Package 'Momocs'

February 19, 2015

Type Package

Title Shape Analysis of Outlines

Version 0.2-6

Date 2014-03-07

Author Vincent Bonhomme, Sandrine Picq, Julien Claude with suggestions and contributions from Cedric Gaucherel, Sarah Ivorra, Ricardo Kriebel, Neus Martinez, Marcelo Reginato, Evan Saitta, Norbert Telmon, Asher Wishkerman.

Maintainer Vincent Bonhomme <bonhomme.vincent@gmail.com>

Description Momocs is intended to ease and popularize shape analysis of outlines (especially using elliptical Fourier analysis). It mostly hinges on the functions developed in Morphometrics with R (Claude, 2008). From outline extraction of images and elliptical Fourier calculation to multivariate analysis and the visualization of transformations within the morphological space, Momocs provides a complete and convenient toolkit to specialists within every field that are, or may be, interested in morphological comparisons of outlines.

License GPL (>= 2)

URL http://www.vincentbonhomme.fr/Momocs

Depends R(>= 2.15.0)

Imports methods, ade4, sp, shapes, jpeg, spdep, ape

Collate global.R Coe.R Coo.R

NeedsCompilation no

Repository CRAN

Date/Publication 2014-03-07 11:58:40

R topics documented:

A.mshape 4
A.plot 5
A.points 6

A.segments . . . 7
a2l . . . 8
baseline(Coo) . . . 8
chc2Coo . . . 9
chc2pix . . . 10
Class: Coe . . . 11
Class: Coo . . . 12
clust . . . 13
Coe2nef . . . 13
coeff.sel . . . 14
coeff.split . . . 15
Color palettes . . . 16
coo.align . . . 17
coo.baseline . . . 17
coo.center . . . 18
coo.centpos . . . 19
coo.centsize . . . 19
coo.close . . . 20
coo.draw . . . 21
coo.force2close . . . 22
coo.ldk . . . 23
coo.list.panel . . . 23
coo.oscillo . . . 24
coo.perim . . . 25
coo.perim.pts . . . 26
coo.plot . . . 27
coo.rotate . . . 28
coo.rotate.center . . . 29
coo.sample . . . 30
coo.sample.int . . . 30
coo.sample.rr . . . 31
coo.scale . . . 32
coo.slide . . . 33
coo.smooth . . . 33
coo.template . . . 34
coo.trans . . . 35
coo.unclose . . . 36
Coo2chc . . . 36
cooLandmarks(Coo) . . . 37
dataset: bot . . . 38
dataset: hearts . . . 39
dataset: mosquito . . . 39
dataset: trilo . . . 40
defLandmarks . . . 41
dev.plot . . . 41
dev.segments . . . 42
diapo(Coo) . . . 43
dudi.plot . . . 44

ed . . . 47
edi . . . 48
edm . . . 49
edm.nearest . . . 50
ef.amplify . . . 51
eFourier . . . 52
efourier . . . 53
efourier.i . . . 55
efourier.norm . . . 56
efourier.shape . . . 58
ellipse.par . . . 59
ellpar . . . 60
harm.pow . . . 61
hcontrib . . . 62
hpow . . . 63
hqual . . . 64
hquant . . . 65
import.jpg . . . 66
import.txt . . . 67
is.closed . . . 68
l2a . . . 69
l2m . . . 70
m2l . . . 71
manova.Coe . . . 71
meanShapes . . . 72
Momocs Package . . . 73
morpho.space . . . 74
nef2Coe . . . 75
panel(Coo) . . . 76
PC.contrib . . . 77
pca . . . 78
pca2shp . . . 80
pix2chc . . . 81
procGPAlign(Coo) . . . 82
Ptolemy . . . 83
rFourier . . . 84
rfourier . . . 85
rfourier.i . . . 86
rfourier.shape . . . 87
smooth.qual . . . 89
stack(Coo) . . . 90
tFourier . . . 91
tfourier . . . 91
tfourier.i . . . 93
tfourier.shape . . . 94
tps.arr . . . 95
tps.grid . . . 96
tps.iso . . . 97

tps2d 98
vecs.param 99

Index **100**

`A.mshape` *Calculates the mean shape of an array of landmarks.*

Description

`A.mshape` returns the mean shape of an array of landmarks when provided with landmarks as rows, $(x; y)$ coordinates as columns, and individuals as the third dimension of the array.

Usage

```
A.mshape(A)
```

Arguments

`A`	An `array` of $(x; y)$ coordinates, typically, landmarks.

Value

Returns a matrix of $(x; y)$ coordinates.

See Also

A.points, A.segments, A.plot coo.plot, and the others coo.utilities.

Examples

```
## Not run:
data(gorf.dat) # we import gorf.data from shapes package
A.plot(gorf.dat, pch=20)
m <- A.mshape(gorf.dat)
points(m, pch=5)

## End(Not run)
```

`A.plot` *Plots an array of homologous coordinates, typically landmarks.*

Description

`A.plot` is used to plot an array of landmarks when provided with landmarks as rows, $(x; y)$ coordinates as columns, and individuals as the third dimension of the array.

Usage

```
A.plot(A, col, palette = col.summer, xlim, ylim,
    border.col = NA, border.lty = 2, pch = 3, cex = 1)
```

Arguments

`A`	An `array` of $(x; y)$ coordinates, typically, landmarks.
`col`	A vector of colors to use for drawing landmarks.
`palette`	If `col` is not provided, then the color palette to use for landmarks.
`xlim`	A vector of length 2 specifying a custom `xlim` of the ploting area.
`ylim`	A vector of length 2 specifying a custom `ylim` of the ploting area.
`border.col`	A color for these segments.
`border.lty`	A `lty` for these segments.
`pch`	A `pch` for points.
`cex`	A `cex` for points.

Value

No returned value.

See Also

A.points, A.segments, A.mshape coo.plot, and the others coo.utilities.

Examples

```
## Not run:
data(gorf.dat) # we import gorf.data from shapes package
A.plot(gorf.dat, pch=20)

## End(Not run)
```

A.points *Plots the points that corresponds to landmarks provided as an array.*

Description

A.points displays landmarks when provided as an array with landmarks as rows, $(x; y)$ coordinates as columns, and individuals as the third dimension of the array.

Usage

```
A.points(A, col, palette=col.summer, pch=3, cex=1)
```

Arguments

A	An array of $(x; y)$ coordinates, typically, landmarks.
col	A vector of colors to use for drawing landmarks.
palette	If col is not provided, then the color palette to use for landmarks.
pch	A pch for points.
cex	A cex for points.

Value

No returned value.

See Also

A.segments, A.mshape, A.plot coo.plot, and the others coo.utilities.

Examples

```
## Not run:
data(gorf.dat) # we import gorf.data from shapes package
A.points(gorf.dat)

## End(Not run)
```

`A.segments` *Utilities to manipulate arrays of homologous coordinates, typically landmarks.*

Description

`A.segments` draws interlandmarks segments as an array with landmarks as rows, $(x; y)$ coordinates as columns, and individuals as the third dimension of the array.

Usage

```
A.segments(A, col=NA, border.col="grey60", border.lty=1)
```

Arguments

`A`	An `array` of $(x; y)$ coordinates, typically, landmarks.
`col`	A vector of colors to use for drawing landmarks.
`border.col`	A color for these segments.
`border.lty`	A `lty` for these segments.

Value

No returned value.

See Also

A.plot, A.points, A.mshape coo.plot, and the others coo.utilities.

Examples

```
## Not run:
data(gorf.dat) # we import gorf.data from shapes package
A.segments(gorf.dat, border.lty=2, border.col="grey95")
m <- A.mshape(gorf.dat)
points(m, pch=5)

## End(Not run)
```

a2l *Converts an array of coordinates to a list.*

Description

a2l converts an array of coordinates into a list of 2-cols matrices.

Usage

```
a2l(a)
```

Arguments

a	An array of coordinates.

Value

A list with 2-cols matrices of (x; y) coordinates.

See Also

l2a

Examples

```
#data(gorf.dat) # we import gorf.data from shapes package
#l <- a2l(gorf.dat)
#a <- l2a(l)
#A.plot(a)
```

baseline(Coo) *Re-register a* Coo *on a new baseline.*

Description

edm returns the euclidean distances between points

$$1 -> n$$

of two 2-col matrices of the same dimension. This function is used internally but may be of interest for other analyses.

Usage

```
baseline(Coo, ldk1=1, ldk2=2, t1=c(-0.5, 0), t2=c(0.5, 0))
```

Arguments

Coo	The Coo object from which to retrieve coordinates.
ldk1	index of the first reference landmark.
ldk2	index of the second reference landmark. See Examples.
t1	$(x; y)$ coordinates of the first target point.
t2	$(x; y)$ coordinates of the second target point.

Value

Returns a Coo registered in the new baseline.

See Also

ed, edm.nearest, dist.

Examples

```
def.par <- par(no.readonly = TRUE)
layout(matrix(1:2, 1, 2))
data(hearts)
stack(hearts, border="#1A1A1A22")
hearts2 <- baseline(hearts, 2, 4)
stack(hearts2, border="#1A1A1A22")
par(def.par)
```

---

chc2Coo — *Imports* .chc *files from SHAPE to* Coo *objects that can be used in Momocs.*

Description

chc2Coo converts .chc files, *e.g.* from the SHAPE suite (see Iwata & Ukai in the **References** below) that can then be used in Momocs.

Usage

```
chc2Coo(chc.path)
```

Arguments

chc.path	A character that indicates the path for the .chc file to convert.

Details

See pix2chc for an illustration of the functions behind and chain-coding of outlines.

Value

Returns a `Coo` object.

References

Freeman H. 1974. Computer processing of line-drawing images. ACM Computing Surveys (CSUR) 6: 57-97.

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

Kuhl FP, Giardina CR. 1982. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18: 236-258.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the `.chc` format.

See Also

Coe2nef and nef2Coe.

`chc2pix` *Converts chain-coded coordinates to a matrix of coordinates.*

Description

`chc2pix` converts chain-coded coordinates such as those used in the SHAPE suite (see Iwata in the **References** below) to a matrix of coordinates.

Usage

```
chc2pix(chc)
```

Arguments

`chc`	A `numeric` that corresponds to a chain-coded outline.

Details

Chain-code is a coding system for describing outlines in numbers from 0 to 7. Given the i^{th} pixel taken on an outline and considering its eight neighbors, the chain-code equivalent for describing the relative position of the $(i+1)^{th}$ pixel is a number from 0 (the next cell is eastward) to, counting counter clockwise, 7 (the next cell is south-eastward). See **References** and **Examples** below.

Value

Returns a 2-col `matrix` that corresponds to the $(x; y)$ coordinates encoded in the chain provided.

References

Freeman H. 1974. Computer processing of line-drawing images. ACM Computing Surveys (CSUR) 6: 57-97.

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

Kuhl FP, Giardina CR. 1982. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18: 236-258.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the `.chc` format.

See Also

pix2chc, for the reverse operation.

Examples

```
data(bot)
coo <- bot@coo[[1]]
chc <- pix2chc(coo)
coo.plot(chc2pix(chc))

# Illustration of chain coding
plot(NA, xlim=c(0, 3), ylim=c(0, 3), axes=FALSE, ann=FALSE, xaxs="i", yaxs="i")
title("Position of the next pixel and corresponding chain-code")
abline(h=0:3, v=0:3)
rect(1, 1, 2, 2, col="grey80")
text(1.5, 1.5, "Starting\npixel", cex=2)
text(x=c(2.5, 2.5, 1.5, rep(0.5, 3), 1.5, 2.5),
y=c(1.5, rep(2.5, 3), 1.5, rep(0.5, 3)), labels=0:7, cex=2)
```

Class: Coe *Class* "Coe"

Description

Objects of the class `Coe` aim at storing harmonic coefficients after a Fourier analysis, either eFourier, rFourier and tFourier.

Slots

`coe`: A `matrix` of harmonic coefficients.

`method`: A `character` that indicates the method used.

`fac`: `data.frame` defining the grouping structure, if any. Columns must be `factors`.

`names`: `character vector` to name every shape.

`nb.h`: `numeric` giving the number of harmonics.

Methods

See `showMethods(class="Coe")`.

Examples

```
## Working zone...
showClass("Coe")
```

Class: Coo *Class* "Coo"

Description

A class for handling morphometric datasets, *i.e.* lists of $(x; y)$ outline coordinates with or without a grouping structure.

Objects from the Class

New Coo-objects can be created with `new("Coo", ...)` or more directly using the `Coo` builder.

Slots

coo: `list` a list of shapes arranged in two columns coordinate matrices and that can have different lengths.

names: `character vector` to name every shape.

fac: `data.frame` defining the grouping structure, if any. Columns must be `factors`.

ldk: `list` defining the coordinates indices that will be considered as homologous points (not yet used in current version).

coo.nb: `numeric`. The number of shapes.

coo.len: a `vector` of `numeric` that contains the number of coordinates per shape.

coo.closed: a `vector` of `logical` that indicates if outlines are closed.

details: any additional details, such as licence and authors can be added in this `list`.

Methods

See `showMethods(class="Coo")`.

Examples

```
## Working zone...
showClass("Coo")
# an example of the Coo builder
data(bot)
list_of_coordinates <- bot@coo
Coo(list_of_coordinates)
```

clust *Hierarchical clustering on a matrix of coefficients*

Description

clust is a method for Coe objects to calculate and display hierarchical clustering of shapes based on the distance between their coefficients, after a distance matrix calculation based on dist.

Usage

```
clust(Coe, fac, method = "euclidean", type = "fan", palette = col.summer)
```

Arguments

Coe	The Coe object on which to perform hierarchical clustering.
fac	factor defining which column of the @fac slot to use for grouping, if any.
method	The distance method used by dist.
type	The plotting method used by plot.phylo.
palette	A color palette to use to plot the different groups, if any.

Value

Returns a dist object.

See Also

dist, hclust and plot.phylo.

Examples

```
data(bot)
botE <- eFourier(bot, nb.h=32)
clust(botE)
```

Coe2nef *Exports* Coe *objects to* .nef *files.*

Description

Coe2nef converts Coe objects to .nef files. This function is intended to ease data exchange between Momocs and SHAPE suite (see Iwata in the **References** below).

Usage

```
Coe2nef(Coe, file="nef.nef")
```

Arguments

Coe	A Coe object.
file	A character to specify where to write the .nef file.

Value

Writes a .nef file.

References

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the .nef format.

See Also

nef2Coe, for the reverse operation.

---

coeff.sel *Helps to select a given number of harmonics from a numerical vector.*

Description

coeff.sel helps to select a given number of harmonics by returning their indices when arranged as a numeric vector. For instance, harmonic coefficients are arranged in the @coeff slot of Coe-objects in that way:

$$A_1, \ldots, A_n, B_1, \ldots, B_n, C_1, \ldots, C_n, D_1, \ldots, D - n$$

after an elliptical Fourier analysis (see eFourier and efourier) while

$$C_n and D_n$$

harmonic are absent for radii variation and tangent angle approaches (see rfourier and tfourier respectively). . This function is used internally but might be of interest elwewhere.

Usage

```
coeff.sel(retain = 8, drop = 0, nb.h = 32, cph = 4)
```

Arguments

retain	numeric. The number of harmonics to retain.
drop	numeric. The number of harmonics to drop
nb.h	numeric. The maximum harmonic rank.
cph	numeric. Must be set to 2 for rfourier and tfourier were used.

Value

`coeff.sel` returns indices that can be used to select columns from an harmonic coefficient matrix. `coeff.split` returns a named list of coordinates.

Examples

```
coeff.sel(retain=8, drop=0, nb.h=12)        #efourier
coeff.sel(retain=8, drop=0, nb.h=12, cph=2) #r/tfourier

# if you want to export the matrix of coefficients but only the first 6 colums.
data(bot)
bot.f <- eFourier(bot, nb.h=12)
bot.f@coe[, coeff.sel(6, 0, 12)]
```

`coeff.split` *Converts a numerical description of harmonic coefficients to a named list.*

Description

`coeff.split` returns a named list of coordinates from a vector of harmonic coefficients. For instance, harmonic coefficients are arranged in the `@coeff` slot of `Coe`-objects in that way:

$$A_1, \ldots, A_n, B_1, \ldots, B_n, C_1, \ldots, C_n, D_1, \ldots, D - n$$

after an elliptical Fourier analysis (see eFourier and efourier) while

$$C_n and D_n$$

harmonic are absent for radii variation and tangent angle approaches (see rfourier and tfourier respectively). This function is used internally but might be of interest elwewhere.

Usage

```
coeff.split(cs, nb.h = 8, cph = 4)
```

Arguments

`cs`	A `vector` of harmonic coefficients.
`nb.h`	`numeric`. The maximum harmonic rank.
`cph`	`numeric`. Must be set to 2 for `rfourier` and `tfourier` were used.

Value

Returns a named list of coordinates.

Examples

```
coeff.split(1:128, nb.h=32, cph=4) # efourier
coeff.split(1:64, nb.h=32, cph=2)  # t/r fourier
```

Color palettes *Some color palettes.*

Description

col.summer, col.india, col.sari and col.gallus are standard color palettes, obtained with colorRampPalette. col.bw generates a palette from black to white. col.bcol and col.wcol are used to generate a palette from one color to black or white, respectively.

Usage

```
col.summer(n)
col.sari(n)
col.gallus(n)
col.blackgallus(n)
col.india(n)
col.bw(n)
col.bcol(col.hex)
col.wcol(col.hex)
```

Arguments

n	integer. The number of colors to create.
col.hex	An hexadecimal character string coding for a color.

Value

A vector of hexadecimal color that can be passed to many methods and functions of Momocs and also used as any palette created with colorRampPalette.

Examples

```
pie(rep(1, 12), col=col.summer(12), main="col.summer")
pie(rep(1, 12), col=col.india(12),  main="col.india")
pie(rep(1, 12), col=col.sari(12),   main="col.sari")
pie(rep(1, 12), col=col.gallus(12), main="col.gallus")
pie(rep(1, 12), col=col.blackgallus(12), main="col.blackgallus")
pie(rep(1, 12), col=col.bw(12), main="col.bw")
pie(rep(1, 12), col=col.bcol(col.hex="#FF6600")(12), main="col.bcol and some orange")
pie(rep(1, 12), col=col.wcol(col.hex="#000080")(12), main="col.bcol and some blue")
barplot(1:33, col=col.summer(33))
```

coo.align *Aligns a list or matrix of coordinates.*

Description

coo.align aligns coo using its best fitting ellipse.

Usage

```
coo.align(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns a matrix of (x; y)coordinates.

Examples

```
def.par <- par(no.readonly = TRUE)
layout(matrix(1:2, 1, 2))
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo, main="A shape")
coo.plot(coo.align(coo), border="red", col=NA, main="An aligned shape")
par(def.par)
```

coo.baseline *Re-register a list or matrix of coordinates.*

Description

coo.baseline registers coo on a (new) baseline, *e.g.* Bookstein's coordinates.

Usage

```
coo.baseline(coo, ldk1 = 1, ldk2 = 2, t1=c(-0.5, 0), t2=c(0.5, 0))
```

Arguments

coo	A list or a matrix of coordinates.
ldk1	Row index of the first reference landmark.
ldk2	Row index of the second reference landmark.
t1	(x; y) coordinates of the first target point.
t2	(x; y) coordinates of the second target point.

Details

coo.baseline returns by default Bookstein's coordinates.

Value

Returns a matrix of (x; y)coordinates.

Examples

```
def.par <- par(no.readonly = TRUE)
layout(matrix(1:2, 1, 2))
data(bot)
coo <- bot@coo[[1]]
coo <- coo.sample(coo, 12)
coo.plot(coo, main="Some landmarks", points=TRUE)
coo.plot(coo.baseline(coo), border="red", col=NA,
    points=TRUE, main="Reregistered using Bookstein's coordiantes")
abline(v=c(-0.5, 0.5), col="grey60", lty=2)
box()
par(def.par)
```

---

coo.center *Centers a list or matrix of coordinates.*

Description

coo.center centers the coo's centroid on the origin.

Usage

```
coo.center(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns a matrix of (x; y)coordinates.

Examples

```
def.par <- par(no.readonly = TRUE)
layout(matrix(1:2, 1, 2))
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo, main="A shape")
abline(v=0, h=0, col="grey60", lty=2)
coo.plot(coo.center(coo), border="red", col=NA, main="A centered shape")
```

```
abline(v=0, h=0, col="grey60", lty=2)
par(def.par)
```

`coo.centpos` *Calculates the position of the centroid of a list or a matrix of coordinates.*

Description

`coo.centpos` returns the centroid position of the shape that is the mean of x and y coordinates.

Usage

```
coo.centpos(coo)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.

Value

Returns the `(x; y)` coordinates of the centroid.

See Also

coo.centsize.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo, cent=FALSE) # we do not plot the centroid
cent <- coo.centpos(coo)
points(cent[1], cent[2], pch=3, cex=5, col="red")
```

`coo.centsize` *Calculates the centroid size of a list or matrix of coordinates.*

Description

`coo.centsize` returns the centroid size of the shape, *i.e.* the square root of the sum of squared distances from each point to the centroid of the shape.

Usage

```
coo.centsize(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns the centroid size.

See Also

coo.centpos.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.centsize(coo)
coo.centsize(coo.scale(coo)) # by default coo.scale scales to centroid size.
```

---

coo.close *Closes a list or matrix of coordinates.*

Description

coo.close closes coo, *i.e.* makes the last coordinates be the first than the first.

Usage

```
coo.close(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns a matrix of (x; y)coordinates.

See Also

coo.unclose and is.closed

Examples

```
## Not run:
data(gorf.dat) # we import gorf.data from shapes package
coo <- gorf.dat[,,1]
is.closed(coo)
coo.c <- coo.close(coo)
is.closed(coo.c)
coo.cu <- coo.unclose(coo)
is.closed(coo.cu)

## End(Not run)
```

`coo.draw` *Adds a single outline on the current plot.*

Description

`coo.draw` is a light version of coo.plot that simply adds a shape on the active plot.

Usage

```
coo.draw(coo = NA, col = "#70809033",
         border = "#708090EE", lwd=1, lty = 1, points = FALSE,
         first.point=TRUE, centroid = FALSE,
         points.col = border, pch = 20, cex = 0.25,  ...)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.
`col`	A color to fill the shape polygon.
`border`	A color for the shape border.
`lwd`	The `lwd` for drawing shapes.
`lty`	The `lty` for drawing shapes.
`points`	`logical`. Whether to display points. If missing and number of points is < 100, then points are plotted.
`first.point`	`logical`. Whether to display the first point.
`centroid`	`logical`. Whether to display centroid.
`points.col`	The color for plotting points.
`pch`	The `pch` for points.
`cex`	The `cex` for points.
`...`	Additional parameters for drawing the first point.

Value

No returned value.

See Also

coo.plot.

Examples

```
data(bot)
coo.plot(bot@coo[[1]])
coo.plot(bot@coo[[2]], lwd=1.2)
```

`coo.force2close` *Forces a list or matrix or coordinates to close.*

Description

`coo.force2close` force one coo to close, i.e. the difference $(x_n - x_1)/n$ is added to each x-coordinates and similarly for y-coordinates.

Usage

```
coo.force2close(coo)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.

Value

Returns a matrix of `(x; y)`coordinates.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo, ylim=c(-200, 1200), xlim=c(0, 500), main="A half bottle forced to close")
coo.draw(coo[1:1500,], border="blue", col=NA)
coo.draw(coo.force2close(coo[1:1500, ]), border="red", col=NA)
```

coo.ldk *Utility to identify landmarks on a matrix or list of coordinates.*

Description

coo.ldk allows user to click to identify the closest points that belong to the outline. Typically used to define landmarks.

Usage

```
coo.ldk(coo, nb.ldk)
```

Arguments

coo	A list or a matrix of coordinates.
nb.ldk	integer. Indicates the number of landmarks to define.

Value

A vector of the coo rows indices that coorespond to the landmarks identified.

See Also

defLandmarks.

Examples

```
## Not run:
data(bot)
coo <- bot@coo[[1]]
coo.ldk(coo, 2) # you have to click now !

## End(Not run)
```

coo.list.panel *Plots sets of shapes.*

Description

coo.list.panel plots a list of shapes if passed with a list of coordinates. Outlines are templated and on the same graphical window with the help of coo.template.

Usage

```
coo.list.panel(coo.list, dim,
               byrow = TRUE, fromtop = TRUE,
               mar = rep(0, 4), cols, borders, density = NULL, angle = 45)
```

Arguments

coo.list	A list of coordinates, such as those in the @coo slot of Coo objects.
dim	A vector of the form (nb.row, nb.cols) to specify the panel display. If missing, shapes are arranged in a square.
byrow	logical. Whether to succesive shape by row or by col.
fromtop	logical. Whether to display shapes from the top of the plotting region.
mar	A vector to define margins.
cols	A vector of colors to fill shapes.
borders	A vector of colors to draw shape borders.
density	A vector for density of shading lines. See polygon
angle	A vector for shading lines.

Value

Returns (invisibly) a data.frame with position of shapes that can be used for other sophisticated plotting design.

See Also

coo.plot and coo.template.

Examples

```
data(bot)
coo.list.panel(bot@coo)
x <- coo.list.panel(bot@coo)
x # positions of shapes returned invisibly
# axis(1) ; axis(2) # that's a single graphical window ;)
```

---

coo.oscillo *Momocs' "oscilloscope" for periodic functions.*

Description

Shape analysis deals with curve fitting, whether $x(t)$ and $y(t)$ positions along the curvilinear abscissa or radius/tangent angle variation. We may need to represent these single or double periodic functions that are ajusted by Fourier-based method. coo.oscillo and coo.oscillo1 compute and provide standardized plot when given a matrix of coordinates or a vector, respectively. These functions are mainly used for development purpose but are included in the package.

Usage

```
coo.oscillo(coo, method = c("d0", "di")[1], plot = TRUE, rug = TRUE,
    legend = TRUE, cols = col.gallus(2), ref=FALSE, ref.nb=8, ...)

coo.oscillo1(coo, method = c("d0", "di")[1], plot = TRUE, rug = TRUE,
    legend = TRUE, cols = col.gallus(1),
    xlab = "Points sampled along the outline",  ylab = "Deviation", ...)
```

Arguments

coo	A list or a matrix of coordinates.
method	character. Whether to calculate differences with the first point ("d0") or the previous ("di"), ie the derivate.
plot	Whether to plot the results.
rug	logical. Whether to display a pseudo rug, that indicate if the derivate is positive.
legend	logical. Whether to add a legend.
cols	A vector of two (for coo.oscillo) or a single (coo.oscillo1) color for lines.
ref	logical. Whether to display the original shape besides the oscillo.
ref.nb	integer. The number or reference points, sampled equidistantly along the curvilinear abscissa and added on the oscillo curves.
xlab	character. Alternate label for x-axis.
ylab	character. Alternate label for y-axis.
...	Additional parameters than can be passed to lines.

Value

Returns a list with two or one component(s), giving the difference calculated.

Examples

```
data(bot)
coo.oscillo(bot@coo[[1]], lty=2)
coo.oscillo1(tfourier(bot@coo[[1]], 24)$phi)
```

---

coo.perim *Calculates the perimeter of a list or matrix of coordinates.*

Description

coo.perim returns the sum of the euclidean distances between all successive (x; y) coordinates.

Usage

```
coo.perim(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns the perimeter length.

Examples

```
data(bot)
coo.perim(bot@coo[[1]])
```

---

coo.perim.pts *Calculates the euclidean distance between points of a list or matrix of coordinates.*

Description

coo.perim.pts returns the euclidean distances between all successive (x; y) coordinates.

Usage

```
coo.perim.pts(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns a vector of euclidean distances.

See Also

ed, coo.perim.

Examples

```
data(bot)
coo.perim.pts(bot@coo[[1]])
```

`coo.plot` *Plots a single outline.*

Description

`coo.plot` is a simple wrapper for plotting shapes. It basically tunes standard plot function to display single shapes as polygons, within a standardised plotting area.

Usage

```
coo.plot(coo=NA, col="#F5F5F5",
         border="#1A1A1A", lwd=1, lty = 1, xlim=c(-1, 1), ylim=c(-1, 1),
         points=FALSE, first.point=TRUE, centroid=TRUE,
         points.col=border, pch=1, cex=0.8, main, ...)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.
`col`	A color to fill the shape polygon.
`border`	A color for the shape border.
`lwd`	The `lwd` for drawing shapes.
`lty`	The `lty` for drawing shapes.
`xlim`	If `coo.plot` is called and `coo` is missing, then a vector of length 2 specifying the `ylim` of the ploting area.
`ylim`	If `coo.plot` is called and `coo` is missing, then a vector of length 2 specifying the `ylim` of the ploting area.
`points`	`logical`. Whether to display points. If missing and number of points is < 100, then points are plotted.
`first.point`	`logical`. Whether to display the first point.
`centroid`	`logical`. Whether to display centroid.
`points.col`	The color for plotting points.
`pch`	The `pch` for points.
`cex`	The `cex` for points.
`main`	`character`. A title for the plot.
`...`	Additional parameters for drawing the first point.

Value

No returned value.

See Also

coo.draw.

Examples

```
data(bot)
coo.plot(bot@coo[[1]])
```

`coo.rotate` *Rotates a list or matrix of coordinates.*

Description

`coo.rotate` rotates (counter-clockwise) `coo` with a `theta` angle (in radians) .

Usage

```
coo.rotate(coo, theta)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.
`theta`	`numeric`. The angle to rotate the shape.

Details

For those not familiar with linear mapping the providence is there: http://en.wikipedia.org/wiki/Linear_mapping

Value

A `list` with `x; y` components or a a matrix of `(x; y)`coordinates.

See Also

coo.rotate.center if one wants to rotate shapes with another center of rotation than the origin.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo,  main="The bottle revolution", xlim=c(-1.5e3, 1.5e3))
r <- seq(pi/6, 2*pi, pi/6)
cols <- col.summer(12)
for (i in seq(along=r)) {
  coo.draw(coo.rotate(coo, r[i]), border=cols[i], col=NA)
}
```

coo.rotate.center *Rotates a list or matrix of coordinates with any center of symmetry.*

Description

coo.rotate.center rotates (counter-clockwise) coo with a theta angle (in radians) and with the center of symmetry specified by center .

Usage

```
coo.rotate.center(coo, theta, center=c(0, 0))
```

Arguments

coo	A list or a matrix of coordinates.
theta	numeric. The angle to rotate the shape.
center	The (x; y) coordinates of the center of rotation.

Details

This function simply centers the shape, rotate it, and then "uncenter" it.

Value

A list with x; y components or a a matrix of (x; y)coordinates.

See Also

coo.rotate.

Examples

```
data(bot)
coo <- coo.scale(coo.center(bot@coo[[1]]))
coo.plot(coo,  main="The bottle's revolution", ylim=c(-10, 50))
r <- seq(pi/6, 2*pi, pi/6)
cols <- col.summer(12)
for (i in seq(along=r)) {
  coo.draw(coo.rotate.center(coo, r[i], center=c(0, 20)), border=cols[i], col=NA)
  }
abline(v=0, h=0, col="grey60", lty=2)
points(0, 20, pch=3, col="red")
```

coo.sample *Samples points along the curvilinear abscissa.*

Description

coo.sample samples n points in coo along the curvilinear abscissa.

Usage

```
coo.sample(coo, n)
```

Arguments

coo	A list or a matrix of coordinates.
n	integer. The number of points to sample.

Details

So far points are just sampled along the coo provided) and not along the "true" curvilinear abscissa, but in most cases, differences should be very small.

Value

Returns a matrix of (x; y)coordinates.

See Also

coo.sample.rr.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo,  main="24 points sampled along the outline")
points(coo.sample(coo, 24), col="red", pch=20)
```

coo.sample.int *Given an outline, interpolates points along the curvilinear abscissa.*

Description

coo.sample.int interpolates n points in coo along the curvilinear abscissa.

Usage

```
coo.sample.int(coo, n)
```

Arguments

coo	A list or a matrix of coordinates.
n	integer. The number of points to sample.

Details

This function circumvent the problem exposed in the Details section of coo.sample, *i.e.* points are here samples along the "true" curvilear abscissa and not based on seq.

Value

Returns a matrix of (x; y)coordinates.

See Also

coo.sample, coo.sample.rr.

Examples

```
data(bot)
coo <- coo.sample(bot@coo[[1]], 12)
coo.plot(coo, points=TRUE)
coo.draw(coo.sample.int(coo, 20), col="#B2222255", border=NA)
coo.draw(coo.sample(coo, 20),     col="#1874CD55", border=NA)
```

---

coo.sample.rr *Samples points with "equally spaced" angles.*

Description

coo.sample.rr samples n points in coo so that the radii departing from the coo's centroid are equals. This function uses polar coordinates and complex algebra.

Usage

```
coo.sample.rr(coo, n)
```

Arguments

coo	A list or a matrix of coordinates.
n	integer. The number of points to sample.

Value

Returns a list with components:

- $pixindices vector of radii indices;
- $radii vector of sampled radii lengths;
- $phase vector of phases;
- $coord coordinates of sampled points for a centered shape;
- $orig.coord coordinates of sampled points on the original shape;

See Also

coo.sample.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo,  main="24 points with equally spaced radii")
rad <- coo.sample.rr(coo, 24)$orig.coord
cent <- coo.centpos(coo)
segments(cent[1], cent[2], rad[, 1], rad[, 2], col=col.summer(24))
points(coo.sample.rr(coo, 24)$orig.coord, pch=20)
```

---

coo.scale *Scales a list or matrix of coordinates.*

Description

coo.scale resizes coo so that it can be included in a square of scale side.

Usage

```
coo.scale(coo, scale)
```

Arguments

coo	A list or a matrix of coordinates.
scale	The scale to use.

Value

Returns a matrix of (x; y)coordinates.

See Also

coo.template.

coo.slide *"Slides" a list or a matrix of coordinates.*

Description

coo.slide "slides" coo, *i.e.* makes the id1 become the first and change the others accordingly.

Usage

```
coo.slide(coo, id1)
```

Arguments

coo	A list or a matrix of coordinates.
id1	integer. Specifies the index of the coordinates that has to be defined as the first coordinate.

Examples

```
m <- matrix(1:10, 5, 2)
m
coo.slide(m, 3)
```

coo.smooth *Smoothes list and matrices of coordinates.*

Description

coo.smooth performs n smoothing iteration on coo.

Usage

```
coo.smooth(coo, n)
```

Arguments

coo	A list or a matrix of coordinates.
n	integer. The number of iterations to perform.

Value

Returns a matrix of (x; y)coordinates.

Examples

```
data(bot)
coo <- coo.sample(bot@coo[[1]], 100)
coo.plot(coo, points=TRUE, main="Take it not too smooth")
s <- seq(10, 1000, length=10)
cols <- col.summer(10)
for (i in seq(along=s)) {
  coo.draw(coo.smooth(coo, s[i]), col=NA, border=cols[i])
  }
```

---

coo.template *"Templates" list and matrix of coordinates.*

Description

`coo.template` returns `coo` so that the shape it is centered on the origin and inscribed in a size-side square, also centered on the origin; see coo.list.panel for an illustration of this function.

Usage

```
coo.template(coo, size)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.
`size`	`numeric`. Indicates the length of the side "inscribing" the shape.

Value

Returns a matrix of `(x; y)`coordinates.

See Also

coo.list.panel.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo.template(coo), xlim=c(-1, 1), ylim=c(-1, 1))
rect(-0.5, -0.5, 0.5, 0.5)

s <- 0.01
coo.plot(coo.template(coo, s))
rect(-s/2, -s/2, s/2, s/2)
```

coo.trans *Transles a list or a matrix of coordinates.*

Description

coo.trans translates coo by x and y on the two dimensions, respectively.

Usage

```
coo.trans(coo, x, y)
```

Arguments

coo	A list or a matrix of coordinates.
x	numeric. The x-axis translation.
y	numeric. The y-axis translation.

Value

Returns a matrix of (x; y)coordinates.

Examples

```
data(bot)
coo <- coo.scale(coo.center(bot@coo[[1]]))

tx <- seq(0, 10*pi, length=50)
ty <- sin(tx)*5
cols <- col.summer(50)

coo.plot(xlim=c(0, 10*pi), main="The bottle's wave")
lines(tx, ty, col="grey60", lty=2)
for (i in seq(along=tx)) {
  coo.draw(coo.trans(coo, tx[i], ty[i]), col=NA, border=cols[i])
}
```

coo.unclose *Uncloses a list or matrix of coordinates.*

Description

coo.unclose tests if coo is closed, then simply removes the last coordinate.

Usage

```
coo.unclose(coo)
```

Arguments

coo	A list or a matrix of coordinates.

Value

Returns a matrix of (x; y)coordinates.

See Also

coo.close and is.closed

Examples

```
## Not run:
data(gorf.dat)
coo <- gorf.dat[,,1]
is.closed(coo)
coo.c <- coo.close(coo)
is.closed(coo.c)
coo.cu <- coo.unclose(coo)
is.closed(coo.cu)

## End(Not run)
```

Coo2chc *Exports* Coo *objects to* .chc *files that can be used in SHAPE.*

Description

Coo2chc converts Coo objects into .chc files that can then be used in the SHAPE suite (see Iwata & Ukai in the **References** below).

Usage

```
Coo2chc(Coo, file="chc.chc")
```

Arguments

Coo	A Coo object.
file	A character to specify where to write the .chc file.

Details

See pix2chc for an illustration of the functions behind and chain-coding of outlines.

Value

Writes a .chc file.

References

Freeman H. 1974. Computer processing of line-drawing images. ACM Computing Surveys (CSUR) 6: 57-97.

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

Kuhl FP, Giardina CR. 1982. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18: 236-258.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the .chc format.

See Also

Coe2nef and nef2Coe.

---

cooLandmarks(Coo) *Retrieves landmarks coordinates from a* Coo *object.*

Description

Retrieves landmarks (if any) coordinates from a Coo object.

Usage

```
cooLandmarks(Coo)
```

Arguments

Coo	The Coo object.

Value

Returns an array of landmark coordiantes such as described in A.plot.

See Also

procGPAlign, defLandmarks, A.plot.

Examples

```
data(hearts)
ldks <- cooLandmarks(hearts)
A.plot(ldks)
```

`dataset: bot` *bot dataset*

Description

The `bot` dataset is a `Coo`-class object that contains outline coordinates of 20 beer and 20 whisky bottles.

Usage

```
data(bot)
```

Format

A `Coo`-object that contains in the slot `@coo`, the lists of $(x; y)$ coordinates and in the slot `@fac` the corresponding `factors`.

Source

Images have been grabbed on the internet and prepared by the package's authors. No particular choice has been made on the dimension of the original images or the brands cited here.

Examples

```
data(bot)
panel(bot, names=TRUE)
```

`dataset: hearts` *hearts dataset*

Description

The `hearts` dataset is a `Coo`-class object that contains outline coordinates of 240 hand-drawn hearts by 8 different persons.

Usage

```
data(hearts)
```

Format

A `Coo`-object that contains in the slot `@coo`, the lists of $(x; y)$ coordinates and in the slot `@fac` a single factor "aut" with the name of the authors. Four landmarks are also registered.

Source

We thank the fellows of the Ecology Department of the French Institute of Pondicherry that drawn the hearts, that then have been smoothed, scaled, centered, and reduced to 80 coordinates per outline.

Examples

```
data(hearts)
levels(hearts@fac[,"aut"]) # thank you guys !
panel(hearts, cols=rep(col.summer(8), each=30), names=TRUE)
```

`dataset: mosquito` *mosquito dataset*

Description

The `mosquito` dataset is a `Coo`-class object that contains outline coordinates of 126 outlines of mosquito wings used in Rohl and Archie (1984).

Usage

```
data(mosquito)
```

Format

A `Coo`-object that contains in the slot `@coo`, the lists of $(x; y)$ coordinates. No grouping factor.

Source

Rohlf F, Archie J. 1984. A comparison of Fourier methods for the description of wing shape in mosquitoes (Diptera: Culicidae). *Systematic Biology*: 302-317.

Arranged from: http://life.bio.sunysb.edu/morph/data/RohlfArchieWingOutlines.nts.

Examples

```
data(mosquito)
panel(mosquito)
```

`dataset: trilo` *trilo dataset*

Description

The `trilo` dataset is a `Coo`-class object that contains 64 coordinates of 50 cephalic outlines from different ontogenetic stages of trilobite.

Usage

```
data(trilo)
```

Format

A `Coo`-object that contains in the slot `@coo`, the lists of $(x; y)$ coordinates and ontogenetic stages in the slot `@fac`.

Source

Hammer & Harper (2005) *Paleontological Data Analysis* ed. Blackwell. See http://folk.uio.no/ohammer/past/.

Arranged from: http://folk.uio.no/ohammer/past/outlines.dat. The original data included 51 outlines and 5 ontogenetic stages, but one of them has just a single outline thas has been removed.

Examples

```
data(trilo)
panel(trilo, names=TRUE)
```

defLandmarks *Define landmarks on a* Coo *object*

Description

defLandmarks helps to define landmarks on a Coo object. The number of landmarks must be specified and rows indices that correspnd to the nearest points clicked on every outlines are stored in the @ldk slot of the Coo object.

Usage

```
defLandmarks(Coo, nb.ldk)
```

Arguments

Coo	A Coo object on which to define landmarks.
nb.ldk	An integer taht indicates the number of landmarks to define along the Coo object.

See Also

coo.ldk.

dev.plot *Calculates and plots series with associated error bars.*

Description

This function is used internally by methods based on deviations for one one many outlines. Yet, it provides a quick way to create plots of series, possibly with deviations, from scratch.

Usage

```
dev.plot(mat, dev, cols, x=1:ncol(mat),
                          lines=TRUE, poly=TRUE, segments=FALSE, bw=0.1,
                    plot=FALSE, main="Deviation plot", xlab="", ylab="Deviations")
```

Arguments

mat	A matrix containing one or many lines (as individuals) with the corresponding y values (as cols).
dev	A matrix of the same dimension as mat but containing the deviation from the mat matrix.
cols	A vector of ncol(mat) colors.
x	An alternative vector of values for every column of mat.

lines	logical. Whether to draw lines for mean values.
poly	logical. Whether to draw polygons for mean + dev values.
segments	logical. Whether to draw segments for these mean + dev values.
bw	numeric. The width of the errors bars to draw.
plot	logical. Whether to plot a new graphical window.
main	character. A title for the plot.
xlab	character. A title for the x-axis.
ylab	character. A title for the y-axis.

Examples

```
# we prepare some fake data
foo.mat  <- matrix(1:10, nr=3, nc=10, byrow=TRUE) + rnorm(30, sd=0.5)
foo.mat  <- foo.mat + matrix(rep(c(0, 2, 5), each=10), 3, byrow=TRUE)
foo.dev  <- matrix(abs(rnorm(30, sd=0.5)), nr=3, nc=10, byrow=TRUE)
# some possible tuning
dev.plot(foo.mat, plot=TRUE)
dev.plot(foo.mat, foo.dev, plot=TRUE)
dev.plot(foo.mat, foo.dev, lines=TRUE, plot=TRUE)
dev.plot(foo.mat, foo.dev, poly=FALSE, segments=TRUE, lines=TRUE, plot=TRUE)
dev.plot(foo.mat, foo.dev, cols=col.sari(3), poly=FALSE, segments=TRUE, lines=TRUE, plot=TRUE)
dev.plot(foo.mat, foo.dev, cols=col.summer(6)[4:6], plot=TRUE)
```

---

dev.segments *Draws colored segments from a matrix of coordinates.*

Description

Given a matrix of (x; y) coordinates, draws segments between every points defined by the row of the matrix and uses a color to display an information.

Usage

```
dev.segments(coo, cols, lwd = 1)
```

Arguments

coo	A matrix of coordinates.
cols	A vector of color of length = nrow(coo).
lwd	The lwd to use for drawing segments.

Examples

```
# we load some data
data(bot)
guinness <- bot[9]

# we calculate the best possible outline and one with 12 harm.
out.best <- l2m(efourier.i(efourier(guinness, nb.h=-1), nb.pts=120))
out.12   <- l2m(efourier.i(efourier(guinness, nb.h=12), nb.pts=120))

# we calculate deviations, you can also try 'edm'
dev <- edm.nearest(out.12, out.best) / coo.centsize(out.12)

# we prepare the color scale
d.cut <- cut(dev, breaks=20, labels=FALSE, include.lowest=TRUE)
cols  <- paste0(col.summer(20)[d.cut], "CC")

# we draw the results
coo.plot(out.best, border="black", col="grey80", main="Guiness fitted by 20 harm.")
dev.segments(out.12, cols=cols, lwd=4)
```

`diapo(Coo)` *Plots a slideshow of each outline in a* `Coo` *object.*

Description

`diapo(Coo)` plots a slideshow of each outline in a `Coo` object with graphical options.

Usage

```
diapo(Coo, id=1, col=NA, border="#708090",
      ldk=TRUE, ldk.pch=3, ldk.col="red", ldk.cex=1, ...)
```

Arguments

`Coo`	The `Coo` object to plot.
`id`	The id of the Coo to start with.
`col`	A color for drawing the outline.
`border`	A color for drawing the border.
`ldk`	`logical`. Whether to display landmarks (if any).
`ldk.pch`	A `pch` for these landmarks.
`ldk.col`	A color for these landmarks.
`ldk.cex`	A `cex` for these landmarks.
`...`	Additional arguments to be passed to coo.draw.

Value

No returned value.

See Also

stack, diapo.

Examples

```
## Not run:
data(mosquito)
diapo(hearts, 230, col="pink", border="black")

## End(Not run)
```

---

dudi.plot *A wrapper for* dudi.pca *graphical functions.*

Description

A wrapper for dudi.pca objects produced by dudi.pca in ade4 or pca in Momocs.

Usage

```
dudi.plot(dudi, fac = NULL, xax = 1, yax = 2, grid = TRUE,
points = TRUE, pch.points = 1, col.points = "black", cex.points = 0.8,
labels = FALSE, label = rownames(dudi$li),
boxes = TRUE, clabel = 0.6,
neighbors = FALSE, col.nei = "grey90", lwd.nei = 0.5,
star = TRUE, col.star = "grey60", cstar = 1,
ellipses = TRUE, col.ellipse = "grey30", cellipse = 1, axesell = TRUE,
chull = FALSE, col.chull = "grey30", optchull = c(0.5, 1),
arrows = FALSE, edge.arrow = FALSE,
        box.arrow = TRUE, maxnb.arrow = 10, dratio.arrow = 0.2,
shapes = TRUE, pos.shp=c("li", "circle", "range", "full")[3],
nr.shp = 6, nc.shp = 5, amp.shp = 1,
scale.shp = 0.666, nb.pts.shp=300,
first.point.shp = FALSE, rotate.shp=0,
circle.nb.shp = 12, circle.r.shp,
col.shp = "#70809011", border.shp = "#708090",
rug = TRUE, rug.ticksize = 0.01, rug.col = "#708090",
eigen = FALSE, eigen.ratio = 0.2,
palette = col.sari, title = substitute(dudi), legend=FALSE,
center.orig = FALSE, zoom.plot = 1)
```

Arguments

dudi	a dudi.pca object.
fac	the (col)name of the @fac slot to use as the grouping factor.
xax	integer. The index of the first PC axis to use.
yax	integer. The index of the first PC axis to use.
grid	logical. Whether to draw the grid.
points	logical. Whether to draw the points.
pch.points	the pch for drawing points.
col.points	the col for drawing points.
cex.points	the cex for drawing points.
labels	logical. Whether to draw labels.
label	character. The labels to draw.
boxes	logical. Whether to draw labels in boxes.
clabel	The cex for labels.
neighbors	logical. Whether to draw a neighboring graph.
col.nei	the col for drawing neighboring graph.
lwd.nei	the lwd for drawing neighboring graph.
star	logical. Whether to draw the star.
col.star	the col for drawing the star.
cstar	numeric. The size of the star.
ellipses	logical. Whether to draw bivariate confidence ellipses.
col.ellipse	The col for drawing these ellipses.
cellipse	numeric. The size of this ellipse.
axesell	logical. Whether to draw the ellipses axes.
chull	logical. Whether to draw a convex hull.
col.chull	The col for drawing convex hulls.
optchull	numeric. A vector of quantiles of the chulls.
arrows	logical. Whether to draw variables arrows.
edge.arrow	logical. Whether to neutralise arrows.
box.arrow	logical. Whether to draw boxes around arrows' labels.
maxnb.arrow	numeric. Only the maxnb.arrow most important variables will be draw.
dratio.arrow	numeric. Same idea as above but here the threshold is a relative to the size of the grid.
shapes	logical. Whether to plot shapes.
pos.shp	character, any of the ("li", "circle", "range") methods. It specifies the way shapes have to be drawn. "li" draws shapes on the actual positions on the factorial map ; "circle" draws shapes on a circle with origin as the center ; "full" (not available in morpho.space) allows to cover the whole plotting region ; "range" draws shape on a rectangle that covers PC1 and PC2 range. Alternatively, a two columns matrix of coordinates where to calculate shapes.

nr.shp	numeric. The number of shape rows.
nc.shp	numeric. The number of shape columns.
amp.shp	numeric. An amplifying factor for shape deformation.
scale.shp	numeric. The size of shapes, relatively to grid size (0.5 = half of the grid size).
nb.pts.shp	numeric. The number of point to use to reconstruct shapes.
first.point.shp	logical. Whether to plot or not the first point when plotting shapes.
rotate.shp	numeric. If specified, the angles (in radians) to counter-clockwise rotate shapes plotted.
circle.nb.shp	integer. When pos == "circle", the number of shapes on the circle.
circle.r.shp	numeric. When pos == "circle", the circle radius.
col.shp	A color string for filling the shapes.
border.shp	A color string for shape borders.
rug	logical. Whether to add rug on axes (see rug).
rug.ticksize	numeric. The relative size of rug.
rug.col	A color string for rug ticks.
eigen	logical. Whether to add the eigen values barplot.
eigen.ratio	numeric. The relative size of the eigen values barplot.
palette	A color palette for group colors such as those produced by colorRampPalette.
title	A character string to be add on the graph.
legend	logical. Whether to add or not a legend..
center.orig	logical. Whether to center the graphical window on the origin.
zoom.plot	numeric. Will help you to keep your distances, e.g. the value of magnification. Requires center.orig to be set true.

Details

This function takes the slightly modified dudi.pca object, obtained with pca on a Coe object and widely hinges on ade4 plotting facilities and graphical layers. Notice that besides dedicated *shapes* argument and options, it can be used for plotting factorial maps on "regular" dudi.pca objects. By default, shapes is set to FALSE for tFourier analyses since the reconstruction of shapes do not always lead to closed shapes (see Rohlf F, Archie J. 1984. A comparison of Fourier methods for the description of wing shape in mosquitoes (Diptera: Culicidae). Systematic Biology: 302-317.).

References

See the papers below that introduce ade4 and also the package's homepage: http://pbil.univ-lyon1.fr/ADE-4/ and particularly the file called "td83" : Lobry JR. 2010. Les fonctions graphiques 2D du paquet ade4.

Dray, S. and Dufour, A.B. (2007): The ade4 package: implementing the duality diagram for ecologists. *Journal of Statistical Software*. **22**(4): 1-20.

Chessel, D. and Dufour, A.B. and Thioulouse, J. (2004): The ade4 package-I- One-table methods. *R News*. **4**: 5-10.

Dray, S. and Dufour, A.B. and Chessel, D. (2007): The ade4 package-II: Two-table and K-table methods. *R News*. **7**(2): 47-52.

See Also

pca, morpho.space.

Examples

```
data(bot)
botF <- eFourier(bot, nb.h=32)
botD <- pca(botF)
dudi.plot(botD)
dudi.plot(botD, 1, title="botD with no class but with ellipses")
dudi.plot(botD, fac=1, chull=TRUE, rug=FALSE, shape=FALSE, title="botD with convex hull")
dudi.plot(botD, fac=1, ellipses=FALSE, neighbors=TRUE, shapes=FALSE, star=FALSE,
col.nei="black", title="botD with Gabriel's neighboring graph")
dudi.plot(botD, labels=TRUE, points=FALSE, boxes=FALSE, shapes=TRUE, pos.shp="li",
title="botD with labels and reconstructed shapes")
dudi.plot(botD, 1, points=FALSE, labels=TRUE, boxes=FALSE, shapes=FALSE,
title="botD with labels and ellipse")
dudi.plot(botD, 1, arrows=TRUE, dratio.arrow=0.2, shapes=FALSE,
title="botD with harmonic correlations")
# With some fake factors
botD <- pca(botF)
dudi.plot(botD, "type", palette=col.gallus,
    rotate.shp=pi/2, title="botD with classes") # rotated shapes
dudi.plot(botD, "type", palette=col.gallus, eigen=TRUE, title="botD with eigen values")
dudi.plot(botD, "type", pos.shp="full", title="botD with shapes(1)")
dudi.plot(botD, "type", pos.shp="range", scale.shp=0.5, shapes=TRUE,
border.shp="firebrick3", col.shp=NA, center.orig=TRUE,
zoom.plot=0.8, title="botD with shapes(2)")
dudi.plot(botD, "type", pos.shp="circle", center.orig=TRUE, title="botD with shapes(3)")
dudi.plot(botD, "type", pos.shp="range", scale.shp=0.5, title="botD with shapes(4)")
dudi.plot(botD, pos.shp=as.matrix(expand.grid(seq(-0.05, 0.05, 0.025),
    seq(-0.05, 0.05, 0.025)))) # an example with a matrix provided to pos.shp
```

---

ed *Calculates euclidean distance between two points.*

Description

ed simply calculates euclidean distance between two points defined by their (x; y) coordinates. This function is used internally but may be of interest for other analyses.

Usage

```
ed(pt1, pt2)
```

Arguments

pt1	(x; y) coordinates of the first point.
pt2	(x; y) coordinates of the second point.

Value

Returns the euclidean distance between the two points.

See Also

edm, edm.nearest, dist.

Examples

```
ed(c(0,1), c(1,0)) # sqrt 2
```

`edi` *Calculates euclidean intermediate between two points.*

Description

`edi` simply calculates coordinates of a points at the relative distance `r` on the `pt1-pt2` defined by their (x; y) coordinates. This function is used internally but may be of interest for other analyses.

Usage

```
edi(pt1, pt2, r = 0.5)
```

Arguments

`pt1`	(x; y) coordinates of the first point.
`pt2`	(x; y) coordinates of the second point.
`r`	the relative distance from `pt1` to `pt2`.

Value

Returns the (x; y) interpolated coordinates.

See Also

ed, edm.

Examples

```
edi(c(0,1), c(1,0), r = 0.5)
```

edm *Calculates euclidean distance every pairs of points in two matrices.*

Description

edm returns the euclidean distances between points

$$1 -> n$$

of two 2-col matrices of the same dimension. This function is used internally but may be of interest for other analyses.

Usage

```
edm(m1, m2)
```

Arguments

m1	The first matrix of coordinates.
m2	The second matrix of coordinates.

Details

If one wishes to align two (or more shapes) Procrustes surimposition may provide a better solution.

Value

Returns a vector of euclidean distances between pairwise coordinates in the two matrices.

See Also

ed, edm.nearest, dist.

Examples

```
x <- matrix(1:10, nc=2)
edm(x, x)
edm(x, x+1)
```

`edm.nearest` *Calculates the shortest euclidean distance found for every point of one matrix among those of a second.*

Description

`edm.nearest` calculates the shortest euclidean distance found for every point of one matrix among those of a second. In other words, if `m1, m2` have `n` rows, the result will be the shortest distance for the first point of `m1` to any point of `m2` and so on, `n` times. This function is used internally but may be of interest for other analyses.

Usage

```
edm.nearest(m1, m2, full=FALSE)
```

Arguments

`m1`	The first `list` or `matrix` of coordinates.
`m2`	The second `list` or `matrix` of coordinates.
`full`	`logical`. Whether to returns a condensed version of the results.

Details

So far this function is quite time consumming since it performs

$$n \times n$$

euclidean distance computation. If one wishes to align two (or more shapes) Procrustes surimposition may provide a better solution.

Value

If `full` is `TRUE`, returns a `list` with two components: `d` which is for every point of `m1` the shortest distance found between it and any point in `m2`, and `pos` the (`m2`) row indices of these points. Otherwise returns `d` as a numeric vector of the shortest distances.

See Also

ed, edm, dist.

Examples

```
x <- matrix(1:10, nc=2)
edm.nearest(x, x+rnorm(10))
edm.nearest(x, x+rnorm(10), full=TRUE)
```

`ef.amplify` *Dilates shapes based on elliptical Fourier decomposition.*

Description

These two functions calculate dilated and eroded shapes based on elliptical Fourier decomposition *i.e.* taking into account the shape as a whole. Lists created by `efourier` objects can be passed to `ef.amplify` or a list or matrix of coordinates to `coo.ef.amplify`.

Usage

```
ef.amplify(ef, amp=rep(0.5, 4))
coo.ef.amplify(coo, amp=rep(0.5, 4), nb.h=5, draw=FALSE, ...)
```

Arguments

`ef`	`list`. A list containing a_n, b_n, c_n and d_n Fourier coefficients, such as returned by `efourier`.
`coo`	A `list` or a `matrix` of coordinates.
`amp`	A vector of `numeric`. If `amp` is of length 4, the value specify the multiplication factor for a_1, b_1, c_1 and d_1 ; if only one value is provided, then the multiplication factor will be the same for the four coefficients $abcd_1$.
`draw`	`logical`. Whether to draw (see coo.draw) the amplified `coo`.
`nb.h`	`integer`. The number of harmonics to calculate.
`...`	Additional parameters to be passed to `coo.draw`.

Value

`ef.amplify` returns the `ef` provided but with "amplified" coefficients for the first harmonics. `coo.amplify` returns the `coo` provided but "amplified" based on elliptical Fourier.

See Also

efourier for a description of the elliptical Fourier analysis and Ptolemy for an illustration of the first ellipse/harmonic defining the shape "amplitude".

Examples

```
# same amp factor for every coeff.
data(bot)
bot1 <- coo.sample(bot@coo[[1]], 50)
coo.plot(bot1, col=NA)
amp <- seq(0.9, 0.3, -0.1)
for (i in seq(along=amp)) {
  coo.ef.amplify(bot1, amp[i], draw=TRUE, first=FALSE,
              col=NA, border=col.summer(length(amp))[i])}
```

```
# random shape, separate amplification
poly <- efourier.shape(nb.h=5, alpha=3, plot=FALSE)
poly2 <- coo.sample(poly, 20)
layout(matrix(1:4, nc=2, byrow=TRUE))
par(oma=rep(0.5, 4), mar=c(2, 2, 3, 2))
coo.plot(poly2, col=NA, main="an")
for (i in seq(-1, 1, 0.5)) {
  coo.ef.amplify(poly2, c(i, 1, 1, 1), draw=TRUE)}

coo.plot(poly2, col=NA, main="bn")
for (i in seq(-1, 1, 0.5)) {
  coo.ef.amplify(poly2, c(1, i, 1, 1), draw=TRUE)}

coo.plot(poly2, col=NA, main="cn")
for (i in seq(-1, 1, 0.5)) {
  coo.ef.amplify(poly2, c(1, 1, i, 1), draw=TRUE)}

coo.plot(poly2, col=NA, main="dn")
for (i in seq(-1, 1, 0.5)) {
  coo.ef.amplify(poly2, c(1, 1, 1, i), draw=TRUE)}
```

`eFourier` *Calculates elliptical Fourier analysis on* `Coo` *objects.*

Description

`eFourier` performs an elliptical Fourier analysis on a `Coo`-class object, with the specified parameters (number of harmonics and number of smoothing iterations) and returns a `Coe`-class object containing harmonic coefficients (usually normalized). It accepts the same arguments as the function efourier.

Usage

```
eFourier(Coo, nb.h = 32, smooth.it = 0, norm = TRUE, start = FALSE)
```

Arguments

`Coo`	A `Coo` object.
`nb.h`	`integer`. The number of harmonics to use
`smooth.it`	`integer`. The number of smoothing iterations to perform.
`norm`	`logical`. Whether to print or not diagnosis messages.
`start`	`logical`. Whether to define the first point of the outlines as an homologous point, and thus preserve this information. See Details and References in efourier.

See Also

efourier and efourier.norm

Examples

```
data(bot)
eFourier(bot)
```

`efourier` *Calculates elliptical Fourier analysis.*

Description

`efourier` computes elliptical Fourier analysis from a matrix or a list of coordinates.

Usage

```
efourier(coo, nb.h = 32, smooth.it = 0, silent = FALSE)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.
`nb.h`	`integer`. The number of harmonics to use
`smooth.it`	`integer`. The number of smoothing iterations to perform.
`silent`	`logical`. Whether to print or not diagnosis messages.

Details

These functions and their mathematical background detailed below are here detailed to ease their use in new methods but are used internally by methods on `Coo`-objects.

Elliptic Fourier analysis and normalization are calculated as follows. Let T be the perimeter of a given closed outline, here considered as the period of the signal. One sets $\omega = 2\pi/T$ to be the pulse. Then, the curvilinear abscissa, t varies from 0 to T. One can express $x(t)$ and $y(t)$ as:

$$x(t) = \frac{a_0}{2} + \sum_{n=1}^{+\infty} a_n \cos n\omega t + b_n \sin n\omega t$$

with

$$a_n = \frac{2}{T} + \int_0^T x(t) \cos(n\omega t) \mathrm{d}t$$

$$b_n = \frac{2}{T} + \int_0^T x(t) \sin(n\omega t) \mathrm{d}t$$

similarly,

$$y(t) = \frac{c_0}{2} + \sum_{n=1}^{+\infty} c_n \cos n\omega t + d_n \sin n\omega t$$

with

$$c_n = \frac{2}{T} + \int_0^T y(t)\cos(n\omega t)\mathrm{d}t$$

$$d_n = \frac{2}{T} + \int_0^T y(t)\sin(n\omega t)\mathrm{d}t$$

Since the outline contains a k finite number of points, one can therefore calculate discrete estimators for every harmonic coefficient of the n^{th} harmonics:

$$a_n = \frac{T}{2\pi^2 n^2}\sum_{p=1}^{k}\frac{\Delta x_p}{\Delta t_p}(\cos\frac{2\pi n t_p}{T} - \cos\frac{2\pi n t_{p-1}}{T})$$

$$b_n = \frac{T}{2\pi^2 n^2}\sum_{p=1}^{k}\frac{\Delta x_p}{\Delta t_p}(\sin\frac{2\pi n t_p}{T} - \sin\frac{2\pi n t_{p-1}}{T})$$

$\Delta x_1 = x_1 - x_k$ and c_n and d_n are calculated similarly. a_0 and c_0 correspond to the estimate of the coordinates of the centroid of original outline and are estimated by:

$$a_0 = \frac{2}{T}\sum_{i=1}^{p} x_i$$

and

$$c_0 = \frac{2}{T}\sum_{i=1}^{p} y_i$$

Intuitively, for all positive integers n, the sum of a cosine curve and a sine curve represent the n^{th} harmonic content of the x and y projections of the k-edged polygon, and for any n, these two curves define an ellipse in the plane. Ferson and colleagues noticed that in the "time" it takes the n^{th} harmonic to traverse its ellipse n times, the $(n+1)^{th}$ harmonic has traversed its own ellipse $n+1$ times. The reconstruction of the original polygon is done by vector adding these ellipses for all harmonics, which echoes astronomical Ptolemy's epicycles (see Ptolemy), and the reconstruction obtained from N harmonics is the best possible fit in a least-squares sense.

Value

A list with these components:

`an`	`vector` of $a_{1->n}$ harmonic coefficients.
`bn`	`vector` of $b_{1->n}$ harmonic coefficients.
`cn`	`vector` of $c_{1->n}$ harmonic coefficients.
`dn`	`vector` of $d_{1->n}$ harmonic coefficients.
`ao`	`ao` Harmonic coefficient.
`co`	`co` Harmonic coefficient.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Ferson S, Rohlf FJ, Koehn RK. 1985. Measuring shape variation of two-dimensional outlines. *Systematic Biology* **34**: 59-68.

See Also

efourier.i for the reverse operation and eFourier the method for `Coo` objects. Ptolemy for an implementation of the Ptolemaic ellipses. rfourier, tfourier for the other members of the Fourier's family.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo)
ef  <- efourier(coo, 12)
ef
efi <- efourier.i(ef)
l2m(efi)
coo.draw(efi, border="red", col=NA)
```

---

efourier.i *Calculates inverse elliptical Fourier analysis.*

Description

`efourier.i` uses the inverse elliptical Fourier transformation to calculate a shape, when given a list with Fourier coefficients, typically obtained computed with efourier.

Usage

```
efourier.i(ef, nb.h, nb.pts = 300)
```

Arguments

`ef`	`list`. A list containing a_n, b_n, c_n and d_n Fourier coefficients, such as returned by `efourier`.
`nb.h`	`integer`. The number of harmonics to use. If not specified, `length(ef$an)` is used.
`nb.pts`	`integer`. The number of points to calculate.

Details

See efourier for the mathematical background.

Value

A list with components:

x	vector of x-coordinates.
y	vector of y-coordinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Ferson S, Rohlf FJ, Koehn RK. 1985. Measuring shape variation of two-dimensional outlines. *Systematic Biology* **34**: 59-68.

See Also

efourier for the reverse operation. l2m, coeff.split may be useful.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo)
ef  <- efourier(coo, 12)
ef
efi <- efourier.i(ef)
l2m(efi)
coo.draw(efi, border="red", col=NA)
```

---

efourier.norm *Normalizes harmonic coefficients.*

Description

efourier.norm normalizes Fourier coefficients for rotation, tranlation, size and orientation of the first ellipse.

Usage

```
efourier.norm(ef, start = FALSE)
```

Arguments

ef	list. A list containing a_n, b_n, c_n and d_n Fourier coefficients, such as returned by efourier.
start	logical. Whether to conserve the position of the first point of the outline.

Details

See efourier for the mathematical background of the normalization. Other approaches implemented in SHAPE are possible such as manually editing or using the longest radius. They will be implemented in further Momocs versions.

Value

A list with following components:

`A`	`vector` of `numeric` $A_{1->n}$ *normalized* harmonic coefficients.
`B`	`vector` of `numeric` $B_{1->n}$ *normalized* harmonic coefficients.
`C`	`vector` of `numeric` $C_{1->n}$ *normalized* harmonic coefficients.
`D`	`vector` of `numeric` $D_{1->n}$ *normalized* harmonic coefficients.
`size`	Magnitude of the semi-major axis of the first fitting ellipse.
`theta`	Angle, in radians, between the starting point and the semi-major axis of the first fitting ellipse.
`psi`	Orientation of the first fitting ellipse.
`ao`	`ao` Harmonic coefficient.
`co`	`co` Harmonic coefficient.
`lnef`	A `list` with A, B, C and D concatenated in a `vector` that may be convenient for some uses.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Ferson S, Rohlf FJ, Koehn RK. 1985. Measuring shape variation of two-dimensional outlines. *Systematic Biology* **34**: 59-68.

See Also

efourier and efourier.i. Also eFourier for normalizing harmonic coefficients when calculating it for `Coo` objects.

Examples

```
data(bot)
q <- efourier(bot@coo[[1]], 24)
efourier.i(q) # equivalent to efourier.shape(q$an, q$bn, q$cn, q$dn)
efourier.norm(q)
efourier.shape(nb.h=5, alpha=1.2)
efourier.shape(nb.h=12, alpha=0.9)
```

`efourier.shape` *Calculates and draw "efourier" shapes.*

Description

`efourier.shape` calculates a "Fourier elliptical shape" given Fourier coefficients (see `Details`) or can generate some "efourier" shapes.

Usage

```
efourier.shape(an, bn, cn, dn, nb.h, nb.pts=80, alpha=2, plot=TRUE)
```

Arguments

`an`	`numeric`. The a_n Fourier coefficients on which to calculate a shape.
`bn`	`numeric`. The b_n Fourier coefficients on which to calculate a shape.
`cn`	`numeric`. The c_n Fourier coefficients on which to calculate a shape.
`dn`	`numeric`. The d_n Fourier coefficients on which to calculate a shape.
`nb.h`	`integer`. The number of harmonics to use.
`nb.pts`	`integer`. The number of points to calculate.
`alpha`	`numeric`. The power coefficient associated with the (usually decreasing) amplitude of the Fourier coefficients (see **Details**).
`plot`	`logical`. Whether to plot or not the shape.

Details

`efourier.shape` can be used by specifying `nb.h` and `alpha`. The coefficients are then sampled in an uniform distribution $(-\pi; \pi)$ and this amplitude is then divided by $harmonicrank^{a}lpha$. If `alpha` is lower than 1, consecutive coefficients will thus increase. See efourier for the mathematical background.

Value

A list with components:

`x`	`vector` of `x`-coordinates.
`y`	`vector` of `y`-coordinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

Ferson S, Rohlf FJ, Koehn RK. 1985. Measuring shape variation of two-dimensional outlines. *Systematic Biology* **34**: 59-68.

See Also

efourier.i.

Examples

```
data(bot)
ef <- efourier(bot@coo[[1]], 24)
efourier.shape(ef$an, ef$bn, ef$cn, ef$dn) # equivalent to efourier.i(ef)
efourier.shape() # is autonomous

efourier.shape(nb.h=12)
efourier.shape(nb.h=12, alpha=0.5)

panel(Coo(replicate(100, l2m(efourier.shape(nb.h=6, alpha=2.5, plot=FALSE))))) # Bubble family
panel(Coo(replicate(100, l2m(
    efourier.shape(nb.h=12, alpha=0.5, nb.pts=80, plot=FALSE))))# some doodle
```

---

ellipse.par *Calculate ellipse parameters on a* Coe*-class object.*

Description

It uses ellpar function to calculate ellipse parameters for all individuals and all harmonics contained in a Coe-class object.

Usage

```
ellipse.par(Coe, range = 1:nrow(Coe@coe), nb.pts = 120)
```

Arguments

Coe	The Coe object on which to calculate ellipses parameters.
range	Vector of integer that specifies the range of harmonics on which to calculate ellipse parameters. By default, all of them.
nb.pts	integer. How many points on the outline are used to estimate ellipse parameters (see **Details** section below).

Details

This method is not exact, see ellpar

Value

Four matrices are returned (with individual as rows and harmonic rank as columns).

a	a matrix of all halves of the major axis.
b	a matrix of all halves of the minor axis.
e	a matrix of all eccentricities.
phi	a matrix of all ellipse angles.

References

See **References** in ellpar.

Examples

```
data(bot)
botF <- eFourier(bot, nb.h=32)
ep <- ellipse.par(botF, 1:5, 64)
names(ep)
layout(matrix(1:4, ncol=2, byrow=TRUE))
boxplot(ep$a, log="y", main="1/2 Major axes")
boxplot(ep$b, log="y", main="1/2 minor axes")
boxplot(ep$e, , main="Eccentricities")
boxplot(ep$phi, main="Ellipse angles")
```

---

ellpar *Calculate ellipse parameters.*

Description

Given a matrix or list of coordinates that defines an ellipse, returns its geometrical parameters, namely: half-length of the major and the minor axis and excentricity.

Usage

```
ellpar(coo)
```

Arguments

coo	a matrix or a list of (x; y) coordinates.

Details

The aim of this function is to provide complementary descriptors for ellipses that are more explicit for humans than harmonic coefficients. For that reason they are also used directly instead of harmonic coefficients (See Schmittbuhl et al. 2003). The same study provided an exact approach to directly obtain ellipse parameters from harmonic coefficients but is not yet implemented in Momocs. So far, these parameters are estimated using matrices of coordinates obtained from harmonic coefficients, *i.e.* by "redrawn" ellipses. Keep in mind that even if error rate is low (moreover if the number of points provided is high), the values returned must not be considered as exact.

Value

a	numeric. Half length of the major axis.
b	numeric. Half length of the minor axis.
e	numeric. Ellipse excentricity.

References

Schmittbuhl M, Allenbach B, Le Minor J-M, Schaaf A, Minor J-marie L. 2003. Elliptical Descriptors: Some Simplified Morphometric Parameters for the Quantification of Complex Outlines. *Mathematical Geology* **35**: 853-871.

See Also

ellipse.par, the corresponding method for Coe-class objects.

Examples

```
data(bot)
```

---

harm.pow *Calculates harmonic power given a list from e/t/rfourier*

Description

Given a list with an, bn (and eventually cn and dn), returns the harmonic power.

Usage

```
harm.pow(xf)
```

Arguments

xf	A list with an, bn (and cn, dn) components, typically from a e/r/tfourier passed on coo.

Value

Returns a vector of harmonic power

Examples

```
data(bot)
ef <- efourier(bot@coo[[1]], 24)
rf <- efourier(bot@coo[[1]], 24)
harm.pow(ef)
harm.pow(rf)

plot(cumsum(harm.pow(ef)[-1]), type="o",
  main="Cumulated harmonic power without the first harmonic",
  ylab="Cumulated harmonic power", xlab="Harmonic rank")
```

`hcontrib` *Calculates and displays the contribution of every harmonic.*

Description

`hcontrib` applies `amp.h` values for each of the `harm.range` harmonics when reconstruction shapes. This thus help to visualize the respective contribution of every harmonic, in other words their contribution in describing shapes.

Usage

```
hcontrib(Coe, id = 1, harm.range = 1:floor(Coe@nb.h / 4),
    amp.h   = c(0, 0.5, 1, 2, 5, 10),
    palette = col.sari, title   = "Harmonic contribution")
```

Arguments

`Coe`	The `Coe` object
`id`	`integer`. The `id` of the shape to display. A range of `id`s can be passed to `harm.pow`
`harm.range`	A `vector` of `integer` giving the harmonic range to calculate. See `nb.h` for `harm.pow`.
`amp.h`	A `vector` of numeric to specify the amplification factors to use.
`palette`	A color palette such those included in Momocs or produced with colorRampPalette.
`title`	A `character`to use as the title.

Value

No returned value.

Examples

```
#FILL
```

hpow *Calculates harmonic power.*

Description

hpow is used to estimate the number of harmonics required for the three Fourier methods implemented so far in Momocs: elliptical Fourier analysis (see efourier), radii variation analysis (see tfourier) and tangent angle analysis (see tfourier). It returns and can plot cumulated harmonic power whether dropping the first harmonic or not.

Usage

```
hpow(Coo, method = c("efourier", "rfourier", "tfourier"),
                  id = 1:Coo@coo.nb, probs = c(0, 0.5, 1), nb.h = 24,
                  drop = 1, smooth.it = 0, plot = TRUE, legend = FALSE,
                  title = "Fourier power spectrum",
                  lineat.y = c(0.9, 0.95, 0.99, 0.999), bw = 0.1)
```

Arguments

Coo	The Coo object
method	A character, either "efourier", "rfourier" or "tfourier" (partial matches are allowed) to use to calculate morphological that indicates which method to use.
id	integer. The id of the shape to display. A range of ids can be passed to harm.pow
probs	A vector of numeric, to define quantiles to calculate ; see quantile
nb.h	integer. The maximal number of harmonics to calculate.
drop	logical. Whether to drop the first harmonic for plotting and power calculation.
smooth.it	integer. The number of smoothing iteration to perform.
plot	logical. Whether to plot or not the shape. If FALSE, only the results are returned.
legend	logical. Whether to display a legend box.
title	character. The title to add.
lineat.y	A vector of numeric to specify where to plot dashed lines on the y-axis.
bw	numeric. The width of horizontal segments drawn for each harmonic.

Details

The power of a given harmonic n is calculated as follows for elliptical Fourier analysis:

$$HarmonicPower_n = \frac{A_n^2 + B_n^2 + C_n^2 + D_n^2}{2}$$

and as follows for radii variation and tangent angle:

$$HarmonicPower_n = \frac{A_n^2 + B_n^2 + C_n^2 + D_n^2}{2}$$

Value

Returns a matrix containing cumulated harmonic power for each harmonic.

Examples

```
data(bot)
hpow(bot)
```

`hqual` *Displays how shapes are described with a given number of harmonics.*

Description

`hqual` is used to calculate and displays reconstructed shapes using a range of harmonic number.

Usage

```
hqual(Coo, method = c("efourier", "rfourier", "tfourier"),
                  id = sample(Coo@coo.nb, 1), smooth.it = 0,
                  harm.range = c(1, 2, 4, 8, 16, 32),
                  scale = TRUE, center = TRUE, align = TRUE,
                  plot.method = c("stack", "panel")[1], legend = TRUE,
                  legend.title = "# harmonics",
                  palette = col.summer, shp.col = "#70809033",
                  shp.border = "#708090EE")
```

Arguments

`Coo`	The `Coo` object
`method`	A character, either "efourier", "rfourier" or "tfourier" (partial matches are allowed) to use to calculate morphological that indicates which method to use.
`id`	`integer`. The `id` of the shape to display. A range of `ids` can be passed to `harm.pow`
`smooth.it`	`integer`. The number of smoothing iteration to perform.
`harm.range`	A `vector` of `integer` giving the harmonic range to calculate. See `nb.h` for `harm.pow`.
`scale`	`logical`. Whether to scale or not the shape.
`center`	`logical`. Whether to center or not the shape.
`align`	`logical`. Whether to align or not the shape.
`plot.method`	A plot method to use, either stack or panel.
`legend`	`logical`. Whether to display a legend box.
`legend.title`	`character`. A title for the legend box.
`palette`	A color palette such those included in Momocs or produced with colorRampPalette.
`shp.col`	A color for the shape body.
`shp.border`	A color for the shape border.

Value

No value returned.

Examples

```
data(bot)
hqual(bot)
```

`hquant` *Calculates deviations between reconstructed and best possible shapes using different distance methods.*

Description

`hquant` is used to estimate the number of harmonics required to describe shapes with the required accuracy. So far, two different methods can be used: edm which calculates point-to-point distance and edm.nearest which calculates for every point of the reconstructed shape the nearest point in the best possible shape. The best possible shape is calculated using the highest possible number of harmonics.

Usage

```
hquant(Coo, method = c("efourier", "rfourier", "tfourier"),
                id = 1, smooth.it = 0, harm.range = seq(4, 20, 4),
                norm.centsize = TRUE,
                dist.method = edm.nearest, dist.nbpts = 120,
            plot = TRUE, dev.plot = TRUE, title = "Deviations along the outline",
                legend = TRUE, legend.title = "# harmonics",
                palette = col.summer, lineat.y = c(0.5, 0.1, 0.01))
```

Arguments

`Coo`	The `Coo` object
`method`	A character, either "efourier", "rfourier" or "tfourier" (partial matches are allowed) to use to calculate morphological that indicates which method to use.
`id`	`integer`. The `id` of the shape to display. A range of `id`s can be passed to `harm.pow`
`smooth.it`	`integer`. The number of smoothing iteration to perform.
`harm.range`	A `vector` of `integer` giving the harmonic range to calculate. See `nb.h` for `harm.pow`.
`norm.centsize`	`logical`. Whether to normalize distances by the centroid size of every outline.
`dist.method`	A distance method to use, e.g. either edm.nearest or edm.
`dist.nbpts`	`integer`. The number of points to sample along the outlines to calculate the deviations. It can be set to "max" and the maximum number of points will be used, e.g. twice the number of the highest possible number of harmonics on the outline(s) analysed.

plot	logical. Whether to plot or not the shape. If FALSE, only the results are returned.
dev.plot	logical. Whether to plot or not the deviation plot.
title	A title for the plot.
legend	logical. Whether to display a legend box.
legend.title	character. A title for the legend box.
palette	A color palette such those included in Momocs or produced with colorRampPalette.
lineat.y	A vector of numeric to specify where to plot dashed lines on the y-axis.

Value

Returns a matrix containing deviations for each harmonic and corresponding quantiles.

Examples

```
## Not run:
data(bot)
hquant(bot)

## End(Not run)
```

---

import.jpg *Everything to convert images to a list of coordinates.*

Description

Functions to import .jpg images and convert them to list of coordinates.

Usage

```
import.jpg(jpg.list)
import.multi1.jpg(path)
import.img.prepare(path)
import.img.Conte(img, x, auto=TRUE, plot=TRUE)
```

Arguments

jpg.list	A vector of character containing the path to your .jpg images.
path	character. A single path to a .jpg image.
img	an imagematrix object.
x	A vector of $(x; y)$ coordinates from where to start Conte algorithm.
auto	logical. Whether to try or not to start at the center of the image(s) before asking the user to click within the shape.
plot	logical. Whether to plot or not the image. Used internally by import.multi1.jpg to not reload the same image.

Details

Typically, an object returned by list.files on a folder containing your images is passed to `import.jpg`. `import.img.prepare` and `import.img.Conte` are typically not used by front-user but internally by `import.jpg`. They clean, binarize, threshold, etc. raw .jpg images and extract a list of coordinates from a black and white `imagematrix`, respectively. The best option is to work with black and white `.jpg` image with the black mask of the outline overlapping the center of the image. `import.multi1.jpg` helps to extract several outlines from the same `.jpg` image.

Due to troubles with ReadImages and the recent change towards the jpeg package for import of images, they MUST be converted to black and white (i.e. 8-bits and grey levels) images before being imported.

If you get this message error : "Error in img[1,] : incorrect number of dimensions", try converting all your images to 8-bit mode, grey levels, and you can also apply a threshold (128 is fine) so that you only have black and white pixels, not grey. Then, save your images without compression. This can be done using automated scripts in editing softwares.

Value

`import.jpg` returns a list of (from 1 to thousands) $(x; y)$ coordinates arranged as matrices and that can be then converted to a `Coo`-object. `import.multi1.jpg` returns a `list` of $(x; y)$ coordinates. `import.img.prepare` returns an `imagematrix` object, `import.img.Conte` returns a matrix of $(x; y)$ coordinates.

See Also

import.txt.

Examples

```
## Not run:
jpg.list <- list.files(path_to_your_folder_containing_.txt_files, full=TRUE)
I <- import.jpg(jpg.list)
Coo(I)

## End(Not run)
```

`import.txt` *Everything to convert* `.txt` *files to a list of coordinates.*

Description

Takes a path list to `.txt` files and returns a list of coordinates arranged as matrices that can be passed to the `Coo` builder.

Usage

```
import.txt(txt.list, ...)
```

Arguments

`txt.list`	a list of full/relative path to `.txt` files. (see **Details**).
`...`	Complementary arguments to be passed to `read.table` within the function.

Details

`.txt` files must have coordinates arranged in two columns (*i.e.* x and y) with no header. The list of paths pointing to `.txt` files must be *full* if your working directory is different from your folder containing images, see **Examples** below.

Value

A `list` of matrices of coordinates.

See Also

import.jpg.

Examples

```
## Not run:
txt.list <- list.files(path_to_your_folder_containing_.txt_files, full=TRUE)
# altenartively :
# setwd(path_to_your_folder_containing_.txt_files)
# txt.list <- list.files()
I <- import.txt(txt.list)
Coo(I)
I # that should be a Coo object

## End(Not run)
```

`is.closed` *Tests if a list or matrix of coordinates is closed.*

Description

`is.closed` tests if the last coordinate of `coo` provided as a `list` or a `matrix` is closed, *i.e.* if the last coordinate is identical as the first.

Usage

```
is.closed(coo)
```

Arguments

`coo`	A `list` or a `matrix` of coordinates.

Value

A logical, either TRUE or FALSE.

See Also

coo.close and coo.unclose.

Examples

```
## Not run:
data(gorf.dat)
coo <- gorf.dat[,,1]
is.closed(coo)
coo.c <- coo.close(coo)
is.closed(coo.c)
coo.cu <- coo.unclose(coo)
is.closed(coo.cu)

## End(Not run)
```

---

l2a *Converts a list of coordinates to an array.*

Description

l2a converts a list of k matrices with n-rows and n-col matrices to a m x n x k array.

Usage

```
l2a(l)
```

Arguments

l	A list of matrices of the same dimension.

Value

An array of coordinates.

See Also

a2l.

Examples

```
## Not run:
data(gorf.dat)
l <- a2l(gorf.dat)
a <- l2a(l)
A.plot(a)

## End(Not run)
```

`l2m` *Converts a list of coordinates to a matrix.*

Description

`l2m` converts a `list` with x and y components to a 2-col `matrix` of coordinates.

Usage

```
l2m(l)
```

Arguments

`l`	A `list` with x and y coordinates as components.

Value

Returns a matrix of `(x; y)`coordinates.

See Also

m2l.

Examples

```
l <- list(x=1:5, y=5:1)
l2m(l)
```

m2l *Convert a matrix of coordinates to a list of coordinates.*

Description

m2l converts a matrix of (x; y)coordinates to a list with x; y components.

Usage

```
m2l(m)
```

Arguments

m	A 2-columns matrix containing x and y coordinates.

Value

Returns a list with x; y components.

See Also

l2m.

Examples

```
## Not run:
data(gorf.dat)
m2l(gorf.dat[,,1])

## End(Not run)
```

manova.Coe *Multivariate ANOVA on* Coe *objects*

Description

A simple wrapper for Multivariate Analysis of Variance on the matrix of harmonic coefficients in a Coe object.

Usage

```
manova.Coe(Coe, fac, retain, drop=0)
```

Arguments

Coe	a Coe object.
fac	factor defining which groups of individuals to compare.
retain	numeric. The number of harmonics to retain.
drop	numeric. The number of harmonics to drop.

Details

The number of harmonic coefficient can not be higher than the number of outlines ; you can specify it with retain and drop (see coeff.sel) or let this method select the highest possible number of harmonics.

Value

The MANOVA summary is printed and also returned invisibly.

See Also

See manova to change the defaults parameters of this method.

Examples

```
data(bot)
botF <- eFourier(bot)
m <- manova.Coe(botF, "type")
m # if you need something from this MANOVA..
```

---

meanShapes *Calculate groups mean shapes from* Coe

Description

meanShapes calculates mean shapes of groups defined through the @fac slot and using the appropriate inverse Fourier method.

Usage

```
meanShapes(Coe,
           fac,
           nb.pts=300)
```

Arguments

Coe	A Coe object on which to define landmarks.
fac	One of the factor names of Coe@fac. See examples below.
nb.pts	The number of points to use for calculation shapes.

Examples

```
data(bot)
botF <- eFourier(bot)

meanShapes(botF) # average shape
shp.type <- meanShapes(botF, "type")
panel(Coo(shp.type), borders=col.gallus(2)) # use Coo plotting facilities !

# below we combine another Fourier approach, one more factor, and one panel plot with names.
bot@fac <- data.frame(type=bot@fac[, 1], plop=rep(letters[1:4], each=10))
botR <- rFourier(bot)
shp.plop <- meanShapes(botR, "plop")
panel(Coo(shp.plop), cols=col.summer(4), names=TRUE)
```

`Momocs Package` *Outline Analysis using R.*

Description

Momocs is intended to ease and popularize shape analysis of outlines (especially using elliptical Fourier analysis). It mostly hinges on the core functions developped in *Morphometrics with R* (Claude, 2008). From outline extraction of images and elliptical analysis, radii variation and tangent angle Fourier calculation to multivariate analysis and calculation of the morphological space, Momocs provides a complete and convenient toolkit to specialists within every field that are, or may be, interested in morphological comparisons of outlines.

Have a look to Momocs' website: http://www.vincentbonhomme.fr/Momocs. You will find a slideshow plus some vignettes and other useful links.

Author(s)

1. Vincent Bonhomme, French Institute of Pondicherry, India. <bonhomme.vincent@gmail.com>
2. Sandrine Picq, UMR CBAE, Montpellier, France.
3. Julien Claude UMR, ISEM, Universite of Montpellier II, France.

References

Please cite the paper below in your great papers. It's freely available in JSS website.

Vincent Bonhomme, Sandrine Picq, Cedric Gaucherel, Julien Claude (2014). Momocs: Outline Analysis Using R. Journal of Statistical Software, 56(13), 1-24. http://www.jstatsoft.org/v56/i13/..

Have a look to Julien Claude's book that greatly inspired Momocs: Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

`morpho.space` *Calculate and plots morphological spaces.*

Description

Calculates and plots morphological spaces with many options.

Usage

```
morpho.space(dudi, xax = 1, yax = 2, xlim, ylim, nb.pts = 300,
                 pos.shp = c("li", "circle", "range")[3], nr.shp = 6,
                 nc.shp = 5, amp.shp = 1, scale.shp = 1, rotate.shp =
                 0, circle.nb.shp = 12, circle.r.shp, plot = TRUE,
                 layer = TRUE, col.shp = "#70809011", border.shp =
                 "#708090", pch.pts = 20, col.pts = "grey40",
                 first.point = FALSE)
```

Arguments

`dudi`	a dudi.pca object.
`xax`	`integer`. The index of the first PC axis to use.
`yax`	`integer`. The index of the first PC axis to use.
`xlim`	A vector of 2 `numeric` indicating the x-range.
`ylim`	A vector of 2 `numeric` indicating the y-range.
`nb.pts`	The number of points to calculate.
`pos.shp`	`character`, any of the (`"li"`, `"circle"`, `"range"`) methods. It specifies the way shapes have to be drawn. `"li"` draws shapes on the actual positions on the factorial map ; `"circle"` draws shapes on a circle with origin as the center ; `"range"` draws shape on a rectangle that covers PC1 and PC2 range. Alternatively, a two columns matrix of coordinates where to calculate shapes.
`nr.shp`	`numeric`. The number of shape rows.
`nc.shp`	`numeric`. The number of shape columns.
`amp.shp`	`numeric`. An amplifying factor for shape deformation.
`scale.shp`	`numeric`. The size of shapes, relatively to 1/8 of the highest range (x or y).
`rotate.shp`	`numeric`. If specified, the angles (in radians) to counter-clockwise rotate shapes plotted.
`circle.nb.shp`	`integer`. When `pos == "circle"`, the number of shapes on the circle.
`circle.r.shp`	`numeric`. When `pos == "circle"`, the circle radius.
`plot`	`logical`. Whether to plot or not the morphological space.
`layer`	`logical`. Whether to add calculated shapes on an existing plot. Used for instance by dudi.plot.
`col.shp`	A color string for filling the shapes.

border.shp	A color string for shape borders.
pch.pts	A pch for plotting the points from the dudi-class object. Use NA to not display these points.
col.pts	A color to plot these points.
first.point	logical. Whether to draw or not the first point of the shapes.

Details

Note that tFourier analyses since the reconstruction of shapes do not always lead to closed shapes (see Rohlf F, Archie J. 1984. A comparison of Fourier methods for the description of wing shape in mosquitoes (Diptera: Culicidae). Systematic Biology: 302-317.)

See Also

dudi.plot.

Examples

```
data(bot)
botF <- eFourier(bot)
botD <- pca(botF)
morpho.space(botD)
```

---

nef2Coe *Imports* .nef *files and creates* Coe *objects.*

Description

nef2Coe converts .nef files to Coe objects. This function is intended to ease data exchange between Momocs and SHAPE suite (see Iwata in the **References** below).

Usage

```
nef2Coe(nef.path)
```

Arguments

nef.path	A character that indicates the path for the .nef file to convert.

Value

nef2Coe returns a Coe object.

References

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the `.nef` format.

See Also

Coe2nef, for the reverse operation.

Examples

```
## Not run:
# I (VB) just finished my post-doc in India and I wont have time for Momocs
# until mid march or maybe a bit after that.
# Ryan Felice found a bug in my own version. I provide hereafter the function
# he kindly sent to me and that worked fine for him.
# Hope this help. Thanks again to him.
NEF2COE<-function (nef.path){
  nef <- readLines(nef.path)
  HARMO.l <- grep(pattern = "HARMO", nef)
  nb.h <- as.numeric(substring(nef[HARMO.l], 8))
  nef <- nef[-(1:HARMO.l)]
  nb.coo <- length(nef)/(nb.h + 1)
  coo.i <- 1:nb.coo
  coo.beg <- (coo.i - 1) * (nb.h + 1) + 1
  coo.end <- coo.beg + nb.h
  res <- matrix(NA, nrow = nb.coo, ncol = nb.h * 4,
      dimnames = list(nef[coo.beg], paste(rep(LETTERS[1:4], each = nb.h),
      1:nb.h, sep = "")))
  for (i in seq(along = coo.i)) {
    nef.i <- nef[(coo.beg[i]+1):coo.end[i]]
    x <- as.numeric(unlist(strsplit(nef.i, " ")))
    x1<-x[!is.na(x)]
    a.i<-x1[seq(1,length(x1),4)]
    b.i<-x1[seq(2,length(x1),4)]
    c.i<-x1[seq(3,length(x1),4)]
    d.i<-x1[seq(4,length(x1),4)]
    res[i, ]<-c(a.i,b.i,c.i,d.i)
  }
  return(Coe(res,method="eFourier"))}

## End(Not run)
```

`panel(Coo)` *Displays a one-page plot of a* `Coo`*-object.*

Description

`panel(Coo)` displays a one-page plot of a `Coo`-object.

Usage

```
panel(Coo, cols, borders, names=NULL, ...)
```

Arguments

Coo	The Coo object to plot.
cols	A vector of colors for drawing the outlines. Either a single value or of length exactly equals to the number of coordinates.
borders	A vector of colors for drawing the borders. Either a single value or of length exactly equals to the number of coordinates.
names	logical. Whether to display names.
...	Additional arguments to be passed to coo.list.panel.

Value

No returned value.

See Also

coo.list.panel is behind the scene. See also stack, diapo.

Examples

```
data(mosquito)
panel(mosquito, borders="grey20", col="grey90")

data(hearts)
panel(hearts, cols=rep(col.summer(8), each=30), names=TRUE)
```

PC.contrib *Shape variation along PC axis*

Description

PC.contrib calculates and plots shape variation along Principal Component axes.

Usage

```
PC.contrib(dudi, PC.r = 1:dudi$nf, sd = 2, cols = rep(NA, 3),
    borders = c("#000080", "#000000", "#EE0000"), lwd = 1, nb.pts = 300,
    plot = TRUE, legend = TRUE)
```

Arguments

dudi	a dudi.pca object.
PC.r	A range of integers indicating the PC axes on which to display shape variation.
sd	A numeric to indicate +/- the number of standard deviations to consider.
cols	A color string of length 3 to use to fill the shapes.
borders	A color string of length 3 to use for the shape borders.
lwd	A numeric to specify the lwd of the shapes.
nb.pts	integer to specify the number of points to draw the shapes.
plot	logical. Whether to plot the results.
legend	logical. Whether to add the legend.

Value

Invisibly returns a list that contains the coordiantes of the calculated shapes.

Examples

```
data(bot)
#botF <- eFourier(bot)
#botD <- pca(botF)
#PC.contrib(botD)
#PC.contrib(botD, sd=1) # only one sd
#PC.contrib(botD, PC.r=1:3, sd=1, cols=paste(col.sari(3), "55", sep=""),
#   borders=rep("black", 3), legend=FALSE)
# only 3 PC axis and some cosmectics.
```

---

pca *A wrapper for* dudi.pca *on* Coe*-objects.*

Description

pca is a wrapper for dudi.pca in ade4 on Coe-objects.

Usage

```
pca(Coe, subset.fac=NULL,
         subset.lev=NULL,
 row.w,
 col.w,
 center=TRUE,
 scale=FALSE,
 scannf=FALSE,
 nf=3)
```

Arguments

Coe	The Coe object
subset.fac	If non NULL, a character to specify the grouping factor on which to perform the PCA.
subset.lev	If non NULL for subset.fac, then a character to specify the level of subset.fac on which to perform the PCA.
row.w	A vector giving the rows weights (see **Details**).
col.w	A vector giving the cols weights (see **Details**).
center	logical. Whether to center or not the harmonic coefficients.
scale	logical. Whether to scale or not the harmonic coefficients.
scannf	logical. Whether to ask or not the number of PC to retain.
nf	integer. If scannf is FALSE, then the number of PC to retain.

Details

The default parameters should be satisfying for most of the cases. For instance, amplitude of coefficients is not rescaled since first harmonics capture most of the information. See the reference below for further technical discussion on PCA on harmonic coefficients.

Value

Returns a dudi.pca object

References

See the papers below that introduce ade4 and also the package's homepage: http://pbil.univ-lyon1.fr/ADE-4/

Dray, S. and Dufour, A.B. (2007): The ade4 package: implementing the duality diagram for ecologists. *Journal of Statistical Software*. **22**(4): 1-20.

Chessel, D. and Dufour, A.B. and Thioulouse, J. (2004): The ade4 package-I- One-table methods. *R News*. **4**: 5-10.

Dray, S. and Dufour, A.B. and Chessel, D. (2007): The ade4 package-II: Two-table and K-table methods. *R News*. **7**(2): 47-52.

See Also

dudi.pca.

Examples

```
data(bot)
botE <- eFourier(bot)
botP <- pca(botE)
dudi.plot(botP, "type")
beer <- pca(botE, "type", "beer")
dudi.plot(beer)
```

```
whisky <- pca(botE, "type", "whisky")
dudi.plot(whisky)
```

---

pca2shp *From a factorial map to a shape.*

Description

Converts a position in a factorial map to a shape. This function is used internally but might be interesting for a direct used on a PCA calculated on a matrix of harmonic coefficients.

Usage

```
pca2shp(pos, rot, mean.shp, method = c("efourier", "rfourier",
                 "tfourier"), scale = 1, amp = 1, trans = TRUE, nb.pts
                 = 64, rotate.shp)
```

Arguments

pos	A vector of positions on a given PC axis.
rot	A vector of multiplying values corresponding to the PC axis passed to pos.
mean.shp	A vector of harmonic coefficients corresponding to the mean shape.
method	A character, either "efourier", "rfourier" or "tfourier" (partial matches are allowed) to use to calculate morphological space.
scale	numeric. To scale the shape returned.
amp	numeric. To amplify the amplitude of the deformation.
trans	logical. Whether to translate of not the coordinate of the shape reconstructed according to pos.
nb.pts	numeirc. The number of points to use for reconstructed shapes.
rotate.shp	numeric. If specified, the angles (in radians) to rotate shapes plotted.

Value

Returns a list of coordinates.

Examples

```
data(bot)
```

pix2chc *Converts lists and matrices of coordinates into* .chc *(chain-coded) files.*

Description

pix2chc converts lists and matrices of coordinates into chain-coded coordinates as used in the SHAPE suite (see Iwata in the **References** below).

Usage

```
pix2chc(coo)
```

Arguments

coo	A list or a 2-col matrix with entire coordinates, see **Details**.

Details

Chain-code is a coding system for describing outlines in numbers from 0 to 7. Given the i^{th} pixel taken on an outline and considering its eight neighbors, the chain-code equivalent for describing the relative position of the $(i+1)^{th}$ pixel is a number from 0 (the next cell is eastward) to, counting counter clockwise, 7 (the next cell is south-eastward). See **References** and **Examples** below.

Value

Returns a vector of numeric that corresponds to chain-coded outline.

References

Freeman H. 1974. Computer processing of line-drawing images. ACM Computing Surveys (CSUR) 6: 57-97.

Iwata H, Ukai Y. 2002. SHAPE: a computer program package for quantitative evaluation of biological shapes based on elliptic Fourier descriptors. The Journal of Heredity 93: 384-385.

Kuhl FP, Giardina CR. 1982. Elliptic Fourier features of a closed contour. Computer Graphics and Image Processing 18: 236-258.

You can also have a look to the SHAPE's manual distributed with the program suite, that gives a description of the .chc format.

See Also

chc2pix for the reverse operation.

Examples

```
data(bot)
coo <- bot@coo[[1]]
chc <- pix2chc(coo)
coo.plot(chc2pix(chc))

# Illustration of chain coding
plot(NA, xlim=c(0, 3), ylim=c(0, 3), axes=FALSE, ann=FALSE, xaxs="i", yaxs="i")
title("Position of the next pixel and corresponding chain-code")
abline(h=0:3, v=0:3)
rect(1, 1, 2, 2, col="grey80")
text(1.5, 1.5, "Starting\npixel", cex=2)
text(x=c(2.5, 2.5, 1.5, rep(0.5, 3), 1.5, 2.5),
y=c(1.5, rep(2.5, 3), 1.5, rep(0.5, 3)), labels=0:7, cex=2)
```

`procGPAlign(Coo)` *Performs a Generalized Procrustes Alignement on a* `Coo` *object.*

Description

Performs a Generalized Procrustes Alignement on a `Coo` object, providing that landmarks have previously been defined.

Usage

```
procGPAlign(Coo, tol=1e-30)
```

Arguments

`Coo`	The `Coo` object.
`tol`	thresold for GPA. See `procGPA` in `shapes` package.

Details

If landmarks are defined, their coordinates are used for the GPA alignment, with the `procGPA` in `shapes`. Then the original outlines are rotated and scaled accordingly.

Value

Returns a `Coo` with centered, rotated and scaled outlines.

See Also

defLandmarks, cooLandmarks.

Examples

```
## Not run:
data(hearts)
# we add a lot of perturbations below
for (i in 1:hearts@coo.nb){
  coo.i <- hearts@coo[[i]] * runif(1, 0, 20)
  coo.i <- coo.rotate(coo.i, runif(1, 0, 2*pi))
  coo.i <- coo.trans(coo.i, runif(1, 0, 200), runif(1, 0, 200))
  hearts@coo[[i]] <- coo.i
}

def.par <- par(no.readonly = TRUE)
layout(matrix(1:3, 1, 3))
stack(hearts, borders="black", ldk.pch=20, ldk.cex=0.5)
title("Troubled hearts")
# we perform GPA
hearts2 <- procGPAlign(hearts)
stack(hearts2, borders="#1A1A1A22", ldk.pch=20, ldk.cex=0.5)
title("After GP alignement")
# we register a new baseline
hearts3 <- baseline(hearts2, 2, 4)
stack(hearts3, borders="#1A1A1A22", ldk.pch=20, ldk.cex=0.5)
title("And with a new baseline")
par(def.par)

## End(Not run)
```

`Ptolemy` *Ptolemaic ellipses and illustration of eFourier*

Description

`Ptolemy` is a method for `Coo` objects to calculate and display Ptolemaic ellipses which illustrates intuitively the principle behing elliptical Fourier analysis.

Usage

```
Ptolemy(Coo, id = 1, t = seq(0, 2 * pi, length = 7)[-1], nb.h = 3,
        nb.pts = 360, palette = col.sari, legend = FALSE)
```

Arguments

`Coo`	The `Coo` object on which to display Ptolemaic ellipses.
`id`	The `id` on which to display Ptolemaic ellipses.
`t`	A `vector` af angles (in radians) on which to display ellipses.
`nb.h`	`integer`. The number of harmonics to display.
`nb.pts`	`integer`. The number of points to use to display shapes.
`palette`	A color palette.
`legend`	`logical`. Whether to plot the legend box.

References

This method has been inspired by the figures found in the followings papers.

Kuhl FP, Giardina CR. 1982. Elliptic Fourier features of a closed contour. *Computer Graphics and Image Processing* **18**: 236-258.

Crampton JS. 1995. Elliptical Fourier shape analysis of fossil bivalves: some practical considerations. *Lethaia* **28**: 179-186.

See Also

efourier. An intuitive explanation of elliptic Fourier analysis can be found in the **Details** section of the efourier function.

Examples

```
data(bot)
Ptolemy(bot, 1)
```

`rFourier` *Calculates radii variation analysis on* `Coo` *objects.*

Description

This method performs a radii variation Fourier analysis on a `Coo`-class object, with the specified parameters (number of harmonics and number of smoothing iterations) and returns a `Coe`-class object containing harmonic coefficients (usually normalized). It accepts the same arguments as the function rfourier.

Usage

```
rFourier(Coo, nb.h= 40, smooth.it = 0, norm=TRUE)
```

Arguments

`Coo`	The `Coo` object.
`nb.h`	`integer`. The number of harmonics to calculate/use.
`smooth.it`	`integer`. The number of smoothing iterations to perform.
`norm`	`logical`. Whether to scale the outlines so that the mean length of the radii used equals 1.

See Also

See rfourier for the mathematical background.

Examples

```
data(bot)
rFourier(bot)
```

`rfourier` *Calculates radii variation Fourier analysis.*

Description

`rfourier` computes radii variation Fourier analysis from a matrix or a list of coordinates.

Usage

```
rfourier(coo, nb.h, smooth.it = 0, norm = FALSE, silent=FALSE)
```

Arguments

`coo`	A `list` or `matrix` of coordinates.
`nb.h`	`integer`. The number of harmonics to calculate/use.
`smooth.it`	`integer`. The number of smoothing iterations to perform.
`norm`	`logical`. Whether to scale the outlines so that the mean length of the radii used equals 1.
`silent`	`logical`. Whether to display diagnosis messages.

Details

Given a closed outline, the radius r, taken as the distance from the outline barycentre and a given point of the outline, can be expressed as a periodic function of the angle θ. Harmonics from 0 to k approximate the function $r(\theta)$:

$$r(\theta) = \frac{1}{2}a_0 + \sum_{n=1}^{k} a_k \cos(w_k\theta + b_k \sin(w_k\theta)$$

with:

$$a_n = \frac{2}{p}\sum_{n=1}^{p} r_i \cos(n\theta_i)$$

$$b_n = \frac{2}{p}\sum_{n=1}^{p} r_i \sin(n\theta_i)$$

with

$$a_0 = \sqrt{\frac{2}{p}}\sum_{n=1}^{p} r_i$$

The a_n and b_n harmonic coefficients, extracted for every individual shape, are then used for multivariate analyses.

Value

A list with these components:

an	vector of $a_{1->n}$ harmonic coefficients.
bn	vector of $b_{1->n}$ harmonic coefficients.
ao	ao Harmonic coefficient.
r	vector of radii lengths.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

rfourier.i for the inverse operation, rfourier.shape. efourier, tfourier for the other members of the Fourier's family.

Examples

```
data(bot)
coo <- coo.center(bot@coo[[1]]) # centering is almost mandatory for rfourier family
coo.plot(coo)
rf  <- rfourier(coo, 12)
rf
rfi <- rfourier.i(rf)
l2m(rfi)
coo.draw(rfi, border="red", col=NA)
```

---

rfourier.i *Calculates inverse radii variation analysis.*

Description

rfourier.i uses the inverse radii variation transformation to calculate a shape, when given a list with Fourier coefficients, typically obtained computed with rfourier.

Usage

```
rfourier.i(rf, nb.h, nb.pts=300)
```

Arguments

rf	A list with ao, an and bn components, typically as returned by rfourier.
nb.h	integer. The number of harmonics to calculate/use.
nb.pts	integer. The number of points to calculate.

Details

See efourier for the mathematical background.

Value

A list with components:

x	vector of x-coordinates.
y	vector of y-coordinates.
angle	vector of angles used.
r	vector of radii calculated.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

efourier for the reverse operation and also `rfourier.shape`. l2m, coeff.split may be useful.

Examples

```
data(bot)
coo <- coo.center(bot@coo[[1]]) # centering is almost mandatory for rfourier family
coo.plot(coo)
rf  <- rfourier(coo, 12)
rf
rfi <- rfourier.i(rf)
l2m(rfi)
coo.draw(rfi, border="red", col=NA)
    # it works since coo.draw and coo.plot retrieve "x"
    # and "y" components (through l2m when passed with a list.
```

`rfourier.shape` *Calculates and draw "rfourier" shapes.*

Description

`rfourier.shape` calculates a "Fourier radii variation shape" given Fourier coefficients (see `Details`) or can generate some "rfourier" shapes.

Usage

```
rfourier.shape(an, bn, nb.h, nb.pts=80, alpha=2, plot=TRUE)
```

Arguments

an	numeric. The a_n Fourier coefficients on which to calculate a shape.
bn	numeric. The b_n Fourier coefficients on which to calculate a shape.
nb.h	integer. The number of harmonics to use.
nb.pts	integer. The number of points to calculate.
alpha	numeric. The power coefficient associated with the (usually decreasing) amplitude of the Fourier coefficients (see **Details**).
plot	logical. Whether to plot or not the shape.

Details

rfourier.shape can be used by specifying nb.h and alpha. The coefficients are then sampled in an uniform distribution $(-\pi;\pi)$ and this amplitude is then divided by $harmonicrank^{a}lpha$. If alpha is lower than 1, consecutive coefficients will thus increase. See rfourier for the mathematical background.

Value

A list with components:

x	vector of x-coordinates.
y	vector of y-coordinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

rfourier.i.

Examples

```
data(bot)
rf <- rfourier(bot@coo[[1]], 24)
rfourier.shape(rf$an, rf$bn) # equivalent to rfourier.i(rf)
rfourier.shape() # not very interesting

rfourier.shape(nb.h=12) # better
rfourier.shape(nb.h=6, alpha=0.4, nb.pts=500)

panel(Coo(replicate(100, l2m(rfourier.shape(
    nb.h=6, alpha=0.4, nb.pts=200, plot=FALSE))))) # Butterflies
```

`smooth.qual` *Displays effect of smoothing on shape reconstruction.*

Description

Helps to visually estimate the number of smoothing iterations required by drawing original shape and smoothed outlines.

Usage

```
smooth.qual(Coo, id=1,
           smooth.range = c(10, 50, 200, 500, 1000),
           palette = col.summer)
```

Arguments

`Coo`	The `Coo`-object
`id`	`integer`. The `id` of the shape to display.
`smooth.range`	A `vector` of `integer` theat specify the number of smoothing iteration to perform and display.
`palette`	A color palette.

References

See Claude and Crampton & Crampton works below for details on smoothing outlines before an outline analysis:

Haines AJ, Crampton JS. 2000. Improvements To The Method Of Fourier Shape Analysis As Applied In Morphometric Studies. *Palaeontology* **43**: 765-783.

Crampton JS. 1995. Elliptical Fourier shape analysis of fossil bivalves: some practical considerations. *Lethaia* **28**: 179-186.

Examples

```
data(bot)
smooth.qual(bot, 15)
```

stack(Coo) *Plots all the outlines from a* Coo *on the same graph.*

Description

stack(Coo) plots all the outlines from a Coo on the same graph with graphical options. Note that you can also use plot(Coo) with the very same arguments.

Usage

```
stack(x, cols, borders,
          ldk=TRUE, ldk.pch=3, ldk.col="red", ldk.cex=1, ...)
```

Arguments

x	The Coo object to plot.
cols	A vector of colors for drawing the outlines. Either a single value or of length exactly equals to the number of coordinates.
borders	A vector of colors for drawing the borders. Either a single value or of length exactly equals to the number of coordinates.
ldk	logical. Whether to display landmarks (if any).
ldk.pch	A pch for these landmarks.
ldk.col	A color for these landmarks.
ldk.cex	A cex for these landmarks.
...	Additional arguments to be passed to coo.draw.

Value

No returned value.

See Also

stack, diapo.

Examples

```
data(mosquito)
stack(mosquito, borders="#1A1A1A22", first.point=FALSE)

data(hearts)
stack(hearts, borders="#1A1A1A22", ldk=FALSE)
stack(hearts, borders="#1A1A1A22", ldk=TRUE, ldk.col="#1A1A1A55")
stack(hearts, borders="#1A1A1A22", ldk=TRUE, ldk.col=col.summer(4), ldk.pch=20)
```

tFourier *Calculates tangent angle analysis on* Coo *objects.*

Description

tFourier performs a tangent angle Fourier analysis on a Coo-class object, with the specified parameters (number of harmonics and number of smoothing iterations) and returns a Coe-class object containing harmonic coefficients (usually normalized). It accepts the same arguments as the function efourier.

Usage

```
tFourier(Coo, nb.h= 40, smooth.it = 0, norm=TRUE)
```

Arguments

Coo	The Coo object.
nb.h	integer. The number of harmonics to calculate/use.
smooth.it	integer. The number of smoothing iterations to perform.
norm	logical. Whether to scale the outlines so that the mean length of the radii used equals 1.

See Also

See tfourier for the mathematical background.

Examples

```
data(bot)
tFourier(bot)
```

tfourier *Calculates tangent angle Fourier analysis.*

Description

tfourier computes tangent angle Fourier analysis from a matrix or a list of coordinates.

Usage

```
tfourier(coo, nb.h, smooth.it=0, norm = FALSE, silent = TRUE)
```

Arguments

coo	A list or matrix of coordinates
nb.h	integer. The number of harmonics to calculate/use
smooth.it	integer. The number of smoothing iterations to perform
norm	logical. Whether to scale and register new coordinates so that the first point used is sent on the origin.
silent	logical. Whether to display diagnosis messages.

Details

Given a closed outline which the outline has been scaled to 2π, $\phi(t)$ can be expressed as follows:

$$\phi(t) = \theta(t) - \theta(0) - t$$

where t is the distance along the outline, $\theta(t)$ the angle of the tangent vector at t and $\theta(0)$ the angle of the tangent vector taken for the first point. It can be removed for normalizing the coefficients obtained. Two coefficients per harmonics can be estimated as follow:

$$a_n = \frac{2}{p}\sum_{n=1}^{p}\phi(t)\cos n\theta_i$$

$$b_n = \frac{2}{p}\sum_{n=1}^{p}\phi(t)\sin n\theta_i$$

with

$$a_0 = \sqrt{\frac{2}{p}}\sum_{n=1}^{p}\phi(t)$$

Value

A list with these components:

ao	ao Harmonic coefficient.
an	vector of $a_{1->n}$ harmonic coefficients.
bn	vector of $b_{1->n}$ harmonic coefficients.
phi	vector of variation of the tangent angle.
t	vector of distance along the perimeter expressed in radians.
perimeter	numeric. The perimeter of the outline.
thetao	numeric. The first tangent angle.
x1	The x-coordinate of the first point.
y1	The y-coordinate of the first point.

References

Zahn CT, Roskies RZ. 1972. Fourier Descriptors for Plane Closed Curves. *IEEE Transactions on Computers* **C-21**: 269-281.

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

efourier, rfourier for the other members of the Fourier's family.

Examples

```
data(bot)
coo <- bot@coo[[1]]
coo.plot(coo)
tf  <- tfourier(coo, 12)
tf
tfi <- tfourier.i(tf)
l2m(tfi)
coo.draw(tfi, border="red", col=NA) # the outline is not closed...
coo.draw(tfourier.i(tf, force2close=TRUE), border="blue", col=NA) # we force it to close.
```

---

tfourier.i *Calculates inverse tangent angle Fourier analysis.*

Description

`tfourier.i` uses the inverse tangent angle Fourier transformation to calculate a shape, when given a list with Fourier coefficients, typically obtained computed with tfourier.

Usage

```
tfourier.i(tf, nb.h, nb.pts = 300, force2close = FALSE,
    rescale =TRUE, perim = 2 * pi, thetao = 0)
```

Arguments

`tf`	a list with ao, an and bn components, typically as returned by tfourier
`nb.h`	`integer`. The number of harmonics to calculate/use
`nb.pts`	`integer`. The number of points to calculate
`force2close`	`logical`. Whether to force the outlines calculated to close (see coo.force2close).
`rescale`	`logical`. Whether to rescale the points calculated so that their perimeter equals `perim`.
`perim`	The perimeter length to rescale shapes.
`thetao`	`numeric`. Radius angle to the reference (in radians)

Details

See tfourier for the mathematical background.

Value

A list with components:

x	vector of x-coordinates.
y	vector of y-coordinates.
phi	vector of interpolated changes on the tangent angle.
angle	vector of position on the perimeter (in radians).

References

Zahn CT, Roskies RZ. 1972. Fourier Descriptors for Plane Closed Curves. *IEEE Transactions on Computers* **C-21**: 269-281.

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

tfourier for the reverse operation and also tfourier.shape. l2m, coeff.split may be useful.

Examples

```
data(bot)
tfourier(bot@coo[[1]], 24)
tfourier.shape()
```

---

tfourier.shape *Calculates and draw "tfourier" shapes.*

Description

tfourier.shape calculates a "Fourier tangent angle shape" given Fourier coefficients (see Details) or can generate some "tfourier" shapes.

Usage

```
tfourier.shape(an, bn, ao = 0, nb.h, nb.pts=80, alpha=2, plot=TRUE)
```

Arguments

an	numeric. The a_n Fourier coefficients on which to calculate a shape.
bn	numeric. The b_n Fourier coefficients on which to calculate a shape.
ao	ao Harmonic coefficient.
nb.h	integer. The number of harmonics to use.
nb.pts	integer. The number of points to calculate.
alpha	numeric. The power coefficient associated with the (usually decreasing) amplitude of the Fourier coefficients (see **Details**).
plot	logical. Whether to plot or not the shape.

Details

tfourier.shape can be used by specifying nb.h and alpha. The coefficients are then sampled in an uniform distribution $(-\pi;\pi)$ and this amplitude is then divided by $harmonicrank^{a}lpha$. If alpha is lower than 1, consecutive coefficients will thus increase. See tfourier for the mathematical background.

Value

A list with components:

x	vector of x-coordinates.
y	vector of y-coordinates.

References

Claude, J. (2008) *Morphometrics with R*, Use R! series, Springer 316 pp.

See Also

tfourier.i.

Examples

```
data(bot)
tf <- tfourier(bot@coo[[1]], 24)
tfourier.shape(tf$an, tf$bn) # equivalent to rfourier.i(rf)
tfourier.shape()
tfourier.shape(nb.h=6, alpha=0.4, nb.pts=500)
panel(Coo(replicate(100, coo.force2close(l2m(
    tfourier.shape(nb.h=6, alpha=2, nb.pts=200, plot=FALSE))))) # biological shapes
```

---

tps.arr *Deformation "vector field" using Thin Plate Splines.*

Description

tps.arr(ows) calculates deformations between two configurations and illustrate them using arrows.

Usage

```
tps.arr(fr, to, amp=1, palette = col.summer,
        arr.nb = 100, arr.levels = 100, arr.len = 0.1,
        arr.ang = 30, arr.lwd = 1, arr.col = "grey50",
        shp = TRUE, shp.col =  rep(NA, 2), shp.border=col.gallus(2),
        shp.lwd = c(2, 2), shp.lty = c(1, 1))
```

Arguments

fr	The reference $(x; y)$ coordinates.
to	The target $(x; y)$ coordinates.
amp	An amplification factor of differences between fr and to.
palette	A color palette such those included in Momocs or produced with colorRampPalette.
arr.nb	A numeric. The number of arrows to calculate.
arr.levels	A numeric. The number of levels for the color of arrows.
arr.len	A numeric. The length of arrows.
arr.ang	A numeric. The angle for arrows' heads.
arr.lwd	A numeric. The lwd for drawing arrows.
arr.col	If palette is not used the color for arrwos.
shp	logical. Whether to draw shapes.
shp.col	Two colors for filling the shapes.
shp.border	Two colors for drawing the borders.
shp.lwd	Two lwd for drawing shapes.
shp.lty	Two lty fro drawing the shapes.

Value

No returned value.

Examples

```
data(bot)
botF <- eFourier(bot)
x <- meanShapes(botF, "type", nb.pts=80)
fr <- x$beer
to <- x$whisky
tps.arr(fr, to, arr.nb=400, palette=col.sari, amp=3)
```

---

tps.grid *Deformation grids using Thin Plate Splines.*

Description

tps.grid calculates and plots deformation grids between two configurations.

Usage

```
tps.grid(fr, to, amp=1, plot.full=TRUE, grid.outside = 0.2,
         grid.size = 20, grid.col   = "grey40",
         shp = TRUE, shp.col =  rep(NA, 2), shp.border=col.gallus(2),
         shp.lwd = c(2, 2), shp.lty = c(1, 1))
```

Arguments

fr	The reference $(x; y)$ coordinates.
to	The target $(x; y)$ coordinates.
amp	An amplification factor of differences between fr and to.
plot.full	logical. If FALSE graphical window will encompasses the entire outlines but maybe not the entire grid.
grid.outside	A numeric that indicates how much the grid extends beyond the range of outlines. Expressed as a proportion of the latter.
grid.size	A numeric to specify the number of grid cells on the longer axis on the outlines.
grid.col	A color for drawing the grid.
shp	logical. Whether to draw shapes.
shp.col	Two colors for filling the shapes.
shp.border	Two colors for drawing the borders.
shp.lwd	Two lwd for drawing shapes.
shp.lty	Two lty fro drawing the shapes.

Value

No returned value.

Examples

```
data(bot)
botF <- eFourier(bot)
x <- meanShapes(botF, "type", nb.pts=80)
fr <- x$beer
to <- x$whisky
tps.grid(fr, to, amp=3, grid.size=40)
```

---

tps.iso *Deformation isolines using Thin Plate Splines.*

Description

tps.iso calculates deformations between two configurations and map them with or without isolines.

Usage

```
tps.iso(fr, to, amp=1, palette = col.summer,
        iso.nb = 500, iso.levels = 12, cont=TRUE, cont.col="black",
        shp = TRUE, shp.col =  rep(NA, 2), shp.border=col.gallus(2),
        shp.lwd = c(2, 2), shp.lty = c(1, 1))
```

Arguments

fr	The reference $(x; y)$ coordinates.
to	The target $(x; y)$ coordinates.
amp	An amplification factor of differences between fr and to.
palette	A color palette such those included in Momocs or produced with colorRampPalette.
iso.levels	numeric. The number of levels for mapping the deformations.
iso.nb	A numeric. The number of points to use for the calculation of deformation.
cont	logical. Whether to draw contour lines.
cont.col	A color for drawing the contour lines.
shp	logical. Whether to draw shapes.
shp.col	Two colors for filling the shapes.
shp.border	Two colors for drawing the borders.
shp.lwd	Two lwd for drawing shapes.
shp.lty	Two lty fro drawing the shapes.

Value

No returned value.

Examples

```
data(bot)
botF <- eFourier(bot)
x <- meanShapes(botF, "type", nb.pts=80)
fr <- x$beer
to <- x$whisky
tps.iso(fr, to, iso.nb=2000, amp=3)
```

---

tps2d *Thin Plate Splines for 2D data.*

Description

tps2d is the core function for Thin Plate Splines. It is used internally but might be useful elsewhere.

Usage

```
tps2d(grid0, fr, to)
```

Arguments

grid0	A matrix of coordinates on which to calculate deformations.
fr	The reference $(x; y)$ coordinates.
to	The target $(x; y)$ coordinates.

Value

Returns a matrix of `(x; y)` coordinates with TPS-interpolated deformations.

See Also

The tps.grid,tps.iso, tps.arr functions use `tps2d`.

`vecs.param` *Some vector utilities.*

Description

Returns ratio of norms and signed angle between two vectors provided as four numeric.

Usage

```
vecs.param(r1, i1, r2, i2)
```

Arguments

`r1`	the "real" part of the first vector, i.e. difference in x-coordinates.
`i1`	the "imaginary" part of the first vector, i.e. difference in y-coordinates.
`r2`	the "real" part of the second vector, i.e. difference in x-coordinates.
`i2`	the "imaginary" part of the second vector, i.e. difference in y-coordinates.

Value

A list with two components: `r.norms` the ratio of (norm of vector 1)/(norm of vector 2) and `d.angle` the signed angle 'from' the first 'to' the second vector.

Examples

```
vecs.param(1, 0, 0, 2)
```

Index

∗Topic **Calibration**
hcontrib, 62
hpow, 63
hqual, 64
hquant, 65
smooth.qual, 89
∗Topic **Classes**
Class: Coe, 11
Class: Coo, 12
∗Topic **Datasets**
dataset: bot, 38
dataset: hearts, 39
dataset: mosquito, 39
dataset: trilo, 40
∗Topic **Import**
chc2Coo, 9
chc2pix, 10
Coe2nef, 13
Coo2chc, 36
import.jpg, 66
import.txt, 67
nef2Coe, 75
pix2chc, 81
∗Topic **Multivariate Analysis**
clust, 13
dudi.plot, 44
manova.Coe, 71
morpho.space, 74
PC.contrib, 77
pca, 78
∗Topic **Outline Analysis**
defLandmarks, 41
ef.amplify, 51
ellipse.par, 59
meanShapes, 72
∗Topic **Package**
Momocs Package, 73
∗Topic **Utilities**
A.mshape, 4
A.plot, 5
A.points, 6
A.segments, 7
baseline(Coo), 8
coeff.sel, 14
coeff.split, 15
Color palettes, 16
coo.draw, 21
coo.list.panel, 23
coo.oscillo, 24
coo.plot, 27
coo.rotate, 28
coo.rotate.center, 29
coo.slide, 33
coo.template, 34
coo.trans, 35
cooLandmarks(Coo), 37
dev.plot, 41
dev.segments, 42
diapo(Coo), 43
ed, 47
edi, 48
edm, 49
edm.nearest, 50
ellpar, 60
harm.pow, 61
panel(Coo), 76
pca2shp, 80
procGPAlign(Coo), 82
stack(Coo), 90
vecs.param, 99
∗Topic **coo Utilities**
a2l, 8
coo.align, 17
coo.baseline, 17
coo.center, 18
coo.centpos, 19
coo.centsize, 19
coo.close, 20

coo.force2close, 22
coo.ldk, 23
coo.perim, 25
coo.perim.pts, 26
coo.sample, 30
coo.sample.int, 30
coo.sample.rr, 31
coo.scale, 32
coo.smooth, 33
coo.unclose, 36
is.closed, 68
l2a, 69
l2m, 70
m2l, 71
tps.arr, 95
tps.grid, 96
tps.iso, 97
tps2d, 98

∗Topic **elliptical Fourier analysis**
eFourier, 52
efourier, 53
efourier.i, 55
efourier.norm, 56
efourier.shape, 58
Ptolemy, 83

∗Topic **radii variation Fourier analysis**
rFourier, 84
rfourier, 85
rfourier.i, 86
rfourier.shape, 87

∗Topic **tangent angle Fourier analysis**
tFourier, 91
tfourier, 91
tfourier.i, 93
tfourier.shape, 94

[,Coo,ANY,ANY,ANY-method *(*Class: Coo*)*, 12
[<-,Coo,ANY,ANY,ANY-method *(*Class: Coo*)*, 12

A.mshape, 4, *5–7*
A.plot, *4*, 5, *6, 7*, *37, 38*
A.points, *4, 5*, 6, *7*
A.segments, *4–6*, 7
a2l, 8, *69*

baseline *(*baseline(Coo)*)*, 8
baseline(Coo), 8
baseline,Coo-method *(*baseline(Coo)*)*, 8
baseline-methods *(*baseline(Coo)*)*, 8
bot *(*dataset: bot*)*, 38
boxplot,Coe-method *(*Class: Coe*)*, 11

chc2Coo, 9
chc2pix, 10, *81*
Class: Coe, 11
Class: Coo, 12
clust, 13
clust,Coe-method *(*clust*)*, 13
clust-methods *(*clust*)*, 13
Coe *(*Class: Coe*)*, 11
Coe-class *(*Class: Coe*)*, 11
Coe2nef, *10*, 13, *37*, *76*
coeff.sel, 14, *72*
coeff.split, 15, *56*, *87*, *94*
col.bcol *(*Color palettes*)*, 16
col.blackgallus *(*Color palettes*)*, 16
col.bw *(*Color palettes*)*, 16
col.gallus *(*Color palettes*)*, 16
col.india *(*Color palettes*)*, 16
col.sari *(*Color palettes*)*, 16
col.summer *(*Color palettes*)*, 16
col.wcol *(*Color palettes*)*, 16
Color palettes, 16
colorRampPalette, *16*, *46*, *62*, *64*, *66*, *96*, *98*
Coo *(*Class: Coo*)*, 12
Coo-class *(*Class: Coo*)*, 12
Coo.align *(*Class: Coo*)*, 12
coo.align, 17
Coo.align,Coo-method *(*Class: Coo*)*, 12
Coo.align-methods *(*Class: Coo*)*, 12
coo.baseline, 17
Coo.center *(*Class: Coo*)*, 12
coo.center, 18
Coo.center,Coo-method *(*Class: Coo*)*, 12
Coo.center-methods *(*Class: Coo*)*, 12
coo.centpos, 19, *20*
coo.centsize, *19*, 19
Coo.close *(*Class: Coo*)*, 12
coo.close, 20, *36*, *69*
Coo.close,Coo-method *(*Class: Coo*)*, 12
Coo.close-methods *(*Class: Coo*)*, 12
coo.draw, 21, *27*, *43*, *51*, *90*
coo.ef.amplify *(*ef.amplify*)*, 51
coo.force2close, 22, *93*
coo.ldk, 23, *41*
coo.list.panel, 23, *34*, *77*

coo.oscillo, 24
coo.oscillo1 *(coo.oscillo)*, 24
coo.perim, 25, *26*
coo.perim.pts, 26
coo.plot, *4–7*, *21*, *22*, *24*, 27
coo.rotate, 28, *29*
coo.rotate.center, *28*, 29
Coo.sample *(Class: Coo)*, 12
coo.sample, 30, *31*, *32*
Coo.sample,Coo-method *(Class: Coo)*, 12
Coo.sample-methods *(Class: Coo)*, 12
coo.sample.int, 30
coo.sample.rr, *30*, *31*, 31
coo.scale, 32
Coo.slide *(Class: Coo)*, 12
coo.slide, 33
Coo.slide,Coo-method *(Class: Coo)*, 12
Coo.slide-methods *(Class: Coo)*, 12
Coo.smooth *(Class: Coo)*, 12
coo.smooth, 33
Coo.smooth,Coo-method *(Class: Coo)*, 12
Coo.smooth-methods *(Class: Coo)*, 12
Coo.template *(Class: Coo)*, 12
coo.template, *23*, *24*, *32*, 34
Coo.template,Coo-method *(Class: Coo)*, 12
Coo.template-methods *(Class: Coo)*, 12
coo.trans, 35
Coo.unclose *(Class: Coo)*, 12
coo.unclose, *20*, 36, *69*
Coo.unclose,Coo-method *(Class: Coo)*, 12
Coo.unclose-methods *(Class: Coo)*, 12
Coo2chc, 36
cooLandmarks, *82*
cooLandmarks *(cooLandmarks(Coo))*, 37
cooLandmarks(Coo), 37
cooLandmarks,Coo-method *(cooLandmarks(Coo))*, 37
cooLandmarks-methods *(cooLandmarks(Coo))*, 37

dataset: bot, 38
dataset: hearts, 39
dataset: mosquito, 39
dataset: trilo, 40
defLandmarks, *23*, *38*, 41, *82*
defLandmarks,Coo-method *(defLandmarks)*, 41
defLandmarks-methods *(defLandmarks)*, 41
dev.plot, 41
dev.segments, 42
diapo, *44*, *77*, *90*
diapo *(diapo(Coo))*, 43
diapo(Coo), 43
diapo,Coo-method *(diapo(Coo))*, 43
diapo-methods *(diapo(Coo))*, 43
dist, *9*, *13*, *48–50*
dudi.pca, *44*, *79*
dudi.plot, 44, *74*, *75*

ed, *9*, *26*, 47, *48–50*
edi, 48
edm, *48*, 49, *50*, *65*
edm.nearest, *9*, *48*, *49*, 50, *65*
ef.amplify, 51
eFourier, *11*, *14*, *15*, 52, *55*, *57*
efourier, *14*, *15*, *51*, *52*, 53, *55–58*, *63*, *84*, *86*, *87*, *91*, *93*
eFourier,Coo-method *(eFourier)*, 52
eFourier-methods *(eFourier)*, 52
efourier.i, *55*, 55, *57*, *59*
efourier.norm, *52*, 56
efourier.shape, 58
ellipse.par, 59, *61*
ellipse.par,Coe-method *(ellipse.par)*, 59
ellipse.par-methods *(ellipse.par)*, 59
ellpar, *59*, *60*, 60

harm.pow, 61
hclust, *13*
hcontrib, 62
hcontrib,Coe-method *(hcontrib)*, 62
hcontrib-methods *(hcontrib)*, 62
hearts *(dataset: hearts)*, 39
hist,Coe-method *(Class: Coe)*, 11
hpow, 63
hpow,Coo-method *(hpow)*, 63
hpow-methods *(hpow)*, 63
hqual, 64
hqual,Coo-method *(hqual)*, 64
hqual-methods *(hqual)*, 64
hquant, 65
hquant,Coo-method *(hquant)*, 65
hquant-methods *(hquant)*, 65

import.img.Conte *(import.jpg)*, 66
import.img.prepare *(import.jpg)*, 66
import.jpg, 66, *68*
import.multi1.jpg *(import.jpg)*, 66

import.txt, *67*, 67
is.closed, *20*, *36*, 68

l2a, *8*, 69
l2m, *56*, 70, *71*, *87*, *94*
list.files, *67*

m2l, *70*, 71
manova, *72*
manova.Coe, 71
manova.Coe,Coe-method *(*manova.Coe*)*, 71
manova.Coe-methods *(*manova.Coe*)*, 71
meanShapes, 72
meanShapes,Coe-method *(*meanShapes*)*, 72
meanShapes-methods *(*meanShapes*)*, 72
Momocs *(*Momocs Package*)*, 73
Momocs Package, 73
Momocs-package *(*Momocs Package*)*, 73
morpho.space, *45*, *47*, 74
mosquito *(*dataset: mosquito*)*, 39

nef2Coe, *10*, *14*, *37*, 75

panel, *64*
panel *(*panel(Coo)*)*, 76
panel(Coo), 76
panel,Coo-method *(*panel(Coo)*)*, 76
panel-methods *(*panel(Coo)*)*, 76
PC.contrib, 77
pca, *44*, *46*, *47*, 78
pca,Coe-method *(*pca*)*, 78
pca-methods *(*pca*)*, 78
pca2shp, 80
pix2chc, *9*, *11*, *37*, 81
plot, *27*
plot,Coo,ANY-method *(*stack(Coo)*)*, 90
plot.phylo, *13*
polygon, *24*
procGPAlign, *38*
procGPAlign *(*procGPAlign(Coo)*)*, 82
procGPAlign(Coo), 82
procGPAlign,Coo-method *(*procGPAlign(Coo)*)*, 82
procGPAlign-methods *(*procGPAlign(Coo)*)*, 82
Ptolemy, *51*, *54*, *55*, 83
Ptolemy,Coo-method *(*Ptolemy*)*, 83
Ptolemy-methods *(*Ptolemy*)*, 83

quantile, *63*

rFourier, *11*, 84
rfourier, *14*, *15*, *55*, *84*, 85, *86*, *88*, *93*
rFourier,Coo-method *(*rFourier*)*, 84
rFourier-methods *(*rFourier*)*, 84
rfourier.i, *86*, 86, *88*
rfourier.shape, *86*, 87
rug, *46*

sample *(*Class: Coo*)*, 12
sample-methods *(*Class: Coo*)*, 12
seq, *31*
show,Coe-method *(*Class: Coe*)*, 11
show,Coo-method *(*Class: Coo*)*, 12
smooth.qual, 89
smooth.qual,Coo-method *(*smooth.qual*)*, 89
smooth.qual-methods *(*smooth.qual*)*, 89
stack, *44*, *64*, *77*, *90*
stack *(*stack(Coo)*)*, 90
stack(Coo), 90
stack,Coo-method *(*stack(Coo)*)*, 90
stack-methods *(*stack(Coo)*)*, 90

tFourier, *11*, *46*, *75*, 91
tfourier, *14*, *15*, *55*, *63*, *86*, *91*, 91, *93–95*
tFourier,Coo-method *(*tFourier*)*, 91
tFourier-methods *(*tFourier*)*, 91
tfourier.i, 93, *95*
tfourier.shape, 94
tps.arr, 95, *99*
tps.grid, 96, *99*
tps.iso, 97, *99*
tps2d, 98
trilo *(*dataset: trilo*)*, 40

vecs.param, 99