Quick Guide to Geomorph: Vignette 1

Emma Sherratt

10 February 2016

Data Input: Importing landmark data

TPS files (`readland.tps`)

Function:

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

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
```

In this case, there is no scale given in the tps file, so the command 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]]).

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 x 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
-.450 -.475 -.590 -.280 -.515 -.120 -.330 0 0 0 .145 -.395 -.045 -.420 -.260 -.465
-.530 -.555 -.685 -.320 -.625 -.120 -.400 0 0 0 .230 -.425 -.005 -.480 -.265 -.525
-.560 -.570 -.700 -.335 -.670 -.120 -.425 0 0 0 .300 -.440 .015 -.495 -.270 -.540
-.590 -.580 -.745 -.355 -.700 -.100 -.435 0 0 0 .330 -.445 .030 -.505 -.285 -.565
-.650 -.580 -.800 -.340 -.715 -.090 -.450 0 0 0 .360 -.445 .040 -.515 -.300 -.580
...
```

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
...
```

The function returns a 3D array containing the coordinate data, and if provided in the file, the names of the specimens (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.

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. 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" ...
```

Where the 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.

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")
```

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)
```

Data Preparation: Manipulating landmark data and classifiers

Data arrays

Landmark data in *geomorph* can be found as objects in two formats: a 2D array (matrix; Figure 1A) or a 3D array (Figure 1B). 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).

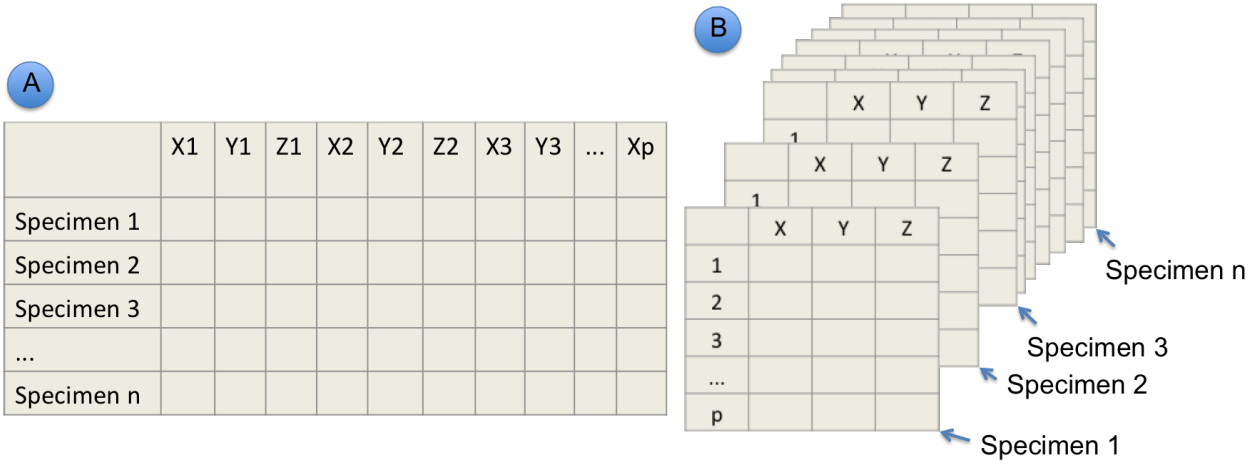


Figure 1 2D array and 3D array of landmark coordinate data. NOTE: This example shows 3D landmark coordinate data, but the same format would be used for 2D coordinate data.

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 x k x 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
```

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 x k x n) of landmark coordinates into a 2D array (n x [p x 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
...
```

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
names(group) <- dimnames(mydata)[[3]] ## assign specimen names from 3D array of data to the group classi
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
...	...	...

```
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)
```

```
Loading required package: rgl
```

```
Loading required package: ape
```

```
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 , *suchasplethodon*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).

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.

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:

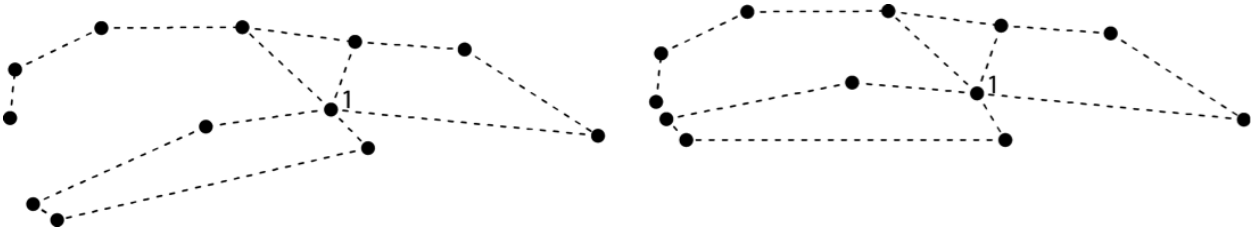
`A` A 3D array (p x k x n) containing landmark coordinates for a set of specimens
`art.pt` A number specifying which landmark is the articulation point between the two landmark subsets
`angle.pts` A vector containing numbers specifying which two points used to define the angle (one per subset)
`rot.pts`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



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

Generalized Procrustes Analysis

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)
```

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

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`, see Vignette 5). 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.

Notes for geomorph 3.0

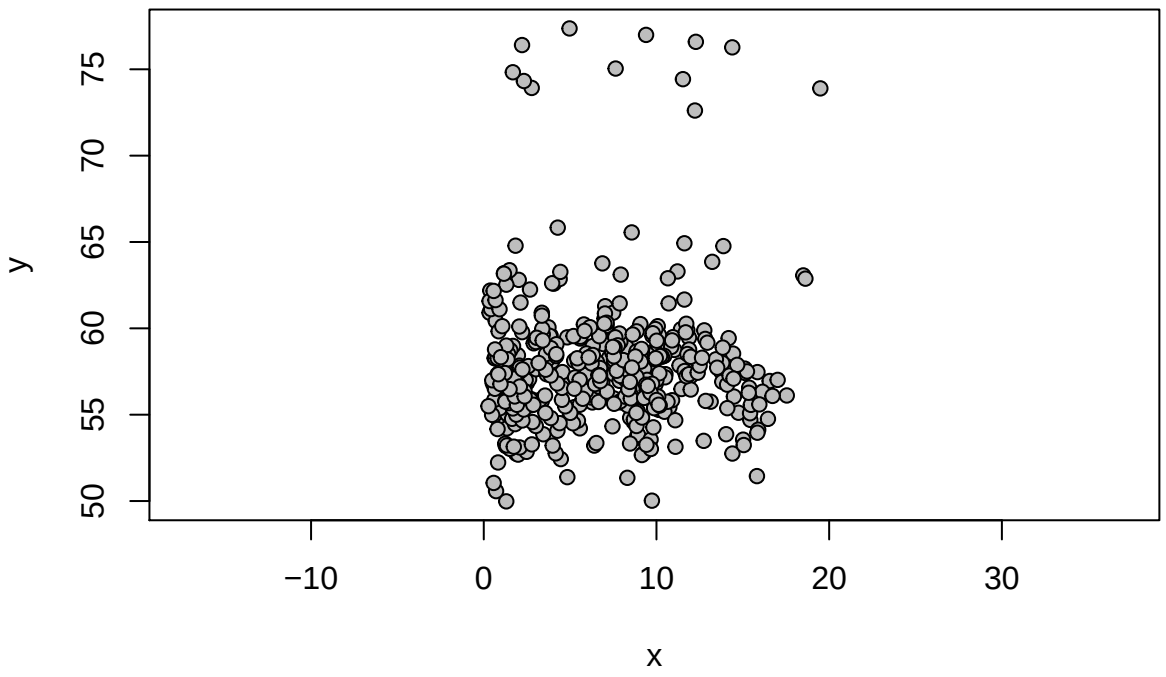
Compared to older versions of *geomorph*, users might notice subtle differences in Procrustes residuals when using semilandmarks (curves or surfaces). This difference is a result of using recursive updates of the consensus configuration with the sliding algorithms (minimized bending energy or Procrustes 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.

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`.

Procrustes Superimposition of Fixed Landmarks only

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

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




```
## Perform GPA-alignment
plethodon.gpa <- gpagen(plethodon$land)
## gpagen runs plotAllSpecimens showing Procrustes residuals around the mean
```

Function returns a list containing:

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

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

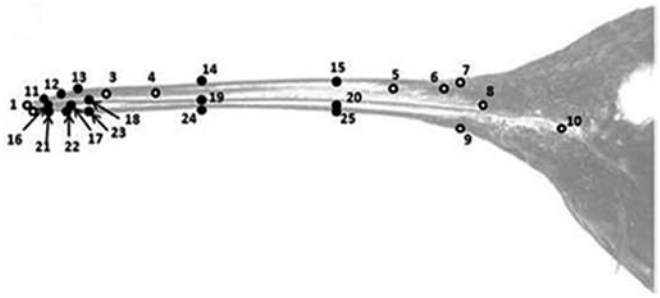
```
  plotAllSpecimens(plethodon.gpa$coords,links=plethodon$links)
```

To view the options for this function

```
  ?plotAllSpecimens
```

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.

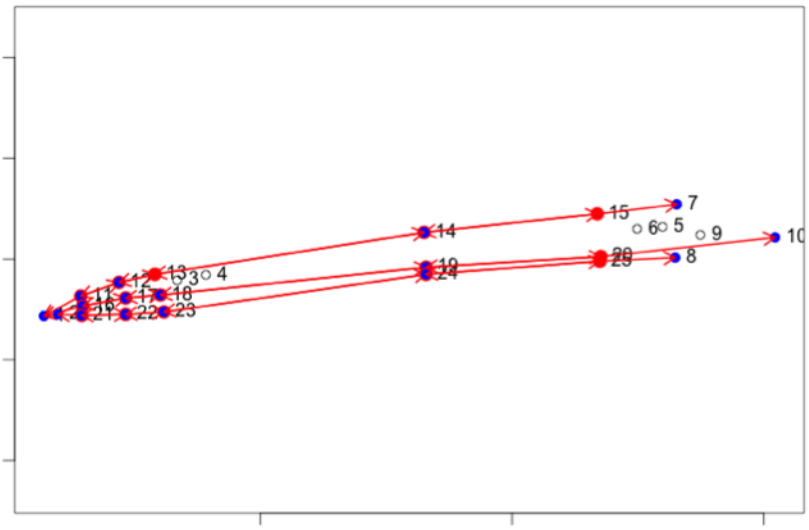


The `hummingbird` dataset, from Berns & Adams 2010 (The Auk 127:626-635), contains fixed landmarks (open circles) and semilandmarks (closed circles) that define the curvature of the beak. When semilandmarks are included in the dataset, a few extra parameters should be specified in the `gpagen` function.

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)
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` (discussed later).

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

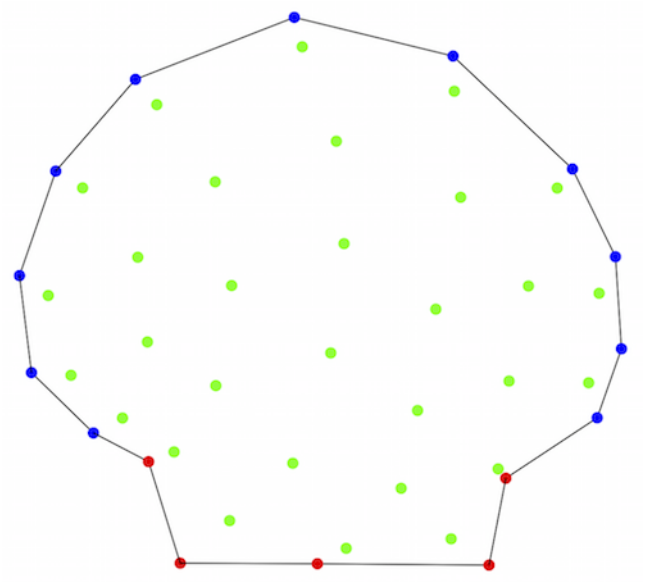
```
A <- gpagen(hummingbirds$land,
            curves=hummingbirds$curvepts,
            ProcD=TRUE) ## Using Procrustes Distance for sliding
B <- gpagen(hummingbirds$land,
            curves=hummingbirds$curvepts,
            ProcD=FALSE) ## Using bending energy for sliding
```

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).

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.



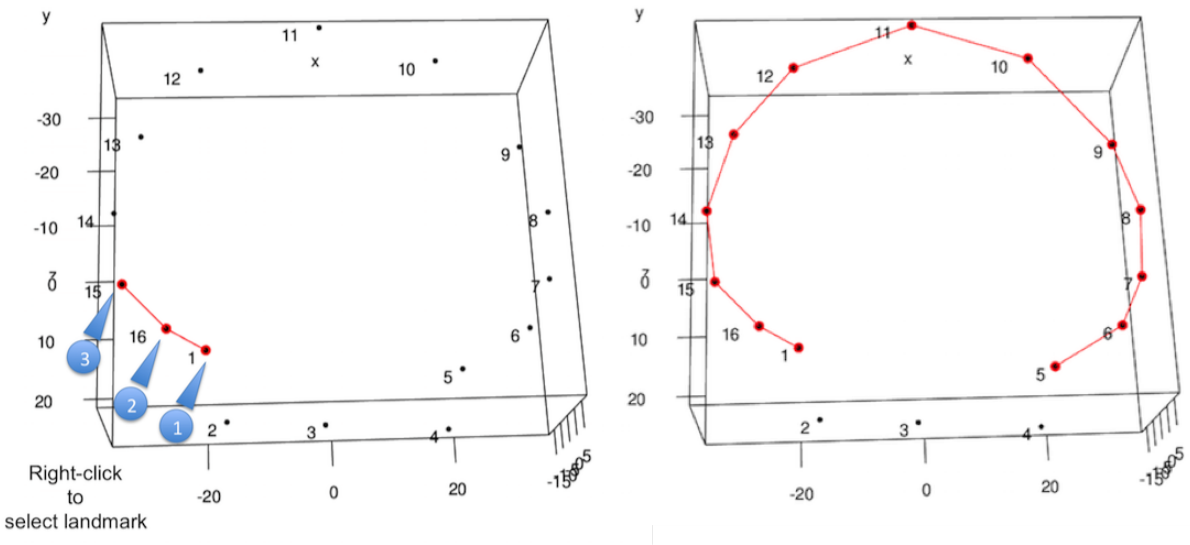
```
data(scallops) ## dataset of scallop shells with semilandmarks
attributes(scallops)
scallops$curvslide ## the curve sliding matrix
head(scallops$surfslide) ## the surfaces sliding matrix
scallop.gpa = gpagen(A=scallops$coorddata,
                     curves=scallops$curvslide,
                     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.

To make one these curve sliding matrices, you can use the interactive function `define.sliders`. Here we will run it using the first specimen, and hiding the surface sliding landmarks to make things easier to see

```
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
```

Once the above function has run, you will need to read in the outputted file for using in the GPA as follows

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

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

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)
```

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

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).

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) ## 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)


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     Rsq      F       Z Pr(>F)
ind                 9 0.104888 0.0116542 0.45533 2.6901 1.34269  0.062 .
side                1 0.003221 0.0032209 0.01398 0.7435 0.60403  0.674
ind:side            9 0.038990 0.0043323 0.16926 1.0407 0.69384  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     Rsq      F       Z Pr(>F)
ind                 9 4.1497e-09 4.6107e-10 0.18555 0.5965 0.28447  0.898
side                1 3.4740e-10 3.4738e-10 0.01553 0.4494 0.18543  0.530
ind:side            9 6.9569e-09 7.7299e-10 0.31108 1.4170 0.72586  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. When `verbose=TRUE`, the function returns the symmetric component of shape variation (`$symm.shape`) and the asymmetric component of shape variation (`$asymm.shape`), to be used in subsequent analyses like Procrustes coordinates.

Example of object symmetry

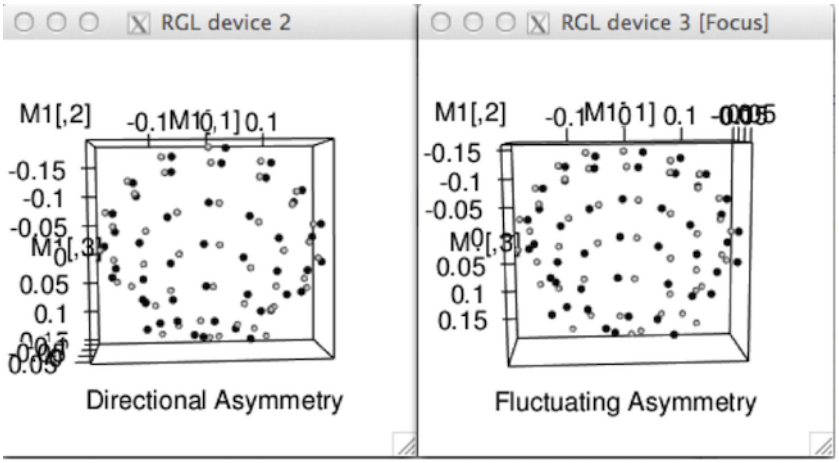
```
data(scallops)
gdf <- geomorph.data.frame(shape = scallops$coorddata, ind=scallops$ind)
scallop.sym <- bilat.symmetry(A = shape, ind = ind, object.sym = TRUE,
               land.pairs=scallops$land.pairs, data = gdf, RRPP = TRUE,
               iter = 499) ## perform object symmetry GPA
summary(scallop.sym)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

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


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     Rsq       F       Z Pr(>F)
ind       4 0.064030 0.0160075 0.64431  9.8348 0.59476  0.978
side      1 0.028838 0.0288375 0.29018 17.7173 2.07959  0.064 .
ind:side  4 0.006511 0.0016276 0.06551
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```



Data Analysis

After the data have been superimposed with `gpagen` or `bilat.symmetry`, the Procrustes coordinates can be used in many statistical analyses, ordination methods and visualization methods.

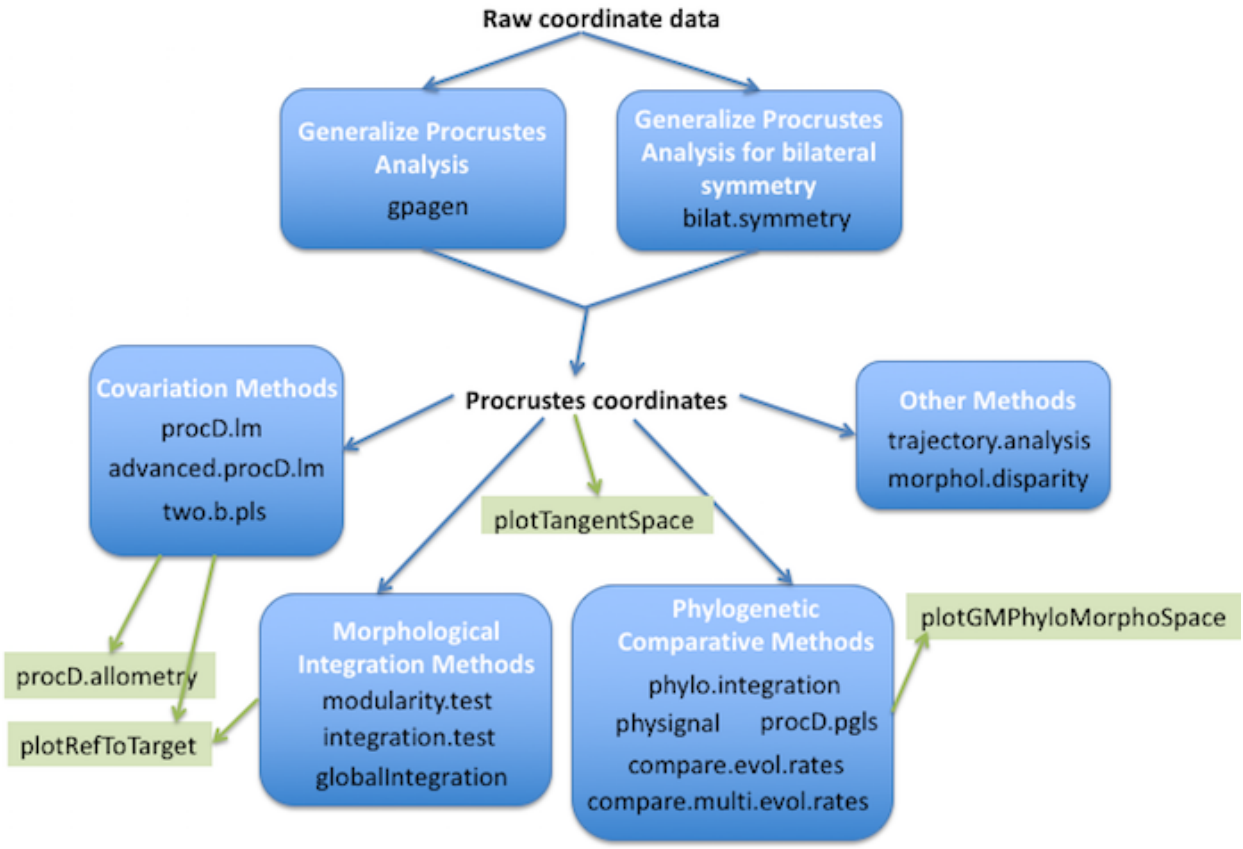


Figure 3 Overview of some of the analysis (blue) and visualization (green) functions in *geomorph*.