Morphops Documentation

Release 0.1.13

Vaibhav Patel

Oct 10, 2021

Contents

Morphops implements common operations and algorithms for Geometric Morphometrics, in Python 3.

CHAPTER 1

Features

Some high-level operations in the current version are

- Centering, rescaling data: *remove_position(lmk_sets)*, *remove_scale(lmk_sets)*
- Rigid Rotation, Ordinary and Generalized Procrustes alignment: *rotate(src_sets,tar_sets)*, *opa(src_set,tar_set)*, *gpa(all_sets)*
- Thin-plate spline warping: *tps_warp(X, Y, pts)*
- Reading from and writing to *.dta files: *read_dta(fn)*, *write_dta(fn,lmk_sets,names)*

1.1 Dependencies

- numpy

1.2 Installation

```
pip install morphops
```

1.3 Usage Examples

```
import morphops as mops

# Create 3 landmark sets, each having 5 landmarks in 2 dimensions.
A = [[0,0],[2,0],[2,2],[1,3],[0,2]]
B = [[0.1,-0.1],[2,0],[2.3,1.8],[1,3],[0.4,2]]
C = [[-0.1,-0.1],[2.1,0],[2,1.8],[0.9,3.1],[-0.4,2.1]]

# Perform Generalized Procrustes alignment to align A, B, C.
```

(continues on next page)

(continued from previous page)

```
res = mops.gpa([A, B, C])
# res['aligned'] contains the aligned A, B, C. res['mean'] is their mean.

# Create a Thin-plate Spline warp from A to B and warp C.
warped_C = mops.tps_warp(A, B, C)
# warped_C contains the image of the pts in C under the TPS warp.
```

CHAPTER 2

What is Geometric Morphometrics?

Geometric Morphometrics is a statistical toolkit for quantifying and studying shapes of forms that are represented by homologous landmark sets.

"Shape" has a specific notion here. For a given landmark set, its shape refers to the spatial information that survives after discarding its absolute position, scale and rotation. So two landmark sets have the same shape if they can be brought in perfect alignment by only changing their positions, scales and rotations.

2.1 Common Operations and Algorithms in Studies

Geometric Morphometrics is often used when pursuing statistical questions involving the morphology of biological forms, like do corvid species that frequently probe have longer bills and more to-the-side orbits than corvid species that frequently peck. It helps inform the Data Collection, Preprocessing and Analysis steps of such statistical studies with sound theoretical or practical justifications.

2.1.1 Data Collection

The most prevalent form of Data Collection involves picking homologous landmarks on each form. For curving forms with few homologous points but well-understood homologous regions, there is a notion of semilandmarks which can "slide" to minimize equidistant sampling artifacts.

A common file format for saving landmarks for a set of specimens is the **.dta* format used by the IDAV Landmark Editor software.

2.1.2 Preprocessing

As discussed before, a central idea in Geometric Morphometrics is extracting the "shapes" of the landmark sets. One way to achieve this is to use the Generalized Procrustes Alignment algorithm or GPA. GPA aligns all the landmark sets by modifying their locations, orientations and sizes so as to minimize their collective interlandmark distances.

After this step, the aligned shapes all lie in a high-dimensional non-linear manifold. For example, if the orignal landmark sets were a set of triangles, the aligned shapes lie on a sphere. Moreover, for naturally arising datasets, the shapes likely lie very close to each other and are distributed around a mean shape. This usually makes it permissible to project all the shapes into the tangent space at the mean shape, and this way the final shape vectors lie in a linear space.

2.1.3 Analysis

With the shapes lying in a high-dimensional linear space after preprocessing, they can now be submitted to various commonly used statistical procedures like Principal Components Analysis and various kinds of regression for further analysis.

CHAPTER 3

morphops module

The morphops module is the primary module of the Morphops library.

It contains implementations of common geometric morphometrics operations. Some examples are -

- IO operations to read/write landmark data
- Common preprocessing like Generalized Procrustes Alignment
- Thin-plate spline warping operations

3.1 Submodules

procrustes	Provides procrustes alignment related operations and algorithms.
tps	Provides thin-plate splines related operations and algorithms.
io	Provides IO functions to read from and write to files in common landmark data file formats.
lmk_util	Provides common functions used in the module.

3.1.1 morphops.procrustes

Provides procrustes alignment related operations and algorithms.

For geometric morphometrics based studies, after landmark data are collected for each specimen, a typical next step is to remove the position, size and orientation information from the landmark set of each specimen so that what remains is the shape information. This can be achieved by, for example, running Generalized Procrustes Aligment (see *gpa()*) on the set of landmark sets.

After procrustes alignment, the shapes lie in a high-dimensional non-euclidean manifold but are usually quite close to each other and can be projected to a euclidean tangent space at their shape mean, whereupon they can be subjected to multivariate analysis techniques like Principal Components Analysis, Partial Least Squares, etc.

morphops.procrustes.**get_position**(*lmks*)
Returns the centroid of the set or sets of landmarks in *lmks*.

The centroid of a p landmarks is simply the arithmetic mean of all the landmark positions. That is

$$\mathbf{x_c} = \sum_{i=1}^{p} \frac{\mathbf{x_i}}{p}$$

Parameters **lmks** (*array-like*) – One of the following

- **Single specimen** A (p,k) array of p landmarks in k dimensions for one specimen.
- **n specimens** A (n,p,k) array of n landmark sets for n specimens, each having p landmarks in k dimensions.

Returns

centroid –

- If *lmks* is a (p,k) array, then *centroid* is a (k,)-shaped array, whose i-th element is the mean of the i-th coordinate in *lmks*.
- If *lmks* is a (n,p,k) array, then *centroid* is a (n,k)-shaped array whose i-th element is the (k,)-shaped centroid of the i-th specimen's landmarks in *lmks*.

Return type numpy.ndarray

morphops.procrustes.**get_scale**(*lmks*)
Returns the euclidean norm of the real matrix or matrices in *lmks*.

The euclidean norm of the real (p x k) matrix X is calculated as

$$\|X\| = \sqrt{Tr(X^T X)}$$

Note: *lmks* is not assumed to have been pre-centered. To pre-center *lmks* you can call *remove_position()* on *lmks* before applying *remove_scale*.

Parameters **lmks** (*array-like*) – One of the following

- **Single specimen** A (p,k) array of p landmarks in k dimensions for one specimen.
- **n specimens** A (n,p,k) array of n landmark sets for n specimens, each having p landmarks in k dimensions.

Returns

scale –

- **Single specimen** If *lmks* is (p,k)-shaped, *scale* is a float representing its euclidean norm.
- **n specimens** If *lmks* is (n,p,k)-shaped, *scale* is an (n,) -shaped array such that the i-th element is the euclidean norm of the i-th specimen's landmarks.

Return type numpy.float64 or numpy.ndarray

morphops.procrustes.**get_ssqd**(*X*)
Alias for *lmk_util.ssqd(X)*.

morphops.procrustes.**gpa**(*X*, *tol=1e-05*, *max_iters=10*, *do_project=False*, *do_scaling=False*, *no_reflect=False*, *unitize_mean=False*)

Performs Generalized Procrustes Alignment to transform all the landmark sets in *X* such that (a quantity proportional to) the sum of squared norms of pairwise differences between all the landmark sets is minimized.

Say `len(X) = n`. *gpa()* tries to find

$$\underset{\beta_i>0,\ R_i\in O(k),\ \gamma_i\in\mathbb{R}^k}{\operatorname{argmin}} g(X) = \frac{1}{n}\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} \|(\beta_i X_i R_i + \mathbf{1_k}\gamma_i^T) - (\beta_j X_j R_j + \mathbf{1_k}\gamma_j^T)\|^2$$

The Generalized (Procrustes) Sum of Squares or G is defined as

$$G(X) = \inf_{\beta_i>0,\ R_i\in O(k),\ \gamma_i\in\mathbb{R}^k} g(X)$$

The GPA algorithm, per [drymar], tries to iteratively rotate and scale the landmark sets in *X* until the sum of squared differences is below *tol*. While the algorithm should converge quite fast, it can be forced to stop the minimization loop after *max_iters* number of iterations.

For an explanation of the other parameters, please see the Parameters section.

Note: Re *do_project* and *do_scaling*: The projection used here is based on [rohlf] and assumes that the aligned shapes are of unit centroid size, which is not generally true when *do_scaling* is *True*. Consequently, if both *do_project* and *do_scaling* are *True*, *gpa()* will issue a warning, but proceed with the projection.

Note: Generally for *opa()*, $OSS(X1, X2) \neq OSS(X2, X1)$.

In contrast to *opa()*, *gpa()* is symmetric for the input matrices in that $G(X1, X2) = G(X2, X1)$.

See also:

rotate(), *opa()*

Parameters

- **X** (*array-like*) – A (n,p,k)-shaped set of landmark sets that have to be aligned to each other.
- **tol** (*float, optional*) – The sum of squared differences value that will be considered "low enough" by the iterative rotation and scaling. The iterations will continue until *tol* has been achieved or *max_iters* is reached, whichever comes first.
- **max_iters** (*int, optional*) – The maximum number of iterations that the iterative rotation and scaling is allowed to run for. The iterations will continue until *tol* has been achieved or *max_iters* is reached, whichever comes first.
- **do_scaling** (*bool, optional*) – If *False*, $\beta_i = \frac{1}{\|X_i'\|}$, where X_i' is the mean-centered X_i. Else β_i is calculated as per [tenb].
- **do_project** (*bool, optional*) – If *True*, the final aligned landmarks are orthogonally projected to the tangent space at the mean of aligned landmark sets *mean*, using equation 1 in [rohlf].
- **no_reflect** (*bool, optional*) – Flag indicating whether the best alignment should exclude reflection (default is False, which means reflection will be used if it achieves better alignment).

- **unitize_mean** (*bool, optional*) – Flag indicating whether the mean of aligned landmark sets *mean* should be rescaled to have unit centroid size.

Returns

result –

aligned: numpy.ndarray A (n,p,k)-shaped set of aligned landmark sets.

mean: numpy.ndarray A (p,k)-shaped array representing the mean of the procrustes aligned landmark sets *aligned*.

b: numpy.ndarray A (n,)-shaped array representing the scaling factor β_i by which the centered X_i' is scaled.

ssq: numpy.float64 This number represents the Generalized (Procrustes) Sum of Squares, which is the infinimum of g. Essentially, the *ssq* is the result of plugging in the optimal β_i, R_i and γ_i into the g objective.

Return type dict

Warns **UserWarning** – If both *do_project* and *do_scaling* are *True*

References

morphops.procrustes.**opa**(*source*, *target*, *do_scaling=False*, *no_reflect=False*)

Performs Ordinary Procrustes Alignment to transform the landmark set *source* such that the squared Euclidean distance between *source* and *target* is minimized.

Say X='source' and Y='target' and *do_scaling* = *True*. *opa()* tries to find

$$\underset{\beta>0,\ R\in O(k),\ \gamma\in\mathbb{R}^k}{\operatorname{argmin}} D^2_{\mathtt{OPA}}(X,Y) = \|Y - \beta XR - \mathbf{1_k}\gamma^T\|^2$$

If *do_scaling* = *False*, $\beta = 1$. If *no_reflect* = *True*, then just as in *rotate()*, *opa()* will force $R \in SO(k)$.

The Ordinary (Procrustes) Sum of Squares or OSS is defined as

$$OSS(X,Y) = \min_{\beta>0,\ R\in O(k),\ \gamma\in\mathbb{R}^k} D^2_{\mathtt{OPA}}(X,Y)$$

Note: Generally for *opa()*, $OSS(X1, X2) \neq OSS(X2, X1)$.

In contrast to *opa()*, *gpa()* is symmetric for the input matrices in that $G(X1, X2) = G(X2, X1)$.

See also:

rotate(), *gpa()*

Parameters

- **source** (*array-like*) – A (p,k)-shaped landmark set corresponding to the source shape.
- **target** (*array-like*) – A (p,k)-shaped landmark set corresponding to the target shape.
- **do_scaling** (*bool, optional*) – Flag indicating whether the best alignment should also find the optimal β that minimizes $D^2_{\mathtt{OPA}}$. The default value of *do_scaling* is False, which means $\beta = 1$, or in other words, *source* will not be scaled.
- **no_reflect** (*bool, optional*) – Flag indicating whether the best alignment should exclude reflection (default is False, which means reflection will be used if it achieves better alignment).

Returns

result –

aligned: numpy.ndarray A (p,k)-shaped landmark set consisting of the *source* landmarks aligned to the *target*.

b: numpy.float64 or int A number representing the scaling factor β by which *source* is scaled.

R: numpy.ndarray A (k,k)-shaped array representing the right rotation matrix R by which *source* is rotated.

c: numpy.ndarray A (k,)-shaped array representing the displacement γ between the centroids of *target* and the scaled+rotated *source*.

oss: numpy.float64 This number represents the Ordinary (Procrustes) Sum of Squares, which is the minimum of $D^2_{\texttt{OPA}}$. Essentially, the *oss* is the result of plugging in the optimal β, R and γ into the $D^2_{\texttt{OPA}}$ objective.

oss_stdized: numpy.float64 This number is the Ordinary Sum of Squares *oss*, divided by the squared norm of the centered target matrix. Loosely speaking it is a kind of "normalization" or "relativization" of the disparity in the *source* and *target* that is captured by the *oss*.

Return type dict

morphops.procrustes.**remove_position**(*lmks*, *position=None*)

If *position* is *None*, `remove_position()` translates *lmks* such that `get_position()` of *translated_lmks* is the origin. Else it is the (`get_position()` of *lmks*) - *position*.

Parameters **lmks** (*array-like*) – One of the following

- **Single specimen** A (p,k) array of p landmarks in k dimensions for one specimen.
- **n specimens** A (n,p,k) array of n landmark sets for n specimens, each having p landmarks in k dimensions.

Returns

translated_lmks –

- **Single specimen** If *lmks* is (p,k)-shaped, *translated_lmks* is (p,k)-shaped such that the centroid of *translated_lmks* + *position* = centroid of *lmks*. When *position* is None, it is taken to be the centroid of *lmks*, which means *translated_lmks* is at the origin.
- **n specimens** If *lmks* is (n,p,k)-shaped, *translated_lmks* is (n,p,k)-shaped such that the i-th element of *translated_lmks* is related to the i-th specimen of *lmks* by a translation calculated as per the single specimen case.

Return type numpy.ndarray

morphops.procrustes.**remove_scale**(*lmks*, *scale=None*)

If *scale* is *None*, `remove_scale()` scales *lmks* such that `get_scale()` of *scaled_lmks* is 1. Else it is (`get_scale()` of *lmks*)/*scale*.

Note: *lmks* is not assumed to have been pre-centered. To pre-center *lmks* you can call `remove_position()` on *lmks* before applying *remove_scale*.

Parameters **lmks** (*array-like*) – One of the following

- **Single specimen** A (p,k) array of p landmarks in k dimensions for one specimen.

- **n specimens** A (n,p,k) array of n landmark sets for n specimens, each having p landmarks in k dimensions.

Returns

scaled_lmks –

- **Single specimen** If *lmks* is (p,k)-shaped, *scaled_lmks* is (p,k)-shaped such that the norm of *scaled_lmks* x *scale* = norm of *lmks*. When *scale* is *None*, it is taken to be the norm of *lmks*, which means *scaled_lmks* has norm 1.
- **n specimens** If *lmks* is (n,p,k)-shaped, *scaled_lmks* is (n,p,k)-shaped such that the i-th element of *scaled_lmks* is related to the i-th specimen of *lmks* by a scaling calculated as per the single specimen case.

Return type numpy.ndarray

morphops.procrustes.**rotate**(*source*, *target*, *no_reflect=False*)

Rotates the landmark set *source* so as to minimize its sum of squared interlandmark distances to *target*.

Say X='source' and Y='target'. By default `rotate()` tries to find

$$\underset{R \in O(k)}{\operatorname{argmin}} \|Y - XR\|^2$$

That is, if *no_reflect* is *False*, `rotate()` might possibly reflect X if it would achieve better alignment to Y. This behavior can be switched off by setting *no_reflect* to *True*, in which case X will be aligned to Y using a pure rotation $R \in SO(k)$.

References

1. Sorkine-Hornung, Olga, and Michael Rabinovich. "Least-squares rigid motion using svd." no 3 (2017): 1-5. I found a pdf here.

Parameters

- **source** (*array-like*) – A (p,k)-shaped landmark set corresponding to the source shape.
- **target** (*array-like*) – A (p,k)-shaped landmark set corresponding to the target shape.
- **no_reflect** (*bool, optional*) – Flag indicating whether the best alignment should exclude reflection (default is False, which means reflection will be used if it achieves better alignment).

Returns

result –

aligned: numpy.ndarray A (p,k)-shaped landmark set consisting of the *source* landmarks rotated to the *target*.

R: numpy.ndarray A (k,k)-shaped array representing the right rotation matrix by which *source* is rotated.

D: numpy.ndarray A (k,)-shaped array representing the diagonal matrix of the SVD of np.dot(target.T, source).

Return type dict

3.1.2 morphops.tps

Provides thin-plate splines related operations and algorithms.

Given two sets of points, the thin-plate spline can interpolate from one to the other in a manner that minimizes the "integral bending norm"[bookstein89].

Importantly, it has a remarkable connection to Kendall's shape space in the following way: The non-zero eigenvectors of the bending energy matrix form an orthonormal basis in the tangent space of shape coordinates [bookstein96].

References

`morphops.tps.`**`K_matrix`**(*X*, *Y=None*)

Calculates the upper-right (p,p) submatrix of the (p+k+1,p+k+1)-shaped L matrix.

Parameters

- **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions.
- **Y** (*(m,2) or (m,3) shaped array-like, optional*) – A (m,k) array of p points in k=2 or k=3 dimensions. *Y* must have the same k as *X*.

  If *Y* is *None*, it is just set to *X*.

Returns

K – A (p,p) array where the element at [i,j] is $U(\|X_i - Y_j\|)$. The definition of U depends on k.

In particular, if k = 2, then $U(r) = r^2 \log(r^2)$, else $U(r) = r$.

Note: Using $\alpha U(r)$ instead of $U(r)$ for some $\alpha \in \mathbb{R}$ will not change the calculated spline. Simple block matrix inverse formulae show that when calculating L^{-1} for the spline using $\alpha U(r)$, the non-uniform coefficients multiplied to the U terms will be scaled by $\frac{1}{\alpha}$ while the uniform coefficients will stay the same.

Return type np.ndarray

`morphops.tps.`**`L_matrix`**(*X*)

Makes the (p+k+1,p+k+1)-shaped L matrix that gets inverted when calculating the thin-plate spline "from" *X*.

Parameters **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p landmarks in k=2 or k=3 dimensions for one specimen.

Returns **L** – A (p+k+1,p+k+1) array of the form [[K | P][P.T | 0]].

Return type np.ndarray

`morphops.tps.`**`P_matrix`**(*X*)

Makes the minor diagonal submatrix P of the (p+k+1,p+k+1)-shaped L matrix.

Basically just stacks a column of 1s before the coordinate columns in *X*.

Parameters **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions.

Returns **P** – A (p,k+1) array, which is 1 in the first column, and exactly *X* in the remaining columns.

Return type np.ndarray

`morphops.tps.`**`bending_energy_matrix`**(*X*)

Returns the upper right (pxp) submatrix of L^(-1).

Parameters **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p landmarks in k=2 or k=3 dimensions for one specimen.

Returns **L_inv** – The upper right (p,p) submatrix of the inverse of the *L_matrix* of *X*.

Return type np.ndarray

morphops.tps.**tps_coefs**(*X*, *Y*)

Finds the thin-plate spline coefficients for the thin-plate spline function that interpolates from X to Y.

Parameters

- **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions.
- **Y** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions. *Y* must have the same shape as *X*.

Returns

- **W** (*np.ndarray*) – A (p,k) array of weights for the non-affine part of the spline.
- **A** (*np.ndarray*) – A (k+1,k) array of weights for the affine part of the spline.

morphops.tps.**tps_warp**(*X*, *Y*, *pts*)

Maps points *pts* to their image under the thin-plate spline function generated by *tps_coefs()* of *X* and *Y*.

Parameters

- **X** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions.
- **Y** (*(p,2) or (p,3) shaped array-like*) – A (p,k) array of p points in k=2 or k=3 dimensions. *Y* must have the same shape as *X*.
- **pts** (*(m,2) or (m,3) shaped array-like, optional*) – A (m,k) array of m points in k=2 or k=3 dimensions. *pts* must have the same coordinate dimensions k as *X*.

Returns **warped_pts** – A (m,k) array of points corresponding to the image of *pts* under the thin-plate spline produced by *X*, *Y*.

Return type (m,2) or (m,3) shaped array-like, optional

3.1.3 morphops.io

Provides IO functions to read from and write to files in common landmark data file formats.

exception morphops.io.**MopsFileReadError**

Bases: Exception

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

exception morphops.io.**MopsFileWriteError**

Bases: Exception

with_traceback()

Exception.with_traceback(tb) – set self.__traceback__ to tb and return self.

morphops.io.**read_dta**(*filename*)

Reads *.dta files, as written by the IDAV Landmark Editor.

dta files typically have the following structure.

1. Few comment lines. Comment lines start with a quotation mark (' or ").

2. A header with structure "1 nL pk 1 9999 Dim=k". Here
    1. n is the number of specimens or number of landmark sets
    2. L in "nL" indicates that the file has specimen labels - assumed true
    3. p is the number of landmarks per landmark set
    4. k is the number of coordinates of each landmark (usually 2 or 3)

    The "1 9999" are ignored (but expected to exist) when reading. This is because those two numbers are a misapplication of the NTS format, which the DTA format is based on. Per the NTS format, the interpretation of the "1 9999" is that the file has missing data indicated by 9999. DTA files always contain the "1 9999" numbers, regardless of whether the file actually has missing data.
3. n lines, each corresponding to the label of 1 specimen.
4. n blocks of p lines. Each line contains k numbers. These correspond to p k-D landmarks in each of the n specimens specified in the order of appearance of their names in the preceding section.

morphops.io.**write_dta**(*filename*, *lmk_sets*, *names=[]*)
Writes *.dta files, as written by the IDAV Landmark Editor.

See also:

read_dta() For an explanation of the *.dta format.

3.1.4 morphops.lmk_util

Provides common functions used in the module.

morphops.lmk_util.**distance_matrix**(*X*, *Y*)
For (p1,k)-shaped X and (p2,k)-shaped Y, returns the (p1,p2) matrix where the element at [i,j] is the distance between X[i,:] and Y[j,:].

morphops.lmk_util.**num_coords**(*X*)
Returns the number of coordinates per landmark k in *X*.

X can be

- a 1-D tensor of shape (k,) corresponding to a landmark point having k coordinates.
- a 2-D tensor of shape (p,k) corresponding to a landmark set of p landmarks, each having k coordinates, or
- a 3-D tensor of shape (n,p,k) corresponding to a set of n landmark sets, each containing p landmarks, each having k coordinates.

morphops.lmk_util.**num_lmk_sets**(*X*)
Returns the number of landmark sets n in *X*.

X must be a 3-D tensor of shape (n,p,k) corresponding to a set of n landmark sets.

morphops.lmk_util.**num_lmks**(*X*)
Returns the number of landmarks per set p in *X*.

X can be

- a 2-D tensor of shape (p,k) corresponding to a landmark set of p landmarks, or
- a 3-D tensor of shape (n,p,k) corresponding to a set of n landmark sets, each containing p landmarks.

morphops.lmk_util.**ssqd**(*X*)
Returns the average sum of squared norms of pairwise differences between all lmk sets in X.

morphops.lmk_util.**transpose**(*X*)
Swaps the last two axes of a N-D tensor.

So for a 2-D matrix, this returns the transpose. For a 3-D tensor of length *n*, this returns the array of *n* transposed matrices.

morphops.**VERSION = '0.1.13'**
The version of this module.

3.2 Changelog

Morphops implements common operations and algorithms for Geometric Morphometrics, in Python 3.

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog.

Contributors to each release are listed in alphabetical order by first name. List entries are sorted in descending chronological order.

3.2.1 Unreleased

Added

- This project now keeps a changelog.

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Bibliography

[drymar] Dryden, I.L. and Mardia, K.V., 1998. Statistical shape analysis.

[tenb] Ten Berge, J.M., 1977. Orthogonal Procrustes rotation for two or more matrices. Psychometrika, 42(2), pp.267-276.

[rohlf] Rohlf, F.J., 1999. Shape statistics: Procrustes superimpositions and tangent spaces. Journal of Classification, 16(2), pp.197-223.

[bookstein89] Bookstein, F.L., 1989. Principal warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on pattern analysis and machine intelligence, 11(6), pp.567-585.

[bookstein96] Bookstein, F.L., 1996. Biometrics, biomathematics and the morphometric synthesis. Bulletin of mathematical biology, 58(2), p.313.

Python Module Index

m

Index