second formula where groups are specified in the form of $\sim \text{group}$. If groups are provided a "homogeneity of slopes" test will be performed.

It is assumed that the order of the specimens in the shape matrix matches the order of values in the independent variables. Linear model fits (using the `lm` function) can also be input in place of formulae. Arguments for `lm` can also be passed on via this function. For further information about ANOVA in `geomorph`, resampling procedures used, and output, see `procD.lm` or `advanced.procD.lm`. If greater flexibility is required for variable order, `advanced.procD.lm` should be used.

It is strongly recommended that `geomorph.data.frame` is used to create and input a data frame. This will

reduce problems caused by conflicts between the global and function environments. In the absence of a specified data frame, `procD.allometry` will attempt to coerce input data into a data frame, but success is not guaranteed.

The generic functions, `print`, `summary`, and `plot` all work with `procD.allometry`. The generic function, `plot`, produces plots of allometric curves, using one of three methods input (see below). If diagnostic plots on model residuals are desired, `procD.lm` should be used with the resulting model formula. This, along with the data frame resulting from analysis with `procD.allometry` can be used directly in `procD.lm`, which might be useful for extracting ANOVA components (as `procD.allometry` is far more basic than `procD.lm`, in terms of output).

A note on allometric models

This function is intended to be used for the graphical visualization of simple allometric patterns. The method is appropriate for models such as $\text{shape} \sim \log(\text{size})$ and $\text{shape} \sim \log(\text{size}) + \text{groups}$. Three plotting options, the common allometric coefficient (CAC), regression scores (RegScore), and predicted lines (PredLine) are implemented as originally described in the literature. NOTE however that for more complex models with additional parameters, one may instead wish to use the plotting capabilities that accompany `procD.lm` (see below for more details).

Notes for experienced or advanced users

Experienced or advanced users will probably prefer using `procD.lm` with a combination of `plot.procD.lm`, `shape.predictor`, and `plotRefToTarget` for publication-quality analyses and graphics. As stated above, use of `procD.allometry` is for visualizing simple allometric models that do not contain additional covariates. Thus, `procD.allometry` may be thought of as a wrapper function for `procD.lm`, but only for a restricted set of models and using a philosophy for model selection based on the outcome of a homogeneity of slopes test. This is not necessary if one wishes to define a model, irrespective of this outcome, or if more complex models are of interest. In these circumstances `procD.lm` offers much greater flexibility, and provides more statistically general approaches to visualizing patterns. Thus, `procD.allometry` might be thought of as an exploratory tool, if one is unsure how to model allometry for multiple groups. One should not necessarily accept the `procD.allometry` result as “truth” and other models can be explored with `procD.lm`. Examples for more flexible approaches to modeling allometry using `procD.lm` are provided below.

Example

Simple allometry

```
data(plethodon) # example dataset
Y.gpa <- gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
# make geomorph data frame
gdf <- geomorph.data.frame(Y.gpa, site = plethodon$site,
species = plethodon$species)

plethAllometry <- procD.allometry(coords~Csize, f2 = NULL, f3=NULL,
logsz = TRUE, data=gdf, iter=249, print.progress = FALSE)
```

Allometry Model

```
summary(plethAllometry) # results identical to procD.lm
```

Call:

```
procD.allometry(f1 = coords ~ Csize, f2 = NULL, logsz = TRUE,
  iter = 249, print.progress = FALSE, data = gdf, f3 = NULL)
```

Type I (Sequential) Sums of Squares and Cross-products
 Randomized Residual Permutation Procedure Used
 250 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
log(size)	1	0.008894	0.0088940	0.045161	1.7973	1.3743	0.08 .
Residuals	38	0.188046	0.0049486				
Total	39	0.196940					

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Obtaining size-adjusted residuals (and allometry-free shapes)

```
plethAnova <- procD.lm(plethAllometry$formula,
  data = plethAllometry$data, iter = 499,
  RRPP=TRUE, print.progress = FALSE)
summary(plethAnova) # same ANOVA Table
```

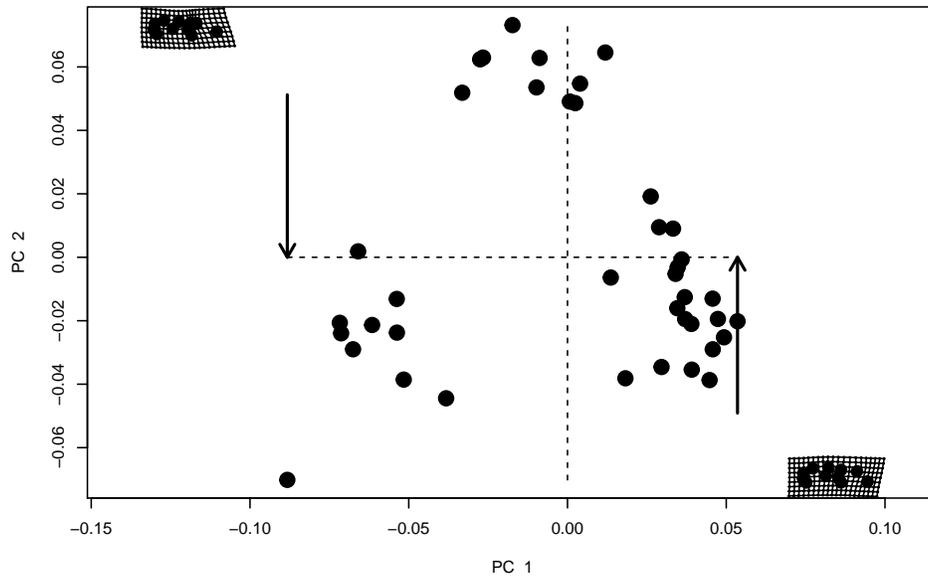
Call:

```
procD.lm(f1 = plethAllometry$formula, iter = 499, RRPP = TRUE,
  data = plethAllometry$data, print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products
 Randomized Residual Permutation Procedure Used
 500 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
log(size)	1	0.008894	0.0088940	0.045161	1.7973	1.2653	0.112
Residuals	38	0.188046	0.0049486				
Total	39	0.196940					

```
# save size-adjusted residuals and make as 3D array
shape.resid <- arrayspecs(plethAnova$residuals,
  p=dim(Y.gpa$coords)[1], k=dim(Y.gpa$coords)[2]) # size-adjusted residuals
# make allometry-free shapes
adj.shape <- shape.resid + array(Y.gpa$consensus, dim(shape.resid))
plotTangentSpace(adj.shape) # PCA of allometry-free shape
```



PC Summary

Importance of first k=20 (out of 24) components:

	PC1	PC2	PC3	PC4	PC5	PC6
Standard deviation	0.04303	0.03781	0.01986	0.01492	0.01314	0.01291
Proportion of Variance	0.38396	0.29650	0.08181	0.04618	0.03579	0.03458
Cumulative Proportion	0.38396	0.68046	0.76227	0.80845	0.84424	0.87882
	PC7	PC8	PC9	PC10	PC11	PC12
Standard deviation	0.01217	0.009205	0.009017	0.00819	0.00724	0.00648
Proportion of Variance	0.03071	0.017570	0.016860	0.01391	0.01087	0.00871
Cumulative Proportion	0.90953	0.927100	0.943960	0.95787	0.96874	0.97745
	PC13	PC14	PC15	PC16	PC17	
Standard deviation	0.005112	0.004286	0.004155	0.003962	0.003687	
Proportion of Variance	0.005420	0.003810	0.003580	0.003260	0.002820	
Cumulative Proportion	0.982870	0.986680	0.990260	0.993520	0.996340	
	PC18	PC19	PC20			
Standard deviation	0.003026	0.002242	0.001862			
Proportion of Variance	0.001900	0.001040	0.000720			
Cumulative Proportion	0.998240	0.999280	1.000000			

6.4 Two-block partial least squares analysis for shape data (two.b.pls)

Function performs two-block partial least squares analysis to assess the degree of association between to blocks of Procrustes-aligned coordinates (or other variables) (see Rohlf and Corti 2000). To see the manual page for this function, type `?geomorph::two.b.pls`.

Function

```
two.b.pls(A1, A2, iter = 999, seed = NULL, print.progress = TRUE)
```

Arguments

- *A1* A 3D array (p x k x n) containing GPA-aligned coordinates, or a matrix (n x variables), for the first block

- *A2* A 3D array (p x k x n) containing GPA-aligned coordinates, or a matrix (n x variables), for the second block
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = “random”, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

NOTE: This function can also be used with non-shape, multivariate data in A1 or A2 (e.g. a set of iterlandmark distances, a.k.a. linear measurements, or ecological variables).

Example

2B-PLS between plethodon head shape and food use data

```
data(plethShapeFood) # example dataset
Y.gpa<-gpagen(plethShapeFood$land, print.progress = FALSE) # GPA-alignment
PLS <-two.b.pls(Y.gpa$coords, plethShapeFood$food,
               iter=999, print.progress = FALSE)
summary(PLS) # Test summary
```

Call:

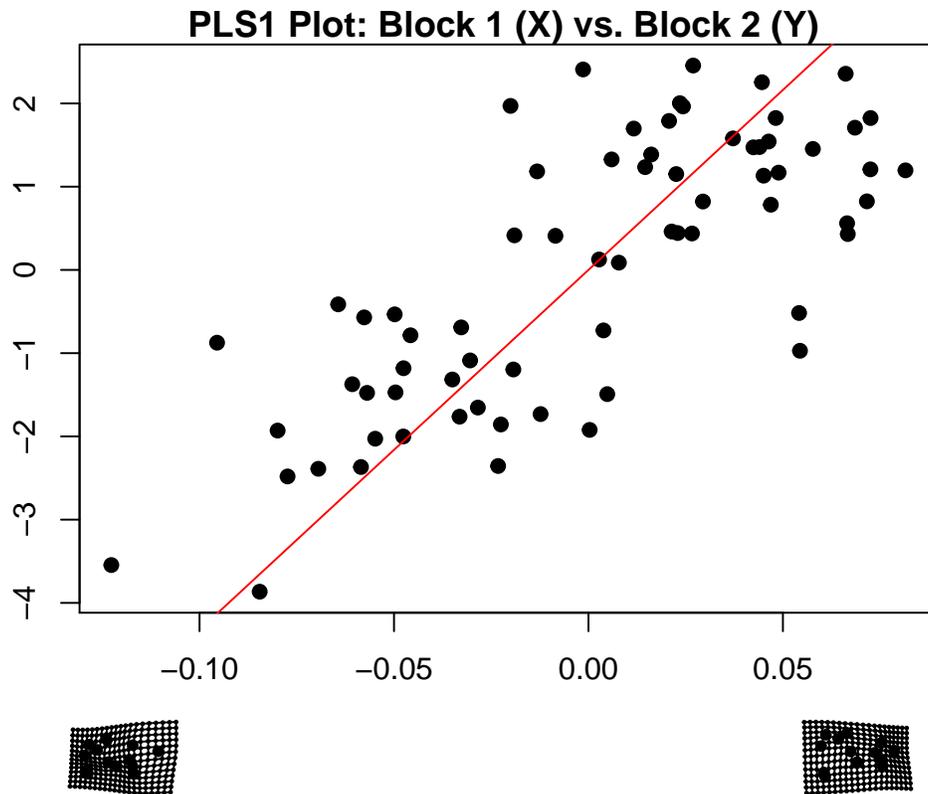
```
two.b.pls(A1 = Y.gpa$coords, A2 = plethShapeFood$food, iter = 999,
          print.progress = FALSE)
```

r-PLS: 0.759

P-value: 0.001

Based on 1000 random permutations

```
plot(PLS) # PLS plot
```



6.5 Helper function: `compare.pls`

The function statistically compares the effect sizes of two or more PLS analyses. Typically, this function might be used to compare levels of integration between two or more samples, each measuring morphological integration between different modules. In such cases, the PLS correlation coefficient, r , is not a good measure of integration effect, as its expected value is dependent on both the number of specimens and number of variables (Adams and Collyer 2016). This analysis calculates effect sizes as standard deviates, z , and performs two-sample z -tests, using the pooled standard error from the sampling distributions of the PLS analyses.

Function

```
compare.pls(...)
```

Arguments

- ... saved analyses of class, "pls"

To use this function, simply perform `two.b.pls`, `integration.test`, or `phylo.integration` on as many samples as desired. Any number of objects of class "pls" can be input.

Example

Example of comparative morphological integration between pupfish head and body shapes

```
data(pupfish) # GPA previously performed
# define the groups
group <- factor(paste(pupfish$Pop, pupfish$Sex, sep = "."))
levels(group)
```

```
[1] "Marsh.F"      "Marsh.M"      "Sinkhole.F"   "Sinkhole.M"
```

```

# subset shape data by body part
tail.LM <- c(1:3, 5:9, 18:38)
head.LM <- (1:56)[-tail.LM]
tail.coords <- pupfish$coords[tail.LM, , ]
head.coords <- pupfish$coords[head.LM, , ]

# Subset 3D array by group, returning a list of 3D arrays
tail.coords.gp <- coords.subset(tail.coords, group)
head.coords.gp <- coords.subset(head.coords, group)

integ.tests <- Map(function(x, y) integration.test(x, y, iter = 499,
  print.progress = FALSE),
  head.coords.gp, tail.coords.gp)
# the map function performs the integration test on each 3D array
# in the lists provided

# View the results
integ.tests$Marsh.F

```

Call:
integration.test(A = x, A2 = y, iter = 499, print.progress = FALSE)

r-PLS: 0.92

P-value: 0

Based on 500 random permutations

```
integ.tests$Marsh.M
```

Call:
integration.test(A = x, A2 = y, iter = 499, print.progress = FALSE)

r-PLS: 0.79

P-value: 0.06

Based on 500 random permutations

```
integ.tests$Sinkhole.F
```

Call:
integration.test(A = x, A2 = y, iter = 499, print.progress = FALSE)

r-PLS: 0.76

P-value: 0.24

Based on 500 random permutations

```
integ.tests$Sinkhole.M
```

Call:

```
integration.test(A = x, A2 = y, iter = 499, print.progress = FALSE)
```

r-PLS: 0.91

P-value: 0

Based on 500 random permutations

```
# Perform test
group.Z <- compare.pls(integ.tests)
summary(group.Z)
```

Effect sizes

Marsh.F	Marsh.M	Sinkhole.F	Sinkhole.M
3.2493074	1.5603308	0.7232586	3.4197781

Effect sizes for pairwise differences in PLS effect size

	Marsh.F	Marsh.M	Sinkhole.F	Sinkhole.M
Marsh.F	0.0000000	1.1836836	1.7628612	0.5225316
Marsh.M	1.1836836	0.0000000	0.5829786	1.6009240
Sinkhole.F	1.7628612	0.5829786	0.0000000	2.1294210
Sinkhole.M	0.5225316	1.6009240	2.1294210	0.0000000

P-values

	Marsh.F	Marsh.M	Sinkhole.F	Sinkhole.M
Marsh.F	1.0000000	0.11826916	0.03896195	0.30065011
Marsh.M	0.11826916	1.0000000	0.27995385	0.05469688
Sinkhole.F	0.03896195	0.27995385	1.0000000	0.01660972
Sinkhole.M	0.30065011	0.05469688	0.01660972	1.0000000

```
# Result: Sexual dimorphism in morphological integration in one population
# but not the other
```

```
# can also list different PLS analyses, separately
compare.pls(MF = integ.tests$Marsh.F, MM = integ.tests$Marsh.M)
```

Effect sizes

MF	MM
3.249307	1.560331

Effect sizes for pairwise differences in PLS effect size

	MF	MM
MF	0.000000	1.183684
MM	1.183684	0.000000

P-values

	MF	MM
MF	1.0000000	0.1182692
MM	0.1182692	1.0000000

7 Morphological Integration methods

Morphological integration is the tendency of morphological traits to vary in a coordinated manner, and usually studied through testing the strength of covariation between two or more sets of traits. The related concept, modularity, is when an organism's structure is compartmentalized, and can be analyzed in a variety of ways. *geomorph* presents a few of the recent methods in studying morphological integration and modularity in high-dimensional data. Suggested reading on the subject: Klingenberg 2008, Klingenberg 2014, Mitteroecker & Bookstein 2007 and references therein.

7.1 Quantify morphological integration between two modules (`integration.test`)

Function quantifies the degree of morphological integration between modules of Procrustes-aligned coordinates, using a two-block partial least squares analysis (PLS). To see the manual page for this function, type `?geomorph::integration.test`. (Formerly known as `morphol.integr`).

Function

```
integration.test(A, A2 = NULL, partition.gp = NULL, iter = 999, seed = NULL, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)
- *A2* Optional 3D array containing GPA-aligned coordinates, or 2D matrix, for the second partition
- *partition.gp* A vector describing which landmarks (or variables) belong in which partition (e.g. A,A,A,B,B,B,C,C,C) (required when only *A* is provided)
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = "random", a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Example

Test for morphological integration between plethodon skull and mandible shape

```
data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
# landmarks on the skull and mandible assigned to partitions B and A respectively
land.gps<-c("A", "A", "A", "A", "A", "B", "B", "B", "B", "B", "B", "B")
```

```
IT <- integration.test(Y.gpa$coords, partition.gp=land.gps,
                      iter=999, print.progress = FALSE)
summary(IT) # Test summary
```

Call:

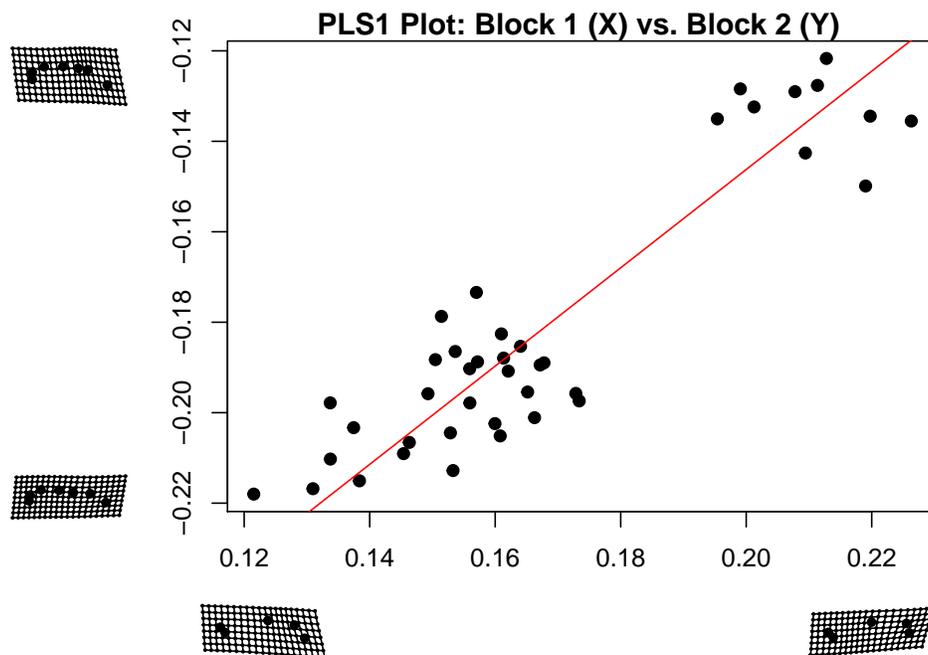
```
integration.test(A = Y.gpa$coords, partition.gp = land.gps, iter = 999,
               print.progress = FALSE)
```

r-PLS: 0.911

P-value: 0.001

Based on 1000 random permutations

```
plot(IT) # PLS plot
```



7.2 Compare modular signal to alternative landmark subsets (`modularity.test`)

Function quantifies the degree of modularity between two or more hypothesized modules of Procrustes-aligned landmark coordinates and compares this to patterns found by randomly assigning landmarks into subsets (Adams 2016). To see the manual page for this function, type `?geomorph::modularity.test`. (Formerly known as `compare.modular.partitions`).

Function

```
modularity.test(A, partition.gp, iter = 999, CI = FALSE, seed = NULL, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)

- *partition.gp* A vector describing which landmarks (or variables) belong in which partition (e.g. A,A,A,B,B,B,C,C,C)
- *iter* Number of iterations for significance testing
- *CI* A logical argument indicating whether bootstrapping should be used for estimating confidence intervals
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = “random”, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Example

Test for morphological integration between plethodon skull and mandible shape

```
data(pupfish) # example dataset
Y.gpa<-gpagen(pupfish$coords, print.progress = FALSE) # GPA-alignment
# Define landmarks on the body (a) and operculum (b)
land.gps<-rep('a',56); land.gps[39:48]<-'b'
MT <- modularity.test(Y.gpa$coords,land.gps,CI=FALSE,iter=499, print.progress = FALSE)
summary(MT) # Test summary
```

Call:

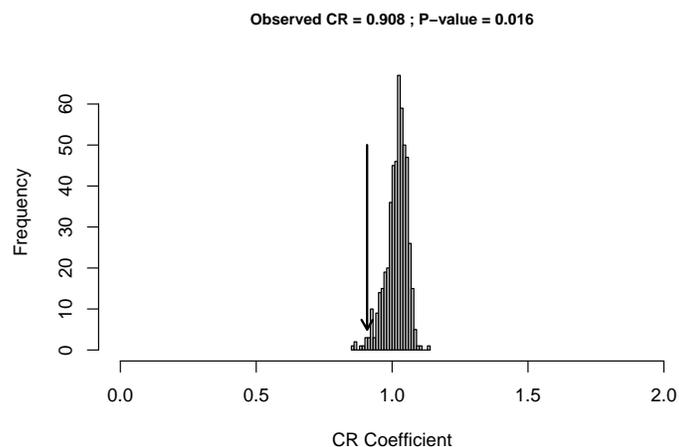
```
modularity.test(A = Y.gpa$coords, partition.gp = land.gps, iter = 499,
  CI = FALSE, print.progress = FALSE)
```

CR: 0.908

P-value: 0.016

Based on 500 random permutations

```
plot(MT) # Histogram of CR sampling distribution
```



7.2.1 Helper function: `define.modules`

Function takes a matrix of digitized landmark coordinates and allows user assign landmarks to each module. The output is a list of which landmarks belong in which partition, to be used by `modularity.test` of `integration.test` option `partition.gp`.

Function

```
define.modules(spec, nmodules)
```

Arguments

- *spec* A $p \times k$ matrix containing landmark coordinates of a single specimen (2D or 3D)
- *nmodules* Number of modules to be defined

7.2.1.1 Selection in 2D

Choosing which landmarks will be included in each module involves landmark selection using a mouse in the plot window. The user is prompted to select each landmark in turn to be assigned to module 1: using the LEFT mouse button (or regular button for Mac users), click on the hollow circle to choose the landmark. Selected landmarks will be filled in. When all landmarks for module 1 are chosen, press 'esc', and then start selecting landmarks for module 2. Repeat until all modules are defined.

7.2.1.2 Selection in 3D

Choosing which landmarks will be included in each module involves landmark selection using a mouse in the rgl plot window. The user is prompted to select one or more landmarks. To do so, use the RIGHT mouse button (or command + LEFT button for Mac users), draw a rectangle around one or more landmarks to select. Selected landmarks will be colored yellow. Then type into the console a letter (e.g. 1, 2, 3...) to assign selected landmark(s) to this module. Repeat until all landmarks are assigned to modules.

```
# using first specimen of plethodon dataset
partition.gp <- define.modules(plethodon$land[, , 1], 2)
Select landmarks in module 1
Press esc when finished

Select landmarks in module 2
Press esc when finished

partition.gp
[1] "1" "1" "1" "1" "1" "1" "2" "2" "2" "2" "2" "2" "2"
```

8 Phylogenetic Comparative methods

The following methods allow users to take phylogenetic non-independence into account during the analysis of multivariate datasets.

All of these functions require a phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional). The tree must have number of tips equal to number of taxa in the data matrix (e.g., `?ape::drop.tip`). And, tip labels of the tree MUST be exactly the same as the taxa names in the landmark data matrix (check using `match`). To learn more about phylogenetic trees in *R*, look at these resources: <http://www.r-phylo.org/wiki/> and http://bodegaphylo.wikispot.org/Phylogenetics_and_Comparative_Methods_in_R

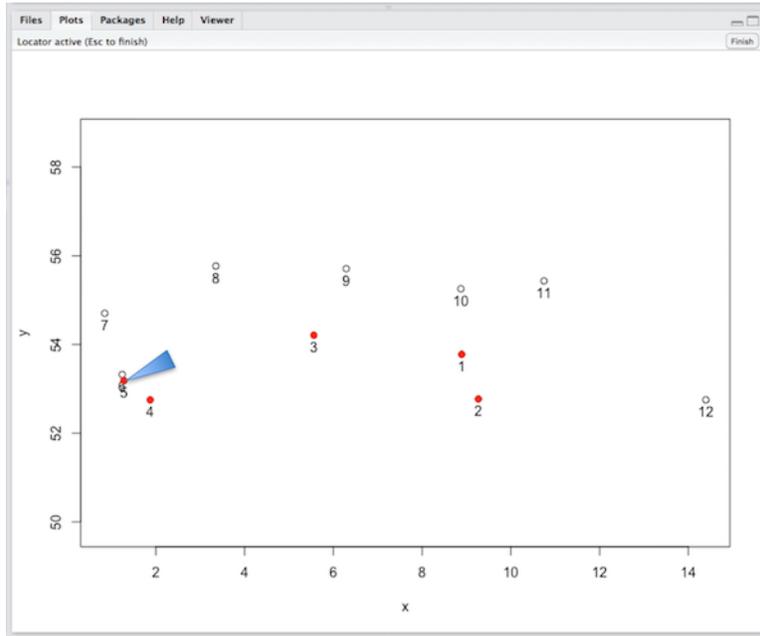


Figure 14: define.modules1

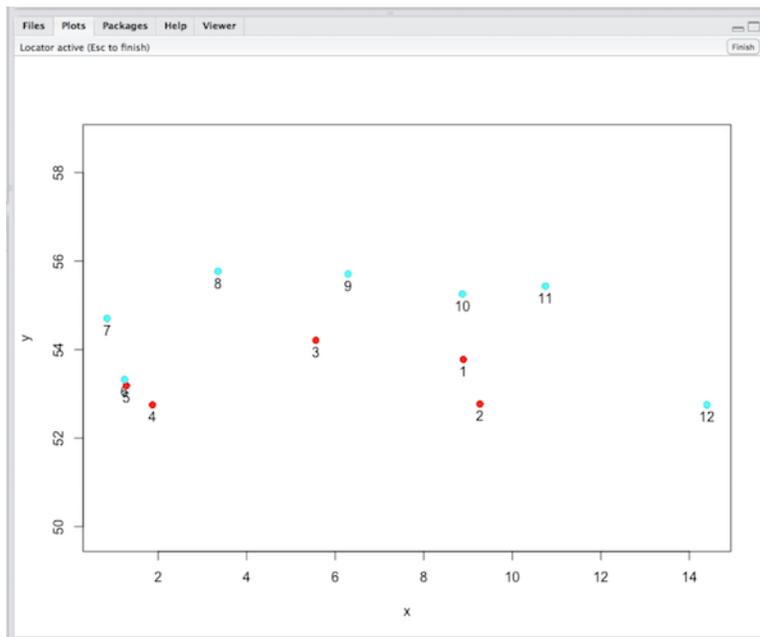


Figure 15: define.modules1

8.1 Assessing phylogenetic signal in morphometric data (physignal)

Function calculates the degree of phylogenetic signal from a set of Procrustes-aligned specimens. The degree of phylogenetic signal in data is estimated using the multivariate version of the K-statistic (Kmult: Adams 2014). This value evaluates the degree of phylogenetic signal in a dataset relative to what is expected under a Brownian motion model of evolution. For geometric morphometric data, the approach is a mathematical generalization of the Kappa statistic (Blomberg et al. 2003) appropriate for highly multivariate data (see Adams 2014). Significance testing is found by permuting the shape data among the tips of the phylogeny. Note that this method can be quite slow as ancestral states must be estimated for every iteration. To see the manual page for this function, type `?geomorph::physignal`.

Function

```
physignal(A, phy, iter = 999, seed = NULL, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)
- *phy* A phylogenetic tree of class phylo - see `?ape::read.tree` (optional)
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = "random", a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Example

Test for phylogenetic signal in plethodon head shape

```
data(plethspecies) # example dataset
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE) # GPA-alignment
PS.shape <- physignal(A=Y.gpa$coords,phy=plethspecies$phy,
                      iter=999, print.progress = FALSE)
summary(PS.shape) # Test summary
```

Call:

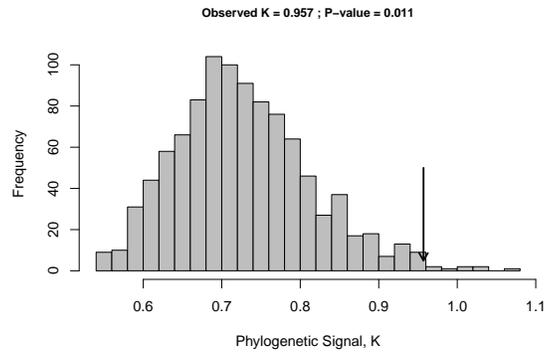
```
physignal(A = Y.gpa$coords, phy = plethspecies$phy, iter = 999,
          print.progress = FALSE)
```

Observed Phylogenetic Signal (K): 0.9573

P-value: 0.011

Based on 1000 random permutations

```
plot(PS.shape) # Histogram
```



Test for phylogenetic signal in plethodon head size

```
PS.size <- physignal(A=Y.gpa$Csize,phy=plethspecies$phy,
                    iter=999, print.progress = FALSE)
summary(PS.size) # Test summary
```

Call:

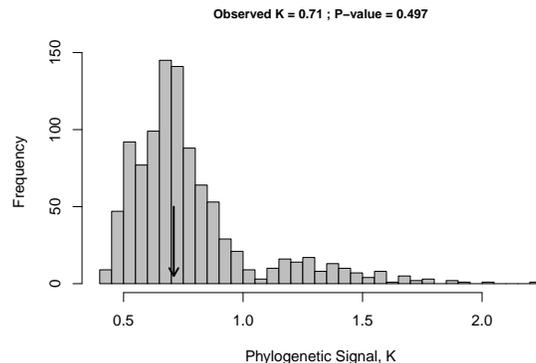
```
physignal(A = Y.gpa$Csize, phy = plethspecies$phy, iter = 999,
          print.progress = FALSE)
```

Observed Phylogenetic Signal (K): 0.7098

P-value: 0.497

Based on 1000 random permutations

```
plot(PS.size) # Histogram
```



8.2 Quantify phylogenetic morphological integration between two sets of variables (`phylo.integration`)

Function quantifies the degree of phylogenetic morphological covariation between two or more sets of Procrustes-aligned coordinates using partial least squares. The degree of morphological covariation is estimated between two or sets of variables while accounting for phylogeny using partial least squares (Adams and Felice 2014), and under a Brownian motion model of evolution. If more than two partitions are defined, the average pairwise PLS correlation is utilized as the test statistic. The observed value is statistically assessed using permutation, where data for one partition are permuted relative to the other partitions. Note that this permutation is performed on phylogenetically- transformed data, so that the probability of phylogenetic

association of A vs. B is similar to that of B vs. A: i.e., $\text{prob}(A,B|\text{phy}) \sim \text{prob}(B,A|\text{phy})$. (Formerly known as `phylo.pls`). To see the manual page for this function, type `?geomorph::phylo.integration`.

Function

```
phylo.integration(A, A2 = NULL, phy, partition.gp = NULL, iter = 999,
  seed = NULL, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)
- *A2* Optional 3D array containing GPA-aligned coordinates, or 2D matrix, for the second partition
- *phy* A phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional)
- *partition.gp* A vector describing which landmarks (or variables) belong in which partition (e.g. A,A,A,B,B,B,C,C,C) (required when only *A* is provided)
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left `NULL` (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If `seed = "random"`, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Example

Test for phylogenetic morphological integration between plethodon cranium and mandible shape

```
data(plethspecies) # example dataset
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE) # GPA-alignment
# landmarks on the skull and mandible assigned to partitions B and A reseptively
land.gps<-c("A","A","A","A","A","B","B","B","B","B","B")
IT<- phylo.integration(Y.gpa$coords,partition.gp=land.gps, phy=plethspecies$phy,
  iter=999, print.progress = FALSE)
summary(IT) # Test summary
```

Call:

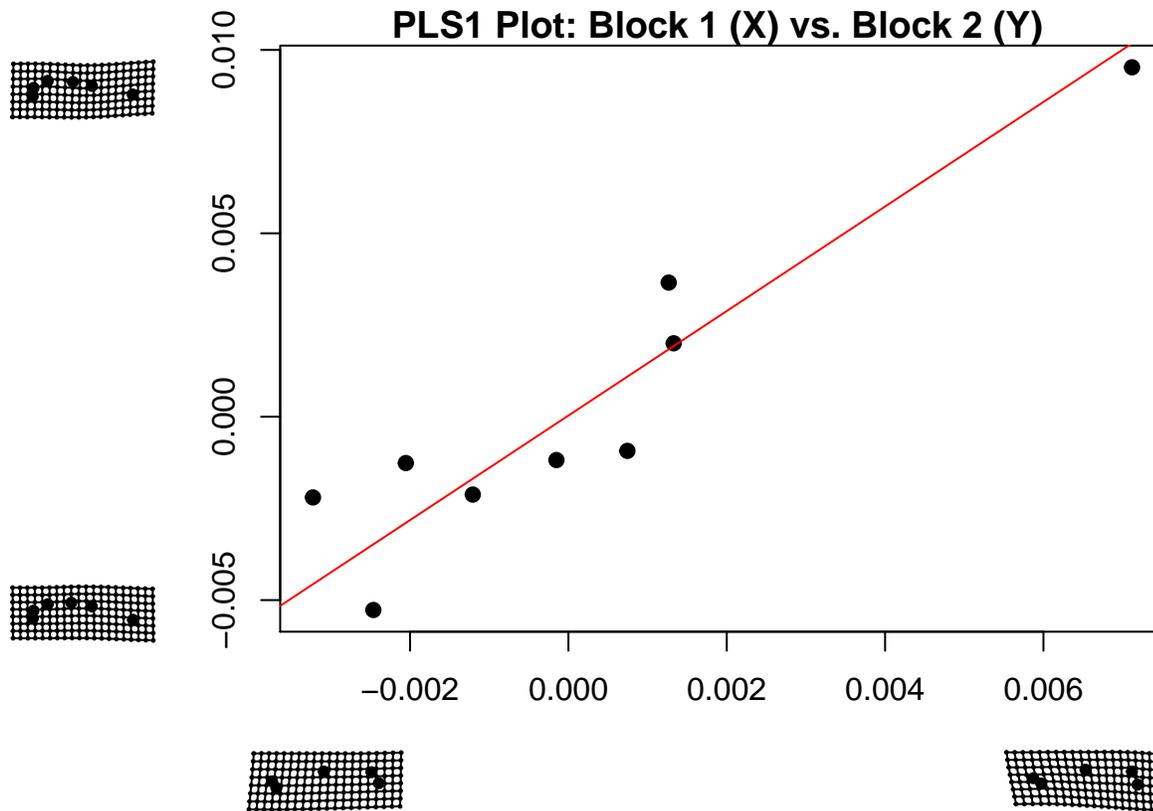
```
phylo.integration(A = Y.gpa$coords, phy = plethspecies$phy, partition.gp = land.gps,
  iter = 999, print.progress = FALSE)
```

r-PLS: 0.933

P-value: 0.023

Based on 1000 random permutations

```
plot(IT) # PLS plot
```



8.3 Comparing rates of shape evolution on phylogenies (`compare.evol.rates`)

Function calculates rates of shape evolution for two or more groups of species on a phylogeny from a set of Procrustes-aligned specimens, under a Brownian motion model of evolution. The approach is based on the distances between species in morphospace after phylogenetic transformation (Adams 2014). From the data the rate of shape evolution for each group is calculated, and a ratio of rates is obtained. If three or more groups of species are used, the ratio of the maximum to minimum rate is used as a test statistic (see Adams 2014). Significance testing is accomplished by phylogenetic simulation in which tips data are obtained under Brownian motion using a common evolutionary rate pattern for all species on the phylogeny. Specifically, the common evolutionary rate matrix for all species is used, with the multi-dimensional rate used along the diagonal elements (see Denton and Adams 2015). This procedure is more general than the original simulation procedure, and retains the desirable statistical properties of earlier methods, and under a wider array of data types. If three or more groups of species are used, pairwise p-values are also calculated. The function can be used to obtain a rate for the whole dataset of species by using a dummy group factor assigning all species to one group. To see the manual page for this function, type `?geomorph::compare.evol.rates`.

Function

```
compare.evol.rates(A, phy, gp, iter = 999, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)
- *phy* A phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional)
- *gp* A factor describing group membership
- *iter* Number of iterations for significance testing

- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Example

Test for phylogenetic morphological integration between plethodon cranium and mandible shape

```
data(plethspecies) # example dataset
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE) # GPA-alignment
# group factor endangered species vs. rest
gp.end<-factor(c(0,0,1,0,0,1,1,0,0))
names(gp.end)<-plethspecies$phy$tip
ER<-compare.evol.rates(A=Y.gpa$coords, phy=plethspecies$phy, gp=gp.end,
                      iter=999, print.progress = FALSE)
summary(ER) # Test summary
```

Call:

Observed Rate Ratio: 1.8372

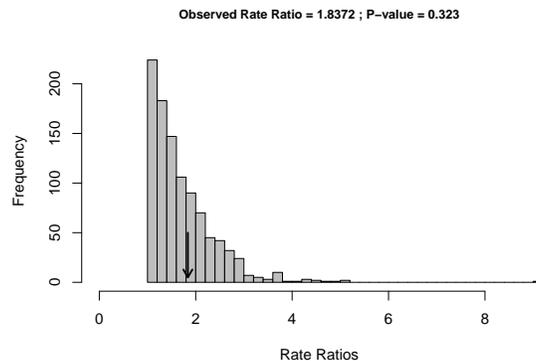
P-value: 0.323

Based on 1000 random permutations

The rate for group 0 is 1.79641754369177e-06

The rate for group 1 is 3.30041235292046e-06

```
plot(ER) # Histogram
```



8.4 Comparing rates of shape evolution among traits on phylogenies (`compare.multi.evol.rates`)

Function compares rates of morphological evolution for two or more multi-dimensional traits on a phylogeny, under a Brownian motion model of evolution following the procedure of Denton and Adams (2015). The approach calculates multivariate evolutionary rates found from the distances between species in morphospace after phylogenetic transformation (sensu Adams 2014). From the data the rate of shape evolution for each multi-dimensional trait is calculated, and a ratio of rates is obtained. If three or more traits are used, the ratio of the maximum to minimum rate is used as a test statistic (see Denton and Adams 2015). Significance testing is accomplished by phylogenetic simulation in which tips data are obtained under Brownian motion using an evolutionary rate matrix for all traits, which contains a common rate for all trait dimensions

(Denton and Adams 2015). If three or more traits are used, pairwise p-values are also returned. To see the manual page for this function, type `?geomorph::compare.multi.evol.rates`.

Function

```
compare.multi.evol.rates(A, gp, phy, Subset = TRUE, iter = 999, print.progress = TRUE)
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates of all specimens, or a matrix (n x variables)
- *gp* A factor describing group membership
- *phy* A phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional)
- *Subset* A logical value indicating whether or not the traits are subsets from a single landmark configuration (default is `TRUE`)
- *iter* Number of iterations for significance testing
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

Comparisons of evolutionary rates between traits may be accomplished in one of two ways. First, if the traits are part of a single shape that was subjected to a single Procrustes superimposition (i.e., they are subsets of landmarks in the configuration), then the procedure is performed without alteration as described above (`Subset = TRUE`). However, if the shapes are derived from different structures (shapes) that were superimposed separately, then the estimates of the rates must take the difference in the number of trait dimensions into account (see discussion in Denton and Adams 2015). This option is identified by selecting `Subset = FALSE`.

Example

Test for phylogenetic morphological integration between plethodon cranium and mandible shape

```
data(plethspecies) # example dataset
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE) # GPA-alignment
# landmarks on the skull and mandible assigned to partitions B and A respectively
land.gps<-c("A","A","A","A","A","B","B","B","B","B","B")
EMR<-compare.multi.evol.rates(A=Y.gpa$coords,gp=land.gps,
  Subset=TRUE, phy= plethspecies$phy, iter=999, print.progress = FALSE)
summary(EMR) # Test summary
```

Call:

Observed Rate Ratio: 1.2997

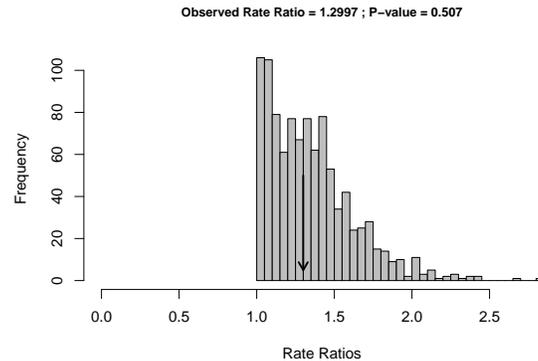
P-value: 0.507

Based on 1000 random permutations

The rate for group A is 1.97486480473745e-06

The rate for group B is 2.56681943179345e-06

```
plot(EMR) # Histogram
```



8.5 Phylogenetic ANOVA/regression for shape data (`procD.pgls`)

Function performs Procrustes ANOVA and regression models in a phylogenetic context under a Brownian motion model of evolution, in a manner that can accommodate high-dimensional datasets. The approach is derived from the statistical equivalency between parametric methods utilizing covariance matrices and methods based on distance matrices (Adams 2014). To see the manual page for this function, type `?geomorph::procD.pgls`.

Function

```
procD.pgls(f1, phy, iter = 999, seed = NULL, int.first = FALSE,  
  effect.type = c("F", "cohen"), RRPP = TRUE,  
  data = NULL, print.progress = TRUE, ...)
```

Arguments

- *f1* A formula for the linear model (e.g., $y \sim x_1 + x_2$)
- *phy* A phylogenetic tree of class `phylo` - see `?ape::read.tree` (optional)
- *iter* Number of iterations for significance testing
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left `NULL` (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = “random”, a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users
- *int.first* A logical value to indicate if interactions of first main effects should precede subsequent main effects
- *effect.type* One of “F” or “cohen”, to choose from which random distribution to estimate effect size. (The default is “F”. The option, “cohen”, refers to Cohen’s f-squared values. Values are log-transformed before z-score calculation to assure normally distributed effect sizes.)
- *RRPP* A logical value indicating whether residual randomization should be used for significance testing (see Introduction VI. Permutation tests)
- *data* A data frame for the function environment, see `geomorph.data.frame`
- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses

The function works in the same manner as `procD.lm`.

Example

Testing for evolutionary allometry with a D-PGLS approach

```
data(plethspecies) # example dataset  
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE) # GPA-alignment  
gdf <- geomorph.data.frame(shape = Y.gpa$coords, cs = Y.gpa$Csize,
```

```

phy = plethspecies$phy) # make geomorph data frame

# permutation option 1: randomize raw values
procD.pgls(shape ~ log(cs), phy = phy, data = gdf, iter = 999, RRPP = FALSE,
           print.progress = FALSE)

```

Call:

```

procD.pgls(f1 = shape ~ log(cs), phy = phy, iter = 999, RRPP = FALSE,
          data = gdf, print.progress = FALSE)

```

Type I (Sequential) Sums of Squares and Cross-products
Randomization of Raw Values used
1000 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
log(cs)	1	0.00006868	6.8679e-05	0.15096	1.2446	0.4356	0.333
Residuals	7	0.00038627	5.5182e-05				
Total	8	0.00045495					

```

# permutation option 1: randomize residuals
procD.pgls(shape ~ log(cs), phy = phy, data = gdf, iter = 999, RRPP = TRUE,
           print.progress = FALSE)

```

Call:

```

procD.pgls(f1 = shape ~ log(cs), phy = phy, iter = 999, RRPP = TRUE,
          data = gdf, print.progress = FALSE)

```

Type I (Sequential) Sums of Squares and Cross-products
Randomized Residual Permutation Procedure Used
1000 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
log(cs)	1	0.00006868	6.8679e-05	0.15096	1.2446	0.4356	0.333
Residuals	7	0.00038627	5.5182e-05				
Total	8	0.00045495					

9 Other methods

9.1 Calculate morphological disparity for one or more groups (`morphol.disparity`)

Function estimates morphological disparity and performs pairwise comparisons among groups. Morphological disparity is a measure of variance in morphological traits. Disparity of landmark-based data is usually measured using the Procrustes Variance, the sum of the diagonal elements of the group covariance matrix. Of course many other methods exist for measuring disparity, but we will not discuss them here. To see the manual page for this function, type `?geomorph::morphol.disparity`.

Function

```

morphol.disparity(f1, groups = NULL, iter = 999, seed = NULL,
                 data = NULL, print.progress = TRUE ...)

```

Arguments

- *f1* A formula describing the linear model used. The left-hand portion of the formula should be a 3D array (p x k x n) containing GPA-aligned coordinates for a set of specimens, or a matrix (n x variables). The right-hand portion of the formula should be "*~1*" to use the overall mean, or "*~ x1 + x2 + x3 + ...*", where each x is a covariate or factor. (Interactions and nested terms also work.)
- *groups* A formula designating groups, e.g., `groups = ~ groups`. If NULL, `morphol.disparity` will attempt to define groups based on the linear model formula, *f1*. If there are no groups inherently indicated in *f1* and *groups* is NULL, a single Procrustes variance will be returned for the entire data set.
- *iter* Number of iterations for permutation test
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = "random", a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *data* A data frame for the function environment, see `geomorph.data.frame`
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses
- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)

Example

```
data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
gdf <- geomorph.data.frame(shape = Y.gpa$coords,
  site = plethodon$site, species = plethodon$species) # make geomorph data frame

# Morphological disparity for entire data set
morphol.disparity(shape ~ 1, groups= NULL, data = gdf, iter=999,
  print.progress = FALSE)
```

No factor in formula from which to define groups.

Procrustes variance:

```
[1] 0.004923493
```

```
# Morphological disparity without covariates, using overall mean
morphol.disparity(shape ~ 1, groups= ~ species*site, data = gdf, iter=999,
  print.progress = FALSE)
```

Call:

```
morphol.disparity(f1 = shape ~ 1, groups = ~species * site, iter = 999,
  data = gdf, print.progress = FALSE)
```

Randomized Residual Permutation Procedure Used
1000 Permutations

Procrustes variances for defined groups

```
Jord:Allo Jord:Symp Teyah:Allo Teyah:Symp
0.003825240 0.005937274 0.003355765 0.006575694
```

Pairwise absolute differences between variances

```
Jord:Allo Jord:Symp Teyah:Allo Teyah:Symp
Jord:Allo 0.000000000 0.0021120344 0.0004694746 0.0027504546
```

```
Jord:Symp 0.0021120344 0.0000000000 0.0025815091 0.0006384202
Teyah:Allo 0.0004694746 0.0025815091 0.0000000000 0.0032199292
Teyah:Symp 0.0027504546 0.0006384202 0.0032199292 0.0000000000
```

P-Values

	Jord:Allo	Jord:Symp	Teyah:Allo	Teyah:Symp
Jord:Allo	1.000	0.020	0.617	0.002
Jord:Symp	0.020	1.000	0.001	0.512
Teyah:Allo	0.617	0.001	1.000	0.001
Teyah:Symp	0.002	0.512	0.001	1.000

```
# Morphological disparity without covariates, using group means
morphol.disparity(shape ~ species*site, groups= ~species*site, data = gdf, iter=999,
  print.progress = FALSE)
```

Call:

```
morphol.disparity(fl = shape ~ species * site, groups = ~species *
  site, iter = 999, data = gdf, print.progress = FALSE)
```

Randomized Residual Permutation Procedure Used
1000 Permutations

Procrustes variances for defined groups

	Jord:Allo	Jord:Symp	Teyah:Allo	Teyah:Symp
	0.001933408	0.001790153	0.001732997	0.001785645

Pairwise absolute differences between variances

	Jord:Allo	Jord:Symp	Teyah:Allo	Teyah:Symp
Jord:Allo	0.0000000000	1.432545e-04	2.004103e-04	1.477632e-04
Jord:Symp	0.0001432545	0.000000e+00	5.715581e-05	4.508733e-06
Teyah:Allo	0.0002004103	5.715581e-05	0.000000e+00	5.264707e-05
Teyah:Symp	0.0001477632	4.508733e-06	5.264707e-05	0.000000e+00

P-Values

	Jord:Allo	Jord:Symp	Teyah:Allo	Teyah:Symp
Jord:Allo	1.000	0.655	0.556	0.661
Jord:Symp	0.655	1.000	0.859	0.986
Teyah:Allo	0.556	0.859	1.000	0.872
Teyah:Symp	0.661	0.986	0.872	1.000

9.2 Quantify and compare shape change trajectories (trajectory.analysis)

Function quantifies phenotypic shape change trajectories from a set of specimens, and assesses variation in attributes of the trajectories via permutation. A shape change trajectory is defined by a sequence of shapes in tangent space. These trajectories can be quantified for various attributes (their size, orientation, and shape), and comparisons of these attribute enable the statistical comparison of shape change trajectories (see Collyer and Adams 2013; Collyer and Adams 2007; Adams and Collyer 2007; Adams and Collyer 2009). To see the manual page for this function, type `?geomorph::trajectory.analysis`.

Function

```
trajectory.analysis(f1, f2 = NULL, iter = 999, seed = NULL,  
  traj.pts = NULL, data = NULL, print.progress = TRUE, ...)
```

Arguments

- *f1* A formula for the linear model, for trajectories (e.g., $Y \sim A$ or $Y \sim A * B$). The right hand side of this formula can contain only one or two factors
- *f2* A formula for additional covariates (e.g., $\sim x1 + x2$)
- *iter* Number of iterations for permutation test
- *seed* An optional argument for setting the seed for random permutations of the resampling procedure. If left NULL (the default), the exact same P-values will be found for repeated runs of the analysis (with the same number of iterations). If *seed* = "random", a random seed will be used, and P-values will vary. One can also specify an integer for specific seed values, which might be of interest for advanced users.
- *traj.pts* An optional value specifying the number of points in each trajectory (if *f1* contains a single factor)
- *data* A data frame for the function environment, see `geomorph.data.frame`
- *print.progress* A logical value to indicate whether a progress bar should be printed to the screen. This is helpful for long-running analyses
- ... Arguments passed on to `procD.fit` (typically associated with the `lm` function)

Data input is specified by a two formulae (e.g., $Y \sim X$), where 'Y' specifies the response variables (trajectory data), and 'X' contains one or more independent variables (discrete or continuous). The response matrix 'Y' can be either in the form of a two-dimensional data matrix of dimension (n x [p x k]), or a 3D array (p x n x k). It is assumed that the order of the specimens 'Y' matches the order of specimens in 'X'.

Linear model fits (using the `lm` function) can also be input in place of a formula. Arguments for `lm` can also be passed on via this function. The first formula, *f1*, must contain the independent variable on the left-hand side of the formula (e.g., $Y \sim$) and either a single factor or a two factor interaction on the right-hand side.

If a single factor is provided, e.g., $Y \sim A$, it is assumed that groups to be described are the levels of factor A and that the data in Y comprise trajectories. In this case, the *traj.pts* = NULL argument must be changed to a numeric value to define the number of points in the trajectory. It is also assumed that the data are structured as variables within points. For example, y11 y21 y31 y12 y22 y32 y13 y23 y33 y14 y24 y34 would be columns of a matrix, Y, describing a 4-point trajectory in a data space defined by three variables. This is the proper arrangement; the following is an improper arrangement: y11 y12 y13 y14 y21 y22 y23 y24 y31 y32 y33 y34, as it groups points within variables. This approach is typical when comparing motion paths (see Adams and Cerney 2007).

Example # Motion paths represented by 5 time points per motion

```
data(motionpaths) # Simulated motion paths  
gdf <- geomorph.data.frame(trajectories = motionpaths$trajectories,  
  groups = motionpaths$groups)  
TA <- trajectory.analysis(f1 = trajectories ~ groups,  
  traj.pts = 5, data=gdf, iter=199, print.progress = FALSE)  
summary(TA)
```

```
trajectory.analysis(f1 = trajectories ~ groups, iter = 199, traj.pts = 5,  
  data = gdf, print.progress = FALSE)
```

Type I (Sequential) Sums of Squares and Cross-products
Randomized Residual Permutation Procedure Used
200 Permutations

	Df	SS	MS	Rsq	F	Z	Pr(>F)
groups	3	6520.9	2173.63	0.98608	849.85	8.3797	0.005 **
Residuals	36	92.1	2.56				
Total	39	6613.0					

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Pairwise statistical results:

*** Path Distances

Observed Path Distances

	1	2	3	4
1	10.144421	6.557011	10.949722	10.097348

Pairwise Absolute Differences Between Path Distances

	1	2	3	4
1	0.0000000	3.587410	0.8053008	0.04707265
2	3.58741011	0.000000	4.3927109	3.54033746
3	0.80530081	4.392711	0.000000	0.85237346
4	0.04707265	3.540337	0.8523735	0.0000000

Effect Sizes

	1	2	3	4
1	0.0000000	6.062112	0.5174543	-1.2512584
2	6.0621124	0.000000	8.0030824	6.6722934
3	0.5174543	8.003082	0.000000	0.6235063
4	-1.2512584	6.672293	0.6235063	0.0000000

P-Values

	1	2	3	4
1	1.000	0.005	0.235	0.955
2	0.005	1.000	0.005	0.005
3	0.235	0.005	1.000	0.265
4	0.955	0.005	0.265	1.000

*** Principal Vector Correlations

Pairwise Correlations

	1	2	3	4
1	1.0000000	0.9989147	0.6122502	0.9996899
2	0.9989147	1.0000000	0.6484120	0.9974452
3	0.6122502	0.6484120	1.0000000	0.5923727
4	0.9996899	0.9974452	0.5923727	1.0000000

Effect Sizes

	1	2	3	4
1	0.0000000	-0.9361053	5.594340	-1.1182119
2	-0.9361053	0.0000000	5.405341	-0.7656184
3	5.5943396	5.4053407	0.000000	6.3543967
4	-1.1182119	-0.7656184	6.354397	0.0000000

P-Values

	1	2	3	4
1	1.000	0.845	0.005	0.915
2	0.845	1.000	0.005	0.765
3	0.005	0.005	1.000	0.005
4	0.915	0.765	0.005	1.000

*** Trajectory Shape Differences

Pairwise Shape Differences (Procrustes Distance)

	1	2	3	4
1	0.0000000	0.10279471	0.10686382	0.2821303
2	0.1027947	0.00000000	0.07843294	0.3649111
3	0.1068638	0.07843294	0.00000000	0.3508621
4	0.2821303	0.36491114	0.35086211	0.0000000

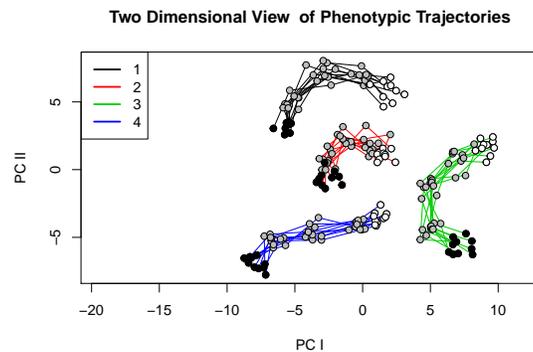
Effect Sizes

	1	2	3	4
1	0.0000000	-0.3456923	-0.2333463	3.703360
2	-0.3456923	0.0000000	-0.9193196	5.599389
3	-0.2333463	-0.9193196	0.0000000	6.095049
4	3.7033600	5.5993894	6.0950485	0.0000000

P-Values

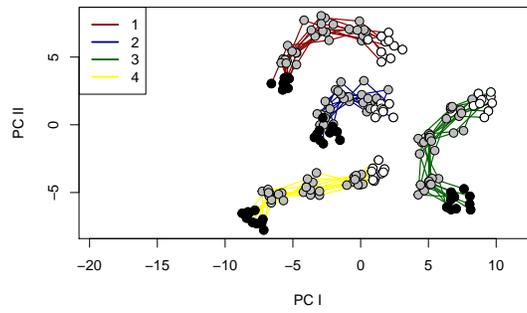
	1	2	3	4
1	1.000	0.610	0.545	0.005
2	0.610	1.000	0.800	0.005
3	0.545	0.800	1.000	0.005
4	0.005	0.005	0.005	1.000

```
# Various plotting examples  
plot(TA)
```



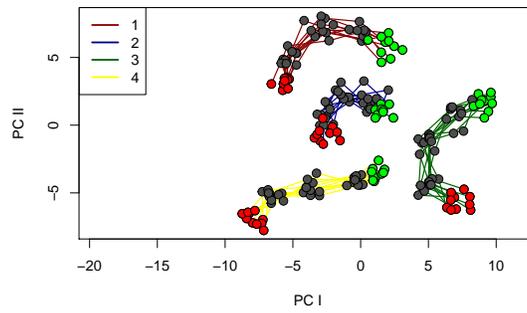
```
plot(TA, group.cols = c("dark red", "dark blue", "dark green", "yellow"), pt.scale = 1.3)
```

Two Dimensional View of Phenotypic Trajectories



```
plot(TA, group.cols = c("dark red", "dark blue", "dark green", "yellow"),  
pt.seq.pattern = c("green", "gray30", "red"), pt.scale = 1.3)
```

Two Dimensional View of Phenotypic Trajectories



10 DATA VISUALIZATION

The next 4 chapters will cover *geomorph*'s visualization functions, organized as follows (hyperlinks): ordination graphs, shape change graphs, data inspection, and plots for analytical functions. Plotting of Procrustes shape data requires many specialist plot functions that we have written to help users to quickly visualize their data, and provide the necessary information for users to produce their own publication quality graphs. Many of *geomorph*'s analytical functions described in Vigentte 3 return information in the form of a list that can simply be used in the S3 generic function plot also.

11 Ordination graphs

11.1 Principal Components Analysis (`plotTangentSpace`)

Usually the first step in a geometric morphometrics study, a Principal Components Analysis (PCA) is an ordination method to visualize the shape variation among individual specimens in the dataset. The function `plotTangentSpace` plots a set of Procrustes-aligned specimens in tangent space along their principal axes, i.e., function performs PCA.

Function

```
plotTangentSpace(A, axis1 = 1, axis2 = 2, warpgrids = TRUE, mesh = NULL, label = FALSE,
                 groups = NULL, legend = FALSE)
```

Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of aligned specimens
- *warpgrids* A logical value indicating whether deformation grids for shapes along X-axis should be displayed
- *mesh* A mesh3d object to be warped to represent shape deformation along X-axis (when warpgrids=TRUE) as described in warpRefMesh.
- *axis1* A value indicating which PC axis should be displayed as the X-axis (default = PC1)
- *axis2* A value indicating which PC axis should be displayed as the Y-axis (default = PC2)
- *label* An optional vector indicating labels for each specimen are to be displayed (or if TRUE, numerical addresses are given)
- *groups* An optional factor vector specifying group identity for each specimen
- *legend* A logical value for whether to add a legend to the plot (only when groups are assigned)

The function performs a principal components analysis of shape variation and plots two dimensions of tangent space for a set of Procrustes-aligned specimens (default is PC1 vs. PC2). The percent variation along each PC-axis is returned. Additionally (and optionally, warpgrids=T), deformation grids can be requested, which display the shape of specimens at the ends of the range of variability along PC1. If groups are provided, specimens from each group are plotted using distinct colors based on the order in which the groups are found in the dataset, and using *R*'s standard color palette: black, red, green, blue, cyan, magenta, yellow, and gray, which are recycled if there are more than 8 groups.

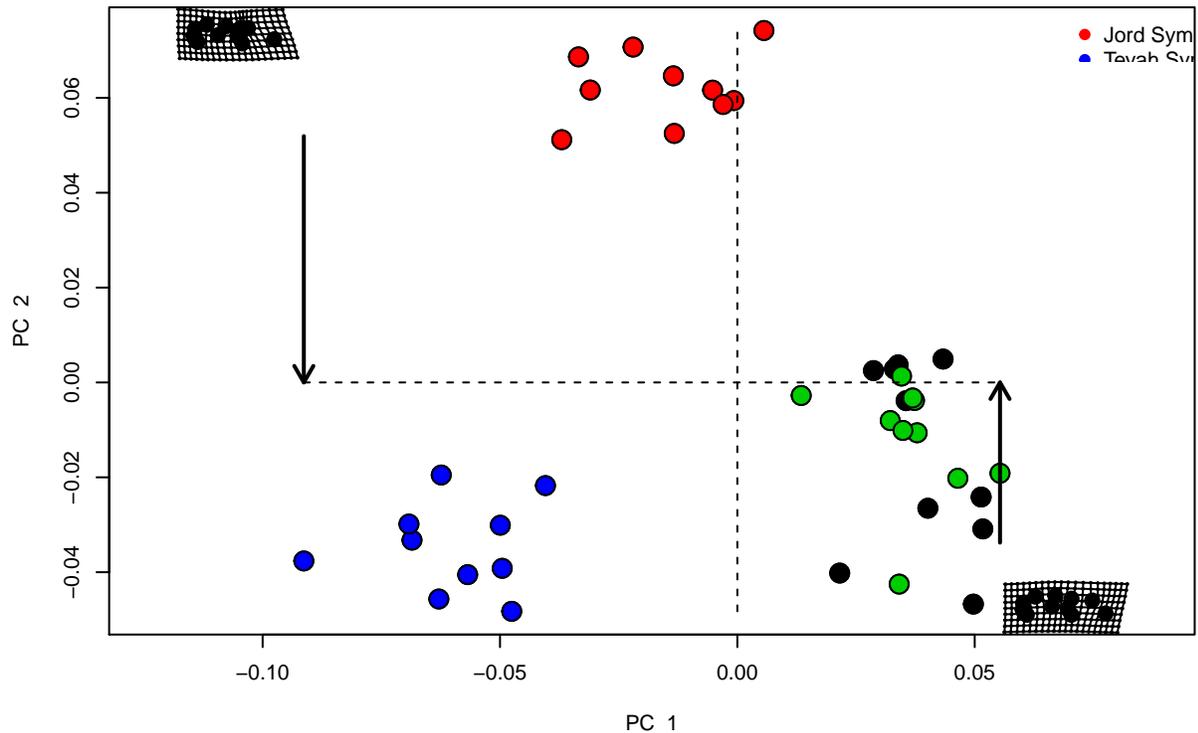
NOTE 1: to change the colors of the groups, simply substitute a vector of the desired colors for each specimen (see example below).

NOTE 2: previous versions of `plotTangentSpace` had option 'verbose' to return the PC scores and PC shapes. From version 3.0.2 this is automatic when assigned to an object.

Example

```
data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
```

```
gp <- as.factor(paste(plethodon$species, plethodon$site)) # create grouping variable
PCA <- plotTangentSpace(Y.gpa$coords, groups = gp, legend = TRUE)
```



```
summary(PCA) # to see the importance of the PC axes
```

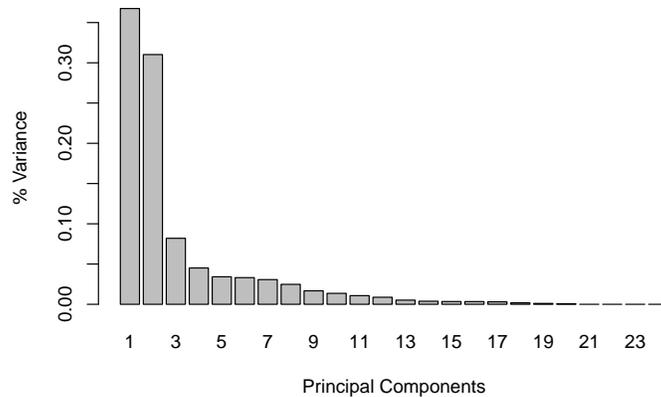
PC Summary

Importance of first k=20 (out of 24) components:

	PC1	PC2	PC3	PC4	PC5	PC6
Standard deviation	0.04307	0.03958	0.02035	0.01509	0.01314	0.01293
Proportion of Variance	0.36743	0.31023	0.08201	0.04512	0.03418	0.03312
Cumulative Proportion	0.36743	0.67767	0.75967	0.80479	0.83897	0.87209
	PC7	PC8	PC9	PC10	PC11	PC12
Standard deviation	0.01245	0.01119	0.009202	0.008289	0.007369	0.006651
Proportion of Variance	0.03068	0.02478	0.016770	0.013600	0.010750	0.008760
Cumulative Proportion	0.90277	0.92755	0.944320	0.957920	0.968670	0.977430
	PC13	PC14	PC15	PC16	PC17	
Standard deviation	0.005116	0.004389	0.004173	0.004141	0.003944	
Proportion of Variance	0.005180	0.003810	0.003450	0.003400	0.003080	
Cumulative Proportion	0.982620	0.986430	0.989880	0.993280	0.996360	
	PC18	PC19	PC20			
Standard deviation	0.003042	0.00231	0.00195			
Proportion of Variance	0.001830	0.00106	0.00075			
Cumulative Proportion	0.998190	0.99925	1.00000			

To plot a graph of the proportion of variance explained by each PC:

```
pvar <- (PCA$sdev^2)/(sum(PCA$sdev^2))
names(pvar) <- seq(1:length(pvar))
barplot(pvar, xlab= "Principal Components", ylab = "% Variance")
```



ProTip! To easily change colors of groups in `plotTangentSpace`

```
col.gp<-c(rep("black",10),rep("red",10),
          rep("yellow",10),rep("orange",10)) # not a factor
plotTangentSpace(Y.gpa$coords, groups = col.gp, legend = TRUE)

# Or even more succinctly using a clever trick with the function match
col.gp <- rainbow(length(levels(gp))) # use rainbow function to generate colours
names(col.gp) <- levels(gp)
col.gp <- col.gp[match(gp, names(col.gp))]
plotTangentSpace(Y.gpa$coords, groups = col.gp, legend = TRUE)
```

The function also returns a list containing: the PC summary (`$pc.summary`), PC scores (`$pc.scores`) – the latter of which can be used by `plot` to reproduce the graphs for publication – and the minimum and maximum shapes (`$pc.shapes`) for all of the PCs, which can be used in `plotRefToTarget`. These shapes and others can also be predicted using the function `shape.predictor`.

```
$pc.scores
      PC1      PC2      PC3      PC4      PC5
[1,] -0.0369931316  0.051182469 -0.0016971188 -3.128812e-03 -0.0109363280
[2,] -0.0007493756  0.059420824  0.0001371746 -2.768676e-03 -0.0081174155
...
$pc.shapes
$pc.shapes$PC1min
      [,1]      [,2]

[1,]  0.16960517 -0.02807690
[2,]  0.21810341 -0.10340535
...
$pc.shapes$PC1max
      [,1]      [,2]
[1,]  0.14235007 -0.020486784
[2,]  0.18012044 -0.086122720
...
$pc.shapes$PC2min
      [,1]      [,2]
[1,]  0.13128088 -0.03616776
[2,]  0.18767216 -0.10979356
...
$pc.shapes$PC2max
      [,1]      [,2]
```

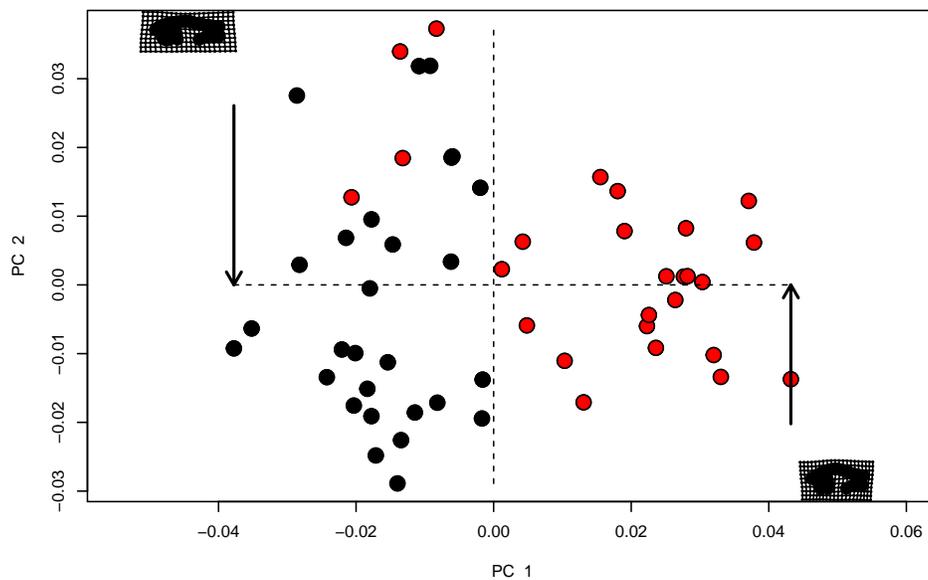
```
[1,] 0.18546661 -0.003640617
[2,] 0.20487416 -0.066270202
...
```

11.1.1 Plotting convex hulls in a PC plot

When shape data are structured into groups, it may be necessary to plot convex hull polygons to show the groups more clearly after PCA.

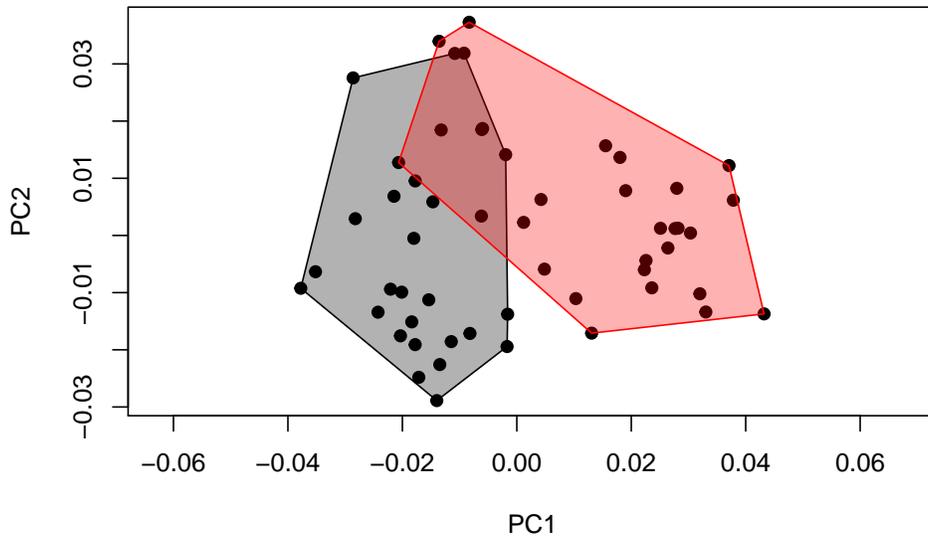
```
data(pupfish) # example dataset
# data already Procrustes Superimposed
# dataset needs dimnames for plotting polygons so we shall assign arbitrary names
dimnames(pupfish$coords)[[3]] <- paste("fish", 1:dim(pupfish$coords)[3], sep="")

PCA <- plotTangentSpace(pupfish$coords, groups = pupfish$Sex) # PCA
```



```
plot(PCA$pc.scores[,c(1,2)], pch=19, asp=1) # make the plot by hand

# Add convex hull polygons to the PCA plot:
colour = c("black", "red") # colour for the two groups
for(j in 1:nlevels(pupfish$Sex)) {
  # Get edge points (used to plot convex hull):
  edge_points <- rownames(PCA$pc.scores[which(pupfish$Sex == levels(pupfish$Sex)[j]),])
  [chull(PCA$pc.scores[which(pupfish$Sex == levels(pupfish$Sex)[j]), c(1,2)])]
  # Plot convex hull as polygon:
  polygon(PCA$pc.scores[edge_points, c(1,2)], col = adjustcolor(colour[j],
    alpha.f = 0.3), border = colour[j])
} # alpha gives the degree of transparency of the polygon
```



11.2 Plot phylogenetic tree and specimens in tangent space (`plotGMPhyloMorphoSpace`)

Another common ordination method is to perform a Principal Components Analysis of species mean shape data and the project a phylogenetic tree into this space, i.e., a phylomorphospace. Function creates a plot of the principal dimensions of tangent space for a set of Procrustes-aligned specimens. Default is a plot of PC axis 1 and 2. The phylogenetic tree for these specimens is superimposed in this plot revealing how shape evolves (e.g., Rohlf 2002; Klingenberg and Gidaszewski 2010; Sidlauskas 2008). The plot also displays the ancestral states for each node of the phylogenetic tree (analogous to `fastAnc` from `phytools`), whose values can optionally be returned. If a tree with branch lengths scaled by time is used, with the option `zaxis = "time"`, the function plots a 3D phylomorphospace, with internal nodes positioned along the Z-axis scaled to time (a.k.a. Chronophylomorphospace, Sakamoto & Ruta 2012).

Function

```
plotGMPhyloMorphoSpace(phy, A, tip.labels = TRUE, node.labels = TRUE,
  ancStates = TRUE, xaxis = 1, yaxis = 2, zaxis = NULL,
  plot.param = list(), shadow = FALSE)
```

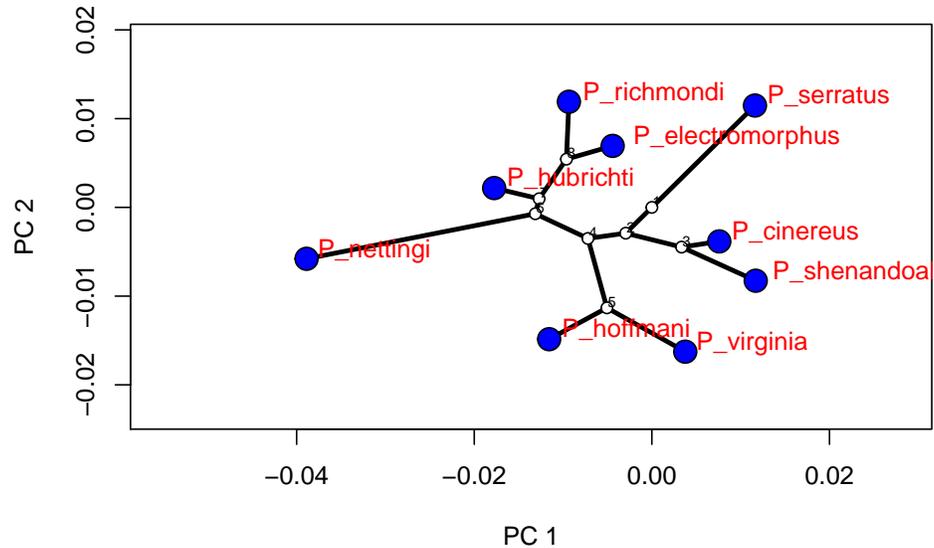
Arguments

- *phy* A phylogenetic tree of class `phylo` - see `read.tree` in library `ape`
- *A* A matrix ($n \times [p \times k]$) or 3D array ($p \times k \times n$) containing GPA-aligned coordinates for a set of specimens
- *tip.labels* A logical value indicating whether taxa labels (tips) should be included
- *node.labels* A logical value indicating whether node labels (ancestors) should be included
- *ancStates* Either a logical value indicating whether ancestral state values should be returned, or a matrix of ancestral states (i.e. calculated with `fastAnc` or `ace`)
- *xaxis* A numeric value indicating which PC axis should be displayed as the X-axis (default = PC1)
- *yaxis* A numeric value indicating which PC axis should be displayed as the Y-axis (default = PC2)
- *zaxis* Optional, a numeric value indicating which PC axis should be displayed as the Z-axis (e.g. PC3) or if `zaxis="time"`, internal nodes are plotted along the Z-axis relative to time
- *plot.param* A list of plotting parameters for the tips (`t.bg`, `t.pch`, `t.cex`), nodes (`n.bg`, `n.pch`, `n.cex`), branches (`l.col`, `l.wd`), taxa labels (`txt.cex`, `txt.adj`, `txt.col`) and node labels (`n.txt.cex`, `n.txt.adj`, `n.txt.col`)
- *shadow* A logical value indicating whether a 2D phylomorphospace should be plotted at the base when `zaxis="time"`

```

data(plethspecies)
Y.gpa<-gpagen(plethspecies$land, print.progress = FALSE)
plotGMPhyloMorphoSpace(plethspecies$phy,Y.gpa$coords,
  plot.param=list(t.bg="blue",txt.col="red",n.cex=1),
  ancStates = F)

```

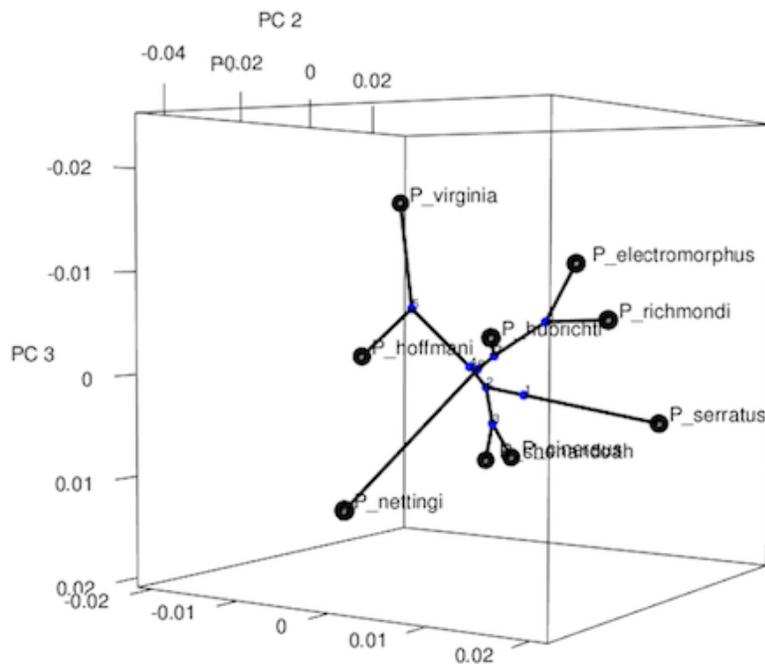


You can also plot a 3D phylomorphospace using the 3rd PC, e.g.,

```

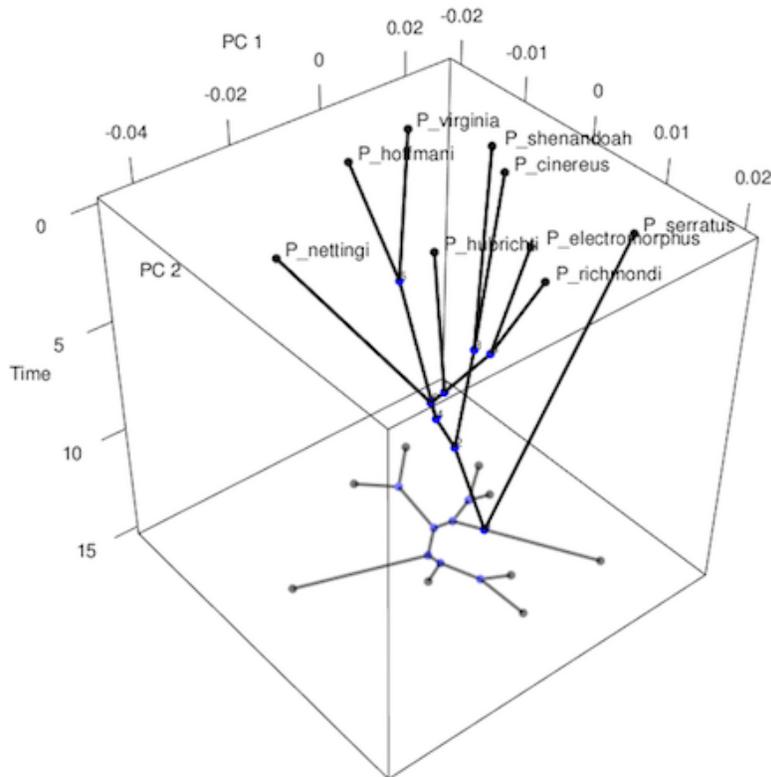
plotGMPhyloMorphoSpace(plethspecies$phy,Y.gpa$coords, zaxis= 3,
  plot.param=list(n.cex=2, n.bg="blue"), shadow=TRUE)

```



Using the option `zaxis = "time"`, the function plots a 3D phylomorphospace, with internal nodes positioned along the Z-axis scaled to time (a.k.a. Chronophylomorphospace, Sakamoto & Ruta 2012).

```
plotGMPhyloMorphoSpace(plethspecies$phy,Y.gpa$coords, zaxis= "time",
                        plot.param=list(n.cex=2, n.bg="blue"), shadow=TRUE)
```



12 Shape change graphs

12.1 Shape prediction from numeric predictors (`shape.predictor`)

Function estimates one or more configurations based on one or more linear predictors, such as PC scores allometric relationships, or any other least squares or partial least squares regression. These configurations can be used with `plotRefToTarget` to generate graphical representations of shape change, based on prediction criteria.

Function

```
shape.predictor(A, x = NULL, Intercept = FALSE, method = c("LS", "PLS"),...)
```

Arguments

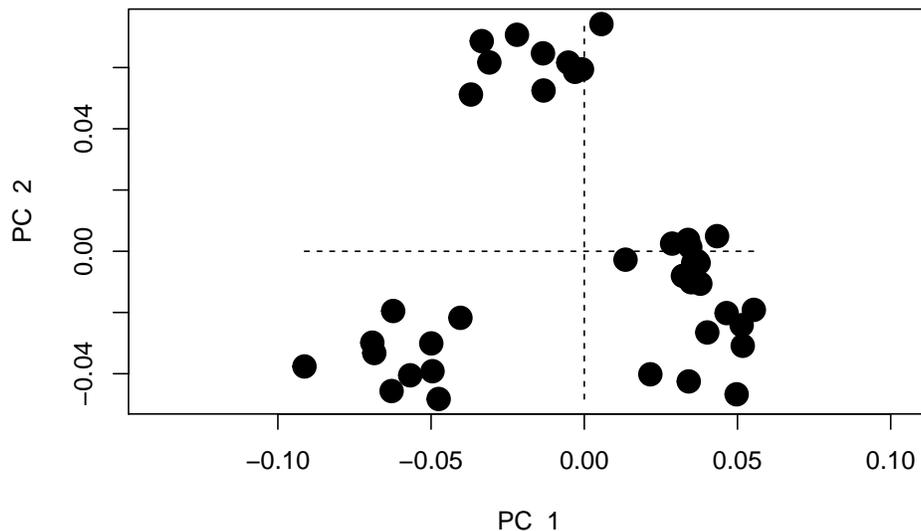
- *A* An array ($p \times k \times n$) containing Procrustes residuals, either from GPA or fitted values from a previous analytical procedure
- *x* Linear (numeric) predictors. Can be a vector or a matrix, or a list containing vectors or matrices. Values must be numeric. If a factor is desired, one should use `model.matrix` to obtain a design matrix. This will impact how prediction criteria need to be provided (see below)
- *Intercept* Logical value to indicate whether an intercept should be used in the linear equation for predictions. Generally, this value will be `FALSE` for shape predictions made in ordination plots. It should be `TRUE` in cases where the expected shape at the point the predictor has a value of 0 is not the mean shape.

- *method* A choice between least squares (LS) or partial least squares (PLS) regression for prediction. The function defaults to LS prediction. PLS might be chosen in cases where correlation is preferred over linear regression. If PLS is chosen, a two-block PLS analysis using `two.b.pls` should be performed first, as only the first singular vector for predictors will be used for defining prediction criteria (see below)
- ... Any number of prediction criteria. Criteria should be presented as either a scalar (if one predictor is provided) or a vector (if more than one predictor or a prediction matrix is provided); e.g., `pred1 = c(0.1, -0.5)`, `pred2 = c(-0.2, -0.1)` (which would be the case if two predictors were provided). It is essential that the number of elements in any prediction criterion matches the number of predictors. Caution should be used when providing a design matrix to ensure that correct dummy variables are used in prediction criteria, and that either 1) an intercept is not included in the design and 2) is TRUE in the Intercept argument; or 1) an intercept is included in the design and 2) is FALSE in the Intercept argument; or 1) an intercept is not included in the design and 2) is FALSE in the Intercept argument, if no intercept is desired

Function returns a list of predicted shapes matching the number of vectors of prediction criteria provides. The predictions also have names matching those of the prediction criteria.

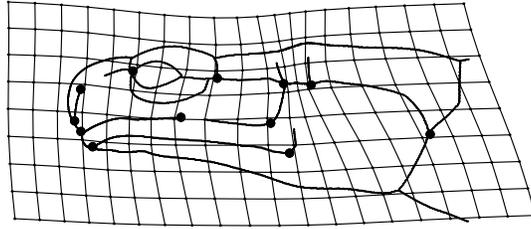
12.1.1 Example 1: Shapes from principal component analysis (`plotTangentSpace`)

```
data("plethodon")
Y.gpa <- gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
M <- mshape(Y.gpa$coords) # mean shape (for all examples)
PCA <- plotTangentSpace(Y.gpa$coords, warpgrids = FALSE) # PCA
```

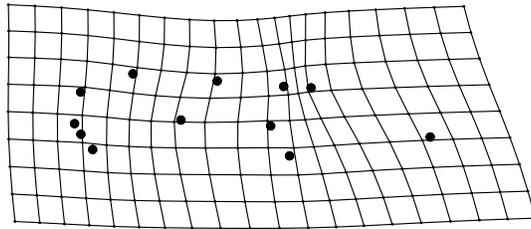


First, we will get the shapes along PC1 using Procrustes values -0.1 and 0.1, corresponding to the x-axis values in the plot above

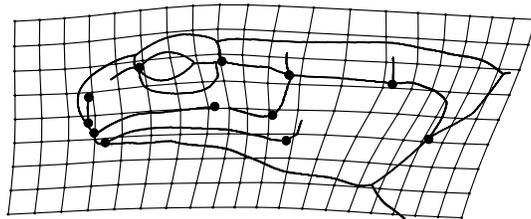
```
preds <- shape.predictor(Y.gpa$coords, x= NULL, Intercept = FALSE,
pred1 = -0.1, pred2 = 0.1) # PC 1 extremes, sort of
plotRefToTarget(M, preds$pred1, outline = plethodon$outline)
```



```
plotRefToTarget(M, preds[[1]]) # can also access list with numerical indexing
```

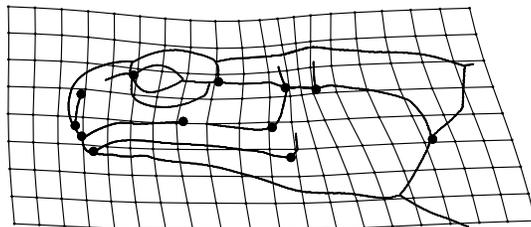


```
plotRefToTarget(M, preds$pred2, outline = plethodon$outline)
```

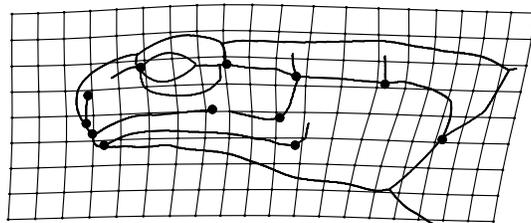


Now, we will get the shapes at the minima and maxima of PC1 using `min` and `max`, corresponding to the x-axis values in the plot above

```
PC <- PCA$pc.scores[,1] # choose first axis
preds <- shape.predictor(Y.gpa$coords, x= PC, Intercept = FALSE,
pred1 = min(PC), pred2 = max(PC)) # PC 1 extremes, more technically
plotRefToTarget(M, preds$pred1, outline = plethodon$outline)
```



```
plotRefToTarget(M, preds$pred2, outline = plethodon$outline)
```

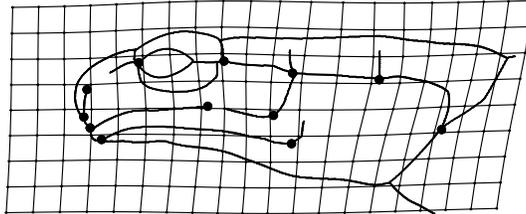


Using both of these techniques, we can choose to visualize a shape from any place in the shape space. Here we set user-picked spots (found with `locator`) - can be anything, but in this case, apparent groups

```

PC <- PCA$pc.scores[,1:2]
preds <- shape.predictor(Y.gpa$coords, x= PC, Intercept = FALSE,
                        pred1 = c(0.045,-0.02),
                        pred2 = c(-0.025,0.06),
                        pred3 = c(-0.06,-0.04))
# shape at middle of bottom-right cluster
plotRefToTarget(M, preds$pred1, outline = plethodon$outline)

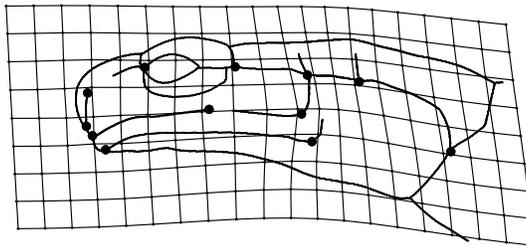
```



```

# shape at middle of top cluster
plotRefToTarget(M, preds$pred2, outline = plethodon$outline)

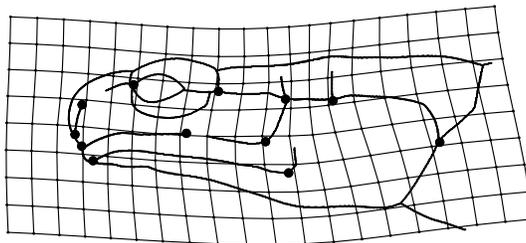
```



```

# shape at middle of bottom-left cluster
plotRefToTarget(M, preds$pred3, outline = plethodon$outline)

```



This example shows how you can predict a shape anywhere in the shape space.

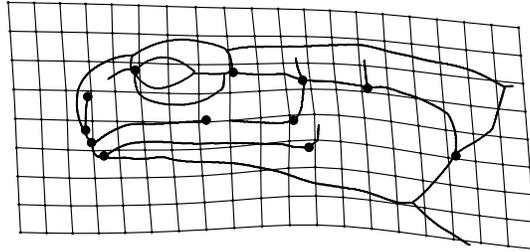
12.1.2 Example 2: Shapes from allometry analyses

First we do a simple allometry model of shape~size (as one would do using `procD.allometry`)

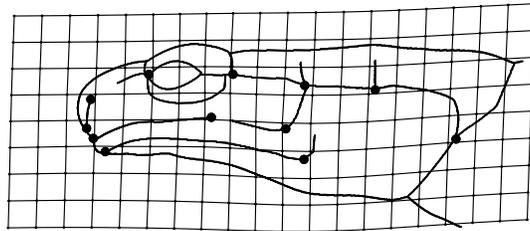
```

preds <- shape.predictor(Y.gpa$coords, x= log(Y.gpa$Csize), Intercept = TRUE,
                        predmin = min(log(Y.gpa$Csize)), predmax = max(log(Y.gpa$Csize)))
plotRefToTarget(M, preds$predmin, outline = plethodon$outline)

```



```
plotRefToTarget(M, preds$predmax, outline = plethodon$outline)
```

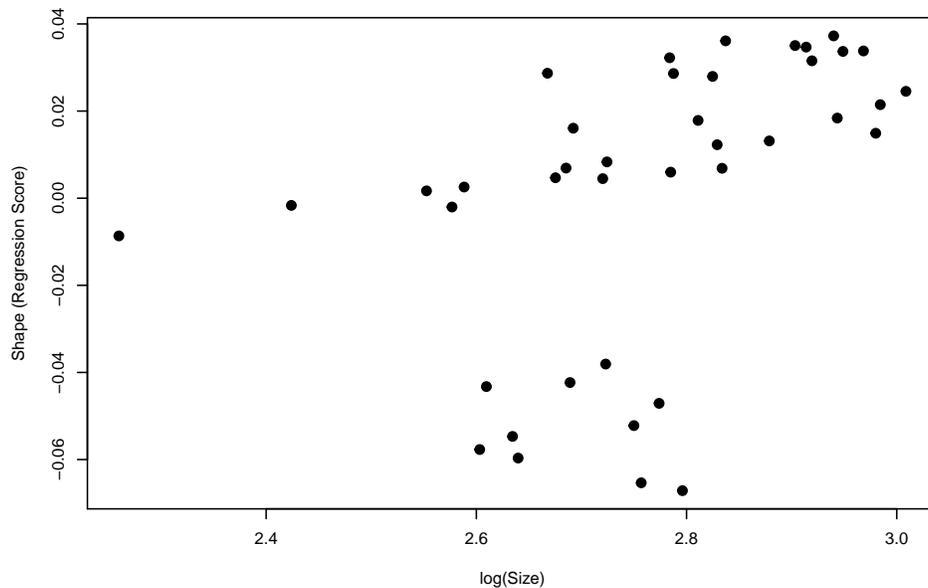


Next we do an allometry example, using RegScore or PredLine via `procD.allometry`

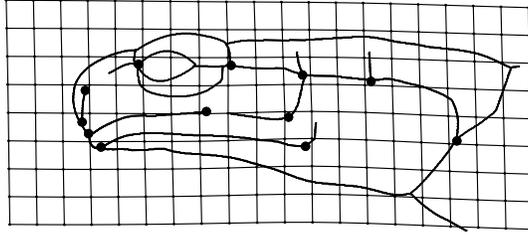
```
gdf <- geomorph.data.frame(Y.gpa)
plethAllometry <- procD.allometry(coords~Csize, data=gdf, logsz = TRUE, print.progress = FALSE)
```

```
##
## Allometry Model
```

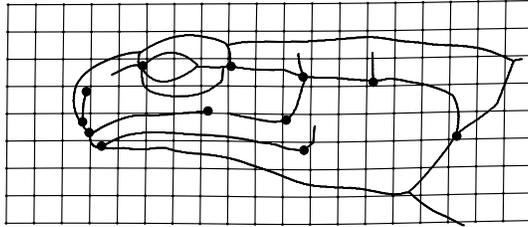
```
plot(plethAllometry, method="RegScore", warpgrids = FALSE)
```



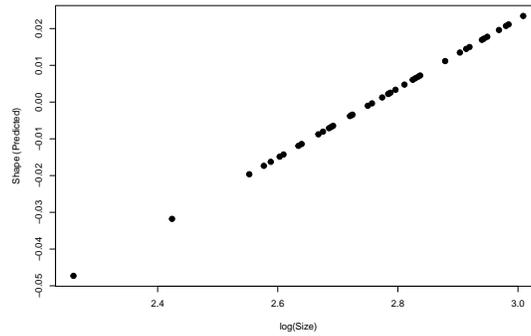
```
preds <- shape.predictor(plethAllometry$Ahat, x= plethAllometry$Reg.proj, Intercept = FALSE,
                        predmin = min(plethAllometry$Reg.proj),
                        predmax = max(plethAllometry$Reg.proj))
plotRefToTarget(M, preds$predmin, outline = plethodon$outline)
```



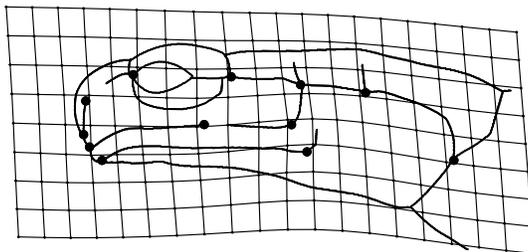
```
plotRefToTarget(M, preds$predmax, outline = plethodon$outline)
```



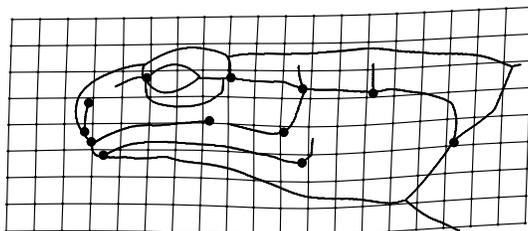
```
plot(plethAllometry, method="PredLine", warpgrids = FALSE)
```



```
preds <- shape.predictor(plethAllometry$Ahat, x= plethAllometry$pred.val, Intercept = FALSE,
  predmin = min(plethAllometry$pred.val),
  predmax = max(plethAllometry$pred.val))
plotRefToTarget(M, preds$predmin, outline = plethodon$outline)
```



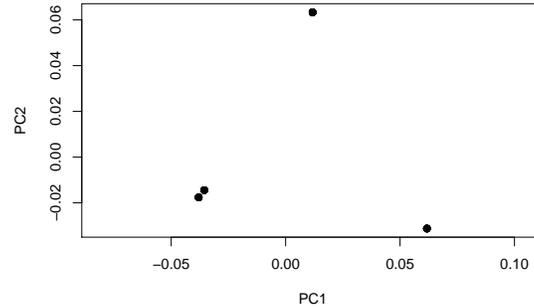
```
plotRefToTarget(M, preds$predmax, outline = plethodon$outline)
```



12.1.3 Example 3: Shapes from factors

Here the mean shapes of the four groups (species*site) are plotted via `procD.lm` and `prcomp`

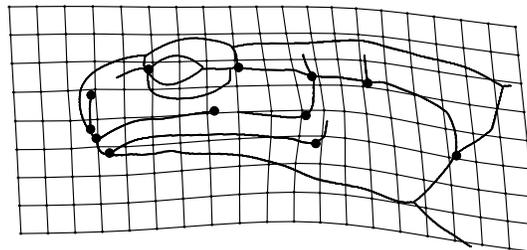
```
gdf <- geomorph.data.frame(Y.gpa, species = plethodon$species, site = plethodon$site)
pleth <- procD.lm(coords ~ species*site, data=gdf, print.progress = FALSE)
PCA <- prcomp(pleth$fitted) # PCA using prcomp
plot(PCA$x, asp=1, pch=19) # plots position of group means
```



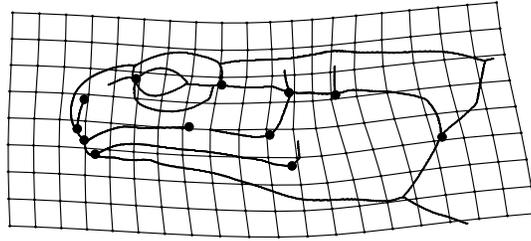
```
means <- unique(round(PCA$x,3))
means # note: suggests 3 PCs useful enough
```

```
##      PC1   PC2   PC3 PC4 PC5 PC6 PC7 PC8 PC9 PC10 PC11 PC12 PC13 PC14
## [1,] 0.012 0.063 0.000  0  0  0  0  0  0  0  0  0  0  0
## [2,] 0.062 -0.031 0.000  0  0  0  0  0  0  0  0  0  0  0
## [3,] -0.038 -0.018 -0.012  0  0  0  0  0  0  0  0  0  0  0
## [4,] -0.036 -0.014 0.012  0  0  0  0  0  0  0  0  0  0  0
##      PC15 PC16 PC17 PC18 PC19 PC20 PC21 PC22 PC23 PC24
## [1,]  0  0  0  0  0  0  0  0  0  0
## [2,]  0  0  0  0  0  0  0  0  0  0
## [3,]  0  0  0  0  0  0  0  0  0  0
## [4,]  0  0  0  0  0  0  0  0  0  0
```

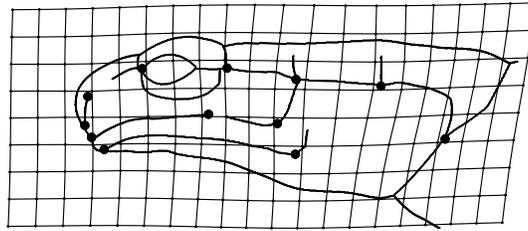
```
preds <- shape.predictor(arrayspecs(pleth$fitted, 12,2), x= PCA$x[,1:3],
  Intercept = FALSE,
  pred1 = means[1,1:3],
  pred2 = means[2,1:3],
  pred3 = means[3,1:3],
  pred4 = means[4,1:3])
plotRefToTarget(M, preds$pred1, outline = plethodon$outline)
```



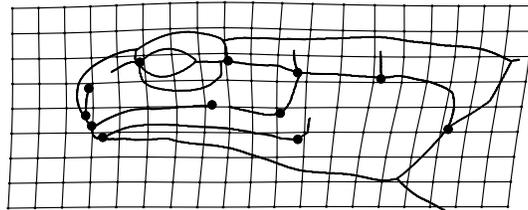
```
plotRefToTarget(M, preds$pred2, outline = plethodon$outline)
```



```
plotRefToTarget(M, preds$pred3, outline = plethodon$outline)
```



```
plotRefToTarget(M, preds$pred4, outline = plethodon$outline)
```



We can also get the mean shapes of the four factors using the model matrix from `procD.lm`

```
X <- pleth$X # The model matrix from `procD.lm`
X # includes intercept; remove for better functioning
```

##	(Intercept)	speciesTeyah	siteSymp	speciesTeyah:siteSymp
## 1	1	0	1	0
## 2	1	0	1	0
## 3	1	0	1	0
## 4	1	0	1	0
## 5	1	0	1	0
## 6	1	0	1	0
## 7	1	0	1	0
## 8	1	0	1	0
## 9	1	0	1	0
## 10	1	0	1	0
## 11	1	1	1	1
## 12	1	1	1	1
## 13	1	1	1	1
## 14	1	1	1	1
## 15	1	1	1	1
## 16	1	1	1	1
## 17	1	1	1	1
## 18	1	1	1	1
## 19	1	1	1	1
## 20	1	1	1	1
## 21	1	0	0	0
## 22	1	0	0	0

```

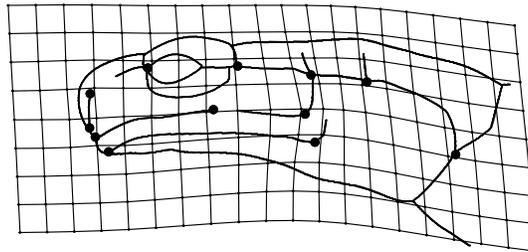
## 23      1      0      0      0
## 24      1      0      0      0
## 25      1      0      0      0
## 26      1      0      0      0
## 27      1      0      0      0
## 28      1      0      0      0
## 29      1      0      0      0
## 30      1      0      0      0
## 31      1      1      0      0
## 32      1      1      0      0
## 33      1      1      0      0
## 34      1      1      0      0
## 35      1      1      0      0
## 36      1      1      0      0
## 37      1      1      0      0
## 38      1      1      0      0
## 39      1      1      0      0
## 40      1      1      0      0

```

```

X <- X[,-1]
symJord <- c(0,1,0) # design for P. Jordani in sympatry
alloJord <- c(0,0,0) # design for P. Jordani in allopatry
preds <- shape.predictor(arrayspecs(pleth$fitted, 12,2), x = X, Intercept = TRUE,
                          symJord=symJord, alloJord=alloJord)
plotRefToTarget(M, preds$symJord, outline = plethodon$outline)

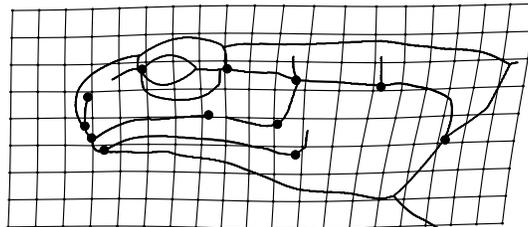
```



```

plotRefToTarget(M, preds$alloJord, outline = plethodon$outline)

```



12.1.4 Example 4: Shapes from Partial Least Squares Analysis (two.b.pls)

We shall use a different example dataset to show how

```

data(plethShapeFood)
Y.gpa <- gpagen(plethShapeFood$land, print.progress = FALSE) #GPA-alignment
# 2B-PLS between head shape and food use data
PLS <- two.b.pls(plethShapeFood$food, Y.gpa$coords, iter=999)

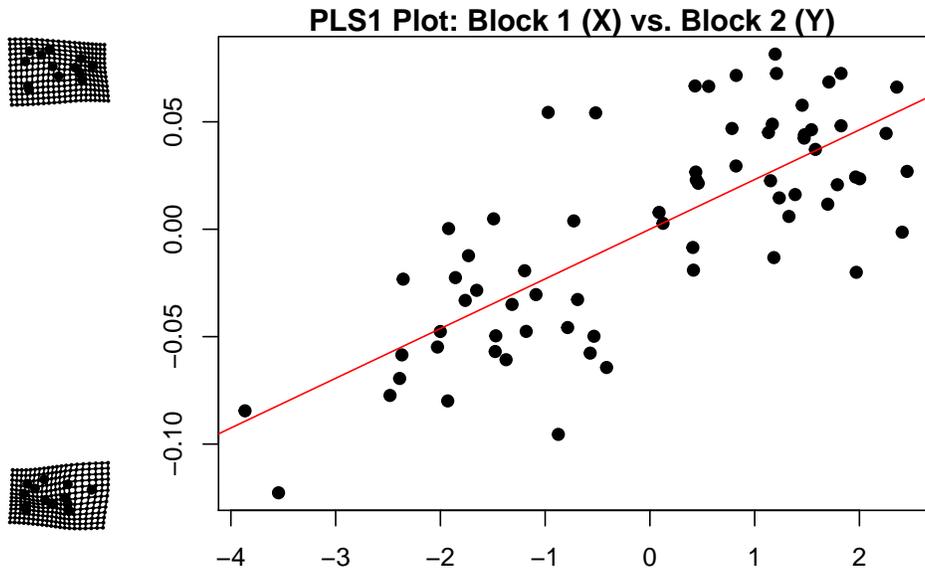
```

```
##
|
|                                     | 0%
|=====                             | 10%
|=====                             | 20%
|=====                             | 30%
|=====                             | 40%
|=====                             | 50%
|=====                             | 60%
|=====                             | 70%
|=====                             | 80%
|=====                             | 90%
|=====                             | 100%
```

```
summary(PLS)
```

```
##
## Call:
## two.b.pls(A1 = plethShapeFood$food, A2 = Y.gpa$coords, iter = 999)
##
##
##
## r-PLS: 0.759
##
## P-value: 0.001
##
## Based on 1000 random permutations
```

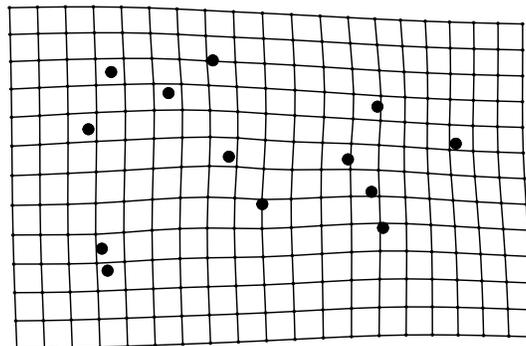
```
plot(PLS)
```



```

preds <- shape.predictor(Y.gpa$coords, plethShapeFood$food, Intercept = FALSE,
  method = "PLS",
  pred1 = 2, pred2 = -4, pred3 = 2.5) # using PLS plot as a guide
M <- mshape(Y.gpa$coords)
plotRefToTarget(M, preds$pred1) # shape at 2 on food axis

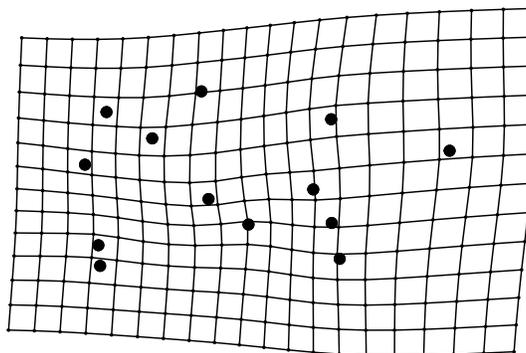
```



```

plotRefToTarget(M, preds$pred2) # shape at -4 on food axis

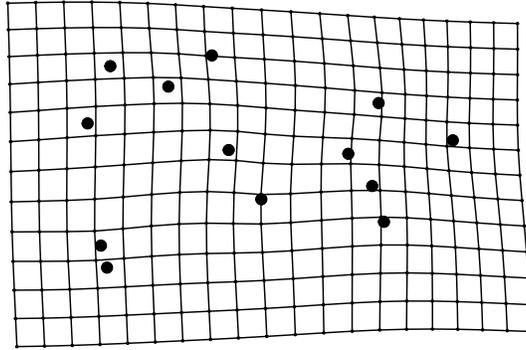
```



```

plotRefToTarget(M, preds$pred3) # shape at 2.5 on food axis

```



This can also be used with an PLS analysis in geomorph.

12.2 Plot shape differences between a reference and target specimen (plotRefToTarget)

Function plots shape differences between a reference and target specimen.

Function

```
plotRefToTarget(M1, M2, mesh = NULL, outline = NULL, method = c("TPS",
  "vector", "points", "surface"), mag = 1, links = NULL, label = FALSE,
  gridPars = NULL, useRefPts = FALSE, ...)
```

Arguments

- *M1* Matrix of landmark coordinates for the first (reference) specimen
- *M2* Matrix of landmark coordinates for the second (target) specimen
- *mesh* A mesh3d object for use with method="surface"
- *outline* An x,y curve or curves warped to the reference (2D only)
- *method* Method used to visualize shape difference; see below for details
- *mag* The desired magnification to be used when visualizing the shape difference (e.g., mag=2)
- *links* An optional matrix defining for links between landmarks
- *label* A logical value indicating whether landmark numbers will be plotted
- *gridPars* An optional object made by `gridPar`
- *useRefPts* An option (logical value) to use reference configuration points rather than target configuration points (when method = "TPS") – NOT RECOMMENDED FOR NOVICE USERS
- ... Additional parameters (not covered by `gridPar`) to be passed to `plot`, `plot3d` or `shade3d`.

The option `mag` allows the user to indicate the degree of magnification to be used when displaying the shape difference. The function will plot either two- or three-dimensional data. This function combines numerous plotting functions found in Claude (2008).

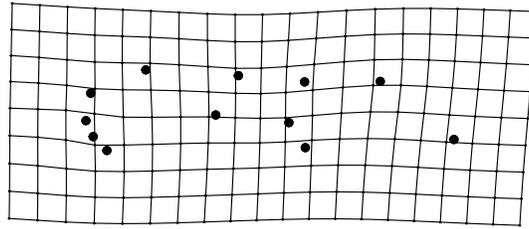
12.2.1 The four plotting methods

A 2D data example

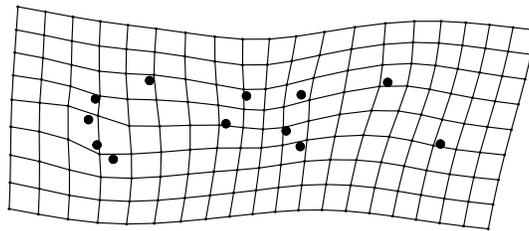
```
data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
ref<-mshape(Y.gpa$coords)
```

1. **TPS** a thin-plate spline deformation grid is generated. For 3D data, this method will generate thin-plate spline deformations in the x-y and x-z planes.

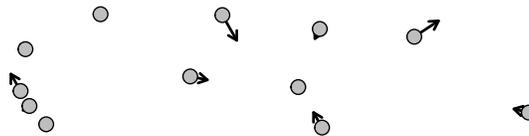
```
plotRefToTarget(ref,Y.gpa$coords[, ,39], method="TPS")
```



```
# magnify difference by 3X  
plotRefToTarget(ref,Y.gpa$coords[, ,39],mag=3, method="TPS")
```

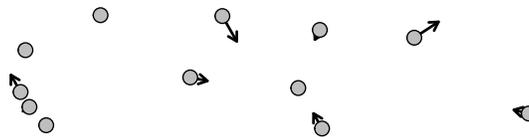


```
# vector a plot showing the vector displacements between corresponding landmarks  
# in the reference and target specimen is shown.  
plotRefToTarget(ref,Y.gpa$coords[, ,39],method="vector", mag=3)
```



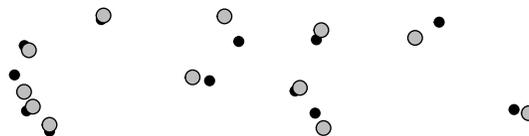
2. **vector** a plot showing the vector displacements between corresponding landmarks in the reference and target specimen is shown (sometimes known as a “lollipop graph, sensu MorphoJ”)

```
plotRefToTarget(ref,Y.gpa$coords[, ,39],method="vector",mag=3)
```



3. **points** a plot is displayed with the landmarks in the target (black) overlaying those of the reference (gray). Additionally, if a matrix of links is provided, the landmarks of the mean shape will be connected by lines. The link matrix is an M x 2 matrix, where M is the desired number of links. Each row of the link matrix designates the two landmarks to be connected by that link.

```
plotRefToTarget(ref,Y.gpa$coords[, ,39],method="points", mag=3)
```



In these first 3 methods, it is possible to add a wireframe or an outline to these plots. For more information on creating and importing an outline, see `warpRefOutline` (helper functions end of this chapter). Using **links** (wireframe) for representation of your specimen. In TPS, vector & points in both 2D and 3D it is possible to add links to these plots. The links argument can be made by hand, using matrix, or there is a graphical assisted function `define.links`.

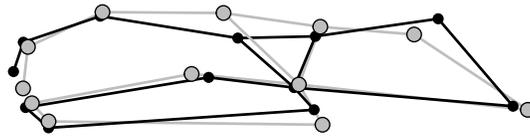
In the plethodon list, there is a matrix called `plethodon$links`

```

      [,1] [,2]
[1,]    4    5
[2,]    3    5
[3,]    2    4
[4,]    1    2
[5,]    1    3
[6,]    6    7
[7,]    7    8
[8,]    8    9
[9,]    9   10
[10,]   10   11
[11,]   11   12
[12,]   12    1
[13,]    1    9
[14,]    1   10

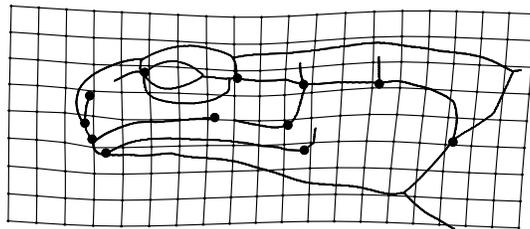
```

```
plotRefToTarget(ref,Y.gpa$coords[, ,39],method="points", mag=3, links = plethodon$links)
```

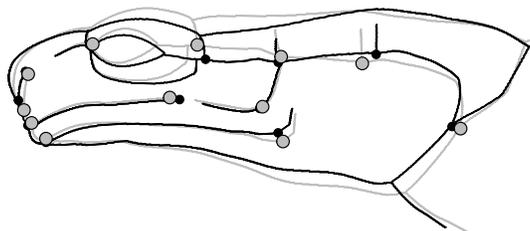


In the plethodon list, there is a matrix called `plethodon$outline` to be added using the **outline** option. This requires first warping a reference outline to the mean shape using the function `warpRefOutline` (see helper functions end of this chapter).

```
plotRefToTarget(ref,Y.gpa$coords[, ,39], method ="TPS",
               outline=plethodon$outline)
```



```
plotRefToTarget(ref,Y.gpa$coords[, ,39], mag=2, method ="points",
               outline=plethodon$outline)
```

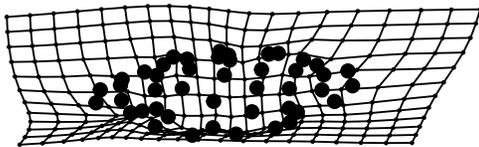


Now, a 3D data example using the three methods shown above

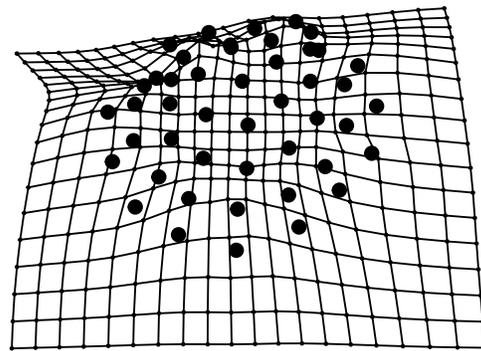
```
data(scallops)
Y.gpa<-gpagen(A=scallops$coorddata, curves=scallops$curvslide,
surfaces=scallops$surfslide, PrinAxes = FALSE, print.progress = FALSE)
ref<-mshape(Y.gpa$coords)
```

```
plotRefToTarget(ref,Y.gpa$coords[, ,1],method="TPS", mag=3)
```

X,Y tps grid



Y,Z tps grid



```
plotRefToTarget(ref,Y.gpa$coords[, ,1],method="vector", mag=3)
```

RGL device 6 [Focus]

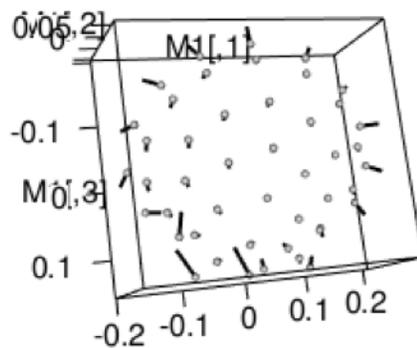
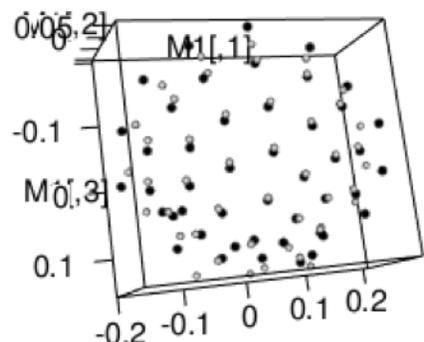


Figure 16: Scallop shell shape represented by method="vector"

```
plotRefToTarget(ref,Y.gpa$coords[, ,1],method="points", mag=3)
```

RGL device 6 [Focus]



4. **surface** a mesh3d surface is warped using thin-plate spline (for 3D data only). Requires mesh3d object in option mesh, made using `warpRefMesh` (see below), which provides a mesh3d object that is the shape of the sample mean. It is recommended that the mean shape is used as the reference for warping (see Rohlf 1998). A 3D data example using the average mesh made with `warpRefMesh` (Note: this example is not included in the *geomorph* package, but show here for illustrative purposes).

```
ref <- mshape(Y.gpa$coords) # calculate the mean shape from set of GPA-aligned specimens
plotRefToTarget(M1=ref, M2=Y.gpa$coords[, ,1], mesh=averagemesh, method="surface")
# averagemesh is a mesh made with warpRefMesh,
# that is the shape of the mean of a set of specimens
```

RGL device 4 [Focus]



Function plots the warped “target” shape (shown here against the mean for illustrative purposes only). This function can be used to show deformations between *any two sets of coordinates*. Coordinate data are provided by several functions, a few examples given below.

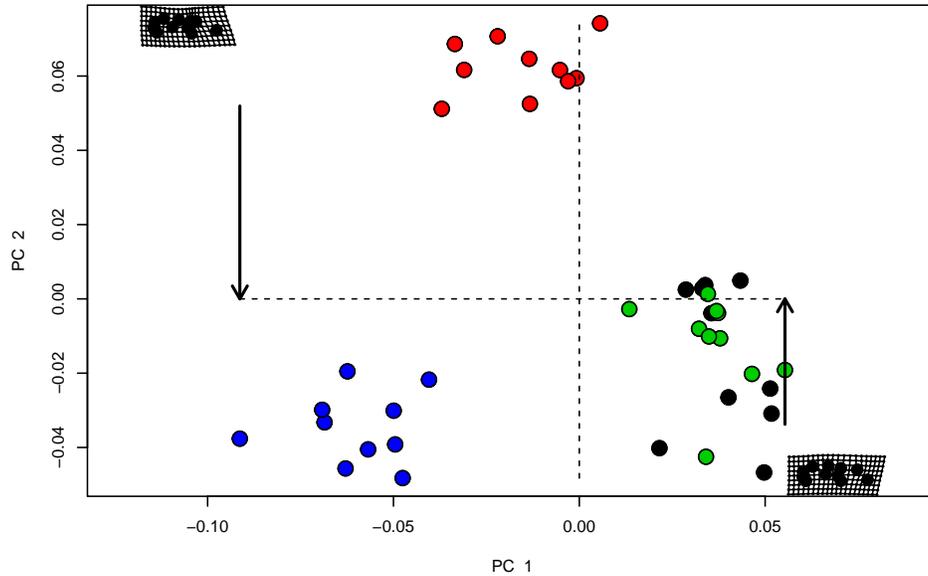
12.2.1.1 Example 1 `plotTangentSpace`

The function returns a list containing shape coordinates (`$pc.shapes`) for the minimum and maximum shape of all PCs. To use, enter the coordinate matrix into position M2, e.g.,

```

data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
res <- plotTangentSpace(Y.gpa$coords,
                       groups = factor(paste(plethodon$species, plethodon$site)))

```



```

layout(mat=matrix(c(1,2), ncol=2)) # set up two part layout for plotting
ref<-mshape(Y.gpa$coords) # calculate mean shape
# shape change along PC1 in the negative direction
plotRefToTarget(M1=ref, M2=res$pc.shapes$PC1min, method="TPS")
# shape change along PC1 in the positive direction
plotRefToTarget(M1=ref, M2=res$pc.shapes$PC1max, method="TPS")

```



```

layout(1) # reset to 1 graph per plot

```

12.2.1.2 Example 2 procD.Allometry

plot of a `procD.allometry` object returns a list containing the shapes at the min and max size, which can be plotted as follows:

```

data(ratland) # example dataset
Y.gpa<-gpagen(ratland, print.progress = FALSE) # GPA-alignment
gdf <- geomorph.data.frame(Y.gpa)
# perform the Multivariate Regression
Allom <- procD.allometry(coords ~ Csize, data=gdf,
                        print.progress = FALSE)

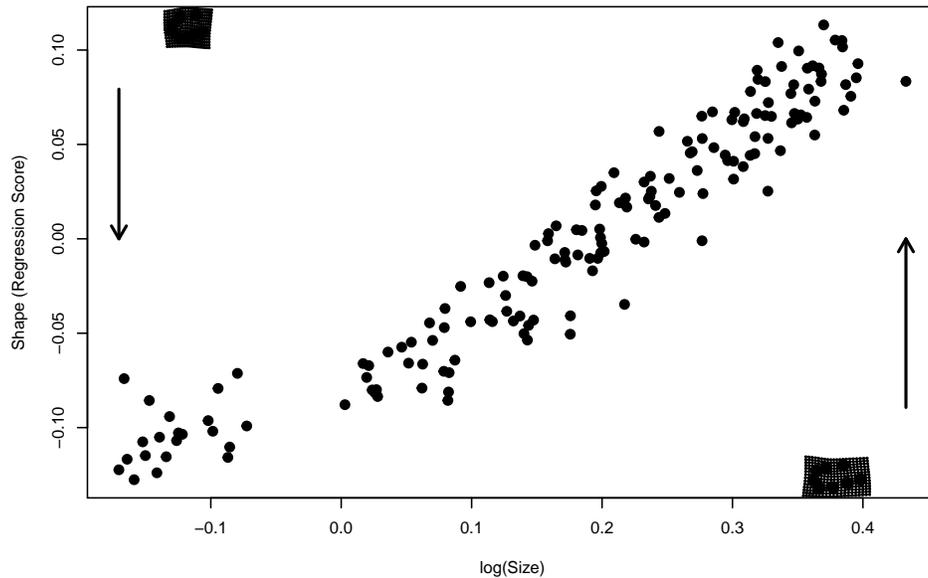
```

```

##
## Allometry Model

```

```
# plot the Multivariate Regression and save the min/max shapes
res <- plot(Allom, method="RegScore", shapes=TRUE)
```



```
layout(mat=matrix(c(1,2), ncol=2)) # set up two part layout for plotting
ref<-mshape(Y.gpa$coords) # calculate mean shape
# Predicted shape at min centroid size
plotRefToTarget(ref, res$min.shape, method="vector", mag=2, axes=F, main = "minCsize")
# Predicted shape at max centroid size
plotRefToTarget(ref, res$max.shape, method="vector", mag=2, axes=F, main = "maxCsize")
```

minCsize

maxCsize

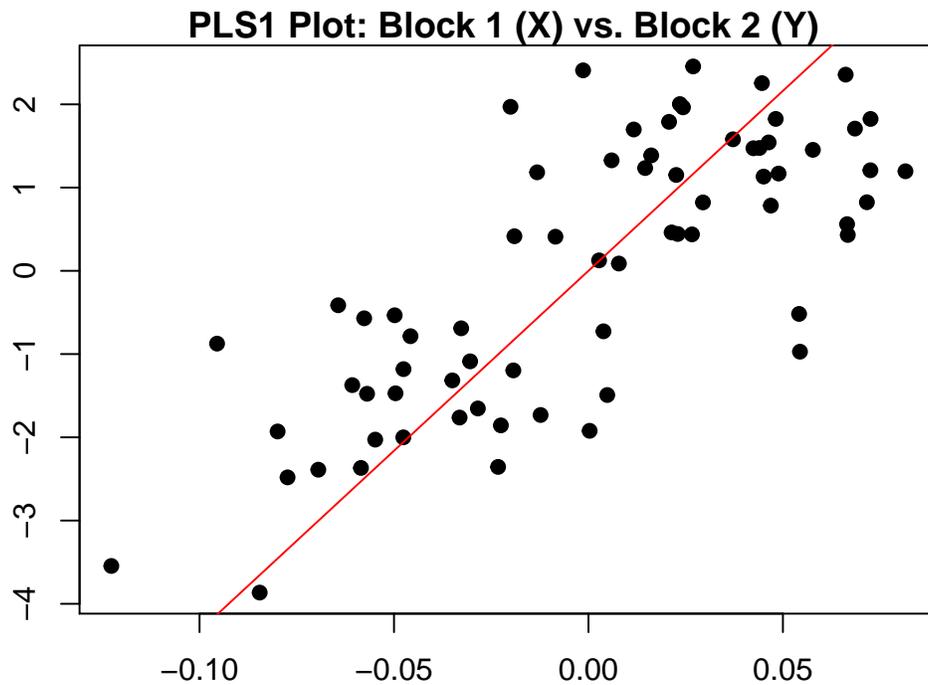


```
layout(1) # reset to 1 graph per plot
```

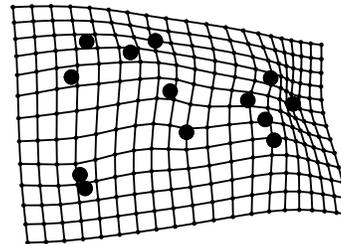
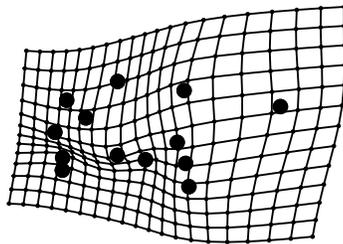
12.2.1.3 Example 3 two.b.pls, integration.test & phylo.integration

plot of a pls object returns a list containing the shapes at the min and max of PLS1 axis for one or both blocks (only if the data were inputted as 3D arrays), which can be plotted as follows:

```
data(plethShapeFood) # example dataset
Y.gpa<-gpagen(plethShapeFood$land, print.progress = FALSE) # GPA-alignment
PLS <-two.b.pls(Y.gpa$coords,plethShapeFood$food,iter=999, print.progress = FALSE)
res <- plot(PLS, shapes=TRUE) # PLS plot
```



```
layout(mat=matrix(c(1,2), ncol=2)) # set up two part layout for plotting
ref<-mshape(Y.gpa$coords) # calculate mean shape
# Predicted shape at min of PLS1 axis 1
plotRefToTarget(ref, res$pls1.min, method="TPS", mag=2, axes=F, main = "min of PLS1")
# Predicted shape at max of PLS1 axis 1
plotRefToTarget(ref, res$pls1.max, method="TPS", mag=2, axes=F, main = "max of PLS1")
```



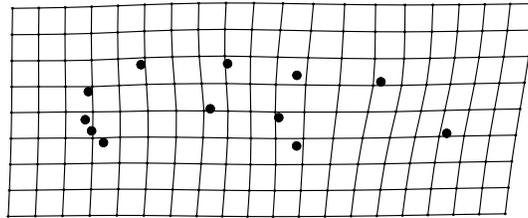
```
layout(1) # reset to 1 graph per plot
```

12.2.1.4 Example 4, differences between groups

for groups shown to be significantly different with `procD.lm`, first average the data by groups, calculate group means and plot those means:

```
data(plethodon)
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
# calculate mean shape for each group
means <- aggregate(two.d.array(Y.gpa$coords) ~ plethodon$site, FUN=mean)
# make mean vectors as matrix
```

```
Allo.mn <- matrix(as.numeric(means[1,-1]), ncol=2, byrow=T)
ref<-mshape(Y.gpa$coords) # calculate mean shape
plotRefToTarget(ref, Allo.mn, method="TPS") # shape of group
```



13 Data inspection

13.1 Plot landmark coordinates for all specimens (plotAllSpecimens)

Function plots landmark coordinates for a set of specimens.

Function

```
plotAllSpecimens(A, mean = TRUE, links = NULL, label = FALSE, plot.param = list())
```

Arguments

- *A* A 3D array (p x k x n) containing GPA-aligned coordinates for a set of specimens
- *mean* A logical value indicating whether the mean shape should be included in the plot
- *links* An optional matrix defining for links between landmarks
- *pointscale* An optional value defining the size of the points for all specimens
- *meansize* An optional value defining the size of the points representing the average specimen

The function creates a plot of the landmark coordinates for all specimens. This is useful for examining patterns of shape variation after GPA. If “mean=TRUE”, the mean shape will be calculated and added to the plot. Additionally, if a matrix of links is provided, the landmarks of the mean shape will be connected by lines. The link matrix is an m x 2 matrix, where m is the desired number of links. Each row of the link matrix designates the two landmarks to be connected by that link. The function will plot either two- or three-dimensional data.

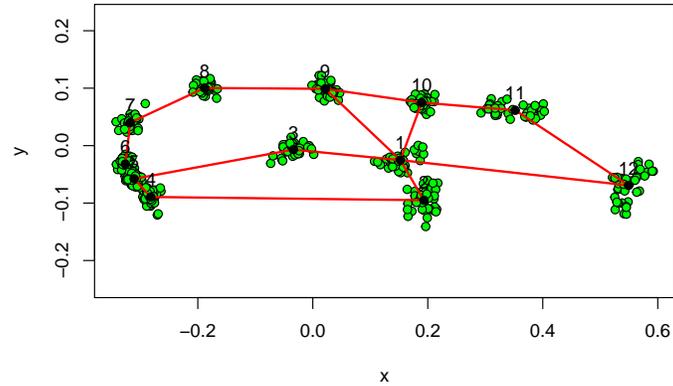
Example for 2D data data(plethodon)

```
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA
plethodon$links # look at links matrix
```

```
      [,1] [,2]
[1,]    4    5
[2,]    3    5
[3,]    2    4
[4,]    1    2
[5,]    1    3
[6,]    6    7
[7,]    7    8
[8,]    8    9
[9,]    9   10
[10,]   10   11
[11,]   11   12
[12,]   12    1
```

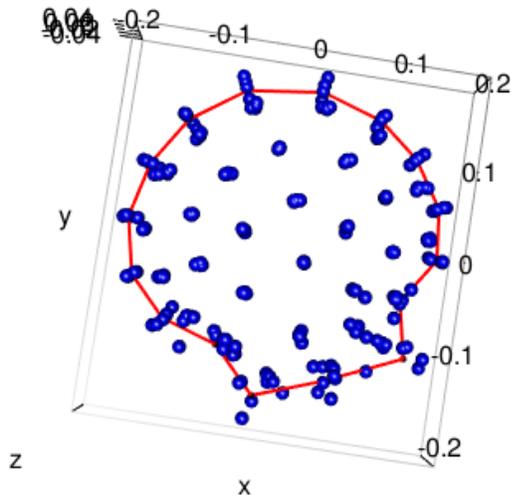
```
[13,] 1 9
[14,] 1 10
```

```
plotAllSpecimens(Y.gpa$coords,links=plethodon$links,label=T,
  plot.param = list(pt.bg = "green", mean.cex=1, link.col="red",
    txt.pos=3, txt.cex=1))
```



Example for 3D data

```
data(scallops)
Y.gpa <- gpagen(A=scallops$cooorddata, curves=scallops$curvslide,
  surfaces=scallops$surfslide, print.progress = FALSE)
scallinks <- matrix(c(1,rep(2:16, each=2),1), nrow=16, byrow=TRUE) # make links matrix
plotAllSpecimens(Y.gpa$coords,links=scallinks,
  plot.param=list(pt.bg = "blue",link.col="red"))
```



Protip! Typing the name of the function in the console brings up the function code. The user can look at this code for ideas of how to customize their own graphs.

13.2 Find potential outliers (plotOutliers)

Function plots a set of Procrustes-aligned specimens ordered by their distance from the mean shape. It is used as a tool to aid identifying specimens that have been digitized wrong (for example, mixing up landmark

order). Specimens falling outside of the upper quartile range are potential outliers. We do not, however, stipulate that they must be removed or are wrong.

Function

```
plotOutliers(A, groups = NULL)
```

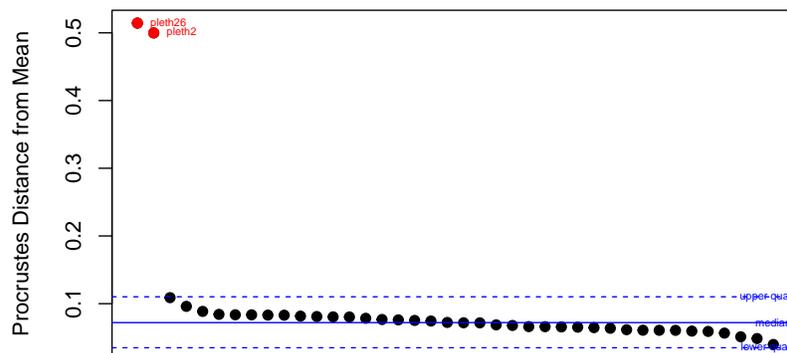
Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of aligned specimens
- *groups* An optional factor defining groups

The function creates a plot of all specimens ordered by their Procrustes distance from the mean shape. In the graph, the median (unbroken line) and upper and lower quartiles (dashed lines) summarize the distances from the mean shape. Specimens falling above the upper quartile are plotted in red and their address returned, for inspection by `plotRefToTarget`.

```
data(plethodon)
# let's make some outliers
newland <- plethodon$land # make copy of example dataset
# dataset needs dimnames for plotting reference so we shall assign arbitrary names
dimnames(newland)[[3]] <- paste("pleth", 1:dim(newland)[3], sep="")
newland[c(1,8),,2] <- newland[c(8,1),,2] # swap lmks 1 and 8 of specimen 2
newland[c(3,11),,26] <- newland[c(11,3),,2] # swap lmks 3 and 11 of specimen 26
Y <- gpagen(newland, print.progress = FALSE) # perform GPA
plotOutliers(Y$coords)
```

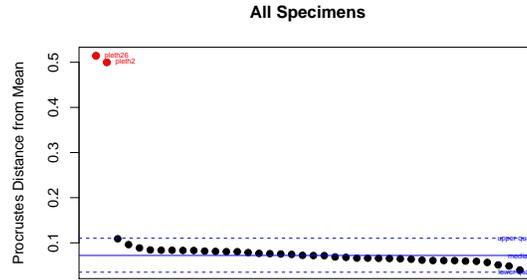
All Specimens



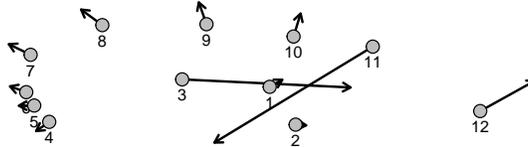
```
## pleth26 pleth2 pleth14 pleth27 pleth4 pleth5 pleth3 pleth12 pleth9
##      26      2      14      27      4      5      3      12      9
## pleth6 pleth10 pleth11 pleth19 pleth13 pleth1 pleth29 pleth20 pleth22
##      6      10      11      19      13      1      29      20      22
## pleth7 pleth15 pleth16 pleth31 pleth37 pleth34 pleth21 pleth40 pleth38
##      7      15      16      31      37      34      21      40      38
## pleth18 pleth8 pleth17 pleth30 pleth32 pleth36 pleth24 pleth33 pleth35
##      18      8      17      30      32      36      24      33      35
## pleth23 pleth25 pleth28 pleth39
##      23      25      28      39
```

If there are outliers, then we can view them:

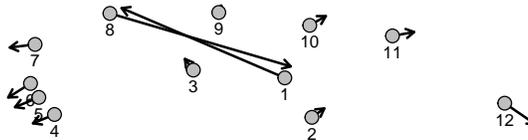
```
outliers <- plotOutliers(Y$coords) # function returns dimname and address of outliers
```



```
plotRefToTarget(mshape(Y$coords), Y$coords[, , outliers[1]], method="vector", label = T)
```



```
plotRefToTarget(mshape(Y$coords), Y$coords[, , outliers[2]], method="vector", label = T)
```



The way that the arrows cross over each other in these two plots are classic examples of when two landmarks have been digitized in the wrong order (switched). In this case, either go back to the original specimen and redigitize. Or in *R*, you can switch the landmarks as we did above.

13.3 Plot 3D specimen, fixed landmarks and surface semilandmarks (plotspec)

A function to plot three-dimensional (3D) specimen along with its landmarks.

Function

```
plotspec(spec, digitspec, fixed = NULL, ptsize = 1, centered = FALSE, ...)
```

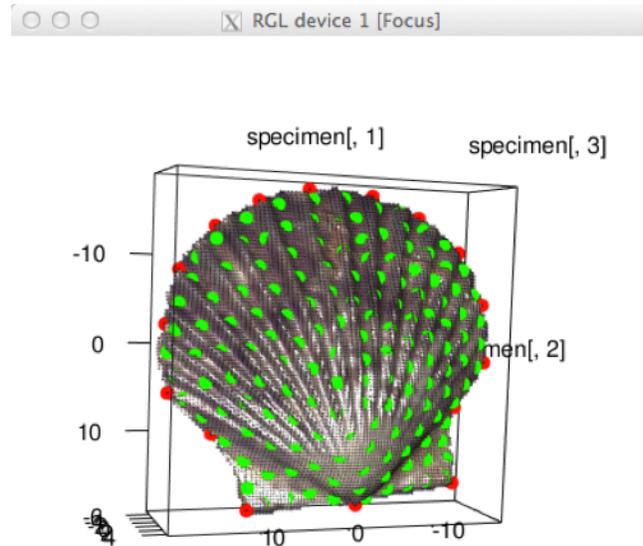
Arguments

- *spec* An object of class shape3d/mesh3d, or matrix of 3D vertex coordinates.
- *digitspec* Name of data matrix containing 3D fixed and/or surface sliding coordinates.
- *fixed* Numeric The number of fixed template landmarks (listed first in digitspec)
- *ptsizes* Numeric Size to plot the mesh points (vertices), e.g., 0.1 for dense meshes, 3 for sparse meshes
- *centered* Logical Whether the data matrix is in the surface mesh coordinate system (*centered*=FALSE) or if the data were collected after the mesh was centered (*centered*=TRUE)- see below
- ... additional parameters which will be passed to plot3d.

Function to plot 3D specimens along with their digitized “fixed” landmarks and semilandmarks “surface sliders” and “curve sliders”. If specimen is a 3D surface (class shape3d/mesh3d) mesh is plotted. For visualization purposes, 3D coordinate data collected using `digit.fixed` or `digit.surface` and `buildtemplate` prior to build 1.1-6 were centered by default. Therefore use this function with *centered*=TRUE. Data collected outside *geomorph* should be read using *centered*=FALSE. The function assumes the fixed landmarks are listed at the beginning of the coordinate matrix (*digitspec*).

13.3.1 Example

```
data(scallopPLY) #load example dataset
plotspec(spec=scallopPLY$ply, digitspec=scallopPLY$coords, fixed=16, centered =TRUE)
```



☒ The fixed landmarks and curve semilandmarks are in red, and the surface semilandmarks are in green.

14 Plots for analytical functions

Many of *geomorph*'s analytical functions described in Vigentte 3 return information in the form of a list that can be used in the S3 generic function plot.

14.1 plot on objects from `gpagen`

Function calls `plotAllSpecimens` and plots landmark coordinates for a set of specimens.

14.2 plot on objects from `bilat.symmetry`

Function plots the symmetric and asymmetric shape component, and the mean directional and fluctuating asymmetry as shape change graphs.

14.2.1 Example

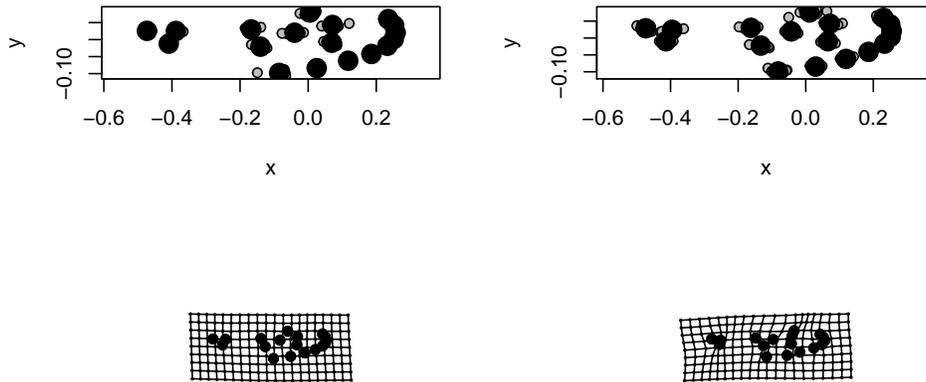
```
data(mosquito) #example dataset
gdf <- geomorph.data.frame(wingshape = mosquito$wingshape,
                           ind=mosquito$ind, side=mosquito$side,
                           replicate=mosquito$replicate) # make geomorph data frame
mosquito.sym <- bilat.symmetry(A = wingshape, ind = ind, side = side,
```

```

replicate = replicate, object.sym = FALSE, RRPP = TRUE, iter = 499,
data = gdf, print.progress = FALSE)
plot(mosquito.sym, warpgrids = TRUE)

```

Symmetric Shape Component (left) and Asymmetric Shape Component (right)



Mean directional (left) and fluctuating (right) asymmetry

14.3 plot on objects from procD.allometry

Function

```

plot(x, method = c("CAC", "RegScore", "PredLine"),
warpgrids = TRUE, label = NULL, gp.label = FALSE, pt.col = NULL,
mesh = NULL, shapes = TRUE, ...)

```

Arguments

- *x* plot object (from procD.allometry)
- *method* Method for estimating allometric shape components
- *warpgrids* A logical value indicating whether deformation grids for small and large shapes should be displayed (note: if groups are provided no TPS grids are shown)
- *label* An optional vector indicating labels for each specimen that are to be displayed
- *gp.label* A logical value indicating labels for each group to be displayed (if group was originally included); “PredLine” only
- *pt.col* An optional vector of colours to use for points (as in points(bg=))
- *mesh* A mesh3d object to be warped to represent shape deformation of the minimum and maximum size if warpgrids=TRUE (see warpRefMesh).
- *shapes* Logical argument whether to return the the shape coordinates shape coordinates of the small and large shapes
- ... other arguments passed to plot

14.3.1 Three plotting methods

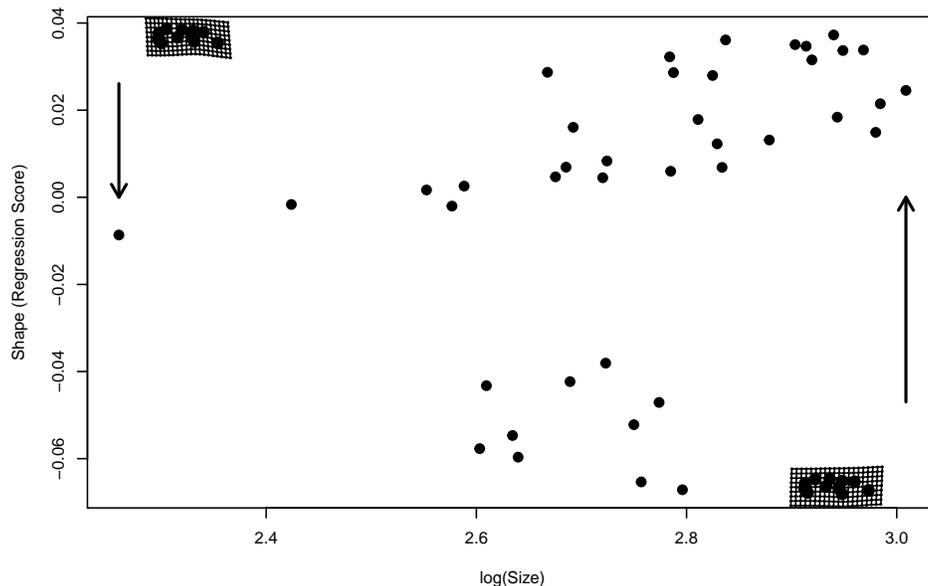
1. If “method=CAC” (the default) the function calculates the common allometric component of the shape data, which is an estimate of the average allometric trend within groups (Mitteroecker et al. 2004). The function also calculates the residual shape component (RSC) for the data.
2. If “method=RegScore” the function calculates shape scores from the regression of shape on size, and plots these versus size (Drake and Klingenberg 2008). For a single group, these shape scores are mathematically identical to the CAC (Adams et al. 2013).
3. If “method=PredLine” the function calculates predicted values from a regression of shape on size, and plots the first principal component of the predicted values versus size as a stylized graphic of the allometric trend (Adams and Nistri 2010).

14.3.2 Example

Comparing allometric slopes between groups (Homogeneity of Slopes Test)

```
data(plethodon) # example dataset
Y.gpa <- gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
# make geomorph data frame
gdf <- geomorph.data.frame(shape = Y.gpa$coords, cs = Y.gpa$Csize,
                           site = plethodon$site, species = plethodon$species)
# simple allometry
plot(procD.allometry(shape~cs, logsz = TRUE, data=gdf, iter=499,
                     RRPP=TRUE, print.progress = FALSE),
     method = "RegScore", warpgrids = T)
```

Allometry Model

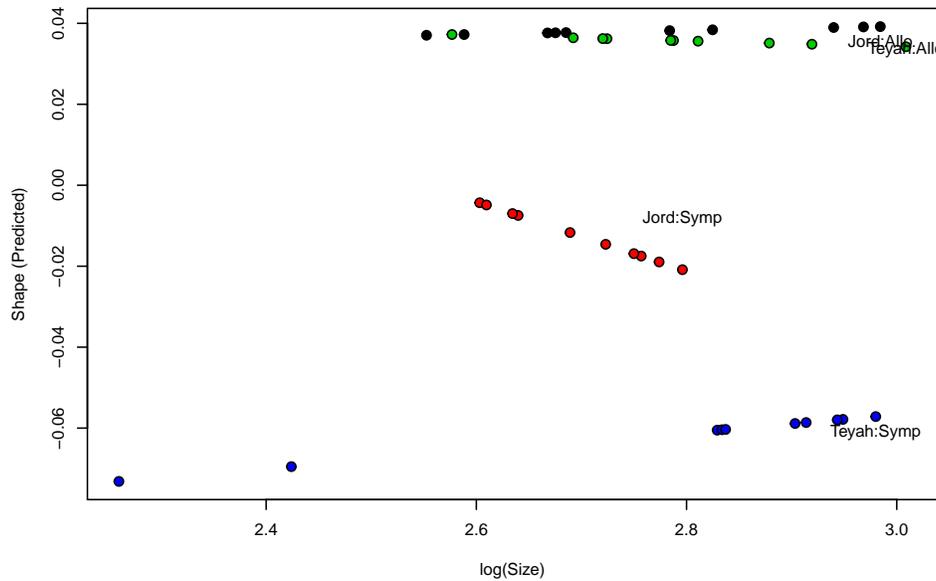


```
# Comparing allometric slopes between groups
plethAllometry <- procD.allometry(shape~cs, ~species*site,
logsz = TRUE, data=gdf, iter=499, RRPP=TRUE, print.progress = FALSE)
```

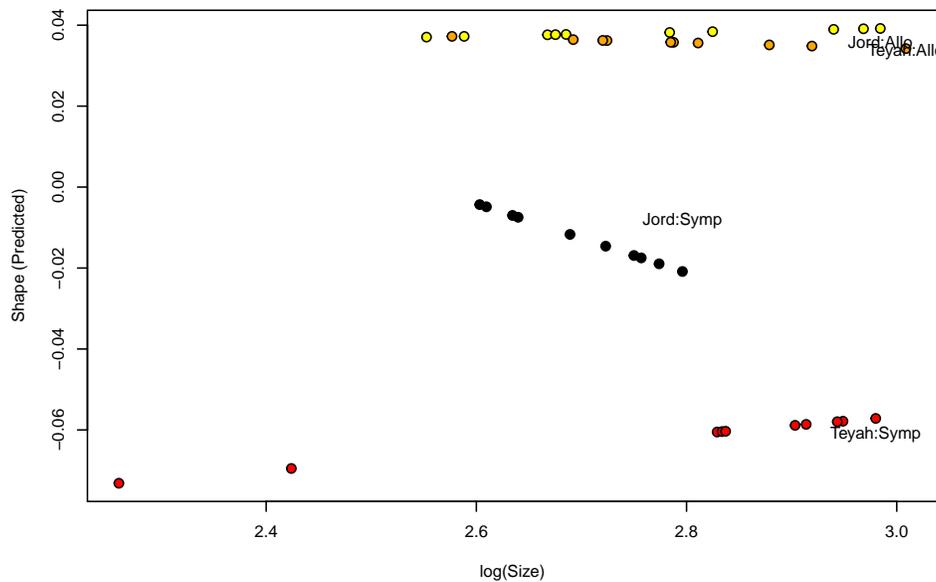
Homogeneity of Slopes Test

Allometry Model

```
plot(plethAllometry, method = "PredLine")
```



```
col.gp<-c(rep("black",10),rep("red",10),  
          rep("yellow",10),rep("orange",10)) # not a factor  
plot(plethAllometry, method = "PredLine", pt.col = col.gp) # change colour of points
```



14.4 plot on objects from advanced.procD.lm, procD.lm, procD.pgls and procD.allometry

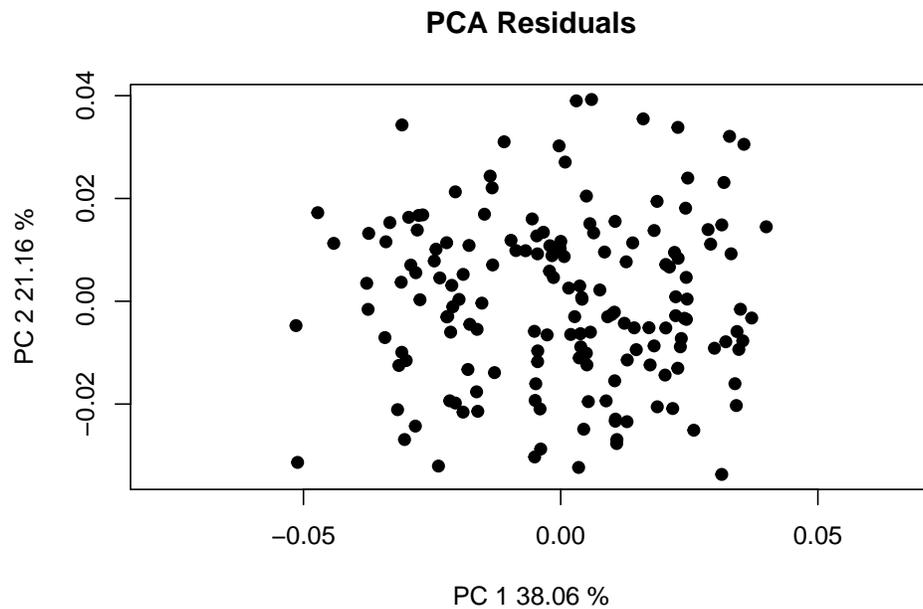
Plots the following graphs:

- *PCA Residuals* A PCA of the residuals (observed responses - fitted responses).
- *Q-Q plot* Evaluate the normality of a variable using a Q-Q plot.

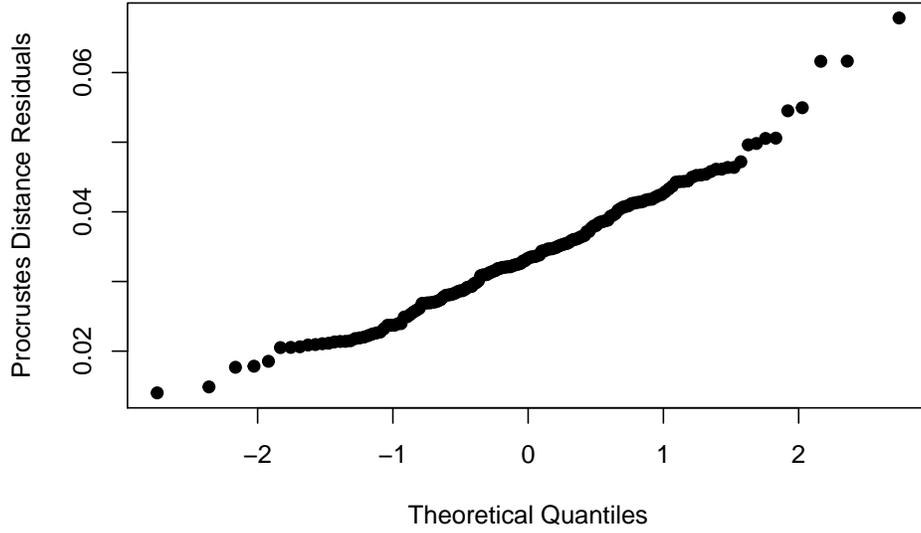
- *Residuals vs. PC1 fitted* Residuals plotted against the first PC axis of fitted values, in order to verify the assumption that the residuals are randomly distributed and have constant variance.
- *Residuals vs. Fitted* Procrustes distance of fitted values vs. Procrustes distance of residuals, alternative to the above plot.
- *Outliers* (if `outliers= TRUE`) calls `plotOutliers`, to identify potential outliers.

14.4.1 Example

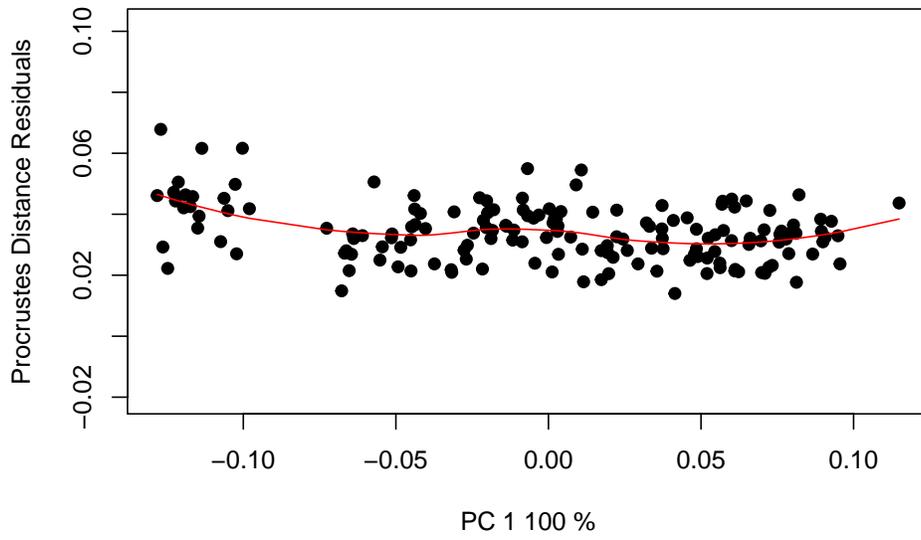
```
data(ratland) # example dataset
rat.gpa<-gpagen(ratland, print.progress = FALSE) # GPA-alignment
gdf <- geomorph.data.frame(rat.gpa) # make geomorph data frame
rat.anova <- procD.lm(coords ~ Csize, data = gdf, iter = 999,
                      RRPP = TRUE, print.progress = FALSE)
plot(rat.anova) # diagnostic plots
```

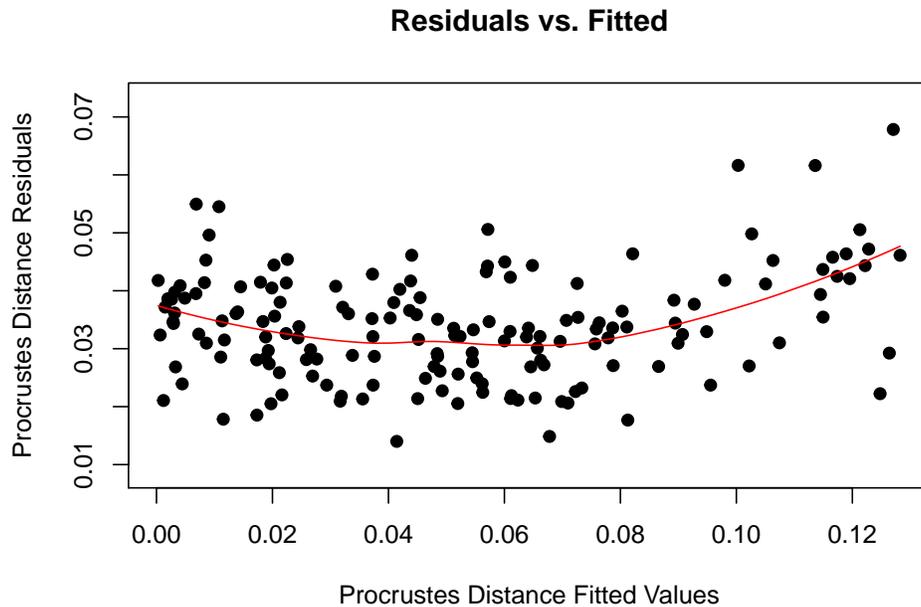


Q-Q plot



Residuals vs. PC 1 fitted





14.5 plot on objects from `phylo.modularity` and `modularity.test`

Plots a histogram of the Covariance ratio (CR, the estimate of the observed modular signal) sampling distribution. The arrow points to the observed CR, and the grey bars are the permuted values.

14.6 plot on objects from `compare.evol.rates` and `compare.multi.evol.rates`

Plots a histogram of the ratio of maximum to minimum evolutionary rates sampling distribution. The arrow points to the observed ratio, and the grey bars are the permuted values.

14.7 plot on objects from `physignal`

Plots a histogram of the ratio of K (estimate of phylogenetic signal) sampling distribution. The arrow points to the observed K, and the grey bars are the permuted values.

14.8 plot on objects from `two.b.pls`, `integration.test` and `phylo.integration`

Function

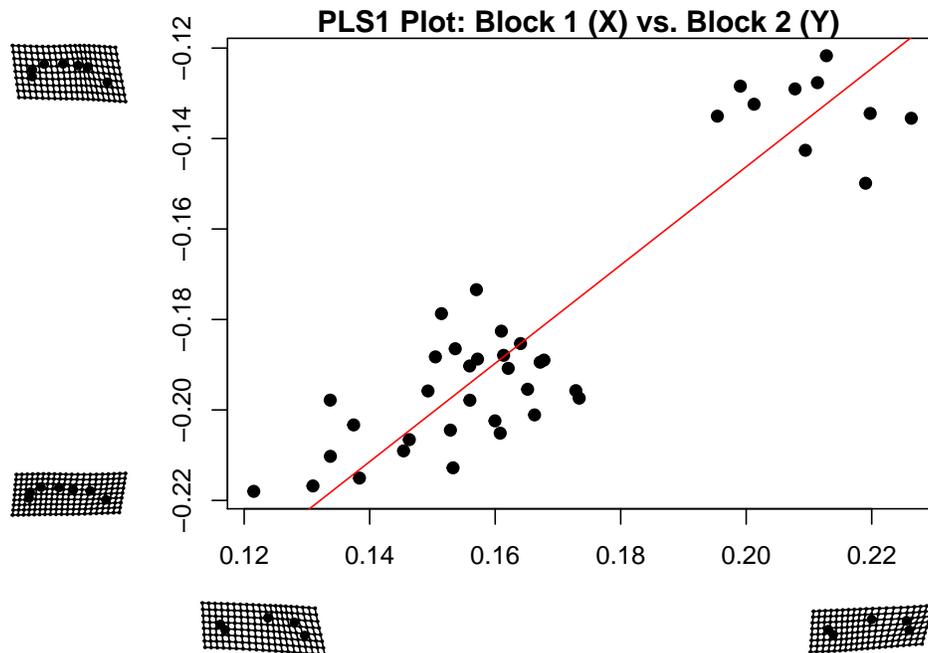
```
plot(x, label = NULL, warpgrids = TRUE, shapes = TRUE, ...)
```

Arguments

- *x* plot object (from `integration.test`, `phylo.integration` or `two.b.pls`)
- *label* An optional vector indicating labels for each specimen that are to be displayed
- *warpgrids* A logical value indicating whether deformation grids for extreme ends of axis1 and axis2 should be displayed (if data were originally input as 3D array)
- *shapes* Logical argument whether to return the the shape coordinates shape coordinates of the extreme ends of axis1 and axis2
- ... other arguments passed to plot

14.8.1 Example

```
data(plethodon) # example dataset
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) # GPA-alignment
#landmarks on the skull and mandible assigned to partitions
land.gps<-c("A","A","A","A","A","B","B","B","B","B","B","B")
IT <- integration.test(Y.gpa$coords, partition.gp=land.gps, iter=999,
  print.progress = FALSE)
plot(IT) # PLS plot
```



14.9 plot on objects from trajectory.analysis

Function

```
plot(x, group.cols = NULL, pt.seq.pattern = c("white", "gray", "black"), pt.scale = 1,
...)
```

Arguments

- *x* plot object (from trajectory.analysis)
- *group.cols* An optional vector of colors for group levels
- *pt.seq.pattern* The sequence of colors for starting, middle, and end points of trajectories, respectively. e.g., c("green", "gray", "red") for gray points but initial points with green color and end points with red color.
- *pt.scale* An optional value to magnify or reduce points (1 = no change)
- ... other arguments passed to plot

14.9.1 Example

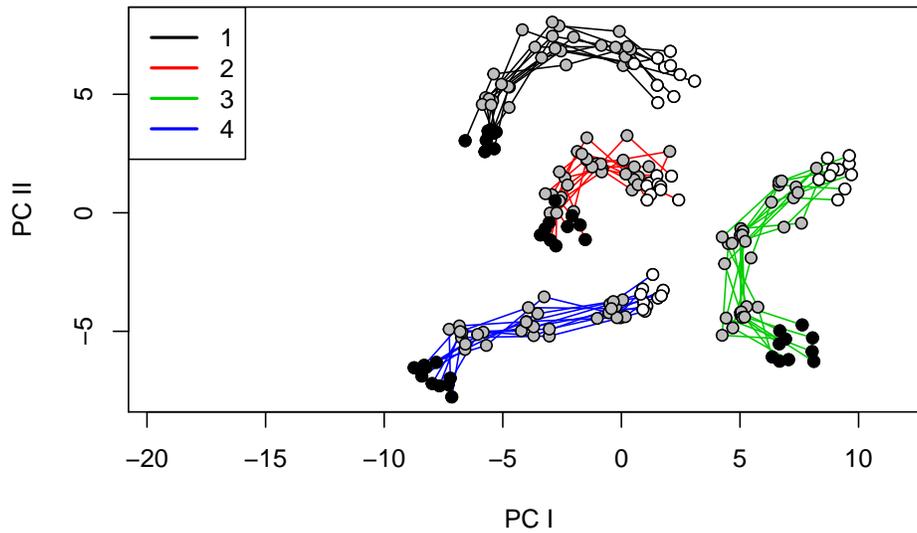
```
# Motion paths represented by 5 time points per motion
data(motionpaths) #example dataset
gdf <- geomorph.data.frame(trajectories = motionpaths$trajectories,
```

```

groups = motionpaths$groups)
TA <- trajectory.analysis(f1 = trajectories ~ groups, traj.pts = 5,
                        data=gdf, iter=199, print.progress = FALSE)
plot(TA)

```

Two Dimensional View of Phenotypic Trajectories

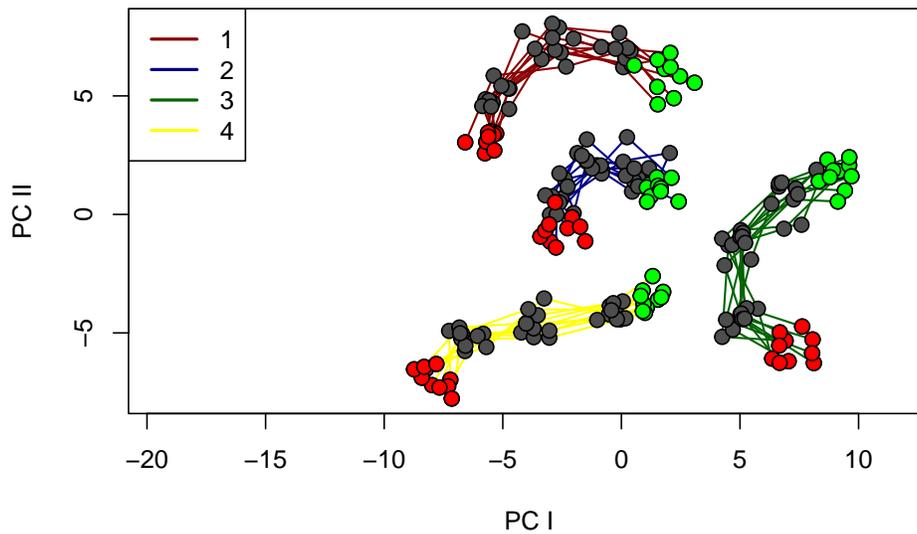


```

plot(TA, group.cols = c("dark red", "dark blue", "dark green", "yellow"),
     pt.seq.pattern = c("green", "gray30", "red"), pt.scale = 1.3)

```

Two Dimensional View of Phenotypic Trajectories



14.10 Helper functions

14.10.1 Set up parameters for grids, points, and links in `plotRefToTarget` (`gridPar`)

Function allows users to vary certain plotting parameters to produce different graphical outcomes for `plotRefToTarget`. Not all parameters need to be adjusted to use this function, as the defaults above will be used.

Function

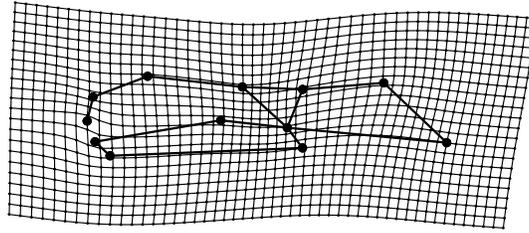
```
gridPar(pt.bg = "gray", pt.size = 1.5, link.col = "gray", link.lwd = 2,  
  link.lty = 1, out.col = "gray", out.cex = 0.1, tar.pt.bg = "black",  
  tar.pt.size = 1, tar.link.col = "black", tar.link.lwd = 2,  
  tar.link.lty = 1, tar.out.col = "black", tar.out.cex = 0.1,  
  n.col.cell = 20, grid.col = "black", grid.lwd = 1, grid.lty = 1,  
  txt.adj = 0.5, txt.pos = 1, txt.cex = 0.8, txt.col = "black")
```

Arguments

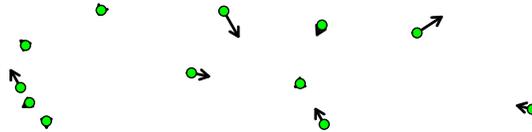
- *pt.bg* Background color of reference configuration points (single value or vector of values)
- *pt.size* Scale factor for reference configuration points (single value or vector of values)
- *link.col* The color of links for reference configurations (single value or vector of values)
- *link.lwd* The line weight of links for reference configurations (single value or vector of values)
- *link.lty* The line type of links for reference configurations (single value or vector of values)
- *out.col* The color of outline for reference configurations (single value or vector of values)
- *out.cex* The size of plotting symbol of outline for reference configurations (single value or vector of values)
- *tar.pt.bg* Background color of target configuration points (single value or vector of values)
- *tar.pt.size* Scale factor for target configuration points (single value or vector of values)
- *tar.link.col* The color of links for target configurations (single value or vector of values)
- *tar.link.lwd* The line weight of links for target configurations (single value or vector of values)
- *tar.link.lty* The line type of links for target configurations (single value or vector of values)
- *tar.out.col* The color of outline for target configurations (single value or vector of values)
- *tar.out.cex* The size of plotting symbol of outline for target configurations (single value or vector of values)
- *n.col.cell* The number of square cells (along x axis) for grids (single numerical value)
- *grid.col* The color of grid lines (single value)
- *grid.lwd* Scale factor for the weight of grid lines (single numerical value)
- *grid.lty* The line type for grid lines (single numerical value, as in base *R* plot)
- *txt.adj* The adjustment value of the landmark label (one or two values, as in base *R* text)
- *txt.pos* The position of the landmark label (single numerical value, as in base *R* text)
- *txt.cex* The size of the landmark label text (single numerical value, as in base *R* text)
- *txt.col* The color of the landmark label text (single numerical value, as in base *R* text)

Examples

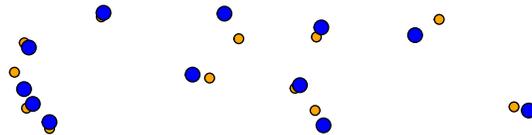
```
data(plethodon) # example dataset  
Y.gpa<-gpagen(plethodon$land, print.progress = FALSE) #GPA-alignment  
ref<-mshape(Y.gpa$coords)  
  
# Altering points and links  
GP1 <- gridPar(pt.bg = "red", pt.size = 1, link.col="blue", link.lwd=2, n.col.cell=50)  
plotRefToTarget(ref,Y.gpa$coords[, ,39], gridPars=GP1, mag=2,  
links=plethodon$links, method="TPS")
```



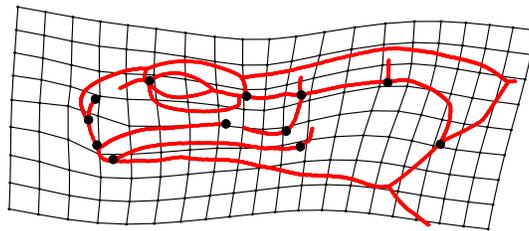
```
# Altering point color
GP2 <- gridPar(pt.bg = "green", pt.size = 1)
plotRefToTarget(ref,Y.gpa$coords[, ,39], gridPars=GP2, mag=3, method="vector")
```



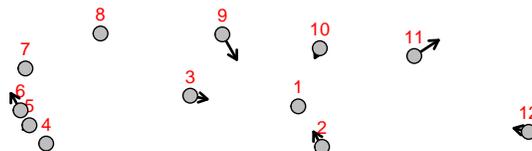
```
# Altering ref and target points
GP3 <- gridPar(pt.bg = "blue", pt.size = 1.5, tar.pt.bg = "orange", tar.pt.size = 1)
plotRefToTarget(ref,Y.gpa$coords[, ,39], gridPars=GP3, mag=3, method="points")
```



```
# Altering outline color
GP4 <- gridPar(tar.out.col = "red", tar.out.cex = 0.3)
plotRefToTarget(ref,Y.gpa$coords[, ,39], gridPars=GP4, mag=3,
outline=plethodon$outline, method="TPS")
```



```
# Altering text labels
GP5 <- gridPar(txt.pos = 3, txt.col = "red")
plotRefToTarget(ref,Y.gpa$coords[, ,39], gridPars=GP5, mag=3, method="vector", label=TRUE)
```



14.10.2 Find the mean specimen (findMeanSpec)

Function

findMeanSpec(A)

Arguments

A function to identify which specimen lies closest to the estimated mean shape for a set of aligned specimens. A is a 3D array ($p \times k \times n$) containing landmark coordinates for a set of aligned specimens. This function is used to facilitate finding a specimen to use with `warpRefMesh` or `warpRefOutline`.

```
findMeanSpec(Y.gpa$coords) # GPA-aligned coordinates
specimen7          # returns the name of the specimen
25                # returns the specimen number (where it appears in the 3D array)
```

14.10.3 Create an outline object warped to the mean shape (`warpRefOutline`)

A function to take an outline (defined by many points) and use thin-plate spline method to warp the outline into the estimated mean shape for a set of aligned specimens. This outline is used `plotRefToTarget` where `outline=` option is available.

Function

```
warpRefOutline(file, mesh.coord, ref, color = NULL, centered = FALSE)
```

Arguments

- *file* A .txt or .csv file of the outline point coordinates, or a .TPS file with `OUTLINES=` or `CURVES=` elements
- *coord* A $p \times k$ matrix of 2D coordinates digitized on the ply file.
- *ref* A $p \times k$ matrix of 2D coordinates made by `mshape`

Function takes an outline (defined by many points) with a set of fixed landmark coordinates and uses the thin-plate spline method (Bookstein 1989) to warp the outline into the shape defined by a second set of landmark coordinates, usually those of the mean shape for a set of aligned specimens. It is highly recommended that the mean shape is used as the reference for warping (see Rohlf 1998). For *file*, it is necessary to have in the working directory a .txt, .csv, .TPS file containing coordinate data of the outline(s). An outline is made up of many points, each defined by an x and y coordinate.

14.10.3.1 To make an outline txt file in ImageJ (<http://imagej.nih.gov/ij/>):

- Import an 8-bit black and white drawing of an outline drawn from one of your digitized specimens
 - a. use `Image > Type > 8-bit` to convert if necessary
 - b. use `Image > Adjust > Threshold` to make B&W
- `Process > Binary > Skeletonize` the image to find a single (pixel width line around the image)
- `Analyze > Tools > Save XY Coordinates` to save the coordinates of all of the pixels (you may need to invert the y coordinate option). This text file will have three columns – the third is a column of RGB values, and needs to be deleted. NOTE: It is important that the coordinate system of these outline points matches that of the digitized landmarks. If this is not the case, digitize a fake specimen on this outline, and save those landmark coordinates to be used in the `coord` argument.

14.10.3.2 To make an outline TPS file in `tpsDIG2` (<http://life.bio.sunysb.edu/morph/soft-dataacq.html>):

- Import an image file using `File > Input source`
- `Using Modes > Outline mode`, highlight one outline on the image
- Right click on the outline, and choose `Save as XY cords`, and you will be prompted to the number of points you want the outline to be saved as (stay with the default all). Click OK
- Repeat for all outline segments
- To finish, `File > Save data as...` and save a .tps file. Once you have an outline, the workflow in *R* is as follows:
- Calculate the mean shape using `mshape`

- Choose an actual specimen to use for the warping. The specimen used as the template for this warping is recommended as one most similar in shape to the average of the sample, but can be any reasonable specimen – do this by eye, or use `findMeanSpec`
- Warp this specimen into the mean shape using `warpRefOutline`
- Use this average outline where it asks for a `outline=` in `plotRefToTarget`

14.10.3.3 Workflow

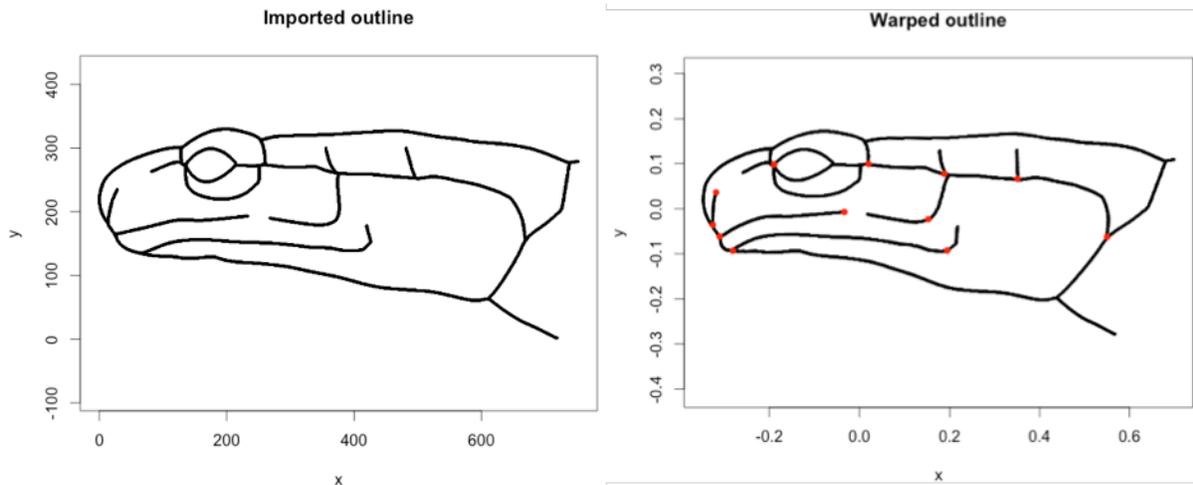
I. Calculate the mean shape using `mshape`

```
# mydata is 3D array of coordinate data
ref <- mshape(mydata)
```

II. Choose an actual specimen to use for the warping. The specimen used as the template for this outline warping is recommended as one most similar in shape to the average of the sample, but can be any reasonable specimen – do this by eye, or use `findMeanSpec`.

III. Warp this specimen's outline into the mean shape using `warpRefOutline`

```
# read in original coordinate data for a specimen chosen in step 2
sp7 <- readmulti.nts("specimen7.nts")
# run function to create mean shape outline
av.outline <- warpRefOutline("specimen7outline.txt", sp7, ref)
attributes(av.outline)
$names
[1] "outline" "npoints"
```



IV.

For further visualiations, use `av.outline$outline` where `outline=` is specified. `npoints` is a list of the number of points for each curve, and is currently not used in *geomorph* but will be in future versions.

14.10.4 Create a mesh3d object warped to the mean shape (`warpRefMesh`)

A function to take a `.ply` file and use thin-plate spline method to warp the file into the estimated mean shape for a set of aligned specimens. This mesh is used in functions where `mesh=` option is available.

Function

```
warpRefMesh(file, mesh.coord, ref, color = NULL, centered = FALSE)
```

Arguments

- *file* An ASCII ply file
- *mesh.coord* A p x k matrix of 3D coordinates digitized on the ply file.
- *ref* A p x k matrix of 3D coordinates made by `mshape`
- *color* Color to set the ply file \$material. If the ply already has color, use NULL. For ply files without color, `color=NULL` will be plotted as grey.
- *centered* Logical If the data in `mesh.coords` were collected from a centered mesh (see below).

Function takes a 3D surface mesh in the format of a .ply file and the digitized landmark coordinates uses the thin-plate spline method (Bookstein 1989) to warp the mesh into the shape defined by a second set of landmark coordinates, usually those of the mean shape for a set of aligned specimens. It is highly recommended that the mean shape is used as the reference for warping (see Rohlf 1998).

14.10.4.1 Workflow

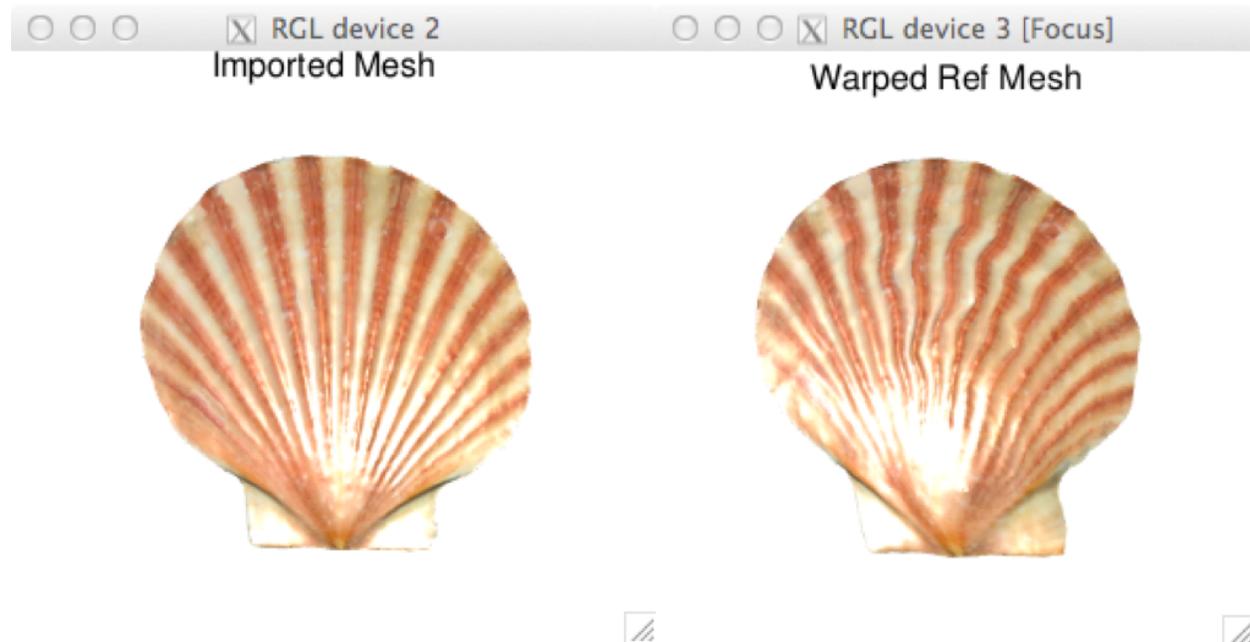
- I. Calculate the mean shape using `mshape`

```
# mydata is 3D array of coordinate data
ref <- mshape(mydata)
```

- II. Choose an actual specimen to use for the warping. The specimen used as the template for this warping is recommended as one most similar in shape to the average of the sample, but can be any reasonable specimen – do this by eye, or use `findMeanSpec`.

- III. Warp this specimen into the mean shape using `warpRefMesh`

```
# read in original coordinate data for a specimen chosen in step 2
sp7 <- readmulti.nts("specimen7.nts")
# read in surface mesh (.ply file) for a specimen chosen in step 2
sp7.ply <- read.ply("specimen7.ply")
# run function to create mean shape mesh
averagemesh <- warpRefMesh(sp7.ply, sp7, mshape(mydata))
```



- IV. Use this average mesh where it asks for a `mesh=` in the visualization functions For landmark coordinates digitized with `geomorph` digitizing functions `centered=TRUE` by default. This refers to the specimen being centered prior to landmark acquisition in the RGL window. For landmark data collected outside of `geomorph`, `centered=FALSE` will usually be the case. The returned `mesh3d` object is for use in `geomorph` functions

where shape deformations are plotted and `mesh=` option is available (`plotTangentSpace`, `plotRefToTarget`, and using `plot` on several analytical functions).

The “averagemesh” can also be saved to the working directory using the `rgl` function `writePLY`.

15 Digitizing

This chapter will cover *geomorph*'s digitizing functions, to collect 2D and 3D data for geometric morphometric analysis. The functions included in *geomorph* are offered as alternatives to other software for 2D landmark data collection such as `tpsDIG2`, and `ImageJ`, and 3D landmark data collection such as `IDAV` landmark.

15.1 2D data collection (`digitize2d`)

An interactive function to digitize two-dimensional landmarks from `.jpg` files. This function works very similarly to `TPSdig` software by J. Rohlf, allowing a list of files to be processed sequentially.

Function

```
digitize2d(filelist, nlandmarks, scale = NULL, tpsfile, verbose = TRUE)
```

Arguments

- *filelist* A list of names of jpeg images to be digitized.
- *nlandmarks* Number of landmarks to be digitized.
- *scale* An optional vector containing the length of the scale to be placed on each image.
- *tpsfile* The name of a TPS file to be created or read.
- *verbose* User decides whether to digitize in verbose or silent format (see details), default is verbose

This function may be used for digitizing 2D landmarks from jpeg images (`.jpg`). The user provides a list of image names, the number of landmarks to be digitized, and the name of an output TPS file. The list can be made manually, using `c` or `list.files` base functions e.g.,

```
filelist <- c("specimen1.jpg", "specimen1.jpg")  
# or  
filelist <- list.files(pattern = "*.jpg")
```

Digitizing landmarks from 2D photos requires that a scale bar is placed in the image in order to scale the coordinate data. The ‘scale’ option requires: a single number (e.g. 10) which means that the scale to be measured in all images is a 10mm scale bar; OR a vector the same length as the filelist containing a number for the scale of each image. If `scale=NULL`, then the digitized coordinates will not be scaled. This option is NOT recommended.

Landmarks to be digitized can include both fixed landmarks and semi-landmarks, the latter of which are to be designated as “sliders” for subsequent analysis (see the function `define.sliders`). Users may digitize all specimens in one session, or may return at a later time to complete digitizing. In the latter case, the user provides the same filelist and TPS file and the function will determine where the user left off. If specimens have missing landmarks, these can be incorporated during the digitizing process using the ‘a’ option as described below (`a = absent`).

15.1.1 Digitizing

Digitizing landmarks involves landmark selection using a mouse in the plot window, using the LEFT mouse button (or regular button for Mac users): Digitize the scale bar by selecting the two end points (single click for start and end). The user is asked whether the system should keep or discard the digitized scale bar.

```
digitize2d(filelist nlandmarks=11, scale=3, tpsfile = "salamaders.tps"),verbose = TRUE)
Only 1 scale measure provided. Will use scale = 3 for all specimens.
Digitizing specimen 1 in filelist
Set scale = 3
```



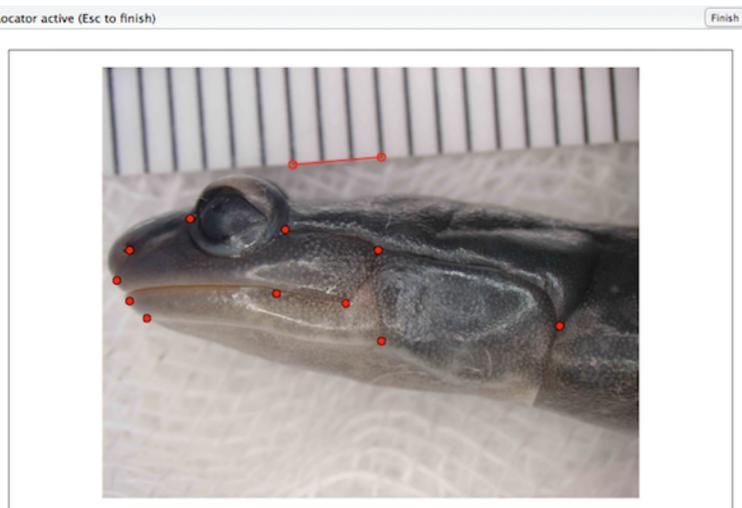
Figure 17: Digitizing window

```
Keep scale (y/n)?
```

```
y
```

Now, digitize each landmark with single click and the landmark is shown in red.

```
Select landmarks 1:11
```



If `verbose = TRUE`, digitizing is interactive between landmark selection using a mouse and the *R* console. Once a landmark is selected, the user is asked if the system should keep or discard the selection (y/n/a). If “y”, the user is asked to continue to select the next landmark. If “n”, the user is asked to select it again. To **digitize a missing landmark**, simply click on any location in the image. Then, when prompted to keep selection, choose ‘a’ (for absent). Missing landmarks can only be included during the digitizing process when `verbose=TRUE`. If `verbose = FALSE` (as in the example above) the digitizing of landmarks is continuous and uninterrupted. Here the user will not be

prompted to approve each landmark selection. At the end of digitizing, the landmark coordinates and the scale are written to a TPS file.

salamander.tps

```
LM = 11
0.913794820018337  0.728605670332276
1.04745503254995  0.58684483885935
0.654575013896415  0.765058455568172
0.168537877417812  0.671901337743106
0.103732925887331  0.736706289273586
0.0551292122394711  0.813662169216032
0.103732925887331  0.927070834394373
0.330550256244013  1.04452980904337
0.686977489661655  1.00402671433682
1.03530410413799  0.927070834394373
1.71575609520803  0.643549171448521
ID=mvz2066081
...
```

15.2 3D data collection: landmarks (`digit.fixed`)

An interactive function to digitize three-dimensional (3D) landmarks. Input for the function is either a matrix of vertex coordinates defining a 3D surface object or a mesh3d object as obtained from `read.ply` (see below).

Function

```
digit.fixed(spec, fixed, index = FALSE, ptsize = 1, center = TRUE)
```

Arguments

- *spec* An object of class `shape3d/mesh3d`, or matrix of 3D vertex coordinates
- *fixed* Numeric The number landmarks (fixed, and curve sliders if desired)
- *index* Logical Whether selected landmark addresses should be returned
- *ptsize* Numeric Size to plot the mesh points (vertices), e.g., 0.1 for dense meshes, 3 for sparse meshes
- *center* Logical Whether the object ‘spec’ should be centered prior to digitizing (default `center=TRUE`)

Function for digitizing “n” three-dimensional landmarks. The landmarks are “fixed” (traditional landmarks). They can be later designated as “curve sliders” (semilandmarks, that will “slide” along curves lacking known landmarks if required. A sliding semi-landmark (“sliders”) will slide between two designated points, along a line tangent to the specified curvature, and must be defined as “sliders” using function `define.sliders` or with similar format matrix made outside *R*. For 3D “surface sliders” (surface semilandmarks that slide over a surface) the function `digitSurface` should be used instead. NOTE: Function centers the mesh before digitizing by default (`center=TRUE`). If one chooses not to center, specimen may be difficult to manipulate in `rgl` window.

15.2.1 Digitizing

3D Digitizing functions in *geomorph* are interactive between landmark selection using a mouse (see below for instructions), and the R console. Once a point is selected, the user is asked if the system should keep or discard the selection (y/n). If “y”, the user is asked to continue to select the next landmark. If “n” the removes the last chosen landmark, and the user is asked to select it again. This can be repeated until the user is comfortable with the landmark chosen. To digitize with a standard 3-button (PC): * the RIGHT mouse button (primary) to select points to be digitized (click on a vertex to select) * the LEFT mouse button (secondary) is used to rotate mesh, * the mouse SCROLLER (third/middle) is used to zoom in and out. NOTE: Digitizing functions on MACINTOSH computers using a standard 3-button mice works as specified.

Macs using platform specific single button mice: * press button to rotate 3D mesh, * press button while pressing COMMAND key to select points to be digitized (click on a vertex to select) * press button while pressing OPTION key to adjust mesh perspective. * the mouse SCROLLER or trackpad two finger scroll is used to zoom in an out. XQuartz must be configured: go to Preferences > Input > tick “Emulate three button mouse”. NOTE: there is no pan (translate) functionality in rgl library for all platforms at this time. This is why the function has a center=TRUE/FALSE option.

Example reading in a mesh with 24700 vertices

```
mandible <- read.ply("Mandible.ply")
digit.fixed(mandible, fixed=20, index=F, ptsize=1, center=T)
Select Landmark 1 #
Keep landmark 1(y/n)? #See picture step 2
y
Select Landmark 2 #See picture step 2
Keep landmark 2(y/n)?
y
Select Landmark 3 #See picture step 1
```

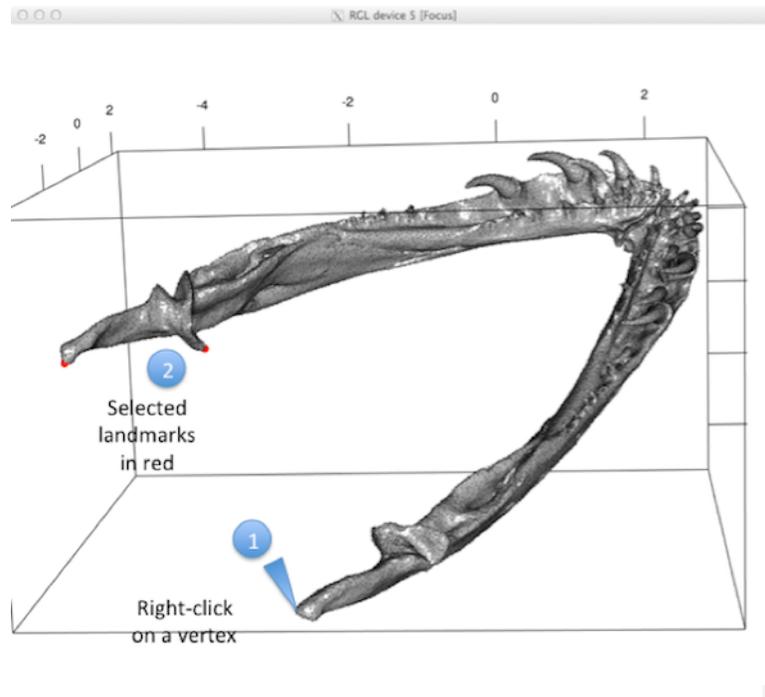


Figure 18: A mesh digitized using `digit.fixed`

Function writes to the working directory an NTS file with the name of the specimen and `.nts` suffix containing the landmark coordinates.

mandible.nts

```
"mandible
1 20 3 0 dim=3
13.2628089578878 10.2361171749175 -3.49646546534654
9.32980895788779 18.7309171749175 -1.94616546534654
-2.70691104211221 17.5351171749175 0.852234534653466
-13.7911610421122 15.6363171749175 0.945934534653464
-14.1681010421122 6.40941717491749 0.853034534653467
```

...

If `index=FALSE` (default) function returns to the console an $n \times 3$ matrix containing the x,y,z coordinates of the digitized landmarks. If `index=TRUE`, function returns a list containing a matrix containing the x,y,z coordinates of the digitized landmarks (*selected*) and a matrix of addresses for landmarks that are "fixed" (fix, primarily for internal use). YouTube video of this function in action here: <http://youtu.be/VK6bLbb4ipY>

15.3 Importing 3D surface files (`read.ply`)

A function to read ply files, which can be used for digitizing landmark coordinates or for shape warps. Other 3D file formats are currently not supported. Other files can be converted to ply using for example **Meshlab**.

Function

```
read.ply(file, ShowSpecimen = TRUE, addNormals = TRUE)
```

Arguments

- *file* An ASCII ply file
- *ShowSpecimen* A logical value indicating whether or not the ply file should be displayed
- *addNormals* A logical value indicating whether or not the normal of each vertex should be calculated (using `addNormals [rgl]`)

Function reads three-dimensional surface data in the form of a single ply file (Polygon File Format; ASCII format only, from 3D scanners such as NextEngine and David scanners). Vertices of the surface may then be used to digitize three-dimensional points, and semilandmarks on curves and surfaces. The function opens the ply file and plots the mesh, with faces rendered if file contains faces, and colored if the file contains vertex color.

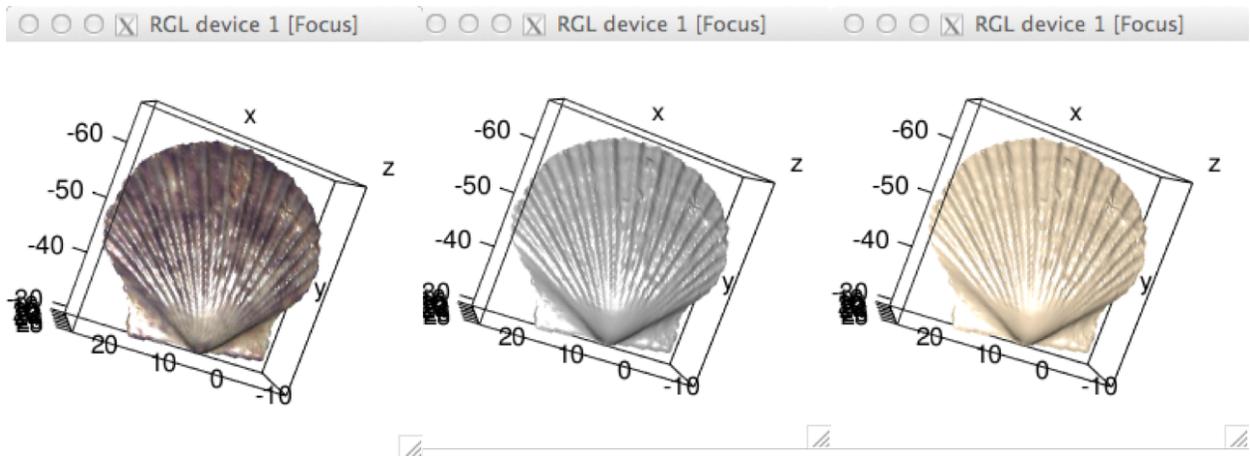
15.3.1 Example

reading a file called `myply.ply` in the working directory

```
read.ply("myply.ply", ShowSpecimen=TRUE)

# view an example in the geomorph package
data(scallopPLY)
myply <- scallopPLY$ply
attributes(myply)
$names
[1] "vb"          "it"          "primitivetype" "material"

$class
[1] "mesh3d" "shape3d"
plot3d(myply) # left image
# change color of mesh
myply$material <- "gray" # using color word (middle image)
myply$material <- "#FCE6C9" # using RGB code (right image)
```



3D data collection: landmarks and semilandmarks (`buildtemplate`) An interactive function to build template of three-dimensional surface sliding semilandmarks. Input for the function is either a matrix of vertex coordinates defining a 3D surface object or a mesh3d object as obtained from `read.ply`.

```
buildtemplate(spec, fixed, surface.sliders, ptsize = 1, center = TRUE)
```

Arguments

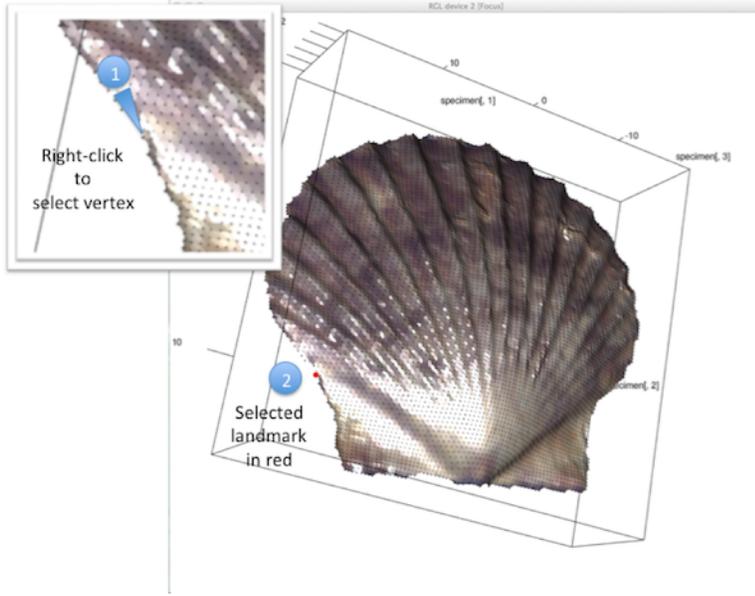
- *spec* Name of surface file, as either an object of class `shape3d`/`mesh3d`, or matrix of three-dimensional vertex coordinates.
- *fixed* Either: a single value designating the number of fixed template landmarks to be selected by `digit.fixed`, OR a p-x-k matrix of 3D coordinates collected previously (e.g. in other software)
- *surface.sliders* The number of template surface sliders desired
- *ptsizes* Size to plot the mesh points (vertices), e.g., 0.1 for dense meshes, 3 for sparse meshes
- *center* Logical Whether the object ‘spec’ should be centered prior to digitizing (default `center=TRUE`)

Function constructs a template of fixed landmarks and n “surface sliders”, semilandmarks that slide over a surface. The user digitizes the fixed points, then the function finds n surface semilandmarks following algorithm outlined in Gunz et al. (2005) and Mitteroecker and Gunz (2009). Surface semilandmarks are roughly equidistant set of predetermined number of points, chosen over the mesh automatically using a nearest-neighbor approach. The set of fixed and surface slider landmarks are exported as a “template”, which is used to extract a set of similarly numbered landmarks on every specimen using function `digit.surface`. Some of the “fixed” landmarks can be later designated as “curve sliders” using function `define.sliders` if required - see details in `digit.fixed`. To ensure a strong match between the scan and the template, it is recommended that a reasonable number of fixed points be used. These fixed points can be designated as “curve sliders” later using function `define.sliders`, see the function `digit.fixed` for details. NOTE: Function centers the mesh before digitizing by default (`center=TRUE`). If one chooses not to center, specimen may be difficult to manipulate in rgl window.

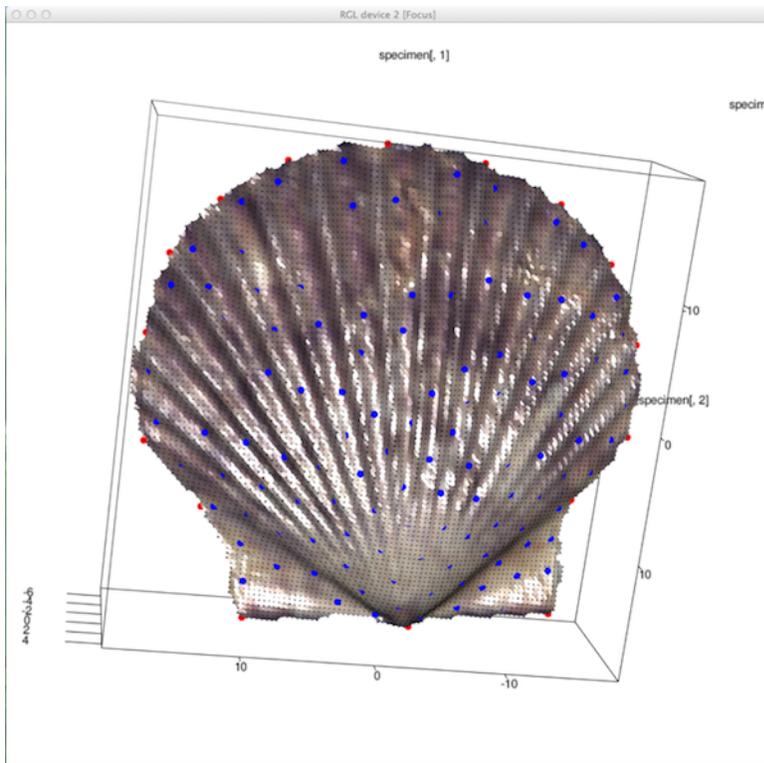
15.3.2 Digitizing

as before in `digit.fixed`, digitizing 16 landmarks on a mesh and then automatically calculate 150 surface sliding semilandmarks

```
myply1 <- read.ply("myply1.ply", ShowSpecimen=FALSE)
buildtemplate(spec = myply1, fixed = 16, surface.sliders = 150, ptsize=1)
Select Landmark 1 #See picture step 1
Keep landmark 1(y/n)? #See picture step 2
y
```



Select all of the fixed landmarks, and then the function will calculate the position of the surface sliding landmarks on the mesh, and plot them in blue.



Function writes to the working directory three files: an NTS file with the name of the specimen and .nts suffix containing the landmark coordinates, “template.txt” containing the same coordinates for use with the function `digit_surface`, and “surfslide.csv”, a file containing the address of the landmarks defined as “surface sliders” for use with `gpagen`. Function also returns to console an $n \times 3$ matrix containing the x,y,z coordinates of the digitized landmarks. NOTE: that the function will overwrite any existing template.txt file in the directory.

myply1.nts

```
"myply1
1 166 3 0 dim=3
```

```

13.2628089578878 10.2361171749175 -3.49646546534654
9.32980895788779 18.7309171749175 -1.94616546534654
-2.70691104211221 17.5351171749175 0.852234534653466
-13.7911610421122 15.6363171749175 0.945934534653464
-14.1681010421122 6.40941717491749 0.853034534653467
...

```

```

template.txt      "xpts" "ypts" "zpts" 13.2628089578878 10.2361171749175 -3.49646546534654
9.32980895788779 18.7309171749175 -1.94616546534654 -2.70691104211221 17.5351171749175
0.852234534653466 -13.7911610421122 15.6363171749175 0.945934534653464 -14.1681010421122
6.40941717491749 0.853034534653467 ...

```

surfslide.csv

```

x
17
18
19
20
21
22
23
...

```

which can be read in for use with `gpagen`.

```
sliders <- as.matrix(read.csv("surfslide.csv", header=T))
```

YouTube video of this function in action here: <http://youtu.be/7WWvImA2QE4>

15.3.3 AUTO mode

The function as described above (for interactive mode) calls `digit.fixed`, prompting the user to select fixed landmarks in the `rgl` window. However if the user has digitized these fixed landmark elsewhere (e.g., in other software), then the input for parameter 'fixed' can be a p-x-k matrix of 3D coordinates. In this case, the function will automatically use these landmarks to build the template of sliding semilandmarks.

To use:

```

myply1 <- read.ply("myply1.ply", ShowSpecimen=FALSE)
# read in fixed landmarks digitized elsewhere (e.g. using IDAV Landmark Editor)
lmks <- readmulti.nts("myply1_fixedlmks.nts")
# readmulti.nts returns a 3D array, so need to use just first specimen, as lmks[, ,1]
buildtemplate(spec = myply1, fixed = lmks[, ,1], surface.sliders = 150ptsize=1)
Select Landmark 1 #See picture step 1
Keep landmark 1(y/n)? #See picture step 2
y

```

Any of the geomorph `readland` functions can be used in this case, or any `read` function of *R*, depending on the data file of the fixed landmarks being imported. If using `read`, make sure the returned matrix contains only numerical values and is a matrix, not a data.frame.

15.3.4 editTemplate

An interactive function to remove landmarks from a 3D template file. **Function**

```
editTemplate(template, fixed, n)
```

Arguments

- *template* Matrix of template 3D coordinates.
- *fixed* Number of “fixed” landmark points (non surface sliding points)
- *n* Number of points to be removed

Function edits a ‘template.txt’ file made by `buildtemplate`, which must be in current working directory. Function overwrites ‘template.txt’ in working directory with edited version. Use `read.table(“template.txt”, header = T)`.

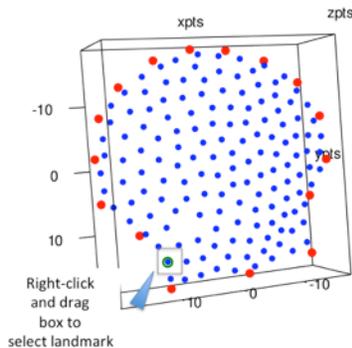
```
editTemplate(template, fixed=16, n=2)
```

```
Remove Template Points
```

```
1 of 2 points have been removed
```

```
2 of 2 points have been removed
```

```
○ ○ ○ RGL device 8 [Focus]
```



3D data collection: landmarks and semilandmarks (`digitsurface`) An interactive function to digitize three-dimensional (3D) landmarks on a surface lacking known landmarks, used after `buildtemplate`. Input for the function is either a matrix of vertex coordinates defining a 3D surface object or a mesh3d object as obtained from `read.ply`. For this function to work the “template.txt” file, made by `buildtemplate` must be in the working directory.

Function

```
digitsurface(spec, fixed, ptsize = 1, center = TRUE)
```

Arguments

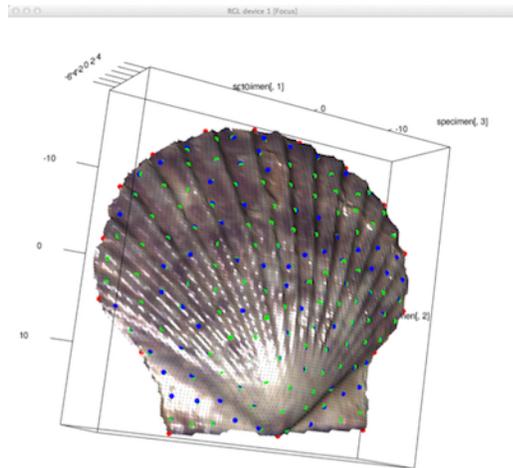
- *spec* Name of surface file, as either an object of class `shape3d/mesh3d`, or matrix of three-dimensional vertex coordinates.
- *fixed* Either: a single value designating the number of fixed template landmarks to be selected by `digit.fixed`, OR a p-x-k matrix of 3D coordinates collected previously (e.g. in other software)
- *ptsizes* numeric: Size to plot the mesh points (vertices), e.g., 0.1 for dense meshes, 3 for sparse meshes
- *center* Logical Whether the object ‘spec’ should be centered prior to digitizing (default `center=TRUE`)

Function for digitizing fixed 3D landmarks and placing “surface sliders”, semilandmarks that slide over a surface. Following selection of fixed points (see digitizing below), function finds surface semilandmarks following algorithm outlined in Gunz et al. (2005) and Mitteroecker and Gunz (2009). `digitsurface` finds the same number of surface semilandmarks as the “template.txt” file (created by `buildtemplate`) by downsampling scanned mesh, registering template with current specimen via GPA. A nearest neighbor algorithm is used to match template surface landmarks to current specimen’s. To use function `digitsurface`, the template must be constructed first, and “template.txt” be in the working directory. Some of the “fixed” landmarks digitized with `digitsurface` can be later designated as “curve sliders” using function

`define.sliders` if required (see details in `digit.fixed`). NOTE: Function centers the mesh before digitizing by default (`center=TRUE`). If one chooses not to center, specimen may be difficult to manipulate in `rgl` window. Digitizing: as before in `digit.fixed`.

Example, digitizing 16 landmarks on a mesh and fit a template to add 150 surface sliding semilandmarks

```
myply2 <- read.ply("myply2.ply", ShowSpecimen=FALSE)
digitSurface(myply2, fixed=16, pysize=1)
Select Landmark 1
Keep Landmark 1(y/n)?
y
...
```



Select all of the fixed landmarks, and then the function will calculate the position of the surface sliding landmarks on the mesh based on the template (blue), and plot them in green.

Function writes to the working directory an NTS file with the name of the specimen and `.nts` suffix containing the landmark coordinates.

myply2.nts

```
"myply2
1 166 3 0 dim=3
13.6450089578878 9.85741717491749 -3.49156546534654
9.70490895788779 18.3504171749175 -1.90256546534653
-2.33683104211221 17.5362171749175 0.855334534653466
-14.1518710421122 15.2515171749175 1.06203453465347
-14.1681010421122 6.40941717491749 0.853034534653467
...
```

YouTube video of this function in action here: <http://youtu.be/4S8XcMMgUyw>

15.3.5 AUTO mode

The function as described above (for interactive mode) calls `digit.fixed`, prompting the user to select fixed landmarks in the `rgl` window. However if the user has digitized these fixed landmark elsewhere (e.g., in other software), then the input for parameter 'fixed' can be a `p-x-k` matrix of 3D coordinates. In this case, the function will automatically use these landmarks and fit the template of sliding semilandmarks. See `buildtemplate` for example.

15.4 Calculate semilandmarks along a curve (`digit.curves`)

A function to calculate equidistant two-dimensional and three-dimensional semilandmarks along a curve. These landmarks will be treated as “sliders” in Generalized Procrustes analysis `gpagen`. This type of semilandmark “slides” along curves lacking known landmarks (see Bookstein 1997 for algorithm details). Each sliding semilandmark (“sliders”) will slide between two designated points, along a line tangent to the specified curvature, as specified by `define.sliders`.

Function

```
digit.curves(start, curve, nPoints, closed=TRUE)
```

Arguments

- *start* A vector of coordinates for the fixed landmark defining the start of the curve
- *curve* A p-x-k matrix of 2D or 3D coordinates for a set of ordered points defining a curve
- *nPoints* Numeric how many semilandmarks to place equidistantly along the curve
- *closed* Logical Whether the curve is closed (TRUE) or open (FALSE)

The function is based upon `tpsDig2` ‘resample curve by length’ for 2D data by James Rohlf. The start of the curve is a fixed landmark on the curve that is equivalent (homologous) in each specimen in the sample (and will be treated as a fixed point during Procrustes Superimposition using `gpagen`). Then `nPoints` are calculated along the curve at equidistant points from the start to the end.

‘curve’ is a p-x-k matrix of 2D or 3D coordinates for a set of ordered points defining a curve. This can be the pixels of an outline calculated in ImageJ (save xy coordinates), or obtained by automatically thresholding a jpeg using `Momocs` `import_jpg` function (<https://github.com/vbonhomme/Momocs/>), or any other reasonable way of obtaining ordered coordinates along a curve (including sampling by hand using `digit.fixed` or `digitize2d` - but note that there should be more points defining the curve than `nPoints` in order to accurately calculate the semilandmarks).

If ‘closed = T’, the function returns the coordinates of the ‘start’ landmark plus `nPoints`. If ‘closed = F’, the function returns the coordinates of the ‘start’ landmark, plus `nPoints` and the end of the curve.

Example Here we have an outline of a leaf in 2D, exported as a .txt file of x,y, coordinates from ImageJ (see `warpRefOutline` for details on how this can be done). The input could also be a .nts file from `digit.fixed`, a .dta from IDAV Landmark, or .tps file from `digitize2d` or any other reasonable way to get coordinates defining a set of points that together describe a curve. The resulting coordinate matrix is to be used in replacement of the input curve coordinates for analyses.

```
curve <- as.matrix(read.table("Chusquea pittieri_1.txt", skip=4))
start <- c(8.5143, 11.8367) # know the x,y coordinates of the start of the curve
plot(curve, asp=T, pch=19, cex=0.1) # plot the curve
points(start[1], start[2], pch=19, cex=0.5, col="red") # show the starting point
```

```
lmks <- digit.curves(start, curve, nPoints=20, closed = T) # run function
points(lmks[,1], lmks[,2], pch=19, cex=0.5, col="green") # plot the sampled points
```

```
lmks
      [,1]    [,2]
[1,]  8.5143 11.8367
[2,]  9.4671 11.2816
[3,] 10.3347 10.5267
[4,] 11.1673  9.7204
[5,] 11.9918  8.9059
[6,] 12.8146  8.0898
[7,] 13.6082  7.2609
[8,] 14.3863  6.4327
```

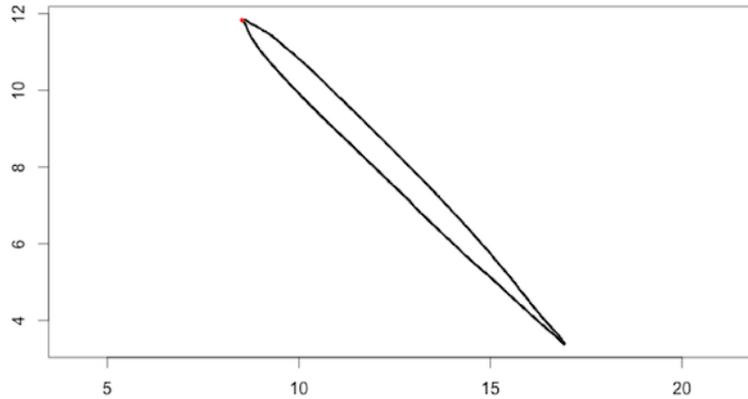


Figure 19: An example curve defined by pixels extracted from a picture in ImageJ. Start is in red

```
[9,] 15.1347 5.5750
[10,] 15.8599 4.6939
[11,] 16.5762 3.8367
[12,] 16.5224 3.7333
[13,] 15.6810 4.4980
[14,] 14.8422 5.2653
[15,] 14.0000 6.0294
[16,] 13.1725 6.8082
[17,] 12.3579 7.6163
[18,] 11.5347 8.4156
[19,] 10.7045 9.2082
[20,] 9.8901 10.0327
[21,] 9.1102 10.8917
```

lmks is the matrix of semilandmarks defining the curve

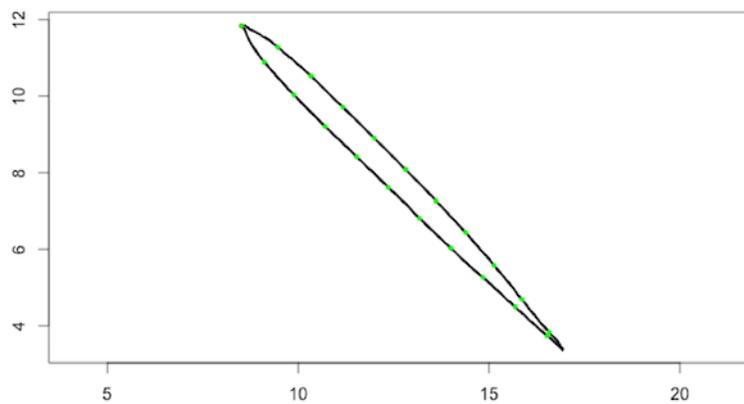


Figure 20: Semilandmarks digitized on the curve (green)

15.5 Calculate linear distances between landmarks (`interlmkdist`)

A simple function to calculate linear distances between a set of landmark coordinates (interlandmark distances)

Function

```
interlmkdist(A, lms)
```

Arguments

- *A* A 3D array (p x k x n) containing landmark coordinates for a set of specimens
- *lms* A matrix or dataframe of landmark addresses for the start and end landmarks defining m linear measurements (can be either 2-x-m or m-x-2). Either the rows or the columns should have names ‘start’ and ‘end’ to define landmarks

Function takes a 3D array of landmark coordinates from a set of specimens and the addresses for the start and end landmarks defining linear measurements and then calculates the interlandmark distances. The function returns a matrix of linear distances for all specimens. If the ‘lms’ matrix has row or column names defining the name of the linear measurements, the returned matrix will use these for column names (see example). If only two interlandmark distances, ‘lms’ input must be m x 2. Function returns a matrix (n x m) of m linear distances for n specimens. _____

16 Frequently Asked Questions

This section will cover some of *geomorph*’s frequently asked questions.

- 1) I’ve loaded my TPS file but I get a warning that no names were extracted. My TPS file has specimen names in it, what do I do?
 - For TPS files it is necessary to specify whether the specimen name should be read from the ID= line or the IMAGE= line. See chapter 2 for more details.
- 2) I want to add a groups to my data analysis, what do I do?
 - Grouping variables can be made in *R* (e.g., `c`) or imported from outside (e.g., as a .csv file). See chapter 3 for more details.
- 3) How many iterations should I use in permutation tests?
 - The default for *geomorph* functions is often 999. Generally more is better, but too many can be excessive and does not lead to more power. See section on permutation tests in the introduction for more details.
- 4) How do I reflect my specimens before using `gpagen` or `bilat.symmetry`?
 - Reflecting a set of specimens may be accomplished by multiplying one coordinate dimension by ‘-1’ for these structures (either the x-, the y-, or the z-dimension). To do this:

```
mydata # this is your 3D array of raw variables needing to be reflected
refl <- array(1, dim=dim(mydata)) # make an array the same dimensions as mydata and fill it with 1s
refl[,1,] <- -1 # replace the x coordinate column with -1
mydata <- mydata * refl # multiply the two arrays to reflect
```

- 5) I’ve loaded my TPS file and I got a warning that “Not all specimens have scale...”. I defined a scale by digitizing two landmarks on the scale bar, what do I do?
 - Let’s say that landmarks 47 and 48 define the ends of the 10mm scale bar, and `coords` is a 3D array of the raw coordinate data:

```
for (i in 1:dim(coords)[3]){
  scb <- coords[47:48,,i]
  scale <- sqrt((scb[1,1]-scb[2,1])^2+(scb[1,2]-scb[2,2])^2)/1000
  coords[, ,i] <- coords[, ,i]*scale }
coords <- coords[-(47:48),,] # remove the landmarks
```

17 References

- Adams, D. C. 1999. Methods for shape analysis of landmark data from articulated structures. *Evol. Ecol. Res.* 1:959-970.
- Adams, D. C. 2014. A generalized K statistic for estimating phylogenetic signal from shape and other high-dimensional multivariate data. *Syst. Biol.* 63:685-697.
- . 2014. A method for assessing phylogenetic least squares models for shape and other high-dimensional multivariate data. *Evolution* 68:2675-2688.
- . 2016. Evaluating modularity in morphometric data: Challenges with the RV coefficient and a new test measure. *Methods in Ecology and Evolution.* 7:565-572.
- Adams, D. C., and C. D. Anthony. 1996. Using randomization techniques to analyse behavioural data. *Anim. Behav.* 51:733-738.
- Adams, D. C., and M. M. Cerney. 2007. Quantifying biomechanical motion using Procrustes Motion Analysis. *J. Biomech.* 40:437-444.
- Adams, D. C., and M. L. Collyer. 2007. The analysis of character divergence along environmental gradients and other covariates. *Evolution* 61:510-515.
- . 2009. A general framework for the analysis of phenotypic trajectories in evolutionary studies. *Evolution* 63:1143-1154. —. 2016. On the comparison of the strength of morphological integration across morphometric datasets. *Evolution.* 70: 2623-2631. Adams, D. C., and R. Felice. 2014. Assessing phylogenetic morphological integration and trait covariation in morphometric data using evolutionary covariance matrices. *Plos ONE* 9:e94335.
- Adams, D. C., and A. Nistri. 2010. Ontogenetic convergence and evolution of foot morphology in European cave salamanders (Family: Plethodontidae). *BMC Evol. Biol.* 10:216.
- Adams, D. C., F. J. Rohlf, and D. E. Slice. 2004. Geometric morphometrics: ten years of progress following the 'revolution'. *Italian Journal of Zoology* 71:5-16.
- Adams, D. C., F. J. Rohlf, and D. E. Slice. 2013. A field comes of age: geometric morphometrics in the 21st century. *Hystrix* 24:7-14.
- Anderson, M. J. 2001. A new method for non-parametric multivariate analysis of variance. *Austral. Ecol.* 26:32-46. Anderson, M. J., and C. J. F. terBraak. 2003. Permutation tests for multi-factorial analysis of variance. *Journal of Statistical Computation and Simulation* 73:85-113.
- Blomberg, S. P., T. Garland, and A. R. Ives. 2003. Testing for phylogenetic signal in comparative data: behavioral traits are more labile. *Evolution* 57:717-745.
- Bookstein, F. L. 1991. *Morphometric tools for landmark data: geometry and biology.* Cambridge Univ. Press., New York.
- Bookstein, F. L. 1997. Shape and the information in medical images: A decade of the morphometric synthesis. *Computer Vision and Image Understanding* 66:97-118.
- . 2015. Integration, disintegration, and self-similarity: Characterizing the scales of shape variation in landmark data. *Evol. Biol.*
- Bookstein, F. L., P. Gunz, P. Mitteroecker, H. Prossinger, K. Schaefer, and H. Seidler. 2003. Cranial integration in Homo: singular warps analysis of the midsagittal plane in ontogeny and evolution. *J. Hum. Evol.* 44:167-187.
- Claude, J. 2008. *Morphometrics with R.* Springer.
- Collyer, M. L., and D. C. Adams. 2007. Analysis of two-state multivariate phenotypic change in ecological studies. *Ecology* 88:683-692.
- . 2013. Phenotypic trajectory analysis: comparison of shape change patterns in evolution and ecology. *Hystrix* 24:75-83.
- Collyer, M. L., D. J. Sekora, and D. C. Adams. 2015. A method for analysis of phenotypic change for phenotypes described by high-dimensional data. *Heredity* 115:357-365.
- Drake, A. G., and C. P. Klingenberg. 2008. The pace of morphological change: historical transformation of skull shape in St Bernard dogs. *Proc. R. Soc. B* 275:71-76.
- Dryden, I. L., and K. V. Mardia. 1993. *Multivariate shape analysis.* Sankhya 55:460-480.
- Good, P. 2000. *Permutation tests: a practical guide to resampling methods for testing hypotheses.* Springer, New York.
- Goodall, C. 1991. Procrustes methods in the statistical analysis of shape. *Journal of the Royal Statistical*

- Society. Series B (Methodological):285-339.
- Gower, J. C. 1975. Generalized Procrustes analysis. *Psychometrika* 40:33-51.
- Gunz, P., P. Mitteroecker, and F. L. Bookstein. 2005. Semilandmarks in three dimensions. Pp. 73-98 in D. E. Slice, ed. *Modern morphometrics in physical anthropology*. Kluwer Academic/Plenum Publishers, New York.
- Gunz, P., P. Mitteroecker, S. Neubauer, G. W. Weber, and F. L. Bookstein. 2009. Principles for the virtual reconstruction of hominin crania. *J. Hum. Evol.* 57:48-62.
- Kendall, D. G. 1984. Shape-manifolds, Procrustean metrics and complex projective spaces. *Bulletin of the London Mathematical Society* 16:81-121.
- Klingenberg, C. P. 2009. Morphometric integration and modularity in configurations of landmarks: tools for evaluating a priori hypotheses. *Evol. Dev.* 11:405-421.
- Klingenberg, C. P., M. Barluenga, and A. Meyer. 2002. Shape analysis of symmetric structures: quantifying variation among individuals and asymmetry. *Evolution* 56:1909-1920.
- Klingenberg, C. P., and N. A. Gidaszewski. 2010. Testing and quantifying phylogenetic signals and homoplasy in morphometric data. *Syst. Biol.* 59:245-261.
- Klingenberg, C. P., and G. S. McIntyre. 1998. Geometric morphometrics of developmental instability: Analyzing patterns of fluctuating asymmetry with procrustes methods. *Evolution* 52:1363-1375.
- Mardia, K. V., F. L. Bookstein, and I. J. Moreton. 2000. Statistical assessment of bilateral symmetry of shapes. *Biometrika* 87:285-300.
- Mitteroecker, P., and P. Gunz. 2009. Advances in geometric morphometrics. *Evol. Biol.* 36:235-247.
- Mitteroecker, P., P. Gunz, M. Bernhard, K. Schaefer, and F. L. Bookstein. 2004. Comparison of cranial ontogenetic trajectories among great apes and humans. *J. Hum. Evol.* 46:679-697.
- O'Higgins, P., and N. Jones. 1998. Facial growth in *Cercocebus torquatus*: an application of three-dimensional geometric morphometric techniques to the study of morphological variation. *J. Anat.* 193:251-272.
- Revell, L. J. 2012. *phytools*: an R package for phylogenetic comparative biology (and other things). *Methods in Ecology and Evolution* 3:217-223.
- Rohlf, F. J. 1998. On applications of geometric morphometrics to studies of ontogeny and phylogeny. *Syst. Biol.* 47:147-158.
- Rohlf, F. J. 1999. Shape statistics: Procrustes superimpositions and tangent spaces. *J. Classif.* 16:197-223.
- Rohlf, F. J. 2002. Geometric morphometrics and phylogeny. Pp. 175-193 in N. MacLeod, and P. L. Forey, eds. *Morphology, shape and phylogeny*. Francis & Taylor, London.
- . 2010. *tpsRelw*: Relative warp analysis. Version 1.49. Version 1.49. Department of Ecology and Evolution, State University of New York at Stony Brook, Stony Brook, NY.
- . 2012. *NTSYSpc*: Numerical taxonomy and multivariate analysis system. Version 2.2. New York: Exeter Software.
- Rohlf, F. J., and M. Corti. 2000. Use of two-block partial least-squares to study covariation in shape. *Syst. Biol.* 49:740-753.
- Rohlf, F. J., and L. F. Marcus. 1993. A revolution in morphometrics. *Trends Ecol. Evol.* 8:129-132.
- Rohlf, F. J., and D. Slice. 1990. Extensions of the Procrustes method for the optimal superimposition of landmarks. *Syst. Zool.* 39:40-59.
- Zelditch, M. L., D. L. Swiderski, and H. D. Sheets. 2012. *Geometric morphometrics for biologists: a primer*. 2nd ed. Elsevier, Amsterdam.