Package ‘Momocs’

October 6, 2020

Title Morphometrics using R
Version 1.3.2
Date 2020-10-05

Description The goal of ‘Momocs’ is to provide a complete, convenient, reproducible and open-source toolkit for 2D morphometrics. It includes most common 2D morphometrics approaches on outlines, open outlines, configurations of landmarks, traditional morphometrics, and facilities for data preparation, manipulation and visualization with a consistent grammar throughout. It allows reproducible, complex morphometrics analyses and other morphometrics approaches should be easy to plug in, or develop from, on top of this canvas.

License GPL-2 | GPL-3
Encoding UTF-8

URL https://github.com/MomX/Momocs/

BugReports https://github.com/MomX/Momocs/issues

Depends R(>= 3.2)

LazyData true

Imports cluster, dendextend, dplyr, magrittr, geometry, geomorph, ggplot2, graphics, grDevices, jpeg, MASS, progress, RColorBrewer, rgeos, sp, utils, vegan, tibble

Suggests devtools, knitr, rmarkdown, testthat, covr, roxygen2

RoxygenNote 7.1.0

NeedsCompilation no

Author Vincent Bonhomme [aut, cre],
           Julien Claude [aut] (core functions in base R)

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

Repository CRAN

Date/Publication 2020-10-06 15:20:11 UTC
R topics documented:

add_ldk .................................................. 7
apodemus .................................................. 8
arrange ................................................... 8
as_df ....................................................... 9
at_least ................................................... 10
bezier ...................................................... 11
bezier_i .................................................... 12
bot .......................................................... 13
boxplot.OutCoe .......................................... 13
boxplot.PCA ............................................. 14
breed ...................................................... 15
bridges ..................................................... 16
calibrate_deviations ................................. 17
calibrate_harmonicpower ......................... 20
calibrate_r2 ............................................ 22
calibrate_reconstructions ....................... 24
chaff ...................................................... 26
charring ................................................... 26
chop ......................................................... 27
classification_metrics ............................ 28
CLUST ................................................... 29
Coe .......................................................... 30
coeff_rearrange ....................................... 32
coeff_sel .................................................. 33
coeff_split ............................................... 34
color_palettes ......................................... 34
col_transp ............................................... 36
combine .................................................... 37
complex ................................................... 38
Coo .......................................................... 39
coo_align .................................................. 41
coo_aligncalliper ...................................... 42
coo_alignminradius .................................... 43
coo_alignnxax .......................................... 44
coo_angle_edges ........................................ 45
coo_angle_tangent ...................................... 46
coo_area ................................................... 47
coo_arrows ............................................... 48
coo_baseline ........................................... 48
coo_bookstein .......................................... 49
coo_boundingbox ....................................... 50
coo_calliper ............................................. 51
coo_centdist ............................................ 52
coo_center ............................................... 53
coo_centpos ............................................. 54
coo_centsize ............................................ 55
R topics documented:

- coo_check
- coo_chull
- coo_circularity
- coo_close
- coo_convexity
- coo_down
- coo_draw
- coo_draw_rads
- coo_dxy
- coo_eccentricity
- coo_elongation
- coo_extract
- coo_flipx
- coo_force2close
- coo_interpolate
- coo_intersect_angle
- coo_intersect_segment
- coo_is_closed
- coo_jitter
- coo_ldk
- coo_left
- coo_length
- coo_likely_clockwise
- coo_listpanel
- coo_lolli
- coo_lw
- coo_nb
- coo_oscillo
- coo_perim
- coo_plot
- coo_range
- coo_rectangularity
- coo_rectilinearity
- coo_rev
- coo_right
- coo_rotate
- coo_rotatecenter
- coo_ruban
- coo_sample
- coo_samplererr
- coo_sample_prop
- coo_scalars
- coo_scale
- coo_sheax
- coo_slice
- coo_slide
- coo_slidedirection
- coo_slidegap
### R topics documented:

- `coo_smooth` .......................................................... 105
- `coo_smoothcurve` ................................................. 106
- `coo_solidity` ..................................................... 107
- `coo_tac` ............................................................ 108
- `coo_template` ..................................................... 109
- `coo_trans` ........................................................ 110
- `coo_trim` .......................................................... 111
- `coo_trimbottom` .................................................. 112
- `coo_trimtop` ..................................................... 113
- `coo_truss` ........................................................ 113
- `coo_untiltx` ..................................................... 114
- `coo_up` ............................................................ 116
- `coo_width` ....................................................... 117
- `d` ................................................................. 117
- `def_ldk` ............................................................ 118
- `def_ldk_angle` ................................................... 119
- `def_ldk_tips` ..................................................... 120
- `def_links` ........................................................ 121
- `def_slidings` ..................................................... 121
- `dfourier` ........................................................ 122
- `dfourier_i` ....................................................... 124
- `dfourier_shape` .................................................. 125
- `dissolve` .......................................................... 126
- `drawers` ........................................................... 127
- `ed` ................................................................. 131
- `edi` ............................................................... 132
- `edm` ............................................................... 132
- `edm_nearest` ..................................................... 133
- `efourier` ........................................................ 134
- `efourier_i` ....................................................... 136
- `efourier_shape` .................................................. 137
- `export` ............................................................ 138
- `fac_dispatcher` .................................................. 140
- `fgProcrustes` .................................................... 141
- `fgsProcrustes` .................................................. 142
- `filter` ............................................................. 143
- `flip_PCaxes` ....................................................... 144
- `flower` ............................................................ 145
- `fProcrustes` ...................................................... 145
- `get_chull_area` ................................................... 146
- `get_ldk` ............................................................ 147
- `get_pairs` ........................................................ 148
- `get_slidings` ..................................................... 149
- `harm_pow` ........................................................ 149
- `hcontrib` ........................................................ 150
- `hearts` ............................................................ 151
- `img_plot` ........................................................ 152
- `import_Conte` .................................................... 152
R topics documented:

import_jpg .................................................. 153
import_jpg1 ................................................ 154
import_SteroMorph_curve1 ......................... 156
import_tps .................................................. 157
import_txt .................................................. 158
inspect ..................................................... 159
is ............................................................... 159
is_equallyspacedradii .................................. 161
KMEANS ...................................................... 162
KMEDOIDS .................................................... 163
layers ......................................................... 164
layers_morphospace ...................................... 167
LDA ............................................................. 168
Ldk ............................................................ 170
ldk_check .................................................... 171
ldk_chull ...................................................... 171
ldk_confell .................................................. 172
ldk_contour .................................................. 173
ldk_labels .................................................... 174
ldk_links ...................................................... 174
lf_structure .................................................. 175
links_all ...................................................... 176
links_delaunay ............................................. 177
MANOVA ....................................................... 177
MANOVA_PW .................................................. 179
MDS .............................................................. 180
measure ......................................................... 181
molars ......................................................... 182
Momocs ......................................................... 182
morphospace_positions ................................. 183
mosaic_engine ............................................. 184
mosquito ....................................................... 187
mouse ........................................................ 187
MSHAPES ..................................................... 188
mutate ......................................................... 189
NMDS .......................................................... 190
npoly .......................................................... 191
nsfishes ....................................................... 192
oak ............................................................. 193
olea ............................................................. 193
Opn ............................................................. 194
OpnCoe ......................................................... 195
opoly .......................................................... 196
opoly_i ........................................................ 197
Out ............................................................. 198
OutCoe ........................................................ 199
palettes ....................................................... 200
panel .......................................................... 202
R topics documented:

papers ................................................................. 204
PCA ................................................................. 205
PCcontrib ............................................................ 207
perm ................................................................. 208
pile ................................................................. 209
pix2chc .............................................................. 212
plot.LDA ............................................................ 213
plot.PCA ............................................................ 217
plot.CV .............................................................. 223
plot.CV2 ............................................................. 225
plot.devssegments .................................................. 227
plot.LDA ............................................................ 228
plot.MSHAPES ....................................................... 231
plot.NMDS .......................................................... 232
plot.PCA ............................................................ 233
plot.silhouette ...................................................... 236
plot_table ........................................................... 236
pProcrustes .......................................................... 237
Ptolemy ............................................................ 238
rearrange_ldk ....................................................... 239
reLDA ............................................................... 240
rename .............................................................. 241
rePCA .............................................................. 242
rescale .............................................................. 243
rfourier ............................................................. 244
rfourier_i ............................................................ 246
rfourier_shape ...................................................... 247
rm_asym ............................................................. 248
rm_harm ............................................................. 249
rm_missing .......................................................... 250
rm_uncomplete ...................................................... 251
rw_fac .............................................................. 252
sample_frac .......................................................... 253
sample_n ............................................................ 254
scree ............................................................... 255
select ............................................................... 256
sfourier ............................................................. 257
sfourier_i ............................................................ 258
sfourier_shape ...................................................... 259
shapes ............................................................... 260
slice ............................................................... 261
slidings_scheme ..................................................... 262
stack ............................................................... 262
subsetize ........................................................... 265
symmetry ........................................................... 266
tfourier ............................................................. 267
tfourier_i ............................................................ 268
tfourier_shape ...................................................... 270
*add_ldk*

**Description**

Helps to add new landmarks on a `Coo` object on top of existing ones. The number of landmarks must be specified and rows indices that correspond to the nearest points clicked on every outlines are stored in the `$ldk` slot of the `Coo` object.

**Usage**

```r
add_ldk(Coo, nb.ldk)
```

**Arguments**

- `Coo`: an Out or Opn object
- `nb.ldk`: the number of landmarks to add on every shape

**Details**

Note that if no landmarks are already defined, then this function is equivalent to `def_ldk`.

**Value**

an Out or an Opn object with some landmarks defined

**See Also**

Other `ldk/slidings` methods: `def_ldk()`, `def_slidings()`, `get_ldk()`, `get_slidings()`, `rearrange_ldk()`, `slidings_scheme()`
Examples

```r
## Not run:
hearts <- slice(hearts, 1:5) # to make it shorter to try
# click on 3 points, 5 times.
hearts <- def_ldk(hearts, 3)
# Don't forget to save the object returned by def_ldk...
hearts2 <- add_ldk(hearts, 3)
stack(hearts2)
hearts2$ldk

## End(Not run)
```

---

**apodemus**

*Data: Outline coordinates of Apodemus (wood mouse) mandibles*

---

**Description**

Data: Outline coordinates of Apodemus (wood mouse) mandibles

**Format**

A **Out** object 64 coordinates of 30 wood molar outlines.

**Source**


**See Also**

Other datasets: bot, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings

---

**arrange**

*Arrange rows by variables*

---

**Description**

Arrange shapes by variables, from the `$fac`. See examples and `?dplyr::arrange`.

**Usage**

`arrange(.data, ...)`
as_df

Arguments

.data a Coo, Coe, PCA object
... logical conditions

Details
dplyr verbs are maintained.

Value

a Momocs object of the same class.

See Also

Other handling functions: at_least(), chop(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsetize()

Examples

olea
# we create a new column
olea %>% mutate(id=1:length(.)) %>% fac$id
# same but now, shapes are arranged in a desc order, based on id
olea %>% mutate(id=1:length(.)) %>% arrange(desc(id)) %>% fac$id

as_df

Turn Momocs objects into tidy data_frames

Description

Used in particular for compatibility with the tidyverse

Usage

as_df(x, ...)

## S3 method for class 'Coo'
as_df(x, ...)

## S3 method for class 'Coe'
as_df(x, ...)

## S3 method for class 'PCA'
as_df(x, retain, ...)

## S3 method for class 'LDA'
as_df(x, retain, ...)
Arguments

\[ x \quad \text{an object, typically a Momocs object} \]
\[ \ldots \quad \text{useless here} \]
\[ \text{retain} \quad \text{numeric for use with scree methods. Default to all. If } <1, \text{ enough axes to retain this proportion of variance; if } >1, \text{ this number of axes.} \]

Value

\[
\text{a dplyr::data_frame}
\]

See Also

Other bridges functions: \texttt{bridges, complex, export()}

Examples

\begin{verbatim}
# first, some (baby) objects
b <- bot %>% coo_sample(12)
bf <- b %>% efourier(5, norm=TRUE)
# Coo object
b %>% as_df
# Coe object
bf %>% as_df

# PCA object
bf %>% PCA %>% as_df  # all PCs by default
bf %>% PCA %>% as_df(2)  # or 2
bf %>% PCA %>% as_df(0.99)  # or enough for 99%

# LDA object
bf %>% LDA(~fake) %>% as_df
# same options apply
\end{verbatim}

\begin{verbatim}
\textbf{at_least} \hspace{1cm} \textit{Retain groups with at least n shapes}
\end{verbatim}

Description

Examples are self-speaking.

Usage

\texttt{at_least(x, fac, N)}

Arguments

\begin{verbatim}
x \quad \text{any Momocs object} \\
fac \quad \text{the id of name of the } \$\text{fac column} \\
N \quad \text{minimal number of individuals to retain the group}
\end{verbatim}
bezier

Description
Calculates Bezier coefficients from a shape

Usage
bezier(coo, n)

Arguments
- coo: a matrix or a list of (x, y) coordinates
- n: the degree, by default the number of coordinates.

Value
a list with components:
- $J$ matrix of Bezier coefficients
- $B$ matrix of Bezier vertices.

Note
Directly borrowed for Claude (2008), and also called bezier there. Not implemented for open outlines but may be useful for other purposes.

References
See Also

Other bezier functions: bezier_i()

Examples

```r
set.seed(34)
x <- coo_sample(efourier_shape(), 5)
plot(x, ylim=c(-3, 3), asp=1, type='b', pch=20)
b <- bezier(x)
bi <- bezier_i(b$B)
lines(bi, col='red')
```

beziers_i

Calculates a shape from Bezier coefficients

Description

Calculates a shape from Bezier coefficients

Usage

`beziers_i(B, nb.pts = 120)`

Arguments

- `B` a matrix of Bezier vertices, such as those produced by `bezier`
- `nb.pts` the number of points to sample along the curve.

Value

a matrix of (x; y) coordinates

Note

Directly borrowed for Claude (2008), and called `beziercurve` there. Not implemented for open outlines but may be useful for other purposes.

References


See Also

Other bezier functions: `bezier()`
Examples

```r
set.seed(34)
x <- coo_sample(efourier_shape(), 5)
plot(x, ylim=c(-3, 3), asp=1, type='b', pch=20)
b <- bezier(x)
bi <- bezier_i(b$B)
lines(bi, col='red')
```

Data: Outline coordinates of beer and whisky bottles.

Description

Data: Outline coordinates of beer and whisky bottles.

Format

A Out object containing the outlines coordinates and a grouping factor for 20 beer and 20 whisky bottles

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.

See Also

Other datasets: apodemus, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings

Boxplot of morphometric coefficients

Description

Explores the distribution of coefficient values.

Usage

```r
## S3 method for class 'OutCoe'
boxplot(x, ...)
```

Arguments

- `x`  
  the Coe object
- `...`  
  useless here
Boxplot on PCA objects

Description

Boxplot on PCA objects

Usage

```r
## S3 method for class 'PCA'
boxplot(x, fac = NULL, nax, ...) 
```

Arguments

- **x**: PCA, typically obtained with `PCA`
- **fac**: factor, or a name or the column id from the $fac$ slot
- **nax**: the range of PC to plot (1 to 99pc total variance by default)
- **...**: useless here

Value

a ggplot object

Examples

```r
bot.f <- efourier(bot, 12)
bot.p <- PCA(bot.f)
boxplot(bot.p)
p <- boxplot(bot.p, 1)
#p + theme_minimal() + scale_fill_grey()
#p + facet_wrap(~PC, scales = "free")
```
breed

*breed*  
*Jitters Coe (and others) objects*

Description

This method applies column-wise on the *coe* of any *Coe* object but relies on a function that can be used on any matrix. It simply uses *morm* with the mean and sd calculated for every column (or row). For a *Coe* object, on every column, randomly generates coefficients values centered on the mean of the column, and with a sd equals to its standard deviates multiplied by *rate*.

Usage

```r
breed(x, ...)  
## Default S3 method:  
breed(x, fac, margin = 2, size, rate = 1, ...)  
## S3 method for class 'Coe'  
breed(x, fac, size, rate = 1, ...)
```

Arguments

- `x`  
  the object to permute
- `...`  
  useless here
- `fac`  
  a column, a formula or a column id from $fac$
- `margin`  
  numeric whether 1 or 2 (rows or columns)
- `size`  
  numeric the required size for the final object, same size by default
- `rate`  
  numeric the number of sd for *morm*, 1 by default.

See Also

Other farming: *perm()*

Examples

```r
m <- matrix(1:12, nrow=3)  
breed(m, margin=2, size=4)  
breed(m, margin=1, size=10)

bot.f <- efourier(bot, 12)  
bot.m <- breed(bot.f, size=80)  
bot.m %>% PCA %>% plot

# breed fac wise  
# bot.f %>% breed(~type, size=50) %>% PCA %>% plot(~type)
```
bridges

Convert between different classes

Description

Convert between different classes

Usage

```
l2m(l)
m2l(m)
d2m(d)
m2d(m)
l2a(l)
a2l(a)
a2m(a)
m2a(m)
m2ll(m, index = NULL)
```

Arguments

- `l`: list with x and y coordinates as components
- `m`: matrix of (x; y) coordinates
- `d`: data.frame with two columns
- `a`: array of (x; y) coordinates
- `index`: numeric, the number of coordinates for every slice

Value

the data in the required class

Note

`a2m/m2a` change, by essence, the dimension of the data. `m2ll` is used internally to handle coo and cur in Ldk objects but may be useful elsewhere

See Also

Other bridges functions: `as_df()`, `complex`, `export()`
Examples

# matrix/list
wings[[1]] %>% coo_sample(4) %>%
m2l() %>% print %>% # matrix to list
l2m() # and back

# data.frame/matrix
wings[[1]] %>% coo_sample(4) %>%
m2d() %>% print %>% # matrix to data.frame
d2m # and back

# list/array
wings %>% slice(1:2) %>%
coo %>% l2a %>% print %>% # list to array
a2l # and back

# array/matrix
wings %>% slice(1:2) %>%
l2a(coo) %>% # and array (from a list)
a2m %>% print %>% # to matrix
m2a # and back

# m2ll
m2ll(wings[[1]], c(6, 4, 3, 5)) # grab slices and coordinates

calibrate_deviations  
Quantitative calibration, through deviations, for Out and Opn objects

description

Calculate deviations from original and reconstructed shapes using a range of harmonic number.

Usage

calibrate_deviations()

calibrate_deviations_efourier(
    x,
    id = 1,
    range,
    norm.centsize = TRUE,
    dist.method = edm_nearest,
    interpolate.factor = 1,
    dist.nbpts = 120,
    plot = TRUE
)

calibrate_deviations_tfourier()
calibrate_deviations

x,
id = 1,
range,
norm.centsize = TRUE,
dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

calibrate_deviations_rfourier(
  x,
id = 1,
range,
norm.centsize = TRUE,
dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

calibrate_deviations_sfourier(
  x,
id = 1,
range,
norm.centsize = TRUE,
dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

calibrate_deviations_npoly(
  x,
id = 1,
range,
norm.centsize = TRUE,
dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

calibrate_deviations_opoly(
  x,
id = 1,
range,
norm.centsize = TRUE,
calibrate_deviations

dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

calibrate_deviations_dfourier(
  x,
id = 1,
range,
norm.centsize = TRUE,
dist.method = edm_nearest,
interpolate.factor = 1,
dist.nbpts = 120,
plot = TRUE
)

Arguments

x and Out or Opn object on which to calibrate_deviations
id the shape on which to perform calibrate_deviations
range vector of harmonics (or degree for opoly and npoly on Opn) on which to perform
  calibrate_deviations. If not provided, the harmonics corresponding to 0.9, 0.95
  and 0.99% of harmonic power are used.
norm.centsize logical whether to normalize deviation by the centroid size
dist.method a method such as edm_nearest to calculate deviations
interpolate.factor a numeric to increase the number of points on the original shape (1 by default)
dist.nbpts numeric the number of points to use for deviations calculations
plot logical whether to print the graph (FALSE is you just want the calculations)

Details

Note that from version 1.1, the calculation changed and fixed a problem. Before, the 'best' possible shape was calculated using the highest possible number of harmonics. This worked well for efourier but not for others (eg rfourier, tfourier) as they are known to be unstable with high number of harmonics. From now on, Momocs uses the 'real' shape, as it is (so it must be centered) and uses coo_interpolate to produce interpolate.factor times more coordinates as the shape has and using the default dist.method, eg edm_nearest, the latter finds the euclidean distance, for each point on the reconstructed shape, the closest point on this interpolated shape. interpolate.factor being set to 1 by default, no interpolation will be made in you do not ask for it. Note, that interpolation to decrease artefactual errors may also be done outside calibrate_deviations and will be probably be removed from it in further versions.

Note also that this code is quite old now and would need a good review, planned for 2018.

For *poly methods on Opn objects, the deviations are calculated from a degree 12 polynom.
Value

a ggplot object and the full list of intermediate results. See examples.

See Also

Other calibration: `calibrate_harmonicpower()`, `calibrate_r2()`, `calibrate_reconstructions`

Examples

```r
b5 <- slice(bot, 1:5) #for the sake of speed
b5 %>% calibrate_deviations_efourier()
b5 %>% calibrate_deviations_rfourier()
b5 %>% calibrate_deviations_tfourier()
b5 %>% calibrate_deviations_sfourier()

o5 <- slice(olea, 1:5) #for the sake of speed
o5 %>% calibrate_deviations_opoly()
o5 %>% calibrate_deviations_npoly()
o5 %>% calibrate_deviations_dfourier()
```

---

`calibrate_harmonicpower`

Quantitative calibration, through harmonic power, for Out and Opn objects

Description

Estimates the number of harmonics required for the four Fourier methods implemented in Momocs: elliptical Fourier analysis (see `efourier`), radii variation analysis (see `rfourier`) and tangent angle analysis (see `tfourier`) and discrete Fourier transform (see `dfourier`). It returns and can plot cumulated harmonic power whether dropping the first harmonic or not, and based and the maximum possible number of harmonics on the Coo object.

Usage

```r
library(momocs)

# Quantitative calibration, through harmonic power, for Out and Opn objects

# Usage

calibrate_harmonicpower()

calibrate_harmonicpower_efourier(
  x,
  id = 1:length(x),
  nb.h,
  drop = 1,
  thresh = c(90, 95, 99, 99.9),
  plot = TRUE
)

calibrate_harmonicpower_rfourier(
  x,
  id = 1:length(x),
  nb.h,
  drop = 1,
  thresh = c(90, 95, 99, 99.9),
  plot = TRUE
)

calibrate_harmonicpower_tfourier(
  x,
  id = 1:length(x),
  nb.h,
  drop = 1,
  thresh = c(90, 95, 99, 99.9),
  plot = TRUE
)

calibrate_harmonicpower_sfourier(
  x,
  id = 1:length(x),
  nb.h,
  drop = 1,
  thresh = c(90, 95, 99, 99.9),
  plot = TRUE
)
```
calibrate_harmonicpower

\[
\begin{align*}
\text{x}, \\
\text{id} = 1: \text{length(x)}, \\
\text{nb.h}, \\
\text{drop} = 1, \\
\text{thresh} = \text{c(90, 95, 99, 99.9)}, \\
\text{plot} = \text{TRUE}
\end{align*}
\]

\[
\begin{align*}
calibrate_{\text{harmonicpower\_tfourier}}( \\
x, \\
\text{id} = 1: \text{length(x)}, \\
\text{nb.h}, \\
\text{drop} = 1, \\
\text{thresh} = \text{c(90, 95, 99, 99.9)}, \\
\text{plot} = \text{TRUE}
\end{align*}
\]

\[
\begin{align*}
calibrate_{\text{harmonicpower\_sfourier}}( \\
x, \\
\text{id} = 1: \text{length(x)}, \\
\text{nb.h}, \\
\text{drop} = 1, \\
\text{thresh} = \text{c(90, 95, 99, 99.9)}, \\
\text{plot} = \text{TRUE}
\end{align*}
\]

\[
\begin{align*}
calibrate_{\text{harmonicpower\_dfourier}}( \\
x, \\
\text{id} = 1: \text{length(x)}, \\
\text{nb.h}, \\
\text{drop} = 1, \\
\text{thresh} = \text{c(90, 95, 99, 99.9)}, \\
\text{plot} = \text{TRUE}
\end{align*}
\]

\section*{Arguments}

\begin{itemize}
  \item \text{x} \quad \text{a Coo of Opn object}
  \item \text{id} \quad \text{the shapes on which to perform calibrate\_harmonicpower. All of them by default}
  \item \text{nb.h} \quad \text{numeric the maximum number of harmonic, on which to base the cumsum}
  \item \text{drop} \quad \text{numeric the number of harmonics to drop for the cumulative sum}
  \item \text{thresh} \quad \text{vector of numeric for drawing horizontal lines, and also used for minh below}
  \item \text{plot} \quad \text{logical whether to plot the result or simply return the matrix Silent message and progress bars (if any) with options(“verbose”=FALSE).}
\end{itemize}
Details

The power of a given harmonic $n$ is calculated as follows for elliptical Fourier analysis and the n-th harmonic: $HarmonicPower_n = \frac{A_n^2 + B_n^2 + C_n^2 + D_n^2}{2}$ 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 list with component:

- gg a ggplot object, q the quantile matrix
- minh a quick summary that returns the number of harmonics required to achieve a certain proportion of the total harmonic power.

See Also

Other calibration: `calibrate_deviations()`, `calibrate_r2()`, `calibrate_reconstructions`

Examples

```r
b5 <- bot %>% slice(1:5)
b5 %>% calibrate_harmonicpower_efourier(nb.h=12)
b5 %>% calibrate_harmonicpower_rfourier(nb.h=12)
b5 %>% calibrate_harmonicpower_tfourier(nb.h=12)
b5 %>% calibrate_harmonicpower_sfourier(nb.h=12)
# on Opn
olea %>% slice(1:5) %>%
  calibrate_harmonicpower_dfourier(nb.h=12)
## Not run:
# let customize the ggplot
library(ggplot2)
cal <- b5 %>% calibrate_harmonicpower_efourier(nb.h=12)
cal$gg + theme_minimal() +
  coord_cartesian(xlim=c(3.5, 12.5), ylim=c(90, 100)) +
  ggtitle("Harmonic power calibration")
## End(Not run)
```

---

**calibrate_r2**

Quantitative r2 calibration for Opn objects

Description

Estimates the r2 to calibrate the degree for npoly and opoly methods. Also returns a plot.
Usage

calibrate_r2()

calibrate_r2_opoly(  
  Opn,  
  id = 1:length(Opn),  
  degree.range = 1:8,  
  thresh = c(0.9, 0.95, 0.99, 0.999),  
  plot = TRUE,  
  ...
)

calibrate_r2_npoly(  
  Opn,  
  id = 1:length(Opn),  
  degree.range = 1:8,  
  thresh = c(0.9, 0.95, 0.99, 0.999),  
  plot = TRUE,  
  ...
)

Arguments

- **Opn**: an Opn object
- **id**: the ids of shapes on which to calculate r2 (all by default)
- **degree.range**: on which to calculate r2
- **thresh**: the threshold to return diagnostic
- **plot**: logical whether to print the plot
- **...**: useless here

Details

May be long, so you can estimate it on a sample either with id here, or one of sample_n or sample_frac

Note

Silent message and progress bars (if any) with options("verbose"=FALSE).

See Also

Other calibration: `calibrate_deviations()`, `calibrate_harmonicpower()`, `calibrate_reconstructions`

Examples

```r
## Not run:
olea %>% slice(1:5) %>% # for the sake of speed
  calibrate_r2_opoly(degree.range=1:5, thresh=c(0.9, 0.99))
```
olea %>% slice(1:5) %>% #for the sake of speed
calibrate_r2_npoly(degree.range=1:5, thresh=c(0.9, 0.99))

## End(Not run)

calibrate_reconstructions

*Calibrate using reconstructed shapes*

**Description**

Calculate and displays reconstructed shapes using a range of harmonic number. Compare them visually with the maximal fit. This explicitly demonstrates how robust efourier is compared to tfourier and rfourier.

**Usage**

```r
calibrate_reconstructions_efourier(x, id, range = 1:9)
calibrate_reconstructions_rfourier(x, id, range = 1:9)
calibrate_reconstructions_tfourier(x, id, range = 1:9)
calibrate_reconstructions_sfourier(x, id, range = 1:9)
calibrate_reconstructions_npoly(
  x,
  id,
  range = 2:10,
  baseline1 = c(-1, 0),
  baseline2 = c(1, 0)
)
calibrate_reconstructions_opoly(
  x,
  id,
  range = 2:10,
  baseline1 = c(-1, 0),
  baseline2 = c(1, 0)
)
calibrate_reconstructions_dfourier(
  x,
  id,
  range = 2:10,
```
calibrate_reconstructions

```r
baseline1 = c(-1, 0),
baseline2 = c(1, 0)
```

**Arguments**

- `x`: the Coo object on which to calibrate_reconstructions
- `id`: the shape on which to perform calibrate_reconstructions
- `range`: vector of harmonics on which to perform calibrate_reconstructions
- `baseline1`: \((x; y)\) coordinates for the first point of the baseline
- `baseline2`: \((x; y)\) coordinates for the second point of the baseline

**Value**

A ggplot object and the full list of intermediate results. See examples.

**See Also**

Other calibration: `calibrate_deviations()`, `calibrate_harmonicpower()`.

**Examples**

```r
### On Out
shapes %>%
calibrate_reconstructions_efourier(id=1, range=1:6)
# you may prefer efourier...
shapes %>%
calibrate_reconstructions_tfourier(id=1, range=1:6)

#' you may prefer efourier...
shapes %>%
calibrate_reconstructions_rfourier(id=1, range=1:6)

#' you may prefer efourier... # todo
#shapes %>%
# calibrate_reconstructions_sfourier(id=5, range=1:6)

### On Opn
olea %>%
calibrate_reconstructions_opoly(id=1)

olea %>%
calibrate_reconstructions_npoly(id=1)

olea %>%
calibrate_reconstructions_dfourier(id=1)
```
chaff

*Data: Landmark and semilandmark coordinates on cereal glumes*

**Description**

Data: Landmark and semilandmark coordinates on cereal glumes

**Format**

An Ldk object with 21 configurations of landmarks and semi-landmarks (4 partitions) sampled on cereal glumes

**Source**

Research support was provided by the European Research Council (Evolutionary Origins of Agriculture (grant no. 269830-EOA) PI: Glynis Jones, Dept of Archaeology, Sheffield, UK. Data collected by Emily Forster.

**See Also**

Other datasets: apodemus, bot, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings

charring

*Data: Outline coordinates from an experimental charring on cereal grains*

**Description**

Data: Outline coordinates from an experimental charring on cereal grains

**Format**

An Out object with 18 grains, 3 views on each, for 2 cereal species, charred at different temperatures for 6 hours (0C (no charring), 230C and 260C).

**Source**

Research support was provided by the European Research Council (Evolutionary Origins of Agriculture (grant no. 269830-EOA) PI: Glynis Jones, Dept of Archaeology, Sheffield, UK. Data collected by Emily Forster.

**See Also**

Other datasets: apodemus, bot, chaff, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings
chop

Split to several objects based on a factor

Description
Rougher slicing that accepts a classifier ie a column name from the $fac$ on Momocs classes. Returns a named (after every level) list that can be lapply-ed and combined. See examples.

Usage

chop(.data, fac)

Arguments

.data a Coo or Coe object

fac a column name from the $fac$

Value

a named list of Coo or Coe objects

See Also
Other handling functions: arrange(), at_least(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsetize()

Examples

olea %>%
  filter(var == "Aglan") %>% # to have a balanced nb of 'view'
  chop(~view) %>% # split into a list of 2
  npoly %>% # separately apply npoly
    # strict equivalent to lapply(npoly)
  combine %>% # recombine
  PCA %>% plot # an illustration of the 2 views
  # treated separately
classification_metrics

*Calculate classification metrics on a confusion matrix*

**Description**

In some cases, the class correctness or the proportion of correctly classified individuals is not enough, so here are more detailed metrics when working on classification.

**Usage**

```r
classification_metrics(x)
```

**Arguments**

- `x`: a table or an LDA object

**Value**

A list with the following components is returned:

1. `accuracy`: the fraction of instances that are correctly classified
2. `macro_prf`: data frame containing precision (the fraction of correct predictions for a certain class); recall, the fraction of instances of a class that were correctly predicted; f1 the harmonic mean (or a weighted average) of precision and recall.
3. `macro_avg`: just the average of the three `macro_prf` indices
4. `ova`: a list of one-vs-all confusion matrices for each class
5. `ova_sum`: a single of all ova matrices
6. `kappa`: measure of agreement between the predictions and the actual labels

**See Also**

The pages below are of great interest to understand these metrics. The code used is partley derived from the Revolution Analytics blog post (with their authorization). Thanks to them!

1. https://en.wikipedia.org/wiki/Precision_and_recall
2. https://blog.revolutionanalytics.com/2016/03/com_class_eval_metrics_r.html
3. https://www.r-bloggers.com/2016/03/is-your-classification-model-making-lucky-guesses/

Other multivariate: `CLUST()`, `KMEANS()`, `KMEDOIDS()`, `LDA()`, `MANOVA_PW()`, `MANOVA()`, `MDS()`, `MSHAPES()`, `NMDS()`, `PCA()`

**Examples**

```r
# some morphometrics on 'hearts'
hearts %>% fgProcrustes(tol=1) %>%
coo_slide(ldk=1) %>% efourier(norm=FALSE) %>% PCA()
# now the LDA and its summary
LDA(~aut) %>% classification_metrics()
```
Description

Performs hierarchical clustering through `dist` and `hclust`. So far it is mainly a wrapper around these two functions, plus plotting using the `dendextend` package facilities.

Usage

```r
CLUST(x, ...)
## Default S3 method:
CLUST(x, ...)
## S3 method for class 'Coe'
CLUST(
  x,
  fac,
  type = c("horizontal", "vertical", "fan")[1],
  k,
  dist_method = "euclidean",
  hclust_method = "complete",
  retain = 0.99,
  labels,
  lwd = 1/4,
  cex = 1/2,
  palette = pal.qual,
  ...
)
```

Arguments

- `x`: a Coe or PCA object
- `...`: useless here
- `fac`: factor specification for `fac_dispatcher`
- `type`: character one of c("horizontal", "vertical", "fan") (default: horizontal)
- `k`: numeric if provided and greater than 1, cut the tree into this number of groups
- `dist_method`: to feed `dist`'s method argument, that is one of euclidean (default), maximum, manhattan, canberra, binary or minkowski.
- `hclust_method`: to feed `hclust`'s method argument, one of ward.D, ward.D2, single, complete (default), average, mcquitty, median or centroid.
- `retain`: number of axis to retain if a PCA object is passed. If a number < 1 is passed, then the number of PCs retained will be enough to capture this proportion of variance via `scree_min`
labels factor specification for labelling tips and to feed `fac_dispatcher`

lwd for branches (default: 0.25)

cex for labels (default: 1)

palette one of available palettes

Value

a ggplot plot

See Also

Other multivariate: `KMEANS()`, `KMEDOIDS()`, `LDA()`, `MANOVA_PW()`, `MANOVA()`, `MDS()`, `MSHAPES()`, `NMDS()`, `PCA()`, `classification_metrics()`

Examples

# On Coe
bf <- bot %>% efourier(6)
CLUST(bf)
# with a factor and vertical
CLUST(bf, ~type, "v")
# with some cutting and different dist/hclust methods
CLUST(bf, 
    dist_method="maximum", hclust_method="average", 
    labels=~type, k=3, lwd=1, cex=1, palette=pal_manual(c("green", "yellow", "red")))

# On PCA
bf %>% PCA %>% CLUST

Description

Coe class is the 'parent' or 'super' class of `OutCoe`, `OpnCoe`, `LdkCoe` and `TraCoe` classes.

Usage

Coe(…)

Arguments

… anything and, anyway, this function will simply returns a message.
Details

Useful shortcuts are described below. See browseVignettes("Momocs") for a detail of the design behind Momocs' classes.

Coe class is the 'parent' class of the following 'child' classes

- **OutCoe** for coefficients from closed **out**lines morphometrics
- **OpnCoe** for coefficients from open **op**en outlines morphometrics
- **LdkCoe** for coefficients from configuration of **landmarks** morphometrics.

In other words, OutCoe, OpnCoe and LdkCoe classes are all, primarily, Coe objects on which we define generic and specific methods. See their respective help pages for more help.

You can access all the methods available for Coe objects with methods(class=Coe).

See Also

Other classes: Coo(), Ldk(), OpnCoe(), Opn(), OutCoe(), Out(), TraCoe()

Examples

```r
## Not run:
# to see all methods for Coe objects.
methods(class='Coe')
# to see all methods for OutCoe objects.
methods(class='OutCoe') # same for OpnCoe, LdkCoe, TraCoe

bot.f <- efourier(bot, 12)
bot.f
class(bot.f)
inherits(bot.f, "Coe")

# if you want to work directly on the matrix of coefficients
bot.f$coe

#getters
bot.f[1]
bot.f[1:5]

#setters
bot.f[1] <- 1:48
bot.f[1]

bot.f[1:5]

# An illustration of Momocs design. See also browseVignettes("Momocs")
op <- opoly(olea, 5)
op
class(op)
op$coe # same thing
```
Rearrange a matrix of (typically Fourier) coefficients

Description
Momocs uses colnamed matrices to store (typically) Fourier coefficients in Coe objects (typically OutCoe). They are arranged as rank-wise: \( A_1, A_2, \ldots, A_n, B_1, \ldots, B_n, C_1, \ldots, C_n, D_1, \ldots, D_n \). From other softwares they may arrive as \( A_1, B_1, C_1, D_1, \ldots, A_n, B_n, C_n, D_n \), this functions helps to go from one to the other format. In short, this function rearranges column order. See examples.

Usage

\[ \text{coeff\_rearrange}(x, \text{by} = \text{c("name", "rank")}[1]) \]

Arguments

- \( x \) matrix (with colnames)
- \( \text{by} \) character either "name" (\( A_1, A_2, \ldots \)) or "rank" (\( A_1, B_1, \ldots \))

Examples

```r
m_name <- m_rank <- matrix(1:32, 2, 16)
# this one is ordered by name
colnames(m_name) <- paste0(rep(letters[1:4], each=4), 1:4)
# this one is ordered by rank
colnames(m_rank) <- paste0(letters[1:4], rep(1:4, each=4))

m_rank
m_rank %>% coeff\_rearrange(by="name")
# no change

m_name
m_name %>% coeff\_rearrange(by="name") # no change
m_name %>% coeff\_rearrange(by="rank")
```
**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 $\text{coe}$ 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 elsewhere.

**Usage**

```r
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**

```r
bot.f <- efourier(bot, 32)
coe <- bot.f$coe # the raw matrix
coe
```

# if you want, say the first 8 harmonics but not the first one
```r
retain <- coeff_sel(retain=8, drop=1, nb.h=32, cph=4)
head(coe[, retain])
```
**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 $coe$ 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 elsewhere.

**Usage**

```r
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**

```r
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**

Colors, colors, colors.
Usage

    col_summer(n)
    col_summer2(n)
    col_spring(n)
    col_autumn(n)
    col_black(n)
    col_solarized(n)
    col_gallus(n)
    col_qual(n)
    col_heat(n)
    col_hot(n)
    col_cold(n)
    col_sari(n)
    col_india(n)
    col_bw(n)
    col_grey(n)

Arguments

    n  the number of colors to generate from the color palette

Value

    colors (hexadecimal format)

Note

Among available color palettes, col_solarized is based on Solarized: https://ethanschoonover.com/solarized/; col_div, col_qual, col_heat, col_cold and col_gallus are based on on ColorBrewer2: https://colorbrewer2.org/.

Examples

    wheel <- function(palette, n=10){

op <- par(mar=rep(0, 4)); on.exit(par(op))
pie(rep(1, n), col=palette(n), labels=NA, clockwise=TRUE)}

# Qualitative
wheel(col_qual)
wheel(col_solarized)
wheel(col_summer)
wheel(col_summer2)
wheel(col_spring)
wheel(col_autumn)

# Divergent
wheel(col_gallus)
wheel(col_india)

# Sequential
wheel(col_heat)
wheel(col_hot)
wheel(col_cold)
wheel(col_sari)
wheel(col_bw)
wheel(col_grey)

# Black only for pubs
wheel(col_black)

<table>
<thead>
<tr>
<th>col_transp</th>
<th>Transparency helpers and palettes</th>
</tr>
</thead>
</table>

**Description**

To ease transparency handling.

**Usage**

```r
col_transp(n, col = "#000000", ceiling = 1)
col_alpha(cols, transp = 0)
```

**Arguments**

- `n` the number of colors to generate
- `col` a color in hexadecimal format on which to generate levels of transparency
- `ceiling` the maximal opacity (from 0 to 1)
- `cols` on or more colors, provided as hexadecimal values
- `transp` numeric between 0 and 1, the value of the transparency to obtain
Examples

```r
x <- col_transp(10, col='#000000')
x
barplot(1:10, col=x, main='a transparent black is grey')

summer10 <- col_summer(10)
summer10
summer10.transp8 <- col_alpha(summer10, 0.8)
summer10.transp8
summer10.transp2 <- col_alpha(summer10, 0.8)
summer10.transp2
x <- 1:10
barplot(x, col=summer10.transp8)
barplot(x/2, col=summer10.transp2, add=TRUE)
```

---

**combine**

*Combine several objects*

**Description**

Combine Coo objects after a slicing, either manual or using `slice` or `chop`. Note that on Coo object, it combines row-wise (ie, merges shapes as a `c` would do); but on Coe it combines column-wise (merges coefficients). In the latter case, Coe must have the same number of shapes (not necessarily the same number of coefficients). Also the `$fac` of the first Coe is retrieved. A separate version may come at some point.

**Usage**

```r
combine(...)```

**Arguments**

```r
...

... a list of Out(Coe), Opn(Coe), Ldk objects (but of the same class)
```

**Note**

Note that the order of shapes or their coefficients is not checked, so anything with the same number of rows will be merged.

**See Also**

Other handling functions: `arrange()`, `at_least()`, `chop()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rm_uncomplete()`, `rw_fac()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`
Examples

```r
w <- filter(bot, type=="whisky")
b <- filter(bot, type=="beer")
combine(w, b)
# or, if you have many levels
bot_s <- chop(bot, ~type)
bot_s$whisky
# note that you can apply something (single function or a more complex pipe) then combine everyone, since combine also works on lists
# eg:
# bot_s2 <- efourier(bot_s, 10) # equivalent to lapply(bot_s, efourier, 10)
# bot_sf <- combine(bot_s2)

# pipe style
efourier(bot_s, 10) %>% combine()
```

---

**complex**

*Convert complex to/from cartesian coordinates*

Description

Convert complex to/from cartesian coordinates

Usage

```r
cpx2coo(Z)
coo2cpx(coo)
```

Arguments

- `Z` coordinates expressed in the complex form
- `coo` coordinates expressed in the cartesian form

Value

coordinates expressed in the cartesian/complex form

See Also

Other bridges functions: `as_df()`, `bridges`, `export()`

Examples

```r
shapes[4] %>%
  coo_sample(24) %>%
  coo2cpx() %>%
  cpx2coo() %>%
# from cartesian
# to complex
# and back
```
Coo

Coo "super" class

Description

Coo class is the 'parent' or 'super' class of Out, Opn and Ldk classes.

Usage

Coo(...)

Arguments

... anything and, anyway, this function will simply returns a message.

Details

Useful shortcuts are described below. See browseVignettes("Momocs") for a detail of the design behind Momocs' classes.

Coo class is the 'parent' class of the following 'child' classes

- Out for closed outlines
- Opn for open outlines
- Ldk for configuration of landmarks

Since all 'child classes' of them handle \((x; y)\) coordinates among other generic methods, but also all have their specificity, this architecture allow to recycle generic methods and to use specific methods.

In other words, Out, Opn and Ldk classes are all, primarily, Coo objects on which we define generic and specific methods. See their respective help pages for more help.

Coo objects all have the following components:

- $coo which is a list of matrices for coordinates
- $fac a data_frame for covariates (if any). You can provide this data_frame directly, as long as it has as many rows as there are matrices in $coo (see examples), or use an helper function such as lf_structure.

You can access all the methods available for Coo objects with methods(class=Coo).

See Also

Other classes: Coe(), Ldk(), OpnCoe(), Opn(), OutCoe(), Out(), TraCoe()
Examples

```r
## Not run:
# to see all methods for Coo objects.
methods(class='Coo')

# to see all methods for Out objects.
methods(class='Out') # same for Opn and Ldk

# Let's take an Out example. But all methods shown here
# work on Ldk (try on 'wings') and on Opn ('olea')
bot

# Primarily a 'Coo' object, but also an 'Out'
class(bot)
inherits(bot, "Coo")
panel(bot)
stack(bot)
plot(bot)

# Getters (you can also use it to set data)
bot[1] %>% coo_plot()
bot[1:5] %>% str()

# Setters
panel(bot)

panel(bot)

# access the different components
# $coo coordinates
head(bot$coo)
# $fac grouping factors
head(bot$fac)
# or if you know the name of the column of interest
bot$type
# table
table(bot$fac)
# an internal view of an Out object
str(bot)

# subsetting
# see ?filter, ?select, and their 'see also' section for the
# complete list of dplyr-like verbs implemented in Momocs

length(bot) # the number of shapes
names(bot) # access all individual names
bot2 <- bot
names(bot2) <- paste0('newnames', 1:length(bot2)) # define new names

# Add a $fac from scratch
```
coo <- bot[1:5] # a list of five matrices
length(coo)
sapply(coo, class)

fac <- data.frame(name=letters[1:5], value=c(5:1))
# Then you have to define the subclass using the right builder
# here we have outlines, so we use Out
x <- Out(coo, fac)
x$coo
x$fac

## End(Not run)

---

### coo_align

**Aligns coordinates**

**Description**

Aligns the coordinates along their longer axis using var-cov matrix and eigen values.

**Usage**

```r
coo_align(coo)
```

**Arguments**

- `coo` matrix of (x; y) coordinates or any `Coo` object.

**Value**

a matrix of (x; y) coordinates, or a `Coo` object.

**See Also**

Other aligning functions: [coo_aligncalliper()], [coo_alignminradius()], [coo_alignxax()]

Examples

```
coo_plot(bot[1])
coo_plot(coo_align(bot[1]))

# on a Coo
b <- bot %>% slice(1:5) # for speed sake
stack(coo_align(b))
```

coo_aligncalliper

**Aligns shapes along their 'calliper length'**

Description

And returns them registered on bookstein coordinates. See coo_bookstein.

Usage

```
coo_aligncalliper(coo)
```

Arguments

```
coo
  matrix of (x; y) coordinates or any Coo object.
```

Value

```
a matrix of (x; y) coordinates, or any Coo object.
```

See Also

Other aligning functions: coo_alignminradius(), coo_alignxax(), coo_align()

Other coo utilities: coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()
coo_alignminradius

Description
And returns them slided with the first coordinate on the east. May be used as an aligning strategy on shapes with a clear ‘invaginate’ part.

Usage
coo_alignminradius(coo)

Arguments
coo matrix of (x; y) coordinates or any Coo object.

Value
a matrix of (x; y) coordinates, or a Coo object.

See Also
Other aligning functions: coo_aligncalliper(), coo_alignxax(), coo_align()
Other coo utilities: coo_aligncalliper(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_sampler(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidetop(), coo_smooth(), coo_smoothcurve(), coo_template(), coo_trans(), coo_trimb(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples
b <- bot %>% slice(1:5) # for speed sake
stack(coo_alignminradius(b))
coo_alignxax  
Aligns shapes along the x-axis

Description
Align the longest axis of a shape along the x-axis.

Usage
coo_alignxax(coo)

Arguments
coo   matrix of (x; y) coordinates or any Coo object.

Details
If some shapes are upside-down (or mirror of each others), try redefining a new starting point (eg with coo_slidedirection) before the alignment step. This may solve your problem because coo_calliper orders the $arr.ind used by coo_aligncalliper.

Value
a matrix of (x; y) coordinates, or any Coo object.

See Also
Other aligning functions: coo_aligncalliper(), coo_alignminradius(), coo_align()
Other coo Utilities: coo_aligncalliper(), coo_alignminradius(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_sampler(), coo_sample(), coo_scale(), coo_shearX(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples
## Not run:
b <- bot[1]
coo_plot(b)
coo_plot(coo_alignxax(b))

## End(Not run)
coo_angle_edges

Calculates the angle of every edge of a shape

Description

Returns the angle (in radians) of every edge of a shape.

Usage

coo_angle_edges(coo, method = c("atan2", "acos")[1])

## Default S3 method:
coo_angle_edges(coo, method = c("atan2", "acos")[1])

## S3 method for class 'Coo'
coo_angle_edges(coo, method = c("atan2", "acos")[1])

Arguments

coo
  a matrix or a list of (x; y) coordinates or any Coo
method
  'atan2' (or 'acos') for a signed (or not) angle.

Value

numeric the angles in radians for every edge.

Note

coo_thetapts is deprecated and will be removed in future releases.

See Also

Other coo_descriptors: coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(),
  coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(),
  coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(),
  coo_tac(), coo_width()

Examples

b <- coo_sample(bot[1], 64)
coo_angle_edges(b)
coo_angle_tangent  

Calculates the tangent angle along the perimeter of a shape

Description

Calculated using complex numbers and returned in radians minus the first one (modulo 2*\pi).

Usage

    coo_angle_tangent(coo)

    ## Default S3 method:
    coo_angle_tangent(coo)

    ## S3 method for class 'Coo'
    coo_angle_tangent(coo)

    coo_tangle(coo)

Arguments

    coo  a matrix of coordinates or any Coo

Value

numeric, the tangent angle along the perimeter, or a list of those for Coo

See Also

tfourier

Other coo_ descriptors: coo_angle_edges(), coo_area(), coo_boundingbox(), coo_chull(),
coo_circularity(), coo_convexity(), coo_eccentricity, coo_elongation(), coo_length(),
coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(),
coo_tac(), coo_width()

Examples

    b <- bot[1]
    phi <- coo_angle_tangent(b)
    phi2 <- coo_angle_tangent(coo_smooth(b, 2))
    plot(phi, type='l')
    plot(phi2, type='l', col='red') # ta is very sensible to noise

    # on Coo
    bot %>% coo_angle_tangent
coo_area

Calculates the area of a shape

Description

Calculates the area for a (non-crossing) shape.

Usage

coo_area(coo)

Arguments

coo a matrix of (x; y) coordinates.

Value

numeric, the area.

Note

Using area.poly in gpc package is a good idea, but their licence impedes Momocs to rely on it. But here is the function to do it, once gpc is loaded: area.poly(as(coo, 'gpc.poly'))

See Also

Other coo_ descriptors: coo_angle_edges(), coo_angle_tangent(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity, coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples

coo_area(bot[1])
# for the distribution of the area of the bottles dataset
hist(sapply(bot$coo, coo_area), breaks=10)
coo_arrows

Plots (lollipop) differences between two configurations

Description

Draws 'arrows' between two configurations.

Usage

```
coo_arrows(coo1, coo2, length = coo_centsize(coo1)/15, angle = 20, ...)
```

Arguments

- `coo1`: A list or a matrix of coordinates.
- `coo2`: A list or a matrix of coordinates.
- `length`: A length for the arrows.
- `angle`: An angle for the arrows.
- `...`: Optional parameters to feed `arrows`.

See Also

Other plotting functions: `coo_draw()`, `coo_listpanel()`, `coo_lolli()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_labels()`, `ldk_links()`, `plot_devsegments()`, `plot_table()`

Examples

```
coo_arrows(coo_sample(olea[3], 50), coo_sample(olea[6], 50))
title("Hi there !")
```

coo_baseline

Register new baselines

Description

A non-exact baseline registration on t1 and t2 coordinates, for the 1dk1-th and 1dk2-th points. By default it returns Bookstein’s coordinates.

Usage

```
coo_baseline(coo, ldk1, ldk2, t1, t2)
```
Arguments

- `coo`: matrix of (x; y) coordinates or any Coo object.
- `ldk1`: numeric the id of the first point of the new baseline (the first, by default)
- `ldk2`: numeric the id of the second point of the new baseline (the last, by default)
- `t1`: numeric the (x; y) coordinates of the 1st point of the new baseline
- `t2`: numeric the (x; y) coordinates of the 2nd point of the new baseline

Value

- a matrix of (x; y) coordinates or a Coo object.

See Also

- Other baselining functions: `coo_bookstein()`
- Other coo_ utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample()`, `coo_sample_prop()`, `coo_sampler()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
h <- hearts %>% slice(1:5) # for speed sake
stack(h)
stack(coo_baseline(h, 2, 4, c(-1, 0), c(1, 1)))
```

---

coo_bookstein | Register Bookstein’s coordinates

Description

Registers a new baseline for the shape, with the `ldk1`-th and `ldk2`-th points being set on \( (x = -0.5; y = 0) \) and \( (x = 0.5; y = 0) \), respectively.

Usage

```r
coo_bookstein(coo, ldk1, ldk2)
```

Arguments

- `coo`: matrix of (x; y) coordinates or any Coo object.
- `ldk1`: numeric the id of the first point of the new baseline (the first, by default)
- `ldk2`: numeric the id of the second point of the new baseline (the last, by default)
Details

For `Out`, it tries to do it using `$ldk` slot. Also the case for `Opn`, but if no landmark is defined, it will do it on the first and the last point of the shape.

For `Out` and `Opn` defines the first landmark as the first point of the new shapes with `coo_slide`.

Value

a matrix of (x; y) coordinates, or a `Coo` object.

See Also

Other baselining functions: `coo_baseline()`

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_sampler()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()` , `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
h <- hearts %>% slice(1:5) # for the sake of speed
stack(h)
stack(coo_bookstein(h, 2, 4))
h <- hearts[1]
coo_plot(h)
coo_plot(coo_bookstein(h, 20, 57), border='red')
```

---

**coo_boundingbox**  
Calculates coordinates of the bounding box

Description

Calculates coordinates of the bounding box

Usage

`coo_boundingbox(coo)`

Arguments

- `coo`  
  matrix of (x; y) coordinates or any `Coo` object.

Value

data.frame with coordinates of the bounding box
See Also

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_sampler()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidetep()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbotttom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()

Other `coo_` descriptors: `coo_angle_edges()`, `coo_angle_tangent()`, `coo_area()`, `coo_chull()`, `coo_circularity()`, `coo_convexity()`, `coo_eccentricity()`, `coo_ellongation()`, `coo_length()`, `coo_lw()`, `coo_rectangularity()`, `coo_rectlinearity()`, `coo_scalars()`, `coo_solidity()`, `coo_tac()`, `coo_width()

Examples

```r
bot[1] %>% coo_boundingbox()
bot %>% coo_boundingbox()
```

---

**coo_calliper**  
*Calculates the calliper length*

**Description**

Also called the Feret’s diameter, the longest distance between two points of the shape provided.

**Usage**

```r
coo_calliper(coo, arr.ind = FALSE)
```

**Arguments**

- `coo` a matrix of (x; y) coordinates or any Coo
- `arr.ind` logical, see below.

**Value**

numeric, the centroid size. If `arr.ind` = TRUE, a data_frame.

See Also

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_sampler()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidetep()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbotttom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`
Examples

```r
b <- bot[1]
coo.calliper(b)
p <- coo.calliper(b, arr.ind=TRUE)
p
p$length
ids <- p$arr_ind[[1]]
coo.plot(b)
segments(b[ids[1], 1], b[ids[1], 2], b[ids[2], 1], b[ids[2], 2], lty=2)

# on a Coo
bot %>%
  coo_sample(32) %>% # for speed sake
  coo_calliper()

bot %>%
  coo_sample(32) %>% # for speed sake
  coo_calliper(arr.ind=TRUE)
```

---

**coo_centdist**  
*Returns the distance between every points and the centroid*

Description

For every point of the shape, returns the (centroid-points) distance.

Usage

```r
coo_centdist(coo)
```

Arguments

- `coo` 
  a matrix of (x; y) coordinates.

Value

a matrix of (x; y) coordinates.

See Also

Other centroid functions: `coo_centpos()`, `coo_centsize()`  
Other coo_utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()` , `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`
**Examples**

```r
b <- coo_sample(bot[1], 64)  
d <- coo_centdist(b)  
barplot(d, xlab="Points along the outline", ylab="Distance to the centroid (pixels")
```

### Description

 Returns a shape centered on the origin. The two functions are strictly equivalent.

### Usage

```r
coo_center(coo)  
coo_centre(coo)
```

### Arguments

- `coo` matrix of (x; y) coordinates or any Coo object.

### Value

a matrix of (x; y) coordinates, or a Coo object.

### See Also

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbotttom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

### Examples

```r
coo_plot(bot[1])  
# same as
coo_plot(coo_centre(bot[1]))  
# this
coo_plot(coo_center(bot[1]))
```

```r
# on Coo objects
b <- slice(bot, 1:5)  
stack(slice(b, 1:5))  
stack(coo_center(b))
```
Calculate centroid coordinates

Description

Returns the (x; y) centroid coordinates of a shape.

Usage

coo_centpos(coo)

Arguments

coo matrix of (x; y) coordinates or any Coo object.

Value

(x; y) coordinates of the centroid as a vector or a matrix.

See Also

Other centroid functions: coo_centdist(), coo_centsize()

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(),
coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_nb(), coo_perm(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(),
coo_slidedirection(), coo_slidemag(), coo_slide(), coo_smoothcurve(), coo_smooth(),
coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(),
coo_up(), is_equallyspacedradii()

Examples

b <- bot[1]
cooplot(b)
xy <- coo_centpos(b)
points(xy[1], xy[2], cex=2, col='blue')
# on a Coo
coo_centpos(bot)
coo_centsize

Description
Calculates centroid size

Usage
coo_centsize(coo)

Arguments
coo matrix of (x; y) coordinates or any Coo object.

Details
This function can be used to integrate size - if meaningful - to Coo objects. See also coo_length and rescale.

Value
numeric, the centroid size.

See Also
Other centroid functions: coo_centdist(), coo_centpos()

Examples
coo_centsize(bot[1])
# on a Coo
coo_centsize(bot)
# add it to $fac
mutate(bot, size=coo_centsize(bot))

coo_check

Description
A simple utility, used internally, mostly in the coo functions and methods. Returns a matrix of coordinates, when passed with either a list or a matrix of coordinates.

Usage
coo_check(coo)
Arguments

coo matrix of (x; y) coordinates or any Coo object.

Value

matrix of (x; y) coordinates or a Coo object.

Examples

#coo_check('Not a shape')
#coo_check(iris)
#coo_check(matrix(1:10, ncol=2))
#coo_check(list(x=1:5, y=6:10))

coo_chull Calculate the (recursive) convex hull of a shape

Description

c oo_chull returns the ids of points that define the convex hull of a shape. A simple wrapper around chull, mainly used in graphical functions.

Usage

c oo_chull(coo)

## Default S3 method:
c oo_chull(coo)

## S3 method for class 'Coo'
c oo_chull(coo)

c oo_chull_onion(coo, close = TRUE)

## Default S3 method:
c oo_chull_onion(coo, close = TRUE)

## S3 method for class 'Coo'
c oo_chull_onion(coo, close = TRUE)

Arguments

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

close logical whether to close onion rings (TRUE by default)

Details

c oo_chull_onion recursively find their convex hull, remove them, until less than 3 points are left.
coo_circularity

Value

coo_chull returns a matrix of points defining the convex hull of the shape; a list for Coo. coo_chull_onion returns a list of successive onions rings, and a list of lists for Coo.

See Also

Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_circularity(), coo_convexity(), coo_eccentricity, coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples

# coo_chull
h <- coo_sample(hearts[4], 32)
coo_plot(h)
ch <- coo_chull(h)
lines(ch, col=‵red‵, lty=2)

bot %>% coo_chull

coo_chull_onion
x <- bot %>% efourier(6) %>% PCA
all_whisky_points <- x %>% as_df() %>% filter(type=‵whisky‵) %>% select(PC1, PC2)
plot(x, ~type, eig=FALSE)
peeling_the_whisky_onion <- all_whisky_points %>% as.matrix %>% coo_chull_onion()
# you may need to par(xpd=NA) to ensure all segments
# even those outside the graphical window are drawn
peeling_the_whisky_onion$coo %>% lapply(coo_draw)
# simulated data
xy <- replicate(2, rnorm(50))
coo_plot(xy, poly=FALSE)
xy %>% coo_chull_onion() %>% lapply(polygon, col=‵#00000022‵)

coo_circularity  Calculates the Haralick's circularity of a shape

Description

coo_circularity calculates the 'circularity measure'. Also called 'compactness' and 'shape factor' sometimes. coo_circularity_haralick calculates Haralick's circularity which is less sensible to digitalization noise than coo_circularity. coo_circularity_norm calculates 'circularity', also called compactness and shape factor, but normalized to the unit circle.
Usage

coo_circularity(coo)

## Default S3 method:
coo_circularity(coo)

## S3 method for class 'Coo'
coo_circularity(coo)

coo_circularity_haralick(coo)

## Default S3 method:
coo_circularity_haralick(coo)

## S3 method for class 'Coo'
coo_circularity_haralick(coo)

coo_circularity_norm(coo)

## Default S3 method:
coo_circularity_norm(coo)

## S3 method for class 'Coo'
coo_circularity_norm(coo)

Arguments

coo  a matrix of (x; y) coordinates or any Coo

Value

numeric for single shapes, list for Coo of the corresponding circularity measurement.

Source


See Also

Other coo descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples

# coo_circularity
bot[1] %>% coo_circularity()
bot %>%
  slice(1:5) %>% # for speed sake only
  coo_circularity

# coo_circularityharalick
bot[1] %>% coo_circularityharalick()
bot %>%
  slice(1:5) %>% # for speed sake only
  coo_circularityharalick

# coo_circularitynorm
bot[1] %>% coo_circularitynorm()
bot %>%
  slice(1:5) %>% # for speed sake only
  coo_circularitynorm

---

**coo_close**  
**Closes/uncloses shapes**

**Description**

Returns a closed shape from (un)closed shapes. See also `coo_unclose`. Returns a unclosed shape from (un)closed shapes. See also `coo_close`.

**Usage**

```r
coo_close(coo)
coo_unclose(coo)
```

**Arguments**

- `coo`  
  matrix of (x; y) coordinates or any Coo object.

**Value**

- a matrix of (x; y) coordinates, or a Coo object.
- a matrix of (x; y) coordinates, or a Coo object.

**See Also**

Other coo Utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `cooBaseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`. 
coo_slidedirection(), coo_slidedirection(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimb(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_sampler(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidedirection(), coo_slidedirection(), coo_smooth(), coo_trimb(), coo_trimb(), coo_trimb(), coo_trimb(), coo_trimb(), coo_trimb(), coo_trimb(), coo_trimb(), is_equallyspacedradii()

Examples

```r
x <- (matrix(1:10, ncol=2))
x2 <- coo_close(x)
x3 <- coo_unclose(x2)
x
coo_is_closed(x)
x2
coo_is_closed(x2)
x3
coo_is_closed(x3)
x <- (matrix(1:10, ncol=2))
x2 <- coo_close(x)
x3 <- coo_unclose(x2)
x
coo_is_closed(x)
x2
coo_is_closed(x2)
x3
coo_is_closed(x3)
```

---

**coo_convexity**

*Calculates the convexity of a shape*

**Description**

Calculated using a ratio of the eigen values (inertia axis)

**Usage**

```r
coo_convexity(coo)
```

**Arguments**

- `coo` a matrix of (x; y) coordinates.
coo_down

Value

numeric for a single shape, list for a Coo

Source


See Also

Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_eccentricity, coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples

coo_convexity(bot[1])
bot %>%
  slice(1:3) %>% # for speed sake only
coo_convexity()

coo_down

coo_down Retains coordinates with negative y-coordinates

Description

Useful when shapes are aligned along the x-axis (e.g. because of a bilateral symmetry) and when one wants to retain just the lower side.

Usage

coo_down(coo, slidegap = FALSE)

Arguments

coo matrix of (x; y) coordinates or any Coo object.
slidegap logical whether to apply coo_slidegap after coo_down

Value

a matrix of (x; y) coordinates or a Coo object (Out are returned as Opn)
Note

When shapes are "sliced" along the x-axis, it usually results on open curves and thus to huge/artefactual
gaps between points neighboring this axis. This is usually solved with coo_slidegap. See examples
there.
Also, when apply a coo_left/right/up/down on an Out object, you then obtain an Opn object, which
is done automatically.

See Also

Other opening functions: coo_left(), coo_right(), coo_up()
Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignnax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_centpos(), coo_close(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(),
coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(),
coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(),
coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(),
coo_up(), is_equallyspacedradii()

Examples

b <- coo_alignnax(bot[1])
coo_plot(b)
coo_draw(coo_down(b), border='red')

---

coo_draw     Adds a shape to the current plot

Description

coo_draw is simply a coo_plot with plot.new=FALSE, ie that adds a shape on the active plot.

Usage

coo_draw(coo, ...)

Arguments

coo     a list or a matrix of coordinates.
...     optional parameters for coo_plot

See Also

Other plotting functions: coo_arrows(), coo_listpanel(), coo_lolli(), coo_plot(), coo_ruban(),
ldk_chull(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_devsegments(),
plot_table()
Examples
b1 <- bot[4]
b2 <- bot[5]
coo_plot(b1)
coo_draw(b2, border='red') # all coo_plot arguments will work for coo_draw

coo_draw_rads

Draw radii to the current plot

Description
Given a shape, all centroid-points radii are drawn using segments that can be passed with options.

Usage
coo_draw_rads(coo, ...)

Arguments
coo a shape
... arguments to feed segments

Examples
shp <- shapes[4] %>% coo_sample(24) %>% coo_plot
coo_draw_rads(shp, col=col_summer(24))

coo_dxy

Calculate abscissa and ordinate on a shape

Description
A simple wrapper to calculate dxi - dx1 and dyi - dx1.

Usage
coo_dxy(coo)

Arguments
coo a matrix (or a list) of (x; y) coordinates or any Coo

Value
a data.frame with two components dx and dy for single shapes or a list of such data.frames for Coo
See Also

Other coo utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignnax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_mb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()` , `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
coo_dxy(coo_sample(bot[1], 12))

bot %>%
  slice(1:5) %>% coo_sample(12) %>% # for readability and speed only
coo_dxy()
```

---

### coo_eccentricity

**Calculates the eccentricity of a shape**

**Description**

`coo_eccentricityeigen` uses the ratio of the eigen values (inertia axes of coordinates). `coo_eccentricityboundingbox` uses the width/length ratio (see `coo lw`).

**Usage**

```r
coo_eccentricityeigen(coo)

## Default S3 method:
coo_eccentricityeigen(coo)

## S3 method for class 'Coo'
coo_eccentricityeigen(coo)

coo_eccentricityboundingbox(coo)

## Default S3 method:
coo_eccentricityboundingbox(coo)

## S3 method for class 'Coo'
coo_eccentricityboundingbox(coo)
```

**Arguments**

- `coo` a matrix of (x; y) coordinates or any Coo
coo_elongation

**Value**

numeric for single shapes, list for Coo.

**Source**


**See Also**

coo_eccentricityboundingbox

Other coo_ descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

**Examples**

```r
# coo_eccentricityeigen
bot[1] %>% coo_eccentricityeigen()
bot %>%
  slice(1:3) %>% # for speed sake only
  coo_eccentricityeigen()

# coo_eccentricityboundingbox
bot[1] %>% coo_eccentricityboundingbox()
bot %>%
  slice(1:3) %>% # for speed sake only
  coo_eccentricityboundingbox()
```

---

**coo_elongation**

*Calculates the elongation of a shape*

**Description**

Calculates the elongation of a shape

**Usage**

coo_elongation(coo)

**Arguments**

coo a matrix of (x; y) coordinates.

**Value**

numeric, the eccentricity of the bounding box
**Source**


**See Also**

Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

**Examples**

```r
coo_elongation(bot[1])
# on Coo
# for speed sake
bot %>% slice(1:3) %>% coo_elongation
```

---

**coo_extract**

*Extract coordinates from a shape*

**Description**

Extract ids coordinates from a single shape or a Coo object.

**Usage**

```r
coo_extract(coo, ids)
```

**Arguments**

- **coo**: either a matrix of (x; y) coordinates or a Coo object.
- **ids**: integer, the ids of points to sample.

**Details**

It probably only make sense for Coo objects with the same number of coordinates and them being homologous, typically on Ldk.

**Value**

A matrix of (x; y) coordinates, or a Coo object.
See Also

Other sampling functions: coo_interpolate(), coo_sample_prop(), coo_sample(),
Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_flipx(), coo_force2close(),
coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likelyclockwise(),
coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(),
coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(),
coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(),
coo_up(), is_equallyspacedradii()

Examples

coo_extract(bot[1], c(3, 9, 12))  # or :
bot[1] %>% coo_extract(c(3, 9, 12))

---

coo_flipx  Flips shapes

Description

coo_flipx flips shapes about the x-axis; coo_flipy about the y-axis.

Usage

coo_flipx(coo)

coo_flipy(coo)

Arguments

coo  matrix of (x; y) coordinates or any Coo object.

Value

a matrix of (x; y) coordinates

See Also

Other transforming functions: coo_shearx()
**coo_force2close**

Forces shapes to close

Description

An exotic function that distribute the distance between the first and the last points of unclosed shapes, so that they become closed. May be useful (?) e.g. for t/rfourier methods where reconstructed shapes may not be closed.

Usage

```r
coo_force2close(coo)
```

Arguments

- `coo`: matrix of (x; y) coordinates or any Coo object.

Value

A matrix of (x; y) coordinates, or a Coo object.

See Also

Other coo_ utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`,
`coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`,
`coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipy()`,
`coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`,
`coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rot()`, `coo_rotatecenter()`, `coo_rotate()`,
`coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_scale()`, `coo_rotate()`,
`coo_slidedirection()`, `coo_slidethrough()`, `coo_smoothgap()`, `coo_smooth()`, `coo_smoothcurve()`,
`coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbtop()`, `coo_trim()`, `coo_trim()`,
`coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
cat <- shapes[4]
cat <- coo_center(cat)
coo_plot(cat)
coo_draw(coo_flipx(cat), border="red")
coo_draw(coo_flipy(cat), border="blue")
```

```r
# to flip an entire Coo:
shapes2 <- shapes
shapes$coo <- lapply(shapes2$coo, coo_flip)
```
**Examples**

b <- coo_sample(bot[1], 64)
b <- b[1:40,]
coo_plot(b)
coo_draw(coo_force2close(b), border='red')

---

**coo_interpolate**

Interpolates coordinates

**Description**

Interpolates n coordinates 'among existing points' 'between' existing points, along the perimeter of the coordinates provided and keeping the first point

**Usage**

coo_interpolate(coo, n)

**Arguments**

- coo: matrix of (x; y) coordinates or any Coo object.
- n: codeinteger, the number fo points to interpolate.

**Value**

a matrix of (x; y) coordinates, or a Coo object.

**See Also**

Other sampling functions: coo_extract(), coo_sample_prop(), coo_samplerr(), coo_sample()
Other coo_utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(),
coo_force2close(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(),
coo_slidedirection(), coo_slidewidth(), coo_slide(), coo_smoothcurve(), coo_smooth(),
coo_template(), coo_trans(), coo_trimbottom(), coo_trimpop(), coo_trim(), coo_untiltx(),
coo_up(), is_equallyspacedradii()

**Examples**

b5 <- bot %>% slice(1:5) # for speed sake
stack(b5)
stack(coo_scale(b5))
stack(b5)
stack(coo_interpolate(coo_sample(b5, 12), 120))
coo_plot(bot[1])
coo_plot(coo_interpolate(coo_sample(bot[1], 12), 120))
**coo_intersect_angle**

Nearest intersection between a shape and a segment specified with an angle

---

**Description**

Take a shape, and segment starting on the centroid and having a particular angle, which point is the nearest where the segment intersects with the shape?

**Usage**

```r
coo_intersect_angle(coo, angle = 0)
```

```r
coo_intersect_direction(coo, direction = c("down", "left", "up", "right")[4])
```

```r
## Default S3 method:
coo_intersect_direction(coo, direction = c("down", "left", "up", "right")[4])
```

```r
## S3 method for class 'Coo'
coo_intersect_direction(coo, direction = c("down", "left", "up", "right")[4])
```

**Arguments**

- `coo` (matrix of (x; y) coordinates or any Coo object).
- `angle` (numeric an angle in radians (0 by default)).
- `direction` (character one of "down", "left", "up", "right" ("right" by default)).

**Value**

numeric the id of the nearest point or a list for Coo. See examples.

**Note**

shapes are always centered before this operation. If you need a simple direction such as (down, left, up, right)ward, then use `coo_intersect_direction` which does not need to find an intersection but relies on coordinates and is about 1000.

**See Also**

Other coo_intersect: *coo_intersect_segment()*

**Examples**

```r
coo <- bot[1] %>% coo_center %>% coo_scale
coo_plot(coo)
coo %>% coo_intersect_angle(pi/7) %>%
  coo[., , drop=FALSE] %>% points(col="red")
```
coo_intersect_segment

# many angles
coo_plot(coo)
sapply(seq(0, pi, pi/12),
    function(x) coo %>% coo_intersect_angle(x)) -> ids
coo[ids, ] %>% points(col="blue")

c oo %>%
coo_intersect_direction("down") %>%
coo[., , drop=FALSE] %>% points(col="orange")

coo_intersect_segment  Nearest intersection between a shape and a segment

Description

Take a shape, and an intersecting segment, which point is the nearest of where the segment intersects
with the shape? Most of the time, centering before makes more sense.

Usage

c oo_intersect_segment(coo, seg, center = TRUE)

Arguments

c oo matrix of (x; y) coordinates or any Coo object.
seg a 2x2 matrix defining the starting and ending points; or a list or a numeric of
length 4.
c enter logical whether to center the shape (TRUE by default)

Value

numeric the id of the nearest point, a list for Coo. See examples.

See Also

Other coo_intersect: coo_intersect_angle()

Examples

c oo <- bot[1] %>% coo_center %>% coo_scale
seg <- c(0, 0, 2, 2) # passed as a numeric of length(4)
coo_plot(coo)
segments(seg[1], seg[2], seg[3], seg[4])
coo %>% coo_intersect_segment(seg) %>% print %>%
# prints on the console and draw it
c oo[., , drop=FALSE] %>% points(col="red")

# on Coo
coo_is_closed

```r
bot %>%
  slice(1:3) %>% # for the sake of speed
coo_center %>%
  coo_intersect_segment(matrix(c(0, 0, 1000, 1000), ncol=2, byrow=TRUE))
```

---

coo_is_closed  Test if shapes are closed

**Description**

Returns TRUE/FALSE whether the last coordinate of the shapes is the same as the first one.

**Usage**

```r
coo_is_closed(coo)

is_open(coo)
```

**Arguments**

- `coo`  matrix of (x; y) coordinates or any Coo object.

**Value**

a single or a vector of logical.

**See Also**

Other coo utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

**Examples**

```r
coo_is_closed(matrix(1:10, ncol=2))
coo_is_closed(coo_close(matrix(1:10, ncol=2)))
coo_is_closed(bot)
coo_is_closed(coo_close(bot))
```
**Description**  
A simple wrapper around `jitter`.

**Usage**  

```r  
coo_jitter(coo, ...)  
```

**Arguments**  

- `coo` matrix of (x; y) coordinates or any `Coo` object.
- `...` additional parameter for `jitter`

**Value**  

a matrix of (x; y) coordinates or a Coo object

**See Also**  

`get_pairs`

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

**Examples**

```r  
b <- bot[1]  
coo_plot(b, zoom=0.2)  
coo_draw(coo_jitter(b, amount=3), border="red")  
```

# for a Coo example, see `\link{get_pairs}`
coo_ldek

**Defines landmarks interactively**

**Description**

Allows to interactively define a `nb_ldek` number of landmarks on a shape. Used in other facilities to acquire/manipulate data.

**Usage**

```
coo_ldek(coo, nb_ldek, close = FALSE, points = TRUE)
```

**Arguments**

- `coo`: a matrix or a list of (x; y) coordinates.
- `nb_ldek`: integer, the number of landmarks to define
- `close`: logical, whether to close (typically for outlines)
- `points`: logical, whether to display points

**Value**

numeric that corresponds to the closest ids, on the shape, from clicked points.

**Examples**

```r
## Not run:
b <- bot[1]  
coo_ldek(b, 3) # run this, and click 3 times  
coo_ldek(bot, 2) # this also works on Out
## End(Not run)
```

---

coo_left

**Retains coordinates with negative x-coordinates**

**Description**

Useful when shapes are aligned along the y-axis (e.g. because of a bilateral symmetry) and when one wants to retain just the lower side.

**Usage**

```
coo_left(coo, slidegap = FALSE)
```
**Arguments**

- `coo` matrix of (x, y) coordinates or any Coo object.
- `slidegap` logical whether to apply `coo_slidegap` after `coo_left`

**Value**

A matrix of (x, y) coordinates or a Coo object (Out are returned as Opn)

**Note**

When shapes are "sliced" along the y-axis, it usually results in open curves and thus to huge/artefactual gaps between points neighboring this axis. This is usually solved with `coo_slidegap`. See examples there.

Also, when apply a `coo_left/right/up/down` on an Out object, you then obtain an Opn object, which is done automatically.

**See Also**

Other opening functions: `coo_down()`, `coo_right()`, `coo_up()`

Other coo utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

**Examples**

```r
b <- coo_center(bot[1])
coo_plot(b)
coo_draw(coo_left(b), border='red')
```

---

**coo_length**  
*Calculates the length of a shape*

**Description**

Nothing more than `coo_lw(coo)[1]`.

**Usage**

`coo_length(coo)`
Arguments

coo

a matrix of (x; y) coordinates or a Coo object

Details

This function can be used to integrate size - if meaningful - to Coo objects. See also coo_centsize and rescale.

Value

the length (in pixels) of the shape

See Also

coo_lw, coo_width

Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo solidity(), coo_tac(), coo_width()

Examples

coo_length(bot[1])
coo_length(bot)
mutate(bot, size=coo_length(bot))

coo_likely_clockwise

Tests if shapes are (likely) developing clockwise or anticlockwise

Description

Tests if shapes are (likely) developping clockwise or anticlockwise

Usage

coo_likely_clockwise(coo)

## Default S3 method:
coo_likely_clockwise(coo)

## S3 method for class 'Coo'
coo_likely_clockwise(coo)

coo_likely_anticlockwise(coo)

Arguments

coo

matrix of (x; y) coordinates or any Coo object.
Value

a single or a vector of logical.

See Also

Other `coo_` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearX()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidedist()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
shapes[4] %>% coo_sample(64) %>% coo_plot() #clockwise cat
shapes[4] %>% coo_likely_clockwise()

# on Coo
shapes %>% coo_likely_clockwise %>% `\`'(4)
```

---

**coo_listpanel**

Plots sets of shapes.

---

**Description**

`coo_listpanel` plots a list of shapes if passed with a list of coordinates. Mainly used by `panel.Coo` functions. If used outside the latter, shapes must be "templated", see `coo_template`. If you want to reorder shapes according to a factor, use `arrange`.

**Usage**

```r
coo_listpanel(
  coo.list,
  dim,
  byrow = TRUE,
  fromtop = TRUE,
  cols,
  borders,
  poly = TRUE,
  points = FALSE,
  points.pch = 3,
  points.cex = 0.2,
  points.col = "#333333",
  ...
)
```
Arguments

coonlist A list of coordinates
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 draw successive shape by row or by col.
fromtop logical. Whether to display shapes from the top of the plotting region.
cols A vector of colors to fill shapes.
borders A vector of colors to draw shape borders.
poly logical whether to use polygon or lines to draw shapes. mainly for use for outlines and open outlines.
points logical if poly is set to FALSE whether to add points
points.pch if points is TRUE, a pch for these points
points.cex if points is TRUE, a cex for these points
points.col if points is TRUE, a col for these points
... additional arguments to feed generic plot

Value

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

See Also

Other plotting functions: coo_arrows(), coo_draw(), coo_lolli(), coo_plot(), coo_ruban(), ldk_chull(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_devsegments(), plot_table()

Examples

coo_listpanel(bot$coo) # equivalent to panel(bot)

coo_lolli(coo1, coo2, pch = NA, cex = 0.5, ...)
Arguments

- **coo1**: A list or a matrix of coordinates.
- **coo2**: A list or a matrix of coordinates.
- **pch**: A pch for the points (default to NA)
- **cex**: A cex for the points
- **...**: Optional parameters to fed points and segments.

See Also

Other plotting functions: `coo_arrows()`, `coo_draw()`, `coo_listpanel()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_labels()`, `ldk_links()`, `plot_devsegments()`, `plot_table()`

Examples

```r
coo_lolli(coo_sample(olea[3], 50), coo_sample(olea[6], 50))
title("A nice title !")
```

---

### coo_lw

*Calculates length and width of a shape*

Description

Returns the length and width of a shape based on their inertia axis i.e. alignment to the x-axis. The length is defined as the range along the x-axis; the width as the range on the y-axis.

Usage

```r
coo_lw(coo)
```

Arguments

- **coo**: A matrix of (x; y) coordinates or Coo object

Value

A vector of two numeric: the length and the width.

See Also

`coo_length`, `coo_width`

Other coo descriptors: `coo_angle_edges()`, `coo_angle_tangent()`, `coo_area()`, `coo_boundingbox()`, `coo_chull()`, `coo_circularity()`, `coo_convexity()`, `coo_eccentricity`, `coo_elongation()`, `coo_length()`, `coo_rectangularity()`, `coo_rectilinearity()`, `coo_scalars()`, `coo solidity()`, `coo_tac()`, `coo_width()`
Examples

coo_nb(bot[1])

counts_coordinates

Description
Returns the number of coordinates, for a single shape or a Coo object

Usage

coo_nb(coo)

Arguments

coo  matrix of (x; y) coordinates or any Coo object.

Value

either a single numeric or a vector of numeric

See Also

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(),
coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shear(), coo_slice(),
coo_slidedirection(), coo_slidemomentary(), coo_smoothcurve(), coo_smooth(),
coo_template(), coo_trans(), coo_trimbottom(), coo_trimmom(), coo_trim(), coo_untiltx(),
coo_up(), is_equallyspacedradii()
coo_oscillo

Momocs' 'oscilloscope' for Fourier-based approaches

Description
Shape analysis deals with curve fitting, whether \( x(t) \) and \( y(t) \) positions along the curvilinear abscissa and/or radius/tangent angle variation. These functions are mainly intended for (self-)teaching of Fourier-based methods.

Usage
```r
coo_oscillo(
  coo,
  method = c("efourier", "rfourier", "tfourier", "all")[4],
  shape = TRUE,
  nb.pts = 12
)
```

Arguments
- **coo**: A list or a matrix of coordinates.
- **method**: character among c("efourier", "rfourier", "tfourier", "all"). 'all' by default
- **shape**: logical whether to plot the original shape
- **nb.pts**: integer. The number or reference points, sampled equidistantly along the curvilinear abscissa and added on the oscillo curves.

Value
the plotted values

See Also
exemplifying functions

Examples
```r
coo_oscillo(shapes[4])
coo_oscillo(shapes[4], "efourier")
coo_oscillo(shapes[4], "rfourier")
coo_oscillo(shapes[4], "tfourier")
#tfourier is prone to high-frequency noise but smoothing can help
coo_oscillo(coo_smooth(shapes[4], 10), "tfourier")
```
coo_perim calculates perimeter and variations

Description

`coo_perim` calculates the perimeter; `coo_perimpts` calculates the euclidean distance between every points of a shape; `coo_perimcum` does the same and calculates and cumulative sum.

Usage

```r
coo_perimpts(coo)
```

## Default S3 method:
```r
coo_perimpts(coo)
```

## S3 method for class 'Coo'
```r
coo_perimpts(coo)
```

coo_perimcum(coo)

## Default S3 method:
```r
coo_perimcum(coo)
```

## S3 method for class 'Coo'
```r
coo_perimcum(coo)
```

coo_perim(coo)

## Default S3 method:
```r
coo_perim(coo)
```

## S3 method for class 'Coo'
```r
coo_perim(coo)
```

Arguments

- `coo` matrix of (x; y) coordinates or any Coo

Value

numeric the distance between every point or a list of those.

See Also

Other coo utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `cooBoundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_flipy()`, `coo_flipz()`, `coo_getarea()`, `coo_getbbox()`.
coo_plot

coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likeclockwise(),
coo_nb(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(),
coo_samplerr(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(),
coo_slidedgapa(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(),
coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

# for speed sake
b1 <- coo_sample(bot[1], 12)
b5 <- bot %>% slice(1:5) %>% coo_sample(12)

# coo_perim
coo_perim(b1)
coo_perim(b5)

# coo_perimpts
coo_perimpts(b1)
b5 %>% coo_perimpts()

# coo_perimcum
b1 %>% coo_perimcum()
b5 %>% coo_perimcum()

---

coo_plot  

Plots a single shape

Description

A simple wrapper around plot for plotting shapes. Widely used in Momocs in other graphical
functions, in methods, etc.

Usage

coo_plot(
  coo,
  xlim,
  ylim,
  border = "#333333",
  col = NA,
  lwd = 1,
  lty = 1,
  points = FALSE,
  first.point = TRUE,
  cex.first.point = 0.5,
  centroid = TRUE,
  xy.axis = TRUE,
  pch = 1,
ldk_plot(coo, ...)

Arguments

- **coo**: A list or a matrix of coordinates.
- **xlim**: If `coo_plot` is called and `coo` is missing, then a vector of length 2 specifying the ylim of the plotting area.
- **ylim**: If `coo_plot` is called and `coo` is missing, then a vector of length 2 specifying the ylim of the plotting area.
- **border**: A color for the shape border.
- **col**: A color to fill the shape polygon.
- **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 plot or not the first point.
- **cex.first.point**: numeric size of this first point
- **centroid**: logical. Whether to display centroid.
- **xy.axis**: logical. Whether to draw the xy axis.
- **pch**: The pch for points.
- **cex**: The cex for points.
- **main**: character. A title for the plot.
- **poly**: logical whether to use `polygon` and `lines` to draw the shape, or just `points`. In other words, whether the shape should be considered as a configuration of landmarks or not (eg a closed outline).
- **plot.new**: logical whether to plot or not a new frame.
- **plot**: logical whether to plot something or just to create an empty plot.
- **zoom**: a numeric to take your distances.
- **...**: further arguments for use in `coo_plot` methods. See examples.

Value

No returned value.
coo_range

Calculate coordinates range

Description

coo_range simply returns the range, coo_range_enlarge enlarges it by a k proportion. coo_diffrange return the amplitude (ie diff after coo_range)

Examples

b <- bot[1]
coo_plot(b)
coo_plot(bot[2], plot.new=FALSE) # equivalent to coo_draw(bot[2])
coo_plot(b, zoom=2)
coo_plot(b, border='blue')
coo_plot(b, first.point=FALSE, centroid=FALSE)
coo_plot(b, points=TRUE, pch=20)
coo_plot(b, xy.axis=FALSE, lwd=2, col='#F2F2F2')

Usage

coo_range(coo)

## Default S3 method:
coo_range(coo)

## S3 method for class 'Coo'
coo_range(coo)

coo_range_enlarge(coo, k)

## Default S3 method:
c oo_range_enlarge(coo, k = 0)

## S3 method for class 'Coo'
coo_range_enlarge(coo, k = 0)

## S3 method for class 'list'
coo_range_enlarge(coo, k = 0)

coo_diffrange(coo)

See Also

Other plotting functions: coo_arrows(), coo_draw(), coo_listpanel(), coo_lolli(), coo_ruban(), ldk_chull(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_devsegments(), plot_table()
coo_rectangularity

Calculates the rectangularity of a shape

Usage

coo_rectangularity(coo)
Arguments

coo  
a matrix of (x; y) coordinates or any Coo

Value

numeric for a single shape, list for Coo

Source


See Also

Other coo_ descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(), coo_lw(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples

c oo_rectangularity(bot[1])

bot %>%
  slice(1:3) %>%  # for speed sake only
  coo_rectangularity

coo_rectilinearity  Calculates the rectilinearity of a shape

Description

As proposed by Zunic and Rosin (see below). May need some testing/review.

Usage

c oo_rectilinearity(coo)

Arguments

coo  
a matrix of (x; y) coordinates or any Coo

Value

numeric for a single shape, list for Coo
Note
due to the laborious nature of the algorithm (in nb.pts^2), and of its implementation, it may be very long to compute.

Source

See Also
Other coo_ descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_scalars(), coo_solidity(), coo_tac(), coo_width()

Examples
bot[1] %>%
  coo_sample(32) %>% # for speed sake only
  coo_rectilinearity

bot %>%
  slice(1:3) %>% coo_sample(32) %>% # for speed sake only
  coo_rectilinearity

---

coo_rev

Reverses coordinates

Description
Returns the reverse suite of coordinates, i.e. change shape’s orientation

Usage
coo_rev(coo)

Arguments
coo matrix of (x; y) coordinates or any Coo object.

Value
a matrix of (x; y) coordinates or a Coo object
**coo_right**

*Retains coordinates with positive x-coordinates*

**Description**

Useful when shapes are aligned along the y-axis (e.g. because of a bilateral symmetry) and when one wants to retain just the upper side.

**Usage**

```r
coo_right(coo, slidegap = FALSE)
```

**Arguments**

- `coo` matrix of (x; y) coordinates or any Coo object.
- `slidegap` logical whether to apply coo_slidegap after coo_right

**Value**

A matrix of (x; y) coordinates or a Coo object (Out are returned as Opn)

**Note**

When shapes are "sliced" along the y-axis, it usually results on open curves and thus to huge/artefactual gaps between points neighboring this axis. This is usually solved with coo_slidegap. See examples there.

Also, when apply a coo_left/right/up/down on an Out object, you then obtain an Opn object, which is done automatically.
See Also

Other opening functions: `coo_down()`, `coo_left()`, `coo_up()`

Other coo_ utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()``, `coo_bookstein()`, `coo_boundingbox()``, `coo_calliper()`, `coo_centdist()`, `coo_center()``, `coo_centpos()``, `coo_close()``, `coo_down()``, `coo_dxy()``, `coo_extract()``, `coo_flipx()``, `coo_force2close()``, `coo_interpolate()``, `coo_is_closed()``, `coo_jitter()``, `coo_left()``, `coo_likely_clockwise()``, `coo_nb()``, `coo_perim()``, `coo_range()``, `coo_rev()``, `coo_rotatecenter()``, `coo_rotate()``, `coo_sample_prop()``, `coo_samplerr()``, `coo_sample()``, `coo_scale()``, `coo_shearx()``, `coo_slice()``, `coo_slidedirection()``, `coo_slidegap()``, `coo_slide()``, `coo_smoothcurve()``, `coo_smooth()``, `coo_template()``, `coo_trans()``, `coo_trimbottom()``, `coo_trimtop()``, `coo_trim()``, `coo_untiltx()``, `coo_up()``, `is_equallyspacedradii()`

Examples

```r
b <- coo_center(bot[1])
coop_plot(b)
coo_draw(coo_right(b), border='red')
```

---

## Description

Rotates the coordinates by a `theta` angle (in radians) in the trigonometric direction (anti-clockwise). If not provided, assumed to be the centroid size. It involves three steps: centering from current position, dividing coordinates by `scale`, translating to the original position.

## Usage

```r
coo_rotate(coo, theta = 0)
```

## Arguments

- **coo**: either a matrix of (x; y) coordinates, or any Coo object.
- **theta**: numeric; the angle (in radians) to rotate shapes.

## Value

A matrix of (x; y) coordinates, or a Coo object.

## See Also

Other coo_ utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()``, `coo_bookstein()`, `coo_boundingbox()``, `coo_calliper()`, `coo_centdist()`, `coo_center()``, `coo_centpos()``, `coo_close()``, `coo_down()``, `coo_dxy()``, `coo_extract()``, `coo_flipx()``, `coo_force2close()``, `coo_interpolate()``, `coo_is_closed()``, `coo_jitter()``, `coo_left()``, `coo_likely_clockwise()``, `coo_nb()``, `coo_perim()``, `coo_range()``, `coo_rev()``, `coo_right()``, `coo_rotatecenter()``, `coo_rotate()``, `coo_sample_prop()``, `coo_samplerr()``, `coo_sample()``, `coo_scale()``, `coo_shearx()``, `coo_slice()``, `coo_slidedirection()``. 
coo_rotatecenter

Other rotation functions: coo_rotatecenter()

Examples

```r
coo_plot(bot[1])
coo_plot(coo_rotate(bot[1], pi/2))

# on Coo
b <- bot %>% slice(1:5) # for speed sake
stack(b)
stack(coo_rotate(b, pi/2))
```

---

coo_rotatecenter  Rotates shapes with a custom center

Description

rotates a shape of 'theta' angles (in radians) and with a (x; y) 'center'.

Usage

```r
coo_rotatecenter(coo, theta, center = c(0, 0))
```

Arguments

- `coo` matrix of (x; y) coordinates or any Coo object.
- `theta` numeric the angle (in radians) to rotate shapes.
- `center` numeric the (x; y) position of the center

Value

a matrix of (x; y) coordinates, or a Coo object.

See Also

Other rotation functions: coo_rotate()

Other coo Utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Other rotation functions: coo_rotate()
Examples

b <- bot[1]
coo_plot(b)
coo_draw(coo_rotatecenter(b, -pi/2, c(200, 200)), border='red')

---

coo_ruban

Plots differences as (colored) segments aka a ruban

Description

Useful to display differences between shapes

Usage

coo_ruban(coo, dev, palette = col_heat, normalize = TRUE, ...)

Arguments

coo
  a shape, typically a mean shape
dev
  numeric a vector of distances or anything relevant
palette
  the color palette to use or any palette
normalize
  logical whether to normalize (TRUE by default) distances
... other parameters to fed segments, eg lwd (see examples)

Value

nothing

See Also

Other plotting functions: coo_arrows(), coo_draw(), coo_listpanel(), coo_lolli(), coo_plot(), ldk_chull(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_devsegments(), plot_table()

Examples

ms <- MSHAPES(efourier(bot, 10), "type")
b <- ms$shp$beer
w <- ms$shp$whisky
# we obtain the mean shape, then euclidean distances between points
m <- MSHAPES(list(b, w))
d <- edm(b, w)
# First plot
coo_plot(m, plot=FALSE)
**coo_sample**

Sample coordinates among existing points.

**Usage**

```r
coo_sample(coo, n)
```

**Arguments**

- `coo` either a matrix of \((x; y)\) coordinates or an `Out` or an `Opn` object.
- `n` integer, the number of points to sample.

**Details**

For the `Out` an `Opn` methods (pointless for Ldk), in an `$ldk` component is defined, it is changed accordingly by multiplying the ids by \(n\) over the number of coordinates.

**Value**

- a matrix of \((x; y)\) coordinates, or an `Out` or an `Opn` object.

**See Also**

Other sampling functions: `coo_extract()`, `coo_interpolate()`, `coo_sample_prop()`, `coo_samplerr()`

Other coo utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `cooBoundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidedegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`
Examples

```r
b <- bot[1]
stack(b)
stack(coo_sample(b, 24))
coo_plot(b)
coo_plot(coo_sample(b, 24))
```

---

**coo_samplerr**

*Samples coordinates (regular radius)*

**Description**

Samples n coordinates with a regular angle.

**Usage**

```r
coo_samplerr(coo, n)
```

**Arguments**

- `coo` matrix of (x; y) coordinates or any Coo object.
- `n` integer, the number of points to sample.

**Details**

By design, this function samples among existing points, so using coo_interpolate prior to it may be useful to have more homogeneous angles. See examples.

**Value**

a matrix of (x; y) coordinates or a Coo object.

**See Also**

Other sampling functions: coo_extract(), coo_interpolate(), coo_sample_prop(), coo_sample()

Other coo_utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()
Examples

```r
stack(bot)
bot <- coo_center(bot)
stack(coo_samplerr(bot, 12))
coo_plot(bot[1],)
coo_plot(rr <- coo_samplerr(bot[1], 12))
cpos <- coo_centpos(bot[1])
segments(cpos[1], cpos[2], rr[, 1], rr[, 2])

# Sometimes, interpolating may be useful:
shp <- hearts[1]
# given a shp, draw segments from each points on it, to its centroid
draw_rads <- function(shp, ...){
  segments(shp[, 1], shp[, 2], coo_centpos(shp)[1], coo_centpos(shp)[2], ...)
}

# calculate the sd of argument difference in successive points,
# in other words a proxy for the homogeneity of angles
sd_theta_diff <- function(shp)
  shp %>% complex(real=., 1), imaginary=., 2)) %>%
  Arg %>% [(-1)] %>% diff %>% sd

# no interpolation: all points are sampled from existing points but the angles are not equal
shp %>% coo_plot(points=TRUE, main="no interpolation")
shp %>% coo_samplerr(64) %>% draw_rads(col="red") %>% sd_theta_diff

# with interpolation: much more homogeneous angles
shp %>% coo_interpolate(360) %>% coo_samplerr(64) %>% draw_rads(col="blue") %>% sd_theta_diff
```

---

**coo_sample_prop**

Sample a proportion of coordinates (among points)

**Description**

A simple wrapper around `coo_sample`

**Usage**

```r
coo_sample_prop(coo, prop = 1)
```

**Arguments**

- **coo**: either a matrix of (x; y) coordinates or an `Out` or an `Opn` object.
- **prop**: numeric, the proportion of points to sample
Details

As for `coo_sample` if an $\ldk$ component is defined, it is changed accordingly by multiplying the
ids by n over the number of coordinates.

Value

a matrix of (x; y) coordinates, or an Out or an Opn object.

See Also

Other sampling functions: `coo_extract()`, `coo_interpolate()`, `coo_samplerr()`, `coo_sample()`

Other coo_utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`,
`coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`,
`coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`,
`coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`,
`coo_likely_clockwise()`,
`coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`,
`coo_rotate()`,
`coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`,
`coo slidemap()`, `coo_smooth()`, `coo_smoothcurve()`, `coo_template()`, `coo_trans()`,
`coo_trimbotttom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()

Examples

# single shape
bot[1] %>% coo_nb()
bot[1] %>% coo_sample_prop(0.5) %>% coo_nb()

coo_scalars `Calculates all scalar descriptors of shape`

Description

See examples for the full list.

Usage

coo_scalars(coo, rectilinearity = FALSE)

Arguments

coo         a matrix of (x; y) coordinates or any Coo
rectilinearity logical whether to include rectilinearity using coo_rectilinearity

Details

coo_rectilinearity being not particularly optimized, it takes around 30 times more time to include it
than to calculate all others and is thus not included by default. by default.
coo_scale

Value
data_frame

See Also
Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo solidity(), coo_tac(), coo_width()

Examples
df <- bot %>% coo_scalars() # pass bot %>% coo_scalars(TRUE) if you want rectilinearity
colnames(df) %>% cat(sep="\n") # all scalars used

# a PCA on all these descriptors
TraCoe(coo_scalars(bot), fac=bot$fac) %>% PCA %>% plot_PCA(~type)

---

coo_scale

Scales coordinates

Description
coo_scale scales the coordinates by a 'scale' factor. If not provided, assumed to be the centroid size. It involves three steps: centering from current position, dividing coordinates by 'scale', pushing back to the original position. coo_scalex applies a scaling (or shrinking) parallel to the x-axis, coo_scaley does the same for the y axis.

Usage
coo_scale(coo, scale)

## Default S3 method:
coo_scale(coo, scale = coo_centsize(coo))

## S3 method for class 'Coo'
coo_scale(coo, scale)

c oo_sca ley(coo, scale = 1)

## Default S3 method:
coo_scalex(coo, scale = 1)

## S3 method for class 'Coo'
coo_scalex(coo, scale = 1)
coo_scaley(coo, scale = 1)

## Default S3 method:
coo_scaley(coo, scale = 1)

## S3 method for class 'Coo'
coo_scaley(coo, scale = 1)

Arguments

- coo: matrix of (x; y) coordinates or any Coo object.
- scale: the scaling factor, by default, the centroid size for coo_scale; 1 for scalex and scaley.

Value

- a single shape or a Coo object

See Also

- Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo Likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

# on a single shape
b <- bot[1] %>% coo_center %>% coo_scale
coo_plot(b, lwd=2)
coo_draw(coo_scalex(b, 1.5), bor="blue")
coo_draw(coo_scaley(b, 0.5), bor="red")

# this also works on Coo objects:
b <- slice(bot, 5) # for speed sake
stack(b)
b %>% coo_center %>% coo_scale %>% stack
b %>% coo_center %>% coo_scaley(0.5) %>% stack
#equivalent to:
#b %>% coo_center %>% coo_scalex(2) %>% stack
Description

`coo_shearx` applies a shear mapping on a matrix of (x; y) coordinates (or a list), parallel to the x-axis (i.e. \(x' = x + ky; y' = y + kx\)). `coo_sheary` does it parallel to the y-axis.

Usage

```r
coo_shearx(coo, k)
coo_sheary(coo, k)
```

Arguments

- **coo**: matrix of (x; y) coordinates or any `Coo` object.
- **k**: numeric shear factor

Value

a matrix of (x; y) coordinates.

See Also

Other transforming functions: `coo_flipx()`

Other coo_ utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()`, `coo_scale()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbottom()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Examples

```r
coo <- coo_template(shapes[11])
coo_plot(coo)
coo_draw(coo_shearx(coo, 0.5), border="blue")
coo_draw(coo_sheary(coo, 0.5), border="red")
```
coo_slice  
Slices shapes between successive coordinates

Description
Takes a shape with n coordinates. When you pass this function with at least two ids (<= n), the shape will be open on the corresponding coordinates and slices returned as a list.

Usage
    coo_slice(coo, ids, ldk)

Arguments
- coo  matrix of (x; y) coordinates or any Coo object.
- ids  numeric of length >= 2, where to slice the shape(s)
- ldk  numeric the id of the ldk to use as ids, only on Out and Opn. If provided, ids will be ignored.

Value
  a list of shapes or a list of Opn

See Also
Have a look to coo_slidegap if you have problems with gaps after slicing around landmarks and/or starting points.

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

    h <- slice(hearts, 1:5)  # speed purpose only
    # single shape, a list of matrices is returned
    sh <- coo_slice(h[[1]], c(12, 24, 36, 48))
    coo_plot(sh[[1]])
    panel(Opn(sh))
    # on a Coo, a list of Opn is returned
    # makes no sense if shapes are not normalized first
    sh2 <- coo_slice(h, c(12, 24, 36, 48))
    panel(sh2[[1]])
# Use coo_slice with 'ldk' instead:
# hearts as an example
x <- h %>% fgProcrustes(tol=1)
# 4 landmarks
stack(x)
x$ldk[1:5]

# here we slice
y <- coo_slice(x, ldk=1:4)

# plotting
stack(y[[1]])
stack(y[[2]])

# new ldks from tipping points, new ldks from angle
olea %>% slice(1:5) %>% # for the sake of speed
def_ldk_tips %>%
def_ldk_angle(0.75*pi) %>% def_ldk_angle(0.25*pi) %>%
coo_slice(ldk =1:4) -> oleas
oleas[[1]] %>% stack
oleas[[2]] %>% stack # etc.

# domestic operations
y[[3]] %>% coo_area()
# shape analysis of a slice
y[[1]] %>% coo_bookstein() %>% npoly %>% PCA %>% plot(~aut)

---

### coo_slide

**Slides coordinates**

**Description**

Slides the coordinates so that the id-th point become the first one.

**Usage**

```r
coo_slide(coo, id, ldk)
```

**Arguments**

- **coo**: matrix of (x; y) coordinates or any Coo object.
- **id**: numeric the id of the point that will become the new first point. See details below for the method on Coo objects.
- **ldk**: numeric the id of the ldk to use as id, only on Out
Details

For Coo objects, and in particular for Out and Opn three different ways of coo_sliding are available:

• **no ldk passed and a single id is passed**: all id-th points within the shapes will become the first points. ldk will be slided accordingly.
• **no ldk passed and a vector of ids matching the length of the Coo**: for every shape, the id-th point will be used as the id-th point. ldk will be slided accordingly.
• **a single ldk is passed**: the ldk-th ldk will be used to slide every shape. If an id is (also) passed, it is ignored with a message.

See examples.

Value

a matrix of (x; y) coordinates, or a Coo object.

See Also

coo_slice and friends.

Other sliding functions: coo_slidedirection(), coo_slidegap()

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), cooLikelyClockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_sampler(), coo_sample(), coo_scale(), coo_rotate(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

h <- hearts %>% slice(1:5) # for speed sake
stack(h)
# set the first landmark as the starting point
stack(coo_slide(h, ldk=1))
# set the 50th point as the starting point (everywhere)
stack(coo_slide(h, id=50))
# set the id-random-th point as the starting point (everywhere)
set.seed(123) # just for the reproducibility
id_random <- sample(x=min(sapply(h$coo, nrow)), size=length(h), replace=TRUE)
stack(coo_slide(h, id=id_random))
coo_slidedirection

**Slides coordinates in a particular direction**

**Description**

Shapes are centered and then, according to direction, the point northwards, southwards, eastwards or westwards the centroid, becomes the first point with coo_slide. ‘right’ is possibly the most sensible option (and is by default), since 0 radians points eastwards, relatively to the origin. This should be followed by a coo_untiltx is most cases to remove any rotationnal dephasing/bias.

**Usage**

```r
coo_slidedirection(  
  coo,  
  direction = c("down", "left", "up", "right")[4],  
  center,  
  id  
)
```

**Arguments**

- **coo** matrix of (x; y) coordinates or any Coo object.
- **direction** character one of "down", "left", "up", "right" ("right" by default)
- **center** logical whether to center or not before sliding
- **id** numeric whether to return the id of the point or the slided shapes

**Value**

a matrix of (x; y) coordinates, or a Coo object.

**See Also**

Other sliding functions: coo_slidegap(), coo_slide()

Other coo Utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samperr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()
Examples

b <- coo_rotate(bot[1], pi/6) # dummy example just to make it obvious
coo_plot(b) # not the first point
coo_plot(coo_slidedirection(b, "up"))
coo_plot(coo_slidedirection(b, "right"))
coo_plot(coo_slidedirection(b, "left"))
coo_plot(coo_slidedirection(b, "down"))

# on Coo objects
b <- bot %>% slice(1:5) # for speed sake
stack(b)
stack(coo_slidedirection(b, "right"))

# This should be followed by a [coo_untiltx] in most (if not all) cases
stack(coo_slidedirection(b, "right") %>% coo_untiltx)

---

coo_slidegap       Slides coordinates using the widest gap

Description

When slicing a shape using two landmarks, or functions such as coo_up, an open curve is obtained and the rank of points make wrong/artefactual results. If the widest gap is > 5 * median of other gaps, then the couple of coordinates forming this widest gap is used as starting and ending points. This switch helps to deal with open curves. Examples are self-speaking. Use force=TRUE to bypass this check.

Usage

coo_slidegap(coo, force)

Arguments

coo          matrix of (x; y) coordinates or any Coo object.
force        logical whether to use the widest gap, with no check, as the real gap

Value

a matrix of (x; y) coordinates or a Coo object.

See Also

Other sliding functions: coo_slidedirection(), coo_slide()

Other coo Utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_smooth

Examples

cat <- coo_center(shapes[4])
coo_plot(cat)

# we only retain the bottom of the cat
cat_down <- coo_down(cat, slidegap=FALSE)

# see? the segment on the x-axis coorespond to the widest gap.
coo_plot(cat_down)

# that's what we meant
coo_plot(coo_slidegap(cat_down))

---

**coo_smooth**  
*Smoothes coordinates*

**Description**

Smoothes coordinates using a simple moving average. May be useful to remove digitization noise, mainly on outlines and open outlines.

**Usage**

`coo_smooth(coo, n)`

**Arguments**

- `coo` matrix of (x; y) coordinates or any Coo object.
- `n` integer the number of smoothing iterations

**Value**

a matrix of (x; y) coordinates, or a Coo object.

**See Also**

Other smoothing functions: `coo_smoothcurve()`

Other coo utilities: `coo_alignalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()``, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()``, `coo_center()``, `coo_centpos()``, `coo_close()``, `coo_down()``, `coo_dxy()``, `coo_extract()``, `coo_flipx()``, `coo_force2close()``, `coo_interpolate()``, `coo_is_closed()``, `coo_jitter()``, `coo_left()``, `coo_likely_clockwise()``, `coo_nb()``, `coo_perim()``, `coo_range()``, `coo_rev()``, `coo_right()``, `coo_rotatecenter()``, `coo_rotate()``, `coo_sample_prop()``, `coo_sampler()``, `coo_sample()``, `coo_scale()``, `coo_shearx()``, `coo_smoothcurve()``, `coo_smooth()``, `coo_template()``, `coo_trans()``, `coo_trimbotttom()``, `coo_trimtop()``, `coo_trim()``, `coo_untiltx()``, `coo_up()``, `is_equallyspacedradii()`
coo_sample_prop(), coo_samlerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

b5 <- slice(bot, 1:5) # for speed sake
stack(b5)
stack(coo_smooth(b5, 10))
coo_plot(b5[1])
coo_plot(coo_smooth(b5[1], 30))

coo_smoothcurve  Smoothes coordinates on curves

Description

Smoothes coordinates using a simple moving average but let the first and last points unchanged. May be useful to remove digitization noise on curves.

Usage

coo_smoothcurve(coo, n)

Arguments

coo  matrix of (x; y) coordinates or any Coo object.

n  integer to specify the number of smoothing iterations

Value

a matrix of (x; y) coordinates, or a Coo object.

See Also

Other smoothing functions: coo_smooth()

Other coo_utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo Likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samlerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()
**coo_solidity**

**Examples**

```r
o <- olea[1]
coo_plot(o, border='grey50', points=FALSE)
coo_draw(coo_smooth(o, 24), border='blue', points=FALSE)
coo_draw(coo_smoothcurve(o, 24), border='red', points=FALSE)
```

---

**coo_solidity**  *Calculates the solidity of a shape*

**Description**

Calculated using the ratio of the shape area and the convex hull area.

**Usage**

```r
coo_solidity(coo)
```

**Arguments**

- `coo` a matrix of (x; y) coordinates or any Coo

**Value**

numeric for a single shape, list for Coo

**Source**


**See Also**

Other coo_descriptors: `coo_angle_edges()`, `coo_angle_tangent()`, `coo_area()`, `coo_boundingbox()`, `coo_chull()`, `coo_circularity()`, `coo_convexity()`, `coo_eccentricity()`, `coo_elongation()`, `coo_length()`, `coo_lw()`, `coo_rectangularity()`, `coo_rectilinearity()`, `coo_scalars()`, `coo_tac()`, `coo_width()`

**Examples**

```r
coo_solidity(bot[1])
```
coo_tac

*Calculates the total absolute curvature of a shape*

**Description**

Calculated using the sum of the absolute value of the second derivative of the `smooth.spline` prediction for each defined point.

**Usage**

`coo_tac(coo)`

**Arguments**

- `coo` a matrix of (x; y) coordinates or any Coo

**Value**

numeric for a single shape and for Coo

**Source**

Siobhan Braybrook.

**See Also**

Other cooDescriptors: `coo_angle_edges()`, `coo_angle_tangent()`, `coo_area()`, `coo_boundingbox()`, `coo_chull()`, `coo_circularity()`, `coo_convexity()`, `coo_eccentricity()`, `coo_elongation()`, `coo_length()`, `coo_lw()`, `coo_rectangularity()`, `coo_rectilinearity()`, `coo_scalars()`, `coo_solidity()`, `coo_width()`

**Examples**

```r
coo_tac(bot[1])

bot %>%
slice(1:3) %>% # for speed sake only
coo_tac
```
### coo_template

**Templates** shapes

coo_template returns shape centered on the origin and inscribed in a size-side square. coo_template_relatively does the same but the biggest shape (as prod(coo_diffrange)) will be of size=size and consequently not defined on single shapes.

#### Usage

```r
coo_template(coo, size)
```

#### Arguments

- **coo**: A list or a matrix of coordinates.
- **size**: numeric. Indicates the length of the side 'inscribing' the shape.

#### Details

See coo_listpanel for an illustration of this function. The morphospaces functions also take profit of this function. May be useful to develop other graphical functions.

#### Value

Returns a matrix of (x; y)coordinates.
See Also

Other coo utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(),
coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(),
coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(),
coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(),
coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(),
coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(),
coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(),
coo_trans(), coo_trimbott(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Other scaling functions: coo_scale()

Examples

```r
coo <- bot[1]
coo_plot(coo_template(coo), xlim=c(-1, 1), ylim=c(-1, 1))
rect(-0.5, -0.5, 0.5, 0.5)

s <- .01
coo_plot(coo_template(coo, s))
rect(-s/2, -s/2, s/2, s/2)
```

---

### coo_trans

Translates coordinates

Description

Translates the coordinates by a 'x' and 'y' value

Usage

```r
coo_trans(coo, x = 0, y = 0)
```

Arguments

- `coo` matrix of (x; y) coordinates or any Coo object.
- `x` numeric translation along the x-axis.
- `y` numeric translation along the y-axis.

Value

A matrix of (x; y) coordinates, or a Coo object.
coo_trim

See Also

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidetrag(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up(), is_equallyspacedradii()

Examples

coo_plot(bot[1])
coo_plot(coo_trans(bot[1], 50, 100))

# on Coo
b <- bot %>% slice(1:5) # for speed sake
stack(b)
stack(coo_trans(b, 50, 100))

coo_trim

Trims both ends coordinates from shape

Description

Removes trim coordinates at both ends of a shape, ie from top and bottom of the shape matrix.

Usage

coo_trim(coo, trim = 1)

Arguments

coo matrix of (x; y) coordinates or any Coo object.
trim numeric, the number of coordinates to trim

See Also

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidetrag(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_untiltx(), coo_up(), is_equallyspacedradii()
Examples

```r
olea[1] %>% coo_sample(12) %>%
  print() %>% ldk_plot() %>%
  coo_trim(1) %>% print() %>% points(col="red")
```

---

**coo_trimbottom**  

*Trims bottom coordinates from shape*

### Description

Removes trim coordinates from the bottom of a shape.

### Usage

```r
coo_trimbottom(coo, trim = 1)
```

### Arguments

- `coo`  
  matrix of (x; y) coordinates or any `Coo` object.

- `trim`  
  numeric, the number of coordinates to trim

### See Also

Other `coo` utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_sampler()`, `coo_sample()`, `coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidegap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimtop()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Other `coo` trimming functions: `coo_trimtop()`, `coo_trim()`

### Examples

```r
olea[1] %>% coo_sample(12) %>%
  print() %>% ldk_plot() %>%
  coo_trimbottom(4) %>% print() %>% points(col="red")
```
**coo_trimtop**

**Description**

Removes trim coordinates from the top of a shape.

**Usage**

```r
coo_trimtop(coo, trim = 1)
```

**Arguments**

- `coo`: matrix of (x; y) coordinates or any Coo object.
- `trim`: numeric, the number of coordinates to trim

**See Also**

Other coo utilities: `coo_aligncalliper()`, `coo_alignminradius()`, `coo_alignxax()`, `coo_align()`, `coo_baseline()`, `coo_bookstein()`, `coo_boundingbox()`, `coo_calliper()`, `coo_centdist()`, `coo_center()`, `coo_centpos()`, `coo_close()`, `coo_down()`, `coo_dxy()`, `coo_extract()`, `coo_flipx()`, `coo_force2close()`, `coo_interpolate()`, `coo_is_closed()`, `coo_jitter()`, `coo_left()`, `coo_likely_clockwise()`, `coo_nb()`, `coo_perim()`, `coo_range()`, `coo_rev()`, `coo_right()`, `coo_rotatecenter()`, `coo_rotate()`, `coo_sample_prop()`, `coo_samplerr()`, `coo_sample()` `, coo_scale()`, `coo_shearx()`, `coo_slice()`, `coo_slidedirection()`, `coo_slidgap()`, `coo_slide()`, `coo_smoothcurve()`, `coo_smooth()`, `coo_template()`, `coo_trans()`, `coo_trimbbottom()`, `coo_trim()`, `coo_untiltx()`, `coo_up()`, `is_equallyspacedradii()`

Other coo trimming functions: `coo_trimbbottom()`, `coo_trim()`

**Examples**

```r
olea[1] %>% coo_sample(12) %>%
print() %>% ldk_plot() %>%
coo_trimtop(4) %>% print() %>% points(col="red")
```

---

**coo_truss**

**Truss measurement**

**Description**

A method to calculate on shapes or on Coo truss measurements, that is all pairwise combinations of euclidean distances.

**Usage**

```r
coo_truss(x)
```
Arguments

x   a shape or an Ldk object

Value

a named numeric or matrix

Note

Mainly implemented for historical/didactical reasons.

See Also

Other premodern: measure()

Examples

# example on a single shape
cat <- coo_sample(shapes[4], 6)
coo_truss(cat)

# example on wings dataset
tx <- coo_truss(wings)
dim(tx)
# we normalize and plot an heatmap
txn <- apply(tx$coe, 2, .normalize)
# heatmap(txn)

txp <- PCA(tx, scale. = TRUE, center=TRUE, fac=wings$fac)
plot(txp, 1)

coo_untiltx  
Removes rotation so that the centroid and a given point are parallel to the x-axis

Description

Rotationnal biases appear after coo_slidedirection (and friends). Typically useful for outline analysis where phasing matters. See examples.

Usage

coo_untiltx(coo, id, ldk)

Arguments

coo          matrix of (x; y) coordinates or any Coo object.
id           numeric the id of the point that will become the new first point. See details below for the method on Coo objects.
ldk          numeric the id of the ldk to use as id, only on Out
Details

For Coo objects, and in particular for Out and Opn two different ways of coo_sliding are available:

- **no ldk passed and an id is passed**: all id-th points within the shapes will become the first points.
- **a single ldk is passed**: the ldk-th ldk will be used to slide every shape. If an id is (also) passed, id is ignored with a message.

Value

a matrix of (x; y) coordinates, or a Coo object.

See Also

coo_slide and friends.

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_up(), is_equallyspacedradii()

Examples

# on a single shape

bot[1] %>% coo_center %>% coo_align %>%
  coo_sample(12) %>% coo_slidedirection("right") %>%
  coo_plot() %>% # the first point is not on the x-axis
  coo_untiltx() %>%
  coo_draw(border="red") # this (red) one is

# on an Out

# prepare bot

prebot <- bot %>% coo_center %>% coo_scale %>%
  coo_align %>% coo_slidedirection("right")

prebot %>% stack # some dephasing remains

prebot %>% coo_slidedirection("right") %>% coo_untiltx() %>% stack # much better

# _here_ there is no change but the second, untilted, is correct

prebot %>% efourier(8, norm=FALSE) %>% PCA %>% plot_PCA(~type)

prebot %>% coo_untiltx %>% efourier(8, norm=FALSE) %>% PCA %>% plot_PCA(~type)

# an example using ldks:

# the landmark #2 is on the x-axis

hearts %>

slice(1:5) %>% fgProcrustes(tol=1e-3) %>% # for speed sake
  coo_center %>% coo_untiltx(ldk=2) %>% stack
coo_up

Retains coordinates with positive y-coordinates

Description
Useful when shapes are aligned along the x-axis (e.g. because of a bilateral symmetry) and when one wants to retain just the upper side.

Usage
coo_up(coo, slidegap = FALSE)

Arguments
- coo: matrix of (x; y) coordinates or any Coo object.
- slidegap: logical whether to apply coo_slidegap after coo_down

Value
a matrix of (x; y) coordinates or a Coo object (Out are returned as Opn)

Note
When shapes are "sliced" along the x-axis, it usually results on open curves and thus to huge/artefactual gaps between points neighboring this axis. This is usually solved with coo_slidegap. See examples there.
Also, when apply a coo_left/right/up/down on an Out object, you then obtain an Opn object, which is done automatically.

See Also
Other opening functions: coo_down(), coo_left(), coo_right()
Other coo_ utilities: coo_alignalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samperr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), is_equallyspacedradii()

Examples
b <- coo_alignxax(bot[1])
coo_plot(b)
coo_draw(coo_up(b), border='red')
coo_width

Calculates the width of a shape

Description
Nothing more than coo_lw(coo)[2].

Usage
coo_width(coo)

Arguments
coo a matrix of (x; y) coordinates or Coo object

Value
the width (in pixels) of the shape

See Also
coo_lw, coo_length.
Other coo_descriptors: coo_angle_edges(), coo_angle_tangent(), coo_area(), coo_boundingbox(), coo_chull(), coo_circularity(), coo_convexity(), coo_eccentricity(), coo_elongation(), coo_length(), coo_lw(), coo_rectangularity(), coo_rectilinearity(), coo_scalars(), coo_solidity(), coo_tac()

Examples
coo_width(bot[1])

d
A wrapper to calculates euclidean distances between two points

Description
The main advantage over ed is that it is a method that can be passed to different objects and used in combination with measure. See examples.

Usage
d(x, id1, id2)
def_ldk

Arguments

- `x`: a Ldk (typically), an Out or a matrix
- `id1`: id of the 1st row
- `id2`: id of the 2nd row

Note

On Out objects, we first `get_ldk`.

See Also

If you want all pairwise combinations, see `coo_truss`.

Examples

```r
# single shape
d(wings[1], 1, 4)
# Ldk object
d(wings, 1, 4)
# Out object
d(hearts, 2, 4)
```

---

def_ldk  

*Defines new landmarks on Out and Opn objects*

Description

Helps to define landmarks on a `Coo` object. The number of landmarks must be specified and rows indices that correspond to the nearest points clicked on every outlines are stored in the `$ldk` slot of the `Coo` object.

Usage

```r
def_ldk(Coo, nb.ldk, close, points)
```

Arguments

- `Coo`: an Out or Opn object
- `nb.ldk`: the number of landmarks to define on every shape
- `close`: logical whether to close (typically for outlines)
- `points`: logical whether to display points

Value

An Out or an Opn object with some landmarks defined
def_ldk_angle

Add new landmarks based on angular positions

Description

A wrapper on coo_intersect_angle and coo_intersect_direction for Out and Opn objects.

Usage

def_ldk_angle(coo, angle)

def_ldk_direction(coo, direction = c("down", "left", "up", "right")[4])

## Default S3 method:
def_ldk_direction(coo, direction = c("down", "left", "up", "right")[4])

## S3 method for class 'Out'
def_ldk_direction(coo, direction = c("down", "left", "up", "right")[4])

## S3 method for class 'Opn'
def_ldk_direction(coo, direction = c("down", "left", "up", "right")[4])

Arguments

coo a codeOut or Opn object
angle numeric an angle in radians (0 by default).
direction character one of "down", "left", "up", "right" ("right" by default)

Note

any existing ldk will be preserved.
See Also

Typically used before coo_slice and coo_slide. See def_ldk_tips as well.

Examples

```r
# adds a new landmark towards south east
hearts %>%
slice(1:5) %>% # for speed purpose only
def_ldk_angle(-pi/6) %>%
stack()

# on Out and towards NW and NE here
olea %>%
slice(1:5) %>% # for speed purpose only
def_ldk_angle(3*pi/4) %>%
def_ldk_angle(pi/4) %>%
stack
```

def_ldk_tips

Define tips as new landmarks

Description

On Opn objects, this can be used before coo_slice. See examples.

Usage

```r
def_ldk_tips(coo)
```

Arguments

- `coo` Opn object

Note

Any existing ldk will be preserved.

Examples

```r
is_ldk(olea) # no ldk for olea
olea %>%
slice(1:3) %>% # for the sake of speed
def_ldk_tips %>%
def_ldk_angle(3*pi/4) %>% def_ldk_angle(pi/4) %T>% stack %>%
coo_slice(ldk=1:4) -> oleas
stack(oleas[[1]])
stack(oleas[[2]]) # etc.
```
\textbf{def_links} \hspace{1cm} \textit{Defines links between landmarks}

\section*{Description}
Works on Ldk objects, on 2-cols matrices, 3-dim arrays (MSHAPES turns it into a matrix).

\section*{Usage}
def_links(x, nb.ldk)

\section*{Arguments}
x \hspace{1cm} Ldk, matric or array
nb.ldk \hspace{1cm} numeric the iterative procedure is stopped when the user click on the top of the graphical window.

\section*{See Also}
Other ldk helpers: ldk_check(), links_all(), links_delaunay()

\section*{Examples}
\begin{verbatim}
## Not run:
wm <- MSHAPES(wings)
links <- def_links(wm, 3) # click to define pairs of landmarks
ldk_links(wm, links)
## End(Not run)
\end{verbatim}

\textbf{def_slidings} \hspace{1cm} \textit{Defines sliding landmarks matrix}

\section*{Description}
Defines sliding landmarks matrix

\section*{Usage}
def_slidings(Coo, slidings)

\section*{Arguments}
Coo \hspace{1cm} an \texttt{Ldk} object
slidings \hspace{1cm} a matrix, a numeric or a list of numeric. See Details
$slidings$ in \texttt{Ldk} must be a 'valid' matrix: containing ids of coordinates, none of them being lower than 1 and higher the number of coordinates in $\texttt{coo}$. 

\texttt{slidings} matrix contains 3 columns (before, slide, after). It is inspired by \texttt{geomorph} and should be compatible with it.

This matrix can be passed directly if the \texttt{slidings} argument is a matrix. Of course, it is strictly equivalent to \texttt{Ldk$slidings <- slidings}$.

\texttt{slidings} can also be passed as "partition(s)", when sliding landmarks identified by their ids (which are a row number) are consecutive in the \texttt{coo}.

A single partition can be passed either as a numeric (eg 4:12), if points 5 to 11 must be considered as sliding landmarks (4 and 12 being fixed); or as a list of numeric.

See examples below.

See Also

Other \texttt{ldk/slidings} methods: \texttt{add_ldk()}, \texttt{def_ldk()}, \texttt{get_ldk()}, \texttt{get_slidings()}, \texttt{rearrange_ldk()}, \texttt{slidings_scheme()}

Examples

```
#waiting for a sliding dataset...
```

dfourier \quad \textit{Discrete cosinus transform}

Description

Calculates discrete cosine transforms, as introduced by Dommergues and colleagues, on a shape (mainly open outlines).

Usage

```
dfourier(coo, nb.h)
```

```
## Default S3 method:
dfourier(coo, nb.h)
```

```
## S3 method for class 'Opn'
dfourier(coo, nb.h)
```

```
## S3 method for class 'list'
dfourier(coo, nb.h)
```

```
## S3 method for class 'Coo'
dfourier(coo, nb.h)
```
dfourier

Arguments

- coo: a matrix (or a list) of (x; y) coordinates
- nb.h: numeric the number of harmonics to calculate

Value

- a list with the following components:
  - an: the A harmonic coefficients
  - bn: the B harmonic coefficients
  - mod: the modules of the points
  - arg: the arguments of the points

Note

This method has been only poorly tested in Momocs and should be considered as experimental. Yet improved by a factor 10, this method is still long to execute. It will be improved in further releases but it should not be so painful right now. It also explains the progress bar. Shapes should be aligned before performing the dct transform.

Silent message and progress bars (if any) with options("verbose"=FALSE).

References

- Many thanks to Remi Laffont for the translation in R.

See Also

Other dfourier: dfourier_i(), dfourier_shape()

Examples

```r
## Not run: # because it's long
od <- dfourier(olea)
od
op <- PCA(od)
plot(op, 1)

## End(Not run)
# dfourier and inverse dfourier
o <- olea[1]
o <- coo_bookstein(o)
coo_plot(o)
o.dfourier <- dfourier(o, nb.h=12)
o.dfourier
o.i <- dfourier_i(o.dfourier)
o.i <- coo_bookstein(o.i)
```
coo_draw(o,i, border='red')

#future calibrate_reconstructions
o <- olea[1]
h.range <- 2:13
coo <- list()
for (i in seq(along=h.range)){
  coo[[i]] <- dfourier_i(dfourier(o, nb.h=h.range[i]))
  names(coo) <- paste0('h', h.range)
panel(Opn(coo), borders=col_india(12), names=TRUE)
title('Discrete Cosine Transforms')

---

dfourier_i  
*Investe discrete cosinus transform*

**Description**

Calculates inverse discrete cosine transforms (see dfourier), given a list of A and B harmonic coefficients, typically such as those produced by dfourier.

**Usage**

dfourier_i(df, nb.h, nb.pts = 60)

**Arguments**

- `df`  
a list with $A$ and $B$ components, containing harmonic coefficients.
- `nb.h`  
a custom number of harmonics to use
- `nb.pts`  
numeric the number of pts for the shape reconstruction

**Value**

a matrix of (x; y) coordinates

**Note**

Only the core functions so far. Will be implemented as an Opn method soon.

**References**

- Many thanks to Remi Laffont for the translation in R.

**See Also**

Other dfourier: `dfourier_shape()`, `dfourier()`
Examples

# dfourier and inverse dfourier
o <- olea[1]
o <- coo_bookstein(o)
coo_plot(o)
o.dfourier <- dfourier(o, nb.h=12)
o.dfourier
o.i <- dfourier_i(o.dfourier)
o.i <- coo_bookstein(o.i)
coo_draw(o.i, border='red')

o <- olea[1]
h.range <- 2:13
coo <- list()
for (i in seq(along=h.range)){
  coo[[i]] <- dfourier_i(dfourier(o, nb.h=h.range[i]))
  names(coo) <- paste0('h', h.range)
  panel(Opn(coo), borders=col_india(12), names=TRUE)
  title('Discrete Cosine Transforms')
}

dfourier_shape

Calculates and draws 'dfourier' shapes

Description

Calculates shapes based on 'Discrete cosine transforms' given harmonic coefficients (see dfourier) or can generate some random 'dfourier' shapes. Mainly intended to generate shapes and/or to understand how dfourier works.

Usage

dfourier_shape(A, B, nb.h, nb.pts = 60, alpha = 2, plot = TRUE)

Arguments

A vector of harmonic coefficients
B vector of harmonic coefficients
nb.h if A and/or B are not provided, the number of harmonics to generate
nb.pts if A and/or B are not provided, the number of points to use to reconstruct the shapes
alpha The power coefficient associated with the (usually decreasing) amplitude of the harmonic coefficients (see efourier_shape)
plot logical whether to plot the shape

See Also

Other dfourier: dfourier_i(), dfourier()
Examples

```r
# some signatures
panel(coo_align(Opn(replicate(48, dfourier_shape(alpha=0.5, nb.h=6)))))
# some worms
panel(coo_align(Opn(replicate(48, dfourier_shape(alpha=2, nb.h=6))))
```

dissolve

Dissolve Coe objects

Description

the opposite of combine, typically used after it. Note that the $\text{fac}$ slot may be wrong since combine...well combines... this $\text{fac}$. See examples.

Usage

dissolve(x, retain)

Arguments

- `x`: a Coe object
- `retain`: the partition id to retain. Or their name if the partitions are named (see `x$\text{method}`)
  eg after a chop

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsetize()

Examples

data(bot)
w <- filter(bot, type="whisky")
b <- filter(bot, type="beer")
wf <- efourier(w, 10)
bf <- efourier(b, 10)
wbf <- combine(wf, bf)
dissolve(wbf, 1)
dissolve(wbf, 2)

# or using chop (yet combine here makes no sense)
bw <- bot %>% chop(~type) %>% lapply(efourier, 10) %>% combine %>% dissolve(1)
bw %>% dissolve(2)
drawers


drawers  grindr drawers for shape plots

description

Useful drawers for building custom shape plots using the grindr approach. See examples and vignettes.

Usage

\begin{verbatim}
draw_polygon(
    coo,
    f,
    col = par("fg"),
    fill = NA,
    lwd = 1,
    lty = 1,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_outline(
    coo,
    f,
    col = par("fg"),
    fill = NA,
    lwd = 1,
    lty = 1,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_outlines(
    coo,
    f,
    col = par("fg"),
    fill = NA,
    lwd = 1,
    lty = 1,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_points(
\end{verbatim}
draw_landmarks(
    coo,
    f,
    col = par("fg"),
    cex = 1/2,
    pch = 20,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_lines(
    coo,
    f,
    col = par("fg"),
    lwd = 1,
    lty = 1,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_centroid(
    coo,
    f,
    col = par("fg"),
    pch = 3,
    cex = 0.5,
    transp = 0,
    pal = pal_qual,
    ...
)

draw_curve(
    coo,
    f,
    col = par("fg"),
    lwd = 1,
lty = 1,
transp = 0,
pal = pal_qual,
...
)
draw_curves(
  coo,
  f,
  col = par("fg"),
  lwd = 1,
  lty = 1,
  transp = 0,
  pal = pal_qual,
  ...
)
draw_firstpoint(
  coo,
  f,
  label = "^",
  col = par("fg"),
  cex = 3/4,
  transp = 0,
  pal = pal_qual,
  ...
)
draw_axes(coo, col = "#999999", lwd = 1/2, ...)
draw_ticks(coo, col = "#333333", cex = 3/4, lwd = 3/4, ...)
draw_labels(coo, labels = 1:nrow(coo), cex = 1/2, d = 1/20, ...)
draw_links(
  coo,
  f,
  links,
  col = "#99999955",
  lwd = 1/2,
  lty = 1,
  transp = 0,
  pal = pal_qual,
  ...
)
draw_title(
  coo,
main = "", sub = "", cex = c(1, 3/4), font = c(2, 1), padding = 1/200, ...)

Arguments

- **coo**: matrix of 2 columns for (x, y) coordinates
- **f**: an optional factor specification to feed. See examples and vignettes.
- **col**: color (hexadecimal) to draw components
- **fill**: color (hexadecimal) to draw components
- **lwd**: to draw components
- **lty**: to draw components
- **transp**: numeric transparency (default: 0, min: 0, max: 1)
- **pal**: a palette to use if no col/border/etc. are provided. See [palettes]
- **...**: additional options to feed core functions for each drawer
- **cex**: to draw components ((c(2,1) by default) for `draw_title`)
- **pch**: to draw components
- **label**: to indicate first point
- **labels**: character name of labels to draw (default to 1:nrow(coo))
- **d**: numeric proportion of d(centroid-each_point) to add when centrifugating landmarks
- **links**: matrix of links to use to draw segments between landmarks. See `wings$ldk` for an example
- **main**: character title (empty by default)
- **sub**: character subtitle (empty by default)
- **font**: numeric to feed text (c(2,1) by default)
- **padding**: numeric a fraction of the graphical window (1/200 by default)

Note

This approach will (soon) replace `coo_plot` and friends in further versions. All comments are welcome.

See Also

- `grindr_layers`
- Other grindr: `layers_morphospace, layers, mosaic_engine(), papers, pile(), plot_LDA(), plot_NMDS(), plot_PCA()`
ed

Calculates euclidean distance between two points.

Description

ed simply calculates euclidean distance between two points defined by their (x; y) coordinates.

Usage

ed(pt1, pt2)

Arguments

pt1 (x; y) coordinates of the first point.
pseudocode: 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))
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.
edm_nearest

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
```r
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
```r
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 consuming since it performs n × 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)

Description

efourier computes Elliptical Fourier Analysis (or Transforms or EFT) from a matrix (or a list) of (x; y) coordinates. efourier_norm normalizes Fourier coefficients. Read Details carefully.

Usage

efourier(x, ...)

## Default S3 method:
efourier(x, nb.h, smooth.it = 0, ...)

## S3 method for class 'Out'
efourier(x, nb.h, smooth.it = 0, norm = TRUE, start = FALSE, ...)

## S3 method for class 'list'
efourier(x, ...)

efourier_norm(ef, start = FALSE)

Arguments

x A list or a matrix of coordinates or a Out object
...
useless here
nb.h integer. The number of harmonics to use. If missing, 12 is used on shapes; 99 percent of harmonic power on Out objects, both with messages.
smooth.it integer. The number of smoothing iterations to perform.
norm whether to normalize the coefficients using efourier_norm
efourier

```
start logical. For efourier whether to consider the first point as homologous; for
efourier_norm whether to conserve the position of the first point of the outline.
```

ef list with a_n, b_n, c_n and d_n Fourier coefficients, typically returned by
efourier

Details

For the maths behind see the paper in JSS.

Normalization of coefficients has long been a matter of trouble, and not only for newcomers. There
are two ways of normalizing outlines: the first, and by far the most used, is to use a "numerical"
alignment, directly on the matrix of coefficients. The coefficients of the first harmonic are consumed
by this process but harmonics of higher rank are normalized in terms of size and rotation. This is
sometimes referred as using the "first ellipse", as the harmonics define an ellipse in the plane, and
the first one is the mother of all ellipses, on which all others "roll" along. This approach is really
convenient as it is done easily by most software (if not the only option) and by Momocs too. It is
the default option of efourier.

But here is the pitfall: if your shapes are prone to bad aligns among all the first ellipses, this
will result in poorly (or even not at all) "homologous" coefficients. The shapes particularly prone
to this are either (at least roughly) circular and/or with a strong bilateral symmetry. You can try
to use stack on the Coe object returned by efourier. Also, and perhaps more explicitly, mor-
phospace usually show a mirroring symmetry, typically visible when calculated in some couple of
components (usually the first two).

If you see these upside-down (or 180 degrees rotated) shapes on the morphospace, you should
seriously consider aligning your shapes before the efourier step, and performing the latter with
norm = FALSE.

Such a pitfall explains the (quite annoying) message when passing efourier with just the Out.

You have several options to align your shapes, using control points (or landmarks), by far the most
time consuming (and less reproducible) but possibly the best one too when alignment is too tricky
to automate. You can also try Procrustes alignment (see fgProcrustes) through their calliper
length (see coo_aligncalliper), etc. You should also make the first point homologous either with
coo_slide or coo_slidedirection to minimize any subsequent problems.

I will dedicate (some day) a vignette or a paper to this problem.

Value

For efourier, a list with components: an, bn, cn, dn harmonic coefficients, plus ao and co. The
latter should have been named a0 and c0 in Claude (2008) but I (intentionnaly) propagated the error.

For efourier_norm, a list with components: A, B, C, D for harmonic coefficients, plus size, the
magnitude of the semi-major axis of the first fitting ellipse, theta angle, in radians, between the
starting and the semi-major axis of the first fitting ellipse, psi orientation of the first fitting ellipse,
ao and do, same as above, and lnef that is the concatenation of coefficients.

Note

Directly borrowed for Claude (2008).

Silent message and progress bars (if any) with options("verbose"=FALSE).
References

See Also
Other efourier: `efourier_i()`, `efourier_shape()

Examples

```r
# single shape
coo <- bot[1]
coo_plot(coo)
ef <- efourier(coo, 12)
# same but silent
efourier(coo, 12, norm=TRUE)
# inverse EFT
efi <- efourier_i(ef)
coo_draw(efi, border="red", col=NA)

# on Out
bot %>% slice(1:5) %>% efourier
```

---

**efourier_i**

*Inverse elliptical Fourier transform*

**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**

```r
efourier_i(ef, nb.h, nb.pts = 120)
```

**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.
efourier_shape

Value
A matrix of (x; y) coordinates.

Note
Directly borrowed for Claude (2008), and also called iefourier there.

References

See Also
Other efourier: efourier_shape(), efourier()

Examples
coo <- bot[1]
coo_plot(coo)
ef <- efourier(coo, 12)
ef
efi <- efourier_i(ef)
coo_draw(efi, border='red', col=NA)

---

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. Mainly intended to generate shapes and/or to understand how efourier works.

Usage
efourier_shape(an, bn, cn, dn, nb.h, nb.pts = 60, alpha = 2, plot = TRUE)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>an</td>
<td>numeric. The (a_n) Fourier coefficients on which to calculate a shape.</td>
</tr>
<tr>
<td>bn</td>
<td>numeric. The (b_n) Fourier coefficients on which to calculate a shape.</td>
</tr>
<tr>
<td>cn</td>
<td>numeric. The (c_n) Fourier coefficients on which to calculate a shape.</td>
</tr>
<tr>
<td>dn</td>
<td>numeric. The (d_n) Fourier coefficients on which to calculate a shape.</td>
</tr>
<tr>
<td>nb.h</td>
<td>integer. The number of harmonics to use.</td>
</tr>
<tr>
<td>nb.pts</td>
<td>integer. The number of points to calculate.</td>
</tr>
</tbody>
</table>
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$^{alpha}$. 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


See Also

Other efourier: efourier_i(), efourier()

Examples

```r
ef <- efourier(bot[1], 24)
efourier_shape(ef$an, ef$bn, ef$cn, ef$dn) # equivalent to efourier_i(ef)
efourier_shape() # is autonomous

panel(Out(a2l(replicate(100,
efourier_shape(nb.h=6, alpha=2.5, plot=FALSE))))) # Bubble family
```

export

Exports Coe objects and shapes

Description

Exports Coe objects and shapes

Usage

`export(x, file, sep, dec)`
export

Arguments

- **x**: a Coe or PCA object
- **file**: the filenames `data.txt` by default
- **sep**: the field separator string to feed `write.table`. (default to tab) tab by default
- **dec**: the string to feed `write.table` (default "." ) by default.

Note

This is a simple wrapper around `write.table`.

Default parameters will write a `.txt` file, readable by foreign programs. With default parameters, numbers will use dots as decimal points, which is considered as a character chain in Excel in many countries (locale versions). This can be solved by using `dec=',',` as in the examples below.

If you are looking for your file, and did not specified `file`, `getwd()` will help.

I have to mention that everytime you use this function, and cowardly run from R to Excel and do 'statistics' there, an innocent and adorable kitten is probably murdered somewhere. Use R!

See Also

Other bridges functions: `as_df()`, `bridges`, `complex`

Examples

```r
## Not run:
# Will write files on your machine!
bf <- efourier(bot, 6)
# Export Coe (here Fourier coefficients)
export(bf) # data.txt which can be opened by every software including MS Excel

# If you come from a country that uses comma as decimal separator (not recommended, but...)
export(bf, dec=',')
export(bf, file='data.xls', dec=',')

# Export PCA scores
bf %>% PCA %>% export()

# for shapes (matrices)
export(bot[1], file='bot1.txt')

# remove these files from your machine
file.remove("coefficients.txt", "data.xls", "scores.txt")

## End(Not run)
```
fac_dispatcher

Brew and serve fac from Momocs object

Description

Ease various specifications for fac specification when passed to Momocs objects. Intensively used (internally).

Usage

fac_dispatcher(x, fac)

Arguments

- **x**: a Momocs object (any Coo, Coe, PCA, etc.)
- **fac**: a specification to extract from fac

Details

fac can be:

- a factor, passed on the fly
- a column id from $fac
- a column name from fac; if not found, return NULL with a message
- a formula in the form: ~column_name (from $fac, no quotes). It expresses more in a concise way. Also allows interacting on the fly. See examples.
- a NULL returns a NULL, with a message

Value

a prepared factor (or a numeric). See examples

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsetize()

Examples

bot <- mutate(bot, s=rnorm(40), fake=factor(rep(letters[1:4], 10)))

# factor, on the fly
fac_dispatcher(bot, factor(rep(letters[1:4], 10)))

# column id
fgProcrustes

Full Generalized Procrustes alignment between shapes

Description

Directly borrowed from Claude (2008), called there the fgpas2 function.

Usage

fgProcrustes(x, tol, coo)

Arguments

x an array, a list of configurations, or an Out, Opn or Ldk object
tol numeric when to stop iterations
coo logical, when working on Out or Opn, whether to use $coo rather than $ldk

Details

If performed on an Out or an Opn object, will try to use the $ldk slot, if landmarks have been previously defined, then (with a message) on the $coo slot, but in that case, all shapes must have the same number of coordinates (coo_sample may help).

Value

a list with components:

- rotated array of superimposed configurations
- iterationnumber number of iterations
- Q convergence criterion
- Q full list of Q
• Qd difference between successive Q
• interproc.dist minimal sum of squared norms of pairwise differences between all shapes in the superimposed sample
• nshape mean shape configuration
• cent.size vector of centroid sizes.

or an Out, Opn or an Ldk object.

Note
Slightly less optimized than procGPA in the shapes package (~20% on my machine). Will be optimized when performance will be the last thing to improve! Silent message and progress bars (if any) with options("verbose"=FALSE).

References

See Also
Other procrustes functions: fProcrustes(), fgsProcrustes(), pProcrustes()

Examples
## Not run:
# on Ldk
stack(wings)
fgProcrustes(wings, tol=0.1) %>% stack()

# on Out
stack(hearts)
fgProcrustes(hearts) %>% stack()

## End(Not run)

fgsProcrustes

Full Generalized Procrustes alignment between shapes with sliding landmarks

Description
Directly wrapped around geomorph::gpagen.

Usage
fgsProcrustes(x)
Arguments

x Ldk object with some $slidings

Note

Landmarks methods are the less tested in Momocs. Keep in mind that some features are still expe-
timental and that your help is welcome.

Source

See ?gpagen in geomorph package

See Also

Other procrustes functions: fProcrustes(), fgProcrustes(), pProcrustes()

Examples

## Not run:
chaffp <- fgsProcrustes(chaff)
chaffp
chaffp %>% PCA() %>% plot("taxa")
## End(Not run)

filter

Subset based on conditions

Description

Return shapes with matching conditions, from the $fac. See examples and ?dplyr::filter.

Usage

filter(.data, ...)

Arguments

.data a Coo, Coe, PCA object

... logical conditions

Details

dplyr verbs are maintained. You should probably not filter on PCA objects. The latter are calcu-
lated using all individuals and filtering may lead to false conclusions. If you want to highlith some
individuals, see examples in plot_PCA.
Value

a Momocs object of the same class.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), facDispatcher(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsetize()

Examples

olea
# we retain on dorsal views
filter(olea, view=="VD")
# only dorsal views and Aglan+PicMa varieties
filter(olea, view=="VD", var %in% c("Aglan", "PicMa"))
# we create an id column and retain the 120 first shapes
olea %>% mutate(id=1:length(olea)) %>% filter(id > 120)

flip_PCaxes

Flips PCA axes

Description

Simply multiply by -1, corresponding scores and rotation vectors for PCA objects. PC orientation being arbitrary, this may help to have a better display.

Usage

flip_PCaxes(x, axs)

Arguments

x a PCA object

axs numeric which PC(s) to flip

Examples

bp <- bot %>% efourier(6) %>% PCA
bp %>% plot
bp %>% flip_PCaxes(1) %>% plot()
**Description**

Data: Measurement of iris flowers

**Format**

A TraCoe object with 150 measurements of 4 variables (petal + sepal) x (length x width) on 3 species of iris. This dataset is the classical iris formatted for Momocs.

**Source**

see linkiris

**See Also**

Other datasets: apodemus, bot, chaff, charring, hearts, molar, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings

---

**fProcrustes**

*Full Procrustes alignment between two shapes*

**Description**

Directly borrowed from Claude (2008), called there the fPsup function.

**Usage**

`fProcrustes(coo1, coo2)`

**Arguments**

- `coo1` configuration matrix to be superimposed onto the centered preshape of coo2.
- `coo2` reference configuration matrix.

**Value**

a list with components:

- `coo1` superimposed centered preshape of coo1 onto the centered preshape of coo2
- `coo2` centered preshape of coo2
- `rotation` rotation matrix
- `scale` scale parameter
- `DF` full Procrustes distance between coo1 and coo2.
get_chull_area

Calculates convex hull area/volume of PCA scores

Reference


See Also

Other procrustes functions: fgProcrustes(), fgsProcrustes(), pProcrustes()

description

May be useful to compare shape diversity. Expressed in PCA units that should only be compared within the same PCA.

Usage

get_chull_area(x, fac, xax = 1, yax = 2)

get_chull_volume(x, fac, xax = 1, yax = 2, zax = 3)

Arguments

x a PCA object
fac (optionnal) column name or ID from the $fac slot.
xax the first PC axis to use (1 by default)
yax the second PC axis (2 by default)
zax the third PC axis (3 by default only for volume)

details

get_chull_area is calculated using coo_chull followed by coo_area; get_chull_volume is calculated using geometry::convexhulln

Value

If fac is not provided global area/volume is returned; otherwise a named list for every level of fac

Examples

bp <- PCA(efourier(bot, 12))
get_chull_area(bp)
get_chull_area(bp, 1)

getchull_volume(bp)
getchull_volume(bp, 1)
get_ldk

Retrieves landmarks coordinates

Description

See Details for the different behaviors implemented.

Usage

get_ldk(Coo)

Arguments

Coo an Out, Opn or Ldk object

Details

Different behaviors depending on the class of the object:

- **Ldk**: retrieves landmarks.
- Ldk with slidings defined: retrieves only the fixed landmarks, not the sliding ones. See also `get_slidings`.
- **Out**: landmarks from $ldk$ and $coo$, if any.
- **Opn**: same as above.

Value

a list of shapes

See Also

Other ldk/slidings methods: `add_ldk()`, `def_ldk()`, `def_slidings()`, `get_slidings()`, `rearrange_ldk()`, `slidings_scheme()`

Examples

# Out example
ldk.h <- get_ldk(hearts)
stack(Ldk(ldk.h))

# on Ldk (no slidings)
get_ldk(wings) # equivalent to wings$coo

# on Ldk (slidings)
get_ldk(chaff)
ge_ldk(chaff) %>% Ldk %>% fgProcrustes(tol=0.1) %>% stack
get_pairs

Get paired individual on a Coe, PCA or LDA objects

Description

If you have paired individuals, i.e. before and after a treatment or for repeated measures, and if you have coded it into $fac$, this method allows you to retrieve the corresponding PC/LD scores, or coefficients for Coe objects.

Usage

get_pairs(x, fac, range)

Arguments

- **x**: any Coe, PCA or LDA object.
- **fac**: factor or column name or id corresponding to the pairing factor.
- **range**: numeric the range of coefficients for Coe, or PC (LD) axes on which to return scores.

Value

A list with components:
- **x1**: all coefficients/scores corresponding to the first level of the fac provided;
- **x2**: same thing for the second level;
- **fac**: the corresponding fac.

Examples

```r
bot2 <- bot1 <- coo_scale(coo_center(coo_sample(bot, 60)))
bot1$fac$session <- factor(rep("session1", 40))  # we simulate an measurement error
bot2 <- coo_jitter(bot1, amount=0.01)
bot2$fac$session <- factor(rep("session2", 40))
botc <- combine(bot1, bot2)
botcf <- efourier(botc, 12)

# we gonna plot the PCA with the two measurement sessions and the two types
botcp <- PCA(botcf)
plot(botcp, "type", col=col_summer(2), pch=rep(c(1, 20), each=40), eigen=FALSE)
bot.pairs <- get_pairs(botcp, fac = "session", range=1:2)
segments(bot.pairs$session1[, 1], bot.pairs$session1[, 2],
         bot.pairs$session2[, 1], bot.pairs$session2[, 2],
         col=col_summer(2)[bot.pairs$fac$type]
```
get_slidings

Extracts sliding landmarks coordinates

Description
From an Ldk object.

Usage
get_slidings(Coo, partition)

Arguments
Coo: an Ldk object
partition: numeric which one(s) to get.

Value
a list of list(s) of coordinates.

See Also
Other ldk/slidings methods: add_ldk(), def_ldk(), def_slidings(), get_ldk(), rearrange_ldk(), slidings_scheme()

Examples
# for each example below a list with partition containign shapes is returned
# extracts the first partition
get_slidings(chaff, 1) %>% names()
# the first and the fourth
get_slidings(chaff, c(1, 4)) %>% names()
# all of them
get_slidings(chaff) %>% names
# here we want to see it
get_slidings(chaff, 1)[[1]] %>% Ldk %>% stack

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/t/fourier passed on coo

Value

Returns a vector of harmonic power

Examples

ef <- efourier(bot[1], 24)
rf <- efourier(bot[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 harmonic contribution to shape

Description

Calculates contribution of harmonics to shape. The amplitude of every coefficients of a given harmonic is multiplied by the coefficients provided and the resulting shapes are reconstructed and plotted. Naturally, only works on Fourier-based methods.

Usage

hcontrib(Coe, ...)

## S3 method for class 'OutCoe'
hcontrib(        Coe,
    id,
    harm.r,
    amp.r = c(0, 0.5, 1, 2, 5, 10),
    main = "Harmonic contribution to shape",
    xlab = "Harmonic rank",
    ylab = "Amplification factor",
    ...)


hearts

Arguments

- Coe: a Coe object (either OutCoe or (soon) OpnCoe)
- ...: additional parameter to pass to coo_draw
- id: the id of a particular shape, otherwise working on the meanshape
- harm.r: range of harmonics on which to explore contributions
- amp.r: a vector of numeric for multiplying coefficients
- main: a title for the plot
- xlab: a title for the x-axis
- ylab: a title for the y-axis

See Also

Other Coe_graphics: boxplot.OutCoe()

Examples

data(bot)
bot.f <- efourier(bot, 12)
hcontrib(bot.f)
hcontrib(bot.f, harm.r=3:10, amp.r=1:8, col="grey20",
main="A huge panel")

hearts  Data: Outline coordinates of hand-drawn hearts

Description

Data: Outline coordinates of hand-drawn hearts

Format

A Out object with the outline coordinates of 240 hand-drawn hearts by 8 different persons, with 4 landmarks.

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 downsampled to 80 coordinates per outline.

See Also

Other datasets: apodemus, bot, chaff, charring, flower, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo, wings
**img_plot**

*Plots a .jpg image*

**Description**

A very simple image plotter. If provided with a path, reads the .jpg and plots it. If not provided with an imagematrix, will ask you to choose interactively a .jpeg image.

**Usage**

```
img_plot(img)

img_plot0(img)
```

**Arguments**

- **img**: a matrix of an image, such as those obtained with `readJPEG`.

**Details**

`img_plot` is used in import functions such as `import_jpg1`; `img_plot0` does the same job but preserves the par and plots axes.

---

**import_Conte**

*Extract outlines coordinates from an image silhouette*

**Description**

Provided with an image 'mask' (i.e. black pixels on a white background), and a point form where to start the algorithm, returns the (x; y) coordinates of its outline.

**Usage**

```
import_Conte(img, x)
```

**Arguments**

- **img**: a matrix of a binary image mask.
- **x**: numeric the (x; y) coordinates of a starting point within the shape.

**Details**

Used internally by `import_jpg1` but may be useful for other purposes.

**Value**

a matrix the (x; y) coordinates of the outline points.
Note

Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.

If you have an image with more than a single shape, then you may want to try imager::highlight function. Momocs may use this at some point.

References


See Also

Other import functions: `import_StereoMorph_curve()`, `import_jpg1()`, `import_jpg()`, `import_tps()`, `import_txt()`, `pix2chc()`

---

**import_jpg**

Extract outline coordinates from multiple .jpg files

**Description**

This function is used to import outline coordinates and is built around `import_jpg1`.

**Usage**

```r
import_jpg(
  jpg.paths = .lf.auto(),
  auto.notcentered = TRUE,
  fun.notcentered = NULL,
  threshold = 0.5
)
```

**Arguments**

- `jpg.paths` a vector of paths corresponding to the .jpg files to import. If not provided (or `NULL`), switches to the automatic version. See Details below.
- `auto.notcentered` logical if TRUE random locations will be used until one of them is (assumed) to be within the shape (because of a black pixel); if FALSE a `locator` will be called, and you will have to click on a point within the shape.
fun.notcentered
threshold

Details
see import_jpg1 for important informations about how the outlines are extracted, and import_Conte
for the algorithm itself.

If jpg.paths is not provided (or NULL), you will have to select any .jpg file in the folder that
contains all your files. All the outlines should be imported then.

Value
a list of matrices of (x; y) coordinates that can be passed to Out

Note
Note this function will be deprecated from Momocs when Momacs and Momit will be fully opera-
tional.

Silent message and progress bars (if any) with options("verbose"=FALSE).

See Also
Other import functions: import_Conte(), import_StereoMorph_curve1(), import_jpg1(), import_tps(),
import_txt(), pix2chc()

Examples
## Not run:
if <- list.files('/quotesingle.Var/foo/jpegs', full.names=TRUE)
coo <- import_jpg(if)
Out(coo)

coo <- import_jpg()

## End(Not run)

import_jpg1

Extract outline coordinates from a single .jpg file

Description
Used to import outline coordinates from .jpg files. This function is used for single images and is
wrapped by import_jpg. It relies itself on import_Conte
Usage

import_jpg1(
    jpg.path,
    auto.notcentered = TRUE,
    fun.notcentered = NULL,
    threshold = 0.5,
    ...
)

Arguments

jpg.path vector of paths corresponding to the .jpg files to import, such as those obtained with list.files.
auto.notcentered logical if TRUE random locations will be used until one of them is (assumed) to be within the shape (because it corresponds to a black pixel) and only if the middle point is not black; if FALSE a locator will be called, and you will have to click on a point within the shape.
fun.notcentered NULL by default but can accept a function that, when passed with an image-matrix and returns a numeric of length two that corresponds to a starting point on the image matrix for the Conte algorithm. A while instruction wraps it, so the function may be wrong in proposing this starting position. See the examples below for a quick example.
threshold the threshold value use to binarize the images. Above, pixels are turned to 1, below to 0.
... arguments to be passed to read.table, eg. 'skip', 'dec', etc.

Details

jpegs can be provided either as RVB or as 8-bit greylevels or monochrome. The function binarizes pixels values using the 'threshold' argument. It will try to start to apply the import_Conte algorithm from the center of the image and 'looking' downwards for the first black/white 'frontier' in the pixels. This point will be the first of the outlines. The latter may be useful if you align manually the images and if you want to retain this information in the consequent morphometric analyses.

If the point at the center of the image is not within the shape, i.e. is 'white' you have two choices defined by the 'auto.notcentered' argument. If it's TRUE, some random starting points will be tried until on of them is 'black' and within the shape; if FALSE you will be asked to click on a point within the shape.

If some pixels on the borders are not white, this functions adds a 2-pixel border of white pixels; otherwise import_Conte would fail and return an error.

Finally, remember that if the images are not in your working directory, list.files must be called with the argument full.names=TRUE!

Note that the use of the fun.notcentered argument will probably leads to serious headaches and will probably imply the dissection of these functions: import_Conte, img_plot and import_jpg itself.
import_StereoMorph_curve1

Value

a matrix of (x; y) coordinates that can be passed to Out

Note

Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.

See Also

import_jpg, import_Conte, import_txt, lf_structure. See also Momocs' vignettes for data import.
Other import functions: import_Conte(), import_StereoMorph_curve1(), import_jpg(), import_tps(), import_txt(), pix2 chc()

---

import_StereoMorph_curve1

*Import files creates by StereoMorph into Momocs*

Description

Helps to read .txt files created by StereoMorph into (x; y) coordinates or Momocs objects. Can be applied to 'curves' or 'ldk' text files.

Usage

import_StereoMorph_curve1(path)
import_StereoMorph_curve(path, names)
import_StereoMorph_ldk1(path)
import_StereoMorph_ldk(path, names)

Arguments

path toward a single file or a folder containing .txt files produced by StereoMorph
names to feed lf_structure

Details

*1 functions import a single .txt file. Their counterpart (no '1') work when path indicates the folder, i.e. 'curves' or 'ldk'. They then return a list of Opn or Ldk objects, respectively. Please do not hesitate to contact me should you have a particular case or need something.

Note

Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.
import_tps

See Also

Other import functions: import_Conte(), import_jpg1(), import_jpg(), import_tps(), import_txt(), pix2chc()
Other import functions: import_Conte(), import_jpg1(), import_jpg(), import_tps(), import_txt(), pix2chc()
Other import functions: import_Conte(), import_jpg1(), import_jpg(), import_tps(), import_txt(), pix2chc()
Other import functions: import_Conte(), import_jpg1(), import_jpg(), import_tps(), import_txt(), pix2chc()

---

import_tps Import a tps file

Description

And returns a list of coordinates, curves, scale

Usage

import_tps(tps.path, curves = TRUE)

    tps2coo(tps, curves = TRUE)

Arguments

    tps.path   lines, typically from readLines, describing a single shape in tps-like format. You
            will need to manually build your Coo object from it: eg Out(coo=your_list$coo).
    curves     logical whether to read curves, if any
    tps        lines for a single tps file tps2coo is used in import_tps and may be useful for
data import. When provided with lines (eg after readLines) from a tps-like description (with "LM", "CURVES", etc.) returns a list of coordinates, curves, etc.

Value

    a list with components: coo a matrix of coordinates; cur a list of matrices; scale the scale as a numeric.

Note

    Note this function will be deprecated from Momocs when Momacs and Momit will be fully operationnal.
import_txt

See Also

Other import functions: import_Conte(), import_StereoMorph_curve1(), import_jpg1(), import_jpg(), import_txt(), pix2chc()

Other import functions: import_Conte(), import_StereoMorph_curve1(), import_jpg1(), import_jpg(), import_txt(), pix2chc()

import_txt

Import coordinates from a .txt file

Description

A wrapper around read.table that can be used to import outline/landmark coordinates.

Usage

import_txt(txt.paths = .lf.auto(), ...)

Arguments

txt.paths a vector of paths corresponding to the .txt files to import. If not provided (or NULL), switches to the automatic version, just as in import_jpg. See Details there.

... arguments to be passed to read.table, eg. 'skip', 'dec', etc.

Details

Columns are not named in the .txt files. You can tune this using the ... argument. Define the read.table arguments that allow to import a single file, and then pass them to this function, ie if your .txt file has a header (eg ('x', 'y')), do not forget header=TRUE.

Value

a list of matrix(ces) of (x; y) coordinates that can be passed to Out, Opn and Ldk.

Note

Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.

Silent message and progress bars (if any) with options("verbose"=FALSE).

See Also

Other import functions: import_Conte(), import_StereoMorph_curve1(), import_jpg1(), import_jpg(), import_tps(), pix2chc()
**inspect**

**Graphical inspection of shapes**

**Description**

Allows to plot shapes, individually, for Coo (Out, Opn or Ldk) objects.

**Usage**

```r
inspect(x, id, ...)
```

**Arguments**

- `x` the Coo object
- `id` the id of the shape to plot, if not provided a random shape is plotted. If passed with 'all' all shapes are plotted, one by one.
- `...` further arguments to be passed to `coo_plot`

**See Also**

Other Coo_graphics: `panel()`, `stack()`

**Examples**

```r
## Not run:
inspect(bot, 5)
inspect(bot)
inspect(bot, 5, pch=3, points=TRUE) # an example of '...' use

## End(Not run)
```

**is**

**Class and component testers**

**Description**

Class testers test if any of the classes of an object *is* of a given class. For instance `is_PCA` on a PCA object (of classes PCA and prcomp) will return TRUE. Component testers check if *there_is* a particular component (eg $fac, etc.) in an object.
Usage

is_Coo(x)
is_PCA(x)
is_LDA(x)
is_Out(x)
is_Opn(x)
is_Ldk(x)
is_Coe(x)
is_OutCoe(x)
is_OpnCoe(x)
is_LdkCoe(x)
is_TraCoe(x)
is_shp(x)
is_fac(x)
is_ldk(x)
is_slidings(x)
is_links(x)

Arguments

x the object to test

Value

logical

Examples

is_Coo(bot)
is_Out(bot)
is_Ldk(bot)
is_ldk(hearts) # mind the capitals!
is_equallyspacedradii  Tests if coordinates likely have equally spaced radii

Description

Returns TRUE/FALSE whether the sd of angles between all successive radii is below/above thesh

Usage

is_equallyspacedradii(coo, thres)

Arguments

coo     matrix of (x; y) coordinates or any Coo object.
thres   numeric a threshold (arbitrarily \(\pi/90\), eg 2 degrees, by default)

Value

a single or a vector of logical. If NA are returned, some coordinates are likely identical, at least for x or y.

See Also

Other coo_ utilities: coo_aligncalliper(), coo_alignminradius(), coo_alignxax(), coo_align(), coo_baseline(), coo_bookstein(), coo_boundingbox(), coo_calliper(), coo_centdist(), coo_center(), coo_centpos(), coo_close(), coo_down(), coo_dxy(), coo_extract(), coo_flipx(), coo_force2close(), coo_interpolate(), coo_is_closed(), coo_jitter(), coo_left(), coo_likely_clockwise(), coo_nb(), coo_perim(), coo_range(), coo_rev(), coo_right(), coo_rotatecenter(), coo_rotate(), coo_sample_prop(), coo_samplerr(), coo_sample(), coo_scale(), coo_shearx(), coo_slice(), coo_slidedirection(), coo_slidegap(), coo_slide(), coo_smoothcurve(), coo_smooth(), coo_template(), coo_trans(), coo_trimbottom(), coo_trimtop(), coo_trim(), coo_untiltx(), coo_up()

Examples

bot[1] %>% is_equallyspacedradii
bot[1] %>% coo_samplerr(36) %>% is_equallyspacedradii
# higher tolerance but wrong
bot[1] %>% coo_samplerr(36) %>% is_equallyspacedradii(thres=5*2*pi/360)
# coo_interpolate is a better option
bot[1] %>% coo_interpolate(1200) %>% coo_samplerr(36) %>% is_equallyspacedradii
# Coo method
bot %>% coo_interpolate(360) %>% coo_samplerr(36) %>% is_equallyspacedradii
KMEANS

KMEANS on PCA objects

Description

A very basic implementation of k-means. Beware that morphospaces are calculated so far for the 1st and 2nd component.

Usage

KMEANS(x, ...)

## S3 method for class 'PCA'
KMEANS(x, centers, nax = 1:2, pch = 20, cex = 0.5, ...)

Arguments

x PCA object
... additional arguments to be passed to kmeans
centers numeric number of centers
nax numeric the range of PC components to use (1:2 by default)
pch to draw the points
cex to draw the points

Value

the same thing as kmeans

See Also

Other multivariate: CLUST(), KMEDOIDS(), LDA(), MANOVA_PW(), MANOVA(), MDS(), MSHAPES(), NMDS(), PCA(), classification_metrics()

Examples

data(bot)
bp <- PCA(efourier(bot, 10))
KMEANS(bp, 2)
Description

A basic implementation of kmedoids on top of cluster::pam. Beware that morphospaces are calculated so far for the 1st and 2nd component.

Usage

KMEDOIDS(x, k, metric = "euclidean", ...)

## Default S3 method:
KMEDOIDS(x, k, metric = "euclidean", ...)

## S3 method for class 'Coe'
KMEDOIDS(x, k, metric = "euclidean", ...)

## S3 method for class 'PCA'
KMEDOIDS(x, k, metric = "euclidean", retain, ...)

Arguments

x a Coe or PCA object

k numeric number of centers

metric one of euclidean (default) or manhattan, to feed cluster::pam

... additional arguments to feed cluster::pam

retain when passing a PCA how many PCs to retain, or a proportion of total variance, see LDA

Value

see cluster::pam. Other components are returned (fac, etc.)

See Also

Other multivariate: CLUST(), KMEANS(), LDA(), MANOVA_P(), MANOVA(), MDS(), MSHAPES(), NMDS(), PCA(), classification_metrics()

Examples

data(bot)
bp <- PCA(efourier(bot, 10))
KMEANS(bp, 2)

set.seed(123) # for reproducibility on a dummy matrix
matrix(rnorm(100, 10, 10)) %>%
KMEDOIDS(5)

# On a Coe
bot_f <- bot %>% efourier()

bot_k <- bot_f %>% KMEDOIDS(2)
# confusion matrix
table(bot_k$fac$type, bot_k$clustering)

# on a PCA
bot_k2 <- bot_f %>% PCA() %>% KMEDOIDS(12, retain=0.9)

# confusion matrix
with(bot_k, table(fac$type, clustering))
# silhouette plot
bot_k %>% plot_silhouette()

# average width as a function of k
k_range <- 2:12
widths <- sapply(k_range, function(k) KMEDOIDS(bot_f, k=k)$silinfo$avg.width)
plot(k_range, widths, type="b")

---

layers

Grindr layers for multivariate plots

Description

Useful layers for building custom multivariate plots using the cheapbabi approach. See examples.

Usage

layer_frame(x, center_origin = TRUE, zoom = 0.9)

layer_axes(x, col = "#999999", lwd = 1/2, ...)

layer_ticks(x, col = "#333333", cex = 3/4, lwd = 3/4, ...)

layer_grid(x, col = "#999999", lty = 3, grid = 3, ...)

layer_box(x, border = "#e5e5e5", ...)

layer_fullframe(x, ...)

layer_points(x, pch = 20, cex = 4/log1p(nrow(x$xy)), transp = 0, ...)

layer_ellipses(x, conf = 0.5, lwd = 1, alpha = 0, ...)

layer_ellipsesfilled(x, conf = 0.5, lwd = 1, alpha = 0, ...)
layer_ellipsesaxes(x, conf = 0.5, lwd = 1, alpha = 0, ...)

layer_chull(x, ...)

layer_chullfilled(x, alpha = 0.8, ...)

layer_stars(x, alpha = 0.5, ...)

layer_delaunay(x, ...)

layer_density(
  x,
  levels_density = 20,
  levels_contour = 4,
  alpha = 1/3,
  n = 200,
  density = TRUE,
  contour = TRUE
)

layer_labelpoints(
  x,
  col = par("fg"),
  cex = 2/3,
  font = 1,
  abbreviate = FALSE,
  ...
)

layer_labelgroups(
  x,
  col = par("fg"),
  cex = 3/4,
  font = 2,
  rect = TRUE,
  alpha = 1/4,
  abbreviate = FALSE,
  ...
)

layer_rug(x, size = 1/200, ...)

layer_histogram_2(x, freq = FALSE, breaks, split = FALSE, transp = 0)

layer_density_2(x, bw, split = FALSE, rug = TRUE, transp = 0)

layer_title(x, title = "", cex = 3/4, ...)
layer_axesnames(x, cex = 3/4, name = "Axis", ...)

layer_eigen(x, nb_max = 5, cex = 1/2, ...)

layer_axesvar(x, cex = 3/4, ...)

layer_legend(x, probs = seq(0, 1, 0.25), cex = 3/4, ...)

Arguments

x a list, typically returned by plot_PCA
center_origin logical whether to center the origin (default TRUE)
zoom numeric to change the zoom (default 0.9)
col color (hexadecimal) to use for drawing components
lwd linewidth for drawing components
... additional options to feed core functions for each layer
cex to use for drawing components
lty linetype for drawing components
grid numeric number of grid to draw
border color (hexadecimal) to use to draw border
pch to use for drawing components
transp transparency to use (min: 0 default:0 max:1)
conf numeric between 0 and 1 for confidence ellipses
alpha numeric between 0 and 1 for the transparency of components
levels_density numeric number of levels to use to feed MASS::kde2d
levels_contour numeric number of levels to use to feed graphics::contour
n numeric number of grid points to feed MASS::kde2d
density logical whether to draw density estimate
contour logical whether to draw contour lines
font to feed text
abbreviate logical whether to abbreviate names
rect logical whether to draw a rectangle below names
size numeric as a fraction of graphical window (default: 1/200)
freq logical to feed hist (default:FALSE')
breaks to feed hist (default: calculated on the pooled values)
split logical whether to split the two distributions into two plots
bw to feed density (default: stats::bw.nrd0)
rug logical whether to add rug (default: TRUE)
title to add to the plot (default "")
name to use on axes (default "Axis")
nb_max numeric number of eigen values to display (default 5)
probs numeric sequence to feed stats::quantile and to indicate where to draw ticks
and legend labels
layers_morphospace

See Also
gindr_drawers

Other grindr: drawers, layers_morphospace, mosaic_engine(), papers.pile(), plot_LDA(), plot_NMDS(), plot_PCA()

layers_morphospace Morphospace layers

Description

Used internally by plot_PCA, plot_LDA, etc. but may be useful elsewhere.

Usage

layer_morphospace_PCA(
  x,
  position = c("range", "full", "circle", "xy", "range_axes", "full_axes")[1],
  nb = 12,
  nr = 6,
  nc = 5,
  rotate = 0,
  size = 0.9,
  col = "#999999",
  flipx = FALSE,
  flipy = FALSE,
  draw = TRUE
)

layer_morphospace_LDA(
  x,
  position = c("range", "full", "circle", "xy", "range_axes", "full_axes")[1],
  nb = 12,
  nr = 6,
  nc = 5,
  rotate = 0,
  size = 0.9,
  col = "#999999",
  flipx = FALSE,
  flipy = FALSE,
  draw = TRUE
)

Arguments

  x layered PCA or LDA. Typically, the object returned by plot_PCA and plot_LDA
**LDA**

Linear Discriminant Analysis on Coe objects

**Description**

Calculates a LDA on Coe on top of MASS::lda.

**Usage**

LDA(x, fac, retain, ...)

## Default S3 method:
LDA(x, fac, retain, ...)

## S3 method for class 'PCA'
LDA(x, fac, retain = 0.99, ...)

**Arguments**

x a Coe or a PCA object

fac the grouping factor (names of one of the $fac column or column id)

retain the proportion of the total variance to retain (if retain<1) using scree, or the number of PC axis (if retain>1).

... additional arguments to feed lda

---

**position**

one of range, full, circle, xy, range_axes, full_axes to feed morphospace_positions (default: range)

**nb** numeric total number of shapes when position="circle" (default: 12)

**nr** numeric number of rows to position shapes (default: 6)

**nc** numeric number of columns to position shapes (default 5)

**rotate** numeric angle (in radians) to rotate shapes when displayed on the morphospace (default: 0)

**size** numeric size to use to feed coo_template (default: 0.9)

**col** color to draw shapes (default: #999999)

**flipx** logical whether to flip shapes against the x-axis (default: FALSE)

**flipy** logical whether to flip shapes against the y-axis (default: FALSE)

**draw** logical whether to draw shapes (default: TRUE)

**See Also**

Other grindr: drawers, layers, mosaic_engine(), papers, pile(), plot_LDA(), plot_NMDS(), plot_PCA()

Other grindr: drawers, layers, mosaic_engine(), papers, pile(), plot_LDA(), plot_NMDS(), plot_PCA()
**Value**

a 'LDA' object on which to apply `plot.LDA`, which is a list with components:

- `x` any `Coe` object (or a matrix)
- `fac` grouping factor used
- `removed` ids of columns in the original matrix that have been removed since constant (if any)
- `mod` the raw lda mod from `lda`
- `mod.pred` the predicted model using `x` and `mod`
- `CV.fac` cross-validated classification
- `CV.tab` cross-validation table
- `CV.correct` proportion of correctly classified individuals
- `CV.ce` class error
- `LDs` unstandardized LD scores see Claude (2008)
- `mshape` mean values of coefficients in the original matrix
- `method` inherited from the `Coe` object (if any)

**Note**

For LDA.PCA, retain can be passed as a vector (eg: 1:5, and retain=1, retain=2, ..., retain=5) will be tried, or as "best" (same as before but retain=1:number_of_pc_axes is used).

Silent message and progress bars (if any) with `options("verbose"=FALSE)`.

**See Also**

Other multivariate: `CLUST()`, `KMEANS()`, `KMEDOIDS()`, `MANOVA_PW()`, `MANOVA()`, `MDS()`, `MShapes()`, `NMDS()`, `PCA()`, `classification_metrics()`

**Examples**

```r
bot.f <- efourier(bot, 24)
bot.p <- PCA(bot.f)
LDA(bot.p, 'type', retain=0.99) # retains 0.99 of the total variance
LDA(bot.p, 'type', retain=5) # retain 5 axis
bot.l <- LDA(bot.p, 'type', retain=0.99)
plot.LDA(bot.l)
bot.f <- mutate(bot.f, plop=factor(rep(letters[1:4], each=10)))
bot.l <- LDA(PCA(bot.f), 'plop')
plot.LDA(bot.l) # will replace the former soon
```
**Ldk**  
Builds an Ldk object

**Description**

In Momocs, Ldk classes objects are lists of configurations of landmarks, with optional components, on which generic methods such as plotting methods (e.g. `stack`) and specific methods (e.g. `fgProcrustes`). Ldk objects are primarily Coo objects. In a sense, morphometrics methods on Ldk objects preserves (x, y) coordinates and LdkCoe are also Ldk objects.

**Usage**

```r
Ldk(coo, fac = dplyr::data_frame(), links = NULL, slidings = NULL)
```

**Arguments**

- `coo`: a list of matrices of (x; y) coordinates, or an array, or an Ldk object or a data.frame (and friends)
- `fac` (optional): a data.frame of factors and/or numerics specifying the grouping structure
- `links` (optional): a 2-columns matrix of 'links' between landmarks, mainly for plotting
- `slidings` (optional): a 3-columns matrix defining (if any) sliding landmarks

**Details**

All the shapes in x must have the same number of landmarks. If you are trying to make an Ldk object from an Out or an Opn object, try `coo_sample` beforehand to homogenize the number of coordinates among shapes. Please note that Ldk methods are as experimental.

Implementation of `$slidings` is inspired by geomorph

**Value**

an Ldk object

**See Also**

Other classes: Coe(), Coo(), OpnCoe(), Opn(), OutCoe(), Out(), TraCoe()

**Examples**

```r
#Methods on Ldk
methods(class=Ldk)

str(mosquito)
```
ldk_check  Checks 'ldk' shapes

Description
A simple utility, used internally, mostly by Ldk methods, in some graphical functions, and notably in l2a. Returns an array of landmarks arranged as (nb.1dk) x (x; y) x (nb. shapes), when passed with either a list, a matrix or an array of coordinates. If a list is provided, checks that the number of landmarks is consistent.

Usage
ldk_check(ldk)

Arguments
ldk  a matrix of (x; y) coordinates, a list, or an array.

Value
an array of (x; y) coordinates.

See Also
Other ldk helpers: def_links(), links_all(), links_delaunay()

Examples
#coo_check('Not a shape')
#coo_check(matrix(1:10, ncol=2))
#coo_check(list(x=1:5, y=6:10))

ldk_chull  Draws convex hulls around landmark positions

Description
A wrapper that uses coo_chull

Usage
ldk_chull(ldk, col = "grey40", lty = 1)
ldk_confell

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldk</td>
<td>an array (or a list) of landmarks</td>
</tr>
<tr>
<td>col</td>
<td>a color for drawing the convex hull</td>
</tr>
<tr>
<td>lty</td>
<td>an lty for drawing the convex hulls</td>
</tr>
</tbody>
</table>

See Also

coo_chull, chull, ldk_confell, ldk_contour

Other plotting functions: coo_arrows(), coo_draw(), coo_listpanel(), coo_lolli(), coo_plot(), coo_ruban(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_devsegments(), plot_table()

Other ldk plotters: ldk_confell(), ldk_contour(), ldk_labels(), ldk_links()

Examples

```r
coo_plot(MSHAPES(wings))
ldk_chull(wings$coo)
```

---

ldk_confell  Draws confidence ellipses for landmark positions

Description

Draws confidence ellipses for landmark positions

Usage

```r
ldk_confell(
  ldk,
  conf = 0.5,
  col = "grey40",
  ell.lty = 1,
  ax = TRUE,
  ax.lty = 2
)
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldk</td>
<td>an array (or a list) of landmarks</td>
</tr>
<tr>
<td>conf</td>
<td>the confidence level (normal quantile, 0.5 by default)</td>
</tr>
<tr>
<td>col</td>
<td>the color for the ellipse</td>
</tr>
<tr>
<td>ell.lty</td>
<td>an lty for the ellipse</td>
</tr>
<tr>
<td>ax</td>
<td>logical whether to draw ellipses axes</td>
</tr>
<tr>
<td>ax.lty</td>
<td>an lty for ellipses axes</td>
</tr>
</tbody>
</table>
ldk_contour

See Also

Other plotting functions: `coo_arrows()`, `coo_draw()`, `coo_listpanel()`, `coo_lolli()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_contour()`, `ldk_labels()`, `ldk_links()`, `plot_devsegments()`, `plot_table()`

Other ldk plotters: `ldk_chull()`, `ldk_contour()`, `ldk_labels()`, `ldk_links()

Examples

c oo_plot(MSHAPES(wings))
ldk_confell(wings$coo)

| ldk_contour | Draws kernel density contours around landmark |

Description

Using `kde2d` in the MASS package.

Usage

```r
ldk_contour(ldk, nlevels = 5, grid.nb = 50, col = "grey60")
```

Arguments

- **ldk**: an array (or a list) of landmarks
- **nlevels**: the number of contour lines
- **grid.nb**: the grid.nb
- **col**: a color for drawing the contour lines

See Also

- `kde2d`, `ldk_confell`, `ldk_chull`

Other plotting functions: `coo_arrows()`, `coo_draw()`, `coo_listpanel()`, `coo_lolli()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_confell()`, `ldk_labels()`, `ldk_links()`, `plot_devsegments()`, `plot_table()`

Other ldk plotters: `ldk_chull()`, `ldk_confell()`, `ldk_labels()`, `ldk_links()

Examples

```r
c oo_plot(MSHAPES(wings))
ldk_contour(wings$coo)
```
ldk_labels

Add landmarks labels

Description

Add landmarks labels

Usage

```r
ldk_labels(ldk, d = 0.05, cex = 2/3, ...)
```

Arguments

- `ldk`: a matrix of (x; y) coordinates: where to plot the labels
- `d`: how far from the coordinates, on a (centroid-landmark) segment
- `cex`: the cex for the label
- `...`: additional parameters to fed text

See Also

Other plotting functions: `coo_arrows()`, `coo_draw()`, `coo_listpanel()`, `coo_lolli()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_links()`, `plot_devsegments()`, `plot_table()`

Other ldk plotters: `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_links()

Examples

```r
coo_plot(wings[1])
ldk_labels(wings[1])
# closer and smaller
coo_plot(wings[1])
ldk_labels(wings[1], d=0.05, cex=0.5)
```

ldk_links

Draws links between landmarks

Description

Cosmetics only but useful to visualize shape variation.

Usage

```r
ldk_links(ldk, links, ...)
```
**lf_structure**

**Arguments**

- **ldk**
  a matrix of (x; y) coordinates
- **links**
  a matrix of links. On the first column the starting-id, on the second column the ending-id (id= the number of the coordinate)
- ...
  additional parameters to fed segments

**See Also**

Other plotting functions: `coo_arrows()`, `coo_draw()`, `coo_listpanel()`, `coo_lolli()`, `coo_plot()`, `coo_ruban()`, `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_labels()`, `plot_devsegments()`, `plot_table()`

Other ldk plotters: `ldk_chull()`, `ldk_confell()`, `ldk_contour()`, `ldk_labels()`

---

**lf_structure**

*bind_db.Coe <- bind_db.Coo Extracts structure from filenames*

**Description**

If filenames are consistently named with the same character separating factors, and with every individual including its belonging levels, e.g.:

- 001_speciesI_siteA_ind1_dorsalview
- 002_speciesI_siteA_ind2_lateralview

etc., this function returns a `data.frame` from it that can be passed to `Out`, `Opn`, `Ldk` objects.

**Usage**

```r
lf_structure(lf, names = character(), split = "_", trim.extension = FALSE)
```

**Arguments**

- **lf**
  a list (its names are used, except if it is a list from `import_tps` in this case names(lf$coo) is used) of a list of filenames, as characters, typically such as those obtained with `list.files`. Alternatively, a path to a folder containing the files. Actually, if lf is of length 1 (a single character), the function assumes it is a path and do a `list.files` on it.
- **names**
  the names of the groups, as a vector of characters which length corresponds to the number of groups.
- **split**
  character, the splitting factor used for the file names.
- **trim.extension**
  logical. Whether to remove the last for characters in filenames, typically their extension, e.g. `'.jpg'`.

**Details**

The number of groups must be consistent across filenames.
Value
data.frame with, for every individual, the corresponding level for every group.

Note
This is, to my view, a good practice to 'store' the grouping structure in filenames, but it is of course not mandatory.
Note also that you can: i) do a import_jpg and save is a list, say 'foo'; then ii) pass 'names(foo)' to lf_structure. See Momocs' vignette for an illustration.
Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.

See Also
import_jpg1, import_Conte, import_txt, lf_structure. See also Momocs' vignettes for data import.
Other babel functions: tie_jpg_txt()

-------

links_all

Description
Creates links (all pairwise combinations) between landmarks

Usage
links_all(coo)

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

Value
a matrix that can be passed to ldk_links, etc. The columns are the row ids of the original shape.

See Also
Other ldk helpers: def_links(), ldk_check(), links_delaunay()

Examples
w <- wings[1]
coo_plot(w)
links <- links_all(w)
ldk_links(w, links)
links_delaunay

*Description*

Creates links (Delaunay triangulation) between landmarks

*Usage*

```r
links_delaunay(coo)
```

*Arguments*

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

*Details*

uses `delaunayn` in the `geometry` package.

*Value*

a matrix that can be passed to `ldk_links`, etc. The columns are the row ids of the original shape.

*See Also*

Other ldk helpers: `def_links()`, `ldk_check()`, `links_all()`

*Examples*

```r
w <- wings[1]
coo_plot(w, poly=FALSE)
links <- links_delaunay(w)
ldk_links(w, links)
```

---

**MANOVA**

*Multivariate analysis of (co)variance on Coe objects*

*Description*

Performs multivariate analysis of variance on PCA objects.
Usage

MANOVA(x, fac, test = "Hotelling", retain, drop)

## S3 method for class 'OpnCoe'
MANOVA(x, fac, test = "Hotelling", retain, drop)

## S3 method for class 'OutCoe'
MANOVA(x, fac, test = "Hotelling", retain, drop)

## S3 method for class 'PCA'
MANOVA(x, fac, test = "Hotelling", retain = 0.99, drop)

Arguments

x a Coe object
fac a name of a column in the $fac slot, or its id, or a formula
test a test for manova ('Hotelling' by default)
retain how many harmonics (or polynomials) to retain, for PCA the highest number of
PC axis to retain, or the proportion of the variance to capture.
drop how many harmonics (or polynomials) to drop

Details

Performs a MANOVA/MANCOVA on PC scores. Just a wrapper around manova. See examples for
multifactorial manova and summary.manova for more details and examples.

Value

a list of matrices of (x,y) coordinates.

Note

Needs a review and should be considered as experimental. Silent message and progress bars (if any)
with options("verbose"=FALSE).

See Also

Other multivariate: CLUST(), KMEANS(), KMEDOIDS(), LDA(), MANOVA_PW(), MDS(), MSHAPES(),
NMDS(), PCA(), classification_metrics()

Examples

# MANOVA
bot.p <- PCA(efourier(bot, 12))
MANOVA(bot.p, 'type')

op <- PCA(npoly(olea, 5))
MANOVA(op, 'domes')
m <- manova(op$x[, 1:5] ~ op$fac$domes * op$fac$var)
summary(m)
summary.aov(m)

# MANCOVA example
# we create a numeric variable, based on centroid size
bot %<>% mutate(cs=coo_centsize(.)
# same pipe
bot %>% efourier %>% PCA %>% MANOVA("cs")

---

**MANOVA_PW**

*Pairwise Multivariate analyses of variance*

**Description**

A wrapper for pairwise MANOVAs on Coe objects. Calculates a MANOVA for every pairwise combination of the factor provided.

**Usage**

`MANOVA_PW(x, ...)`

```r
# S3 method for class 'PCA'
MANOVA_PW(x, fac, retain = 0.99, ...)
```

**Arguments**

- `x`: a PCA object
- `...`: more arguments to feed MANOVA
- `fac`: a name (or its id) of a grouping factor in $fac or a factor or a formula.
- `retain`: the number of PC axis to retain (1:retain) or the proportion of variance to capture (0.99 par default).

**Value**

A list with the following components is returned (invisibly because $manovas may be very long, see examples):

- `manovas` a list containing all the raw manovas
- `summary`
- `stars.tab` a table with 'significance stars', discutable but largely used: ’’ if Pr(F) < 0.001; ’’ of < 0.01; ’’ if < 0.05; ’’ if < 0.10 and ’’ if above.

**Note**

Needs a review and should be considered as experimental. If the fac passed has only two levels, there is only pair and it is equivalent to MANOVA. MANOVA_PW.PCA works with the regular manova.
MDS

(Metric) multidimensional scaling

Description

A wrapper around stats::cmdscale.

Usage

MDS(x, method = "euclidean", k = 2, ...)

Arguments

x any Coe object
method a dissiminarity index to feed method in stats::dist (default: euclidean)
k numeric number of dimensions to feed stats::cmdscale (default: 2)
... additional parameters to feed stats::cmdscale

Details

For Details, see vegan::metaMDS

Value

what is returned by stats::dist plus $fac. And prepend MDS class to it.

See Also

MANOVA, manova.

Other multivariate: CLUST(), KMEANS(), KMEDOIDS(), LDA(), MANOVA(), MDS(), MSHAPES(), NMDS(), PCA(), classification_metrics

Examples

# we create a fake factor with 4 levels
bot$fac$fake <- factor(rep(letters[1:4], each=10))
bot.p <- PCA(efourier(bot, 8))
MANOVA_PW(bot.p, 'fake') # or MANOVA_PW(bot.p, 2)

# an example on open outlines
op <- PCA(npoly(olea))
MANOVA_PW(op, 'domes')
# to get the results
res <- MANOVA_PW(op, 'domes')
res$manovas
res$stars.tab
res$summary
measure

**See Also**

Other multivariate: `CLUST()`, `KMEANS()`, `KMEANSODS()`, `LDA()`, `MANOVA_PW()`, `MANOVA()`, `MSHAPES()`, `NMDS()`, `PCA()`, `classification_metrics()`

**Examples**

```r
x <- bot %>% efourier %>% MDS
x
```

<table>
<thead>
<tr>
<th>measure</th>
<th>Measures shape descriptors</th>
</tr>
</thead>
</table>

**Description**

Calculates shape descriptors on Coo and other objects. Any function that returns a scalar when fed coordinates can be passed and naturally those of Momocs (pick some there `apropos("coo_")`). Functions without arguments (eg `coo_area`) have to be passed without brackets but functions with arguments (eg `d`) have to be passed "entirely". See examples.

**Usage**

```r
measure(x, ...)
```

**Arguments**

- `x`: any Coo object, or a list of shapes, or a shape as a matrix.
- `...`: a list of functions. See examples.

**Value**

a `TraCoe` object, or a raw data.frame

**See Also**

Other premodern: `coo_truss()`

**Examples**

```r
bm <- measure(bot, coo_area, coo_perim)
bm
bm$coe
```

# how to use arguments, eg with the d() function
```r
measure(wings, coo_area, d(1, 3), d(4, 5))
```

# alternatively, to get a data_frame
molars  

**Description**

Courtesy of Julien Corny and Florent Detroit.

**Format**

A `Out` object containing 79 equi-linearly spaced (x; y) coordinates for 360 crown outlines, of modern human molars, along with their type (`$type`) - 90 first upper molars (UM1), 90 second upper molars (UM2), 90 first lower molars (LM1), 90 second lower molars (LM2) - and the individual (`ind`) they come from (the data of the 360 molars are taken from 180 individuals).

**Source**


**See Also**

Other datasets: `apodemus`, `bot`, `chaff`, `charring`, `flower`, `hearts`, `mosquito`, `mouse`, `nsfishes`, `oak`, `olea`, `shapes`, `trilo`, `wings`

---

**Description**

The goal of Momocs is to provide a complete, convenient, reproducible and open-source toolkit for 2D morphometrics. It includes most common 2D morphometrics approaches on outlines, open outlines, configurations of landmarks, traditional morphometrics, and facilities for data preparation, manipulation and visualization with a consistent grammar throughout. It allows reproducible, complex morphometric analyses and other morphometrics approaches should be easy to plug in, or develop from, on top of this canvas.

**Details**

To cite Momocs in publications: `citation("Momocs")`. 

```r
measure(bot$coo, coo_area, coo_perim)

# and also, to get a data_frame (one row)
measure(bot[1], coo_area, coo_perim)
```
Cheers

We are very grateful to (in alphabetical order): Sean Asselin, Laurent Bouby, Matt Bulbert, Simon Crameri, Julia Cooke, April Dinwiddie, Carl Lipo, Cedric Gaucherel, Catherine Girard, QGouil (GitHub), Christian Steven Hoggard, Sarah Ivorra, Glynis Jones, Nathalie Keller, Ricardo Kriebel, Remi Laffont, Fabien Lafuma, Matthias Mace, Stas Malavin, Neus Martinez, Ben Marwick, Sabrina Renaud, Marcelo Reginato, Evan Saitta, David Siddons, Eleanor Stillman, Theodore Stammer, Tom Stubbs, Norbert Telmon, Jean-Frederic Terral, Bill Venables, Daniele Ventura, Michael Wallace, Asher Wishkerman, John Wood for their helpful ideas and bug reports.

References


See Also

- Homepage: https://github.com/MomX/Momocs
- Issues: https://github.com/MomX/Momocs/issues
- Tutorial: browseVignettes("Momocs") or http://momx.github.io/Momocs/
- Email: bonhomme.vincent@gmail.com to contribute to dev, ask for something, propose collaboration, share your data, etc.

morphospace_positions  Calculates nice positions on a plane for drawing shapes

Description

Calculates nice positions on a plane for drawing shapes

Usage

```r
morphospace_positions(
  xy,
  pos.shp = c("range", "full", "circle", "xy", "range_axes", "full_axes")[1],
  nb.shp = 12,
  nr.shp = 6,
  nc.shp = 5,
  circle.r.shp
)
```
mosaic_engine

Arguments

xy a matrix of points typically from a PCA or other multivariate method on which morphospace can be calculated

pos.shp how shapes should be positioned: range of xy, full extent of the plane, circle as a rosewind, on xy values provided, range_axes on the range of xy but on the axes, full_axes same thing but on (0.85) range of the axes. You can also directly pass a matrix (or a data.frame) with columns named ("x","y").

nb.shp the total number of shapes

nr.shp the number of rows to position shapes

nc.shp the number of cols to position shapes

circle.r.shp if circle, its radius

Details

See plot.PCA for self-speaking examples

mosaic_engine PLOTS MOSAICS OF SHAPES.

Description

Will soon replace panel. See examples and vignettes.

Usage

```r
mosaic_engine(
  coo_list,
  dim,
  asp = 1,
  byrow = TRUE,
  fromtop = TRUE,
  sample = 60,
  relatively = FALSE,
  template_size = 0.92
)
```

```r
mosaic(x, ...)
```

```r
## S3 method for class 'Out'
mosaic(
  x,
  f,
  relatively = FALSE,
  pal = pal_qual,
  sample = 60,
)```
paper_fun = paper_white,
draw_fun = draw_outlines,
legend = TRUE,
dim = NA,
asp = 1,
byrow = TRUE,
fromtop = TRUE,
...
)

## S3 method for class 'Opn'
mosaic(
  x,
  f,
  relatively = FALSE,
  pal = pal_qual,
  sample = 60,
  paper_fun = paper_white,
  draw_fun = draw_curves,
  legend = TRUE,
  dim = NA,
  asp = 1,
  byrow = TRUE,
  fromtop = TRUE,
  ...
)

## S3 method for class 'Ldk'
mosaic(
  x,
  f,
  relatively = FALSE,
  pal = pal_qual,
  sample = 60,
  paper_fun = paper_white,
  draw_fun = draw_landmarks,
  legend = TRUE,
  dim = NA,
  asp = 1,
  byrow = TRUE,
  fromtop = TRUE,
  ...
)

Arguments

- coo_list: list of shapes
- dim: numeric of length 2, the desired dimensions for rows and columns
mosaic_engine

asp numeric the yx ratio used to calculate dim (1 by default).
byrow logical whether to order shapes by rows
fromtop logical whether to order shapes from top
sample numeric number of points to coo_sample
relatively logical if TRUE use coo_template_relatively or, if FALSE(by default) coo_template. In other words, whether to preserve size or not.
template_size numeric to feed coo_template(_relatively). Only useful to add padding around shapes when the default value (0.95) is lowered.
x any Coo object
... additional arguments to feed the main drawer if the number of shapes is > 1000 (default: 64). If non-numeric (eg FALSE) do not sample.
f factor specification to feed fac_dispatcher
pal one of palettes
paper_fun a papers function (default: paper)
draw_fun one of drawers for pile.list
legend logical whether to draw a legend (will be improved in further versions)

Value
a list of templated and translated shapes

See Also
Other grindr: drawers, layers_morphospace, layers, papers, pile(), plot_LDA(), plot_NMDS(), plot_PCA()

Examples

# On Out ---
bot %>% mosaic
bot %>% mosaic(~type)

# As with other grindr functions you can continue the pipe
bot %>% mosaic(~type, asp=0.5) %>% draw_firstpoint

# On Opn ---- same grammar
olea %>% mosaic(~view+var, paper_fun=paper_dots)

# On Ldk
mosaic(wings, ~group, pal=pal_qual_Dark2, pch=3)

# On Out with different sizes
# would work on other Coo too
shapes2 <- shapes
sizes <- runif(30, 1, 2)
shapes2 %>% mosaic(relatively=FALSE)
shapes2 %>% mosaic(relatively=TRUE) %>% draw_centroid()
mosquito

Data: Outline coordinates of mosquito wings.

Description

Data: Outline coordinates of mosquito wings.

Format

A Out object with the 126 mosquito wing outlines used Rohlf and Archie (1984). Note that the links defined here are quite approximate.

Source


See Also

Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mouse, nsfishes, oak, olea, shapes, trilo, wings

mouse

Data: Outline coordinates of mouse molars

Description

Data: Outline coordinates of mouse molars

Format

A Out object 64 coordinates of 30 wood molar outlines.

Source


See Also

Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, nsfishes, oak, olea, shapes, trilo, wings
Mean shape calculation for Coo, Coe, etc.

Description

Quite a versatile function that calculates mean (or median, or whatever function) on list or an array of shapes, an Ldk object. It can also be used on Coe objects. In that case, the reverse transformation (from coefficients to shapes) is calculated, (within groups defined with the fac argument if provided) and the Coe object is also returned (in $Coe) along with a list of shapes (in $shp) and can then be passed to plot_MSHAPES.

Usage

MSHAPES(x, fac = NULL, FUN = mean, nb.pts = 120, ...)

Arguments

- **x**: a list, array, Ldk, LdkCoe, OutCoe or OpnCoe or PCA object
- **fac**: factor specification for fac_dispatcher
- **FUN**: a function to compute the mean shape (mean by default, by median can be considered)
- **nb.pts**: numeric the number of points for calculated shapes (only Coe objects)
- **...**: useless here.

Value

the averaged shape; on Coe objects, a list with two components: $Coe object of the same class, and $shp a list of matrices of (x, y) coordinates. On PCA and LDA objects, the FUN (typically mean or median) of scores on PCs or LDs. This method used on the latter objects may be moved to another function at some point.

See Also

Other multivariate: CLUST(), KMEANS(), KMEDOIDS(), LDA(), MANOVA_PW(), MANOVA(), MDS(), NMDS(), PCA(), classification_metrics()

Examples

```r
### on shapes
MSHAPES(wings)
MSHAPES(wings$coo)
MSHAPES(coo_sample(bot, 24)$coo)
stack(wings)
coo_draw(MSHAPES(wings))

bot.f <- efourier(bot, 12)
MSHAPES(bot.f) # the mean (global) shape
```
mutate

```r
ms <- MSHAPES(bot.f, 'type')
ms$Coe
class(ms$Coe)
ms <- ms$shp
coo_plot(ms$beer)
coo_draw(ms$whisky, border='forestgreen')
```

---

### Description

Add new variables to the `$fac`. See examples and `?dplyr::mutate`.

### Usage

```r
mutate(.data, ...)
```

### Arguments

- `.data` a Coo, Coe, PCA object
- `...` comma separated list of unquoted expressions

### Details

dplyr verbs are maintained.

### Value

a Momocs object of the same class.

### See Also

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rm_uncomplete()`, `rw_fac()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`

### Examples

```r
olea
mutate(olea, id=factor(1:length(olea)))
```
NMDS

Non metric multidimensional scaling

Description

A wrapper around vegan::metaMDS.

Usage

NMDS(x, distance = "bray", k = 2, try = 20, trymax = 20, ...)

Arguments

- x: any Coe object
- distance: a dissimilarity index to feed vegan::vegdist (default: bray)
- k: numeric number of dimensions to feed vegan::metaMDS (default: 2)
- try: numeric minimum number of random starts to feed vegan::metaMDS (default: 20)
- trymax: numeric minimum number of random starts to feed vegan::metaMDS (default: 20)
- ...: additional parameters to feed vegan::metaMDS

Details

For Details, see vegan::metaMDS

Value

what is returned by vegan::metaMDS plus $fac. And prepend NMDS class to it.

See Also

Other multivariate: CLUST(), KMEANS(), KMEDOIDS(), LDA(), MANOVA_PW(), MANOVA(), MDS(), MSHAPES(), PCA(), classification_metrics()

Examples

x <- bot %>% efourier %>% NMDS

# Shepard diagram # before a Momocs wrapper
# vegan::stressplot(x)
Description

Calculates natural polynomial coefficients, through a linear model fit (see \texttt{lm}), from a matrix of \((x; y)\) coordinates or an \texttt{Opn} object.

Usage

\begin{verbatim}
npoly(x, 
## Default S3 method: 
npoly(x, degree, ...) 
## S3 method for class 'Opn'
npoly(
   x, 
   degree, 
   baseline1 = c(-0.5, 0), 
   baseline2 = c(0.5, 0), 
   nb.pts = 120, 
   ... 
)
## S3 method for class 'list'
npoly(x, ...)
\end{verbatim}

Arguments

\begin{itemize}
  \item \texttt{x} a matrix (or a list) of \((x; y)\) coordinates or an \texttt{Opn} object
  \item \texttt{...} useless here
  \item \texttt{degree} polynomial degree for the fit (the Intercept is also returned)
  \item \texttt{baseline1} numeric the \((x; y)\) coordinates of the first baseline by default \((x = -0.5; y = 0)\)
  \item \texttt{baseline2} numeric the \((x; y)\) coordinates of the second baseline by default \((x = 0.5; y = 0)\)
  \item \texttt{nb.pts} number of points to sample and on which to calculate polynomials
\end{itemize}

Value

when applied on a single shape, a list with components:

\begin{itemize}
  \item \texttt{coeff} the coefficients (including the intercept)
  \item \texttt{ortho} whether orthogonal or natural polynomials were fitted
  \item \texttt{degree} degree of the fit (could be retrieved through \texttt{coeff} though)
\end{itemize}
- baseline1 the first baseline point (so far the first point)
- baseline2 the second baseline point (so far the last point)
- r2 the r2 from the fit
- mod the raw lm model

otherwise, an OpnCoe object.

See Also

Other polynomials: opoly_i(), opoly()

Examples

data(olea)
o <- olea[1]
op <- opoly(o, degree=4)
op
# shape reconstruction
opi <- opoly_i(op)
coo_plot(o)
coo_draw(opi, border="red")
# R2 for degree 1 to 10
r <- numeric()
for (i in 1:10) { r[i] <- npoly(o, degree=i)$r2 }
plot(2:10, r[2:10], type='b', pch=20, col='red', main='R2 / degree')

---

nsfishes  

Data: Outline coordinates of North Sea fishes

Description

Data: Outline coordinates of North Sea fishes

Format

A Out object containing the outlines coordinates for 218 fishes from the North Sea along with taxonomical cofactors.

Source

Caillon F, Frelat R, Mollmann C, Bonhomme V (submitted)

See Also

Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, mouse, oaks, olea, shapes, trilo, wings
Description

Format
A Ldk object containing 11 (x; y) landmarks from 176 oak leaves wings, from

Source

See Also
Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, olea, shapes, trilo, wings

olea
Data: Outline coordinates of olive seeds open outlines.

Description
Data: Outline coordinates of olive seeds open outlines.

Format
An Opn object with the outline coordinates of olive seeds.

Source
We thank Jean-Frederic Terral and Sarah Ivorra (UMR CBAE, Montpellier, France) from allowing us to share the data.

See Also
Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, shapes, trilo, wings
Description

In Momocs, Opn classes objects are lists of open outlines, with optional components, on which generic methods such as plotting methods (e.g. stack) and specific methods (e.g. npoly) can be applied. Opn objects are primarily Coo objects.

Usage

Opn(x, fac = dplyr::data_frame(), ldk = list())

Arguments

- x: list of matrices of (x; y) coordinates, or an array, or a data.frame (and friends)
- fac: (optional) a data.frame of factors and/or numerics specifying the grouping structure
- ldk: (optional) list of landmarks as row number indices

Value

an Opn object

See Also

Other classes: Coe(), Coo(), Ldk(), OpnCoe(), OutCoe(), Out(), TraCoe()

Examples

```r
# Methods on Opn
methods(class=Opn)
# we load some open outlines. See ?olea for credits
olea
panel(olea)
# orthogonal polynomials
op <- opoly(olea, degree=5)
# we print the Coe
op
# Let's do a PCA on it
op.p <- PCA(op)
plot(op.p, 'domes')
plot(op.p, 'var')
# and now an LDA after a PCA
olda <- LDA(PCA(op), 'var')
# for CV table and others
olda
plot_LDA(olda)
```
OpnCoe

Builds an OpnCoe object

Description

In Momocs, OpnCoe classes objects are wrapping around lists of morphometric coefficients, along with other informations, on which generic methods such as plotting methods (e.g. `boxplot`) and specific methods can be applied. OpnCoe objects are primarily `Coe` objects.

Usage

```r
OpnCoe(
  coe = matrix(),
  fac = dplyr::data_frame(),
  method = character(),
  baseline1 = numeric(),
  baseline2 = numeric(),
  mod = list(),
  r2 = numeric()
)
```

Arguments

- `coe` matrix of morphometric coefficients
- `fac` (optionnal) a `data.frame` of factors, specifying the grouping structure
- `method` used to obtain these coefficients
- `baseline1` \((x; y)\) coordinates of the first baseline point
- `baseline2` \((x; y)\) coordinates of the second baseline point
- `mod` an R \texttt{lm} object, used to reconstruct shapes
- `r2` numeric, the r-squared from every model

Value

an OpnCoe object

See Also

Other classes: `Coe()`, `Coo()`, `Ldk()`, `Opn()`, `OutCoe()`, `Out()`, `TraCoe()`

Examples

```r
# all OpnCoe classes
methods(class='OpnCoe')
```
Calculate orthogonal polynomial fits on open outlines

Description

Calculates orthogonal polynomial coefficients, through a linear model fit (see \texttt{lm}), from a matrix of \( (x; y) \) coordinates or a \texttt{Opn} object.

Usage

\texttt{opoly}(x, \ldots)

## Default S3 method:
\texttt{opoly}(x, \texttt{degree}, \ldots)

## S3 method for class 'Opn'
\texttt{opoly}(
  x,
  \texttt{degree},
  \texttt{baseline1} = \texttt{c}(-0.5, 0),
  \texttt{baseline2} = \texttt{c}(0.5, 0),
  \texttt{nb.pts} = 120,
  \ldots
)

## S3 method for class 'list'
\texttt{opoly}(x, \ldots)

Arguments

\texttt{x} a matrix (or a list) of \( (x; y) \) coordinates

\texttt{...} useless here

\texttt{degree} polynomial degree for the fit (the Intercept is also returned)

\texttt{baseline1} numeric the \( (x; y) \) coordinates of the first baseline by default \((x = -0.5; y = 0)\)

\texttt{baseline2} numeric the \( (x; y) \) coordinates of the second baseline by default \((x = 0.5; y = 0)\)

\texttt{nb.pts} number of points to sample and on which to calculate polynomials

Value

a list with components when applied on a single shape:

- \texttt{coeff} the coefficients (including the intercept)
- \texttt{ortho} whether orthogonal or natural polynomials were fitted
- \texttt{degree} degree of the fit (could be retrieved through \texttt{coeff} though)
• baseline1 the first baseline point (so far the first point)
• baseline2 the second baseline point (so far the last point)
• $r^2$ the $r^2$ from the fit
• mod the raw lm model

otherwise an OpnCoe object.

Note

Orthogonal polynomials are sometimes called Legendre’s polynomials. They are preferred over natural polynomials since adding a degree do not change lower orders coefficients.

See Also

Other polynomials: npoly(), opoly_i()

Examples

data(olea)
o <- olea[1]
op <- opoly(o, degree=4)
op
# shape reconstruction
opi <- opoly_i(op)
coo_plot(o)
coo_draw(opi)
lines(opi, col='red')
# R2 for degree 1 to 10
r <- numeric()
for (i in 1:10) { r[i] <- opoly(o, degree=i)$r2 }
plot(2:10, r[2:10], type='b', pch=20, col='red', main='R2 / degree')
Arguments

pol  a pol list such as created by npoly or opoly
nb.pts  the number of points to predict. By default (and cannot be higher) the number of points in the original shape.
reregister  logical whether to reregister the shape with the original baseline.

Value

a matrix of (x; y) coordinates.

See Also

Other polynomials: npoly(), opoly()

Examples

data(olea)
o <- olea[5]
coo_plot(o)
for (i in 2:7){
x <- opoly_i(opoly(o, i))
coo_draw(x, border=col_summer(7)[i], points=FALSE) }

Out  Builds an Out object

Description

In Momocs, Out-classes objects are lists of closed outlines, with optional components, and on which generic methods such as plotting methods (e.g. stack) and specific methods (e.g. efourier can be applied. Out objects are primarily Coo objects.

Usage

Out(x, fac = dplyr::data_frame(), ldk = list())

Arguments

x  a list of matrices of (x; y) coordinates, or an array or an Out object or an Ldk object, or a data.frame (and friends)
fac  (optional) a data.frame of factors and/or numerics specifying the grouping structure
ldk  (optional) list of landmarks as row number indices

Value

an Out object
**Description**

In Momocs, `OutCoe` classes objects are wrapping around lists of morphometric coefficients, along with other informations, on which generic methods such as plotting methods (e.g. `boxplot`) and specific methods can be applied. `OutCoe` objects are primarily `Coe` objects.

**Usage**

```r
OutCoe(coe = matrix(), fac = dplyr::data_frame(), method, norm)
```

**Arguments**

- **coe**: matrix of harmonic coefficients
- **fac**: (optional) a `data.frame` of factors, specifying the grouping structure
- **method**: used to obtain these coefficients
- **norm**: the normalisation used to obtain these coefficients

**Details**

These methods can be applied on `Out` objects:

**Value**

- an `OutCoe` object

**See Also**

- Other classes: `Coe()`, `Coo()`, `Ldk()`, `OpnCoe()`, `Opn()`, `OutCoe()`, `TraCoe()`

**Examples**

```r
# all OutCoe methods
methods(class='OutCoe')
```
**Description**

All colorblind friendly RColorBrewer palettes recreated without the number of colors limitation and with transparency support thanks to `pal_alpha` that can be used alone. Also, all viridis palettes (see the [package on CRAN](https://ethanschoonover.com/solarized/)), yet color ramps are borrowed and Momocs does not depend on it. Also, `pal_qual_solarized` based on Solarized: [https://ethanschoonover.com/solarized/](https://ethanschoonover.com/solarized/) and `pal_seq_grey` only shades of grey from `grey10` to `grey90`.

**Usage**

```r
pal_alpha(cols, transp = 0)
pal_manual(cols, transp = 0)
pal_qual_solarized(n, transp = 0)
pal_seq_grey(n, transp = 0)
pal_div_BrBG(n, transp = 0)
pal_div_PiYG(n, transp = 0)
pal_div_PRGn(n, transp = 0)
pal_div_PuOr(n, transp = 0)
pal_div_RdBu(n, transp = 0)
pal_div_RdYlBu(n, transp = 0)
pal_qual_Dark2(n, transp = 0)
pal_qual_Paired(n, transp = 0)
pal_qual_Set2(n, transp = 0)
pal_seq_Blues(n, transp = 0)
pal_seq_BuGn(n, transp = 0)
pal_seq_BuPu(n, transp = 0)
pal_seq_GnBu(n, transp = 0)
```
pal_seq_Greens(n, transp = 0)
pal_seq_Greys(n, transp = 0)
pal_seq_Oranges(n, transp = 0)
pal_seq_OrRd(n, transp = 0)
pal_seq_PuBu(n, transp = 0)
pal_seq_PuBuGn(n, transp = 0)
pal_seq_PuRd(n, transp = 0)
pal_seq_Purples(n, transp = 0)
pal_seq_RdPu(n, transp = 0)
pal_seq_Reds(n, transp = 0)
pal_seq_YlGn(n, transp = 0)
pal_seq_YlGnBu(n, transp = 0)
pal_seq_YlOrBr(n, transp = 0)
pal_seq_YlOrRd(n, transp = 0)
pal_seq_magma(n, transp = 0)
pal_seq_inferno(n, transp = 0)
pal_seq_plasma(n, transp = 0)
pal_seq_viridis(n, transp = 0)
pal_qual(n, transp = 0)
pal_seq(n, transp = 0)
pal_div(n, transp = 0)

Arguments

cols: color(s) as hexadecimal values
transp: numeric between 0 and 1 (0, eg opaque, by default)
n: numeric number of colors
Details

Default color palettes are currently:

- \texttt{pal\_qual=pal\_qual\_Set2}
- \texttt{pal\_seq=pal\_seq\_viridis}
- \texttt{pal\_div=pal\_div\_RdBu}

Note

RColorBrewer palettes are not happy when \texttt{n} is lower than 3 and above a given number for each palette. If this is the case, these functions will create a color palette with \texttt{colorRampPalette} and return colors even so.

Examples

\begin{verbatim}
pal_div_BrBG(5) %>% barplot(rep(1, 5), col=.)
pal_div_BrBG(5, 0.5) %>% barplot(rep(1, 5), col=.)
\end{verbatim}

Description

Plots all the outlines, side by side, from a \texttt{Coo} (\texttt{Out}, \texttt{Opn} or \texttt{Ldk}) objects.

Usage

\begin{verbatim}
panel(x, ...)

## S3 method for class 'Out'
panel(
x,
dim,
cols,
borders,
fac,
palette = col_summer,
coo_sample = 120,
names = NULL,
cex.names = 0.6,
points = TRUE,
points.pch = 3,
points.cex = 0.2,
points.col,
...)
\end{verbatim}
## S3 method for class 'Opn'
panel(
  x,
  cols,
  borders,
  fac,
  palette = col_summer,
  coo_sample = 120,
  names = NULL,
  cex.names = 0.6,
  points = TRUE,
  points.pch = 3,
  points.cex = 0.2,
  points.col,
  ...
)

## S3 method for class 'Ldk'
panel(
  x,
  cols,
  borders,
  fac,
  palette = col_summer,
  names = NULL,
  cex.names = 0.6,
  points = TRUE,
  points.pch = 3,
  points.cex = 0.2,
  points.col = "#333333",
  ...
)

### Arguments

- **x**  The Coo object to plot.
- **...** additional arguments to feed generic `plot`
- **dim** for `coo_listpanel`: a numeric of length 2 specifying the dimensions of the panel
- **cols** A vector of colors for drawing the outlines. Either a single value or of length exactly equal 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.
- **fac** a factor within the $fac$ slot for colors
- **palette** a color `palette`
- **coo_sample** if not NULL the number of point per shape to display (to plot quickly)
- **names** whether to plot names or not. If TRUE uses shape names, or something for `fac_dispatcher`
cex.names a cex for the names
points logical (for Ldk) whether to draw points
points.pch (for Ldk) and a pch for these points
points.cex (for Ldk) and a cex for these points
points.col (for Ldk) and a col for these points

**Note**

If you want to reorder shapes according to a factor, use `arrange`.

**See Also**

Other Coo_graphics: `inspect()`, `stack()`

**Examples**

```r
panel(mosquito, names=TRUE, cex.names=0.5)
panel(olea)
panel(bot, c(4, 10))
# an illustration of the use of fac
panel(bot, fac='type', palette=col_spring, names=TRUE)
```

---

papers grindr papers for shape plots

**Description**

Papers on which to use drawers for building custom shape plots using the grindr approach. See examples and vignettes.

**Usage**

```r
paper(coo, ...)
paper_white(coo)
paper_grid(coo, grid = c(10, 5), cols = c("#ffa500", "#e5e5e5"), ...)
paper_chess(coo, n = 50, col = "#E5E5E5")
paper_dots(coo, pch = 20, n = 50, col = "#7F7F7F")
```
PCA

Arguments

- coo: a single shape or any Coe object
- grid: numeric of length 2 to (roughly) specify the number of majors lines, and the number of minor lines within two major ones
- cols: colors (hexadecimal preferred) to use for grid drawing
- n: numeric number of squares for the chessboard
- col: color (hexadecimal) to use for chessboard drawing
- pch: to use for dots

Note

This approach will (soon) replace coo_plot and friends in further versions. All comments are welcome.

See Also

Other grindr: drawers, layers_morphospace, layers, mosaic_engine(), pile(), plot_LDA(), plot_NMDS(), plot_PCA()

---

**Description**

Performs a PCA on Coe objects, using prcomp.

**Usage**

```r
PCA(x, scale., center, fac)
```

- **S3 method** for class 'OutCoe'
  ```r
  PCA(x, scale. = FALSE, center = TRUE, fac)
  ```

- **S3 method** for class 'OpnCoe'
  ```r
  PCA(x, scale. = FALSE, center = TRUE, fac)
  ```

- **S3 method** for class 'LdkCoe'
  ```r
  PCA(x, scale. = FALSE, center = TRUE, fac)
  ```

- **S3 method** for class 'TraCoe'
  ```r
  PCA(x, scale. = TRUE, center = TRUE, fac)
  ```

- **Default S3 method:**
  ```r
  PCA(x, scale. = TRUE, center = TRUE, fac = dplyr::data_frame())
  ```

  as_PCA(x, fac)
Arguments

- **x**: a Coe object or an appropriate object (e.g., prcomp) for as_PCA
- **scale**: logical whether to scale the input data
- **center**: logical whether to center the input data
- **fac**: any factor or data.frame to be passed to as_PCA and for use with plot.PCA

Details

By default, methods on Coe object do not scale the input data but center them. There is also a generic method (e.g., for traditional morphometrics) that centers and scales data.

Value

A 'PCA' object on which to apply plot.PCA, among others. This list has several components, most of them inherited from the prcomp object:

1. **sdev**: the standard deviations of the principal components (i.e., the square roots of the eigenvalues of the covariance/correlation matrix, though the calculation is actually done with the singular values of the data matrix)
2. **eig**: the cumulated proportion of variance along the PC axes
3. **rotation**: the matrix of variable loadings (i.e., a matrix whose columns contain the eigenvectors). The function princomp returns this in the element loadings.
4. **center**: scale the centering and scaling used
5. **x**: PCA scores (the value of the rotated data (the centred (and scaled if requested) data multiplied by the rotation matrix))
6. **other components**: are inherited from the Coe object passed to PCA, e.g., fac, mshape, method, baseline1 and baseline2, etc. They are documented in the corresponding *Coe file.

See Also

Other multivariate: CLUST(), KMEANS(), KMEDOIDS(), LDA(), MANOVA_PW(), MANOVA(), MDS(), MSHAPES(), NMDS(), classification_metrics()

Examples

```r
bot.f <- efourier(bot, 12)
bot.p <- PCA(bot.f)
bot.p
plot(bot.p, morpho=FALSE)
plot(bot.p, 'type')

op <- npoly(olea, 5)
op.p <- PCA(op)
op.p
plot(op.p, 1, morpho=TRUE)

wp <- fgProcrustes(wings, tol=1e-4)
wp <- PCA(wp)
```
wpp
plot(wpp, 1)

# "foreign prcomp"
head(iris)
iris.p <- prcomp(iris[, 1:4])
iris.p <- as_PCA(iris.p, iris[, 5])
class(iris.p)
plot(iris.p, 1)

---

**PCcontrib**

Shape variation along PC axes

**Description**

Calculates and plots shape variation along Principal Component axes.

**Usage**

```r
PCcontrib(PCA, ...)
```

```r
## S3 method for class 'PCA'
PCcontrib(PCA, nax, sd.r = c(-2, -1, -0.5, 0, 0.5, 1, 2), gap = 1, ...)
```

**Arguments**

- `PCA`: a PCA object
- `...`: additional parameter to pass to `coo_draw`
- `nax`: the range of PCs to plot (1 to 99pc total variance by default)
- `sd.r`: a single or a range of mean +/- sd values (eg: `c(-1, 0, 1)`)
- `gap`: for combined-Coe, an adjustment variable for gap between shapes. (bug)Default to 1 (which should never superimpose shapes), reduce it to get a more compact plot.

**Value**

(invisibly) a list with `gg` the ggplot object and `shp` the list of shapes.

**Examples**

```r
bot.p <- PCA(efourier(bot, 12))
PCcontrib(bot.p, nax=1:3)
```

```r
## Not run:
library(ggplot2)

gg <- PCcontrib(bot.p, nax=1:8, sd.r=c(-5, -3, -2, -1, -0.5, 0, 0.5, 1, 2, 3, 5))
gg$gg + geom_polygon(fill="slategrey", col="black") + ggtitle("A nice title")

## End(Not run)
```
perm

Permutates and breed Coe (and others) objects

Description

This method applies permutations column-wise on the coe of any Coe object but relies on a function that can be used on any matrix. For a Coe object, it uses sample on every column (or row) with (or without) replacement.

Usage

perm(x, ...)

## Default S3 method:
perm(x, margin = 2, size, replace = TRUE, ...)

## S3 method for class 'Coe'
perm(x, size, replace = TRUE, ...)

Arguments

x the object to permute
...
useless here
margin numeric whether 1 or 2 (rows or columns)
size numeric the required size for the final object, same size by default.
replace logical, whether to use sample with replacement

See Also

Other farming: breed()

Examples

m <- matrix(1:12, nrow=3)
m
perm(m, margin=2, size=5)
perm(m, margin=1, size=10)

bot.f <- efourier(bot, 12)
bot.m <- perm(bot.f, 80)
bot.m
Description

Pile all shapes in the same graphical window. Useful to check their normalization in terms of size, position, rotation, first point, etc. It is, essentially, a shortcut around paper + drawers of the grindr family.

Usage

```r
pile(coo, f, sample, subset, pal, paper_fun, draw_fun, transp, ...)
```

## Default S3 method:
pile(
  coo,
  f,
  sample,
  subset,
  pal = pal_qual,
  paper_fun = paper,
  draw_fun = draw_curves,
  transp = 0,
  ...
)

## S3 method for class 'list'
pile(
  coo,
  f,
  sample = 64,
  subset = 1000,
  pal = pal_qual,
  paper_fun = paper,
  draw_fun = draw_curves,
  transp = 0,
  ...
)

## S3 method for class 'array'
pile(
  coo,
  f,
  sample = 64,
  subset = 1000,
  pal = pal_qual,
  paper_fun = paper,
  ...
## S3 method for class 'Out'
pile(
  coo,
  f,
  sample = 64,
  subset = 1000,
  pal = pal_qual,
  paper_fun = paper,
  draw_fun = draw_outlines,
  transp = 0,
  ...
)

## S3 method for class 'Opn'
pile(
  coo,
  f,
  sample = 64,
  subset = 1000,
  pal = pal_qual,
  paper_fun = paper,
  draw_fun = draw_curves,
  transp = 0,
  ...
)

## S3 method for class 'Ldk'
pile(
  coo,
  f,
  sample = 64,
  subset = 1000,
  pal = pal_qual,
  paper_fun = paper,
  draw_fun = draw_landmarks,
  transp = 0,
  ...
)

### Arguments

- **coo**: a single shape or any `Coo` object
- **f**: factor specification
sample: numeric number of points to coo_sample if the number of shapes is > 1000 (default: 64). If non-numeric (eg FALSE) do not sample.

subset: numeric only draw this number of (randomly chosen) shapes if the number of shapes is > 1000 (default: 1000). If non-numeric (eg FALSE) do not sample.

default: 64). If non-numeric (eg FALSE) do not sample.

subset: numeric only draw this number of (randomly chosen) shapes if the number of shapes is > 1000 (default: 1000). If non-numeric (eg FALSE) do not sample.

subset: numeric only draw this number of (randomly chosen) shapes if the number of shapes is > 1000 (default: 1000). If non-numeric (eg FALSE) do not sample.

pal: palette among palettes (default: pal_qual)

pal: palette among palettes (default: pal_qual)

draw_fun: a papers function (default: paper)

draw_fun: a papers function (default: paper)

draw_fun: one of drawers for pile.list

draw_fun: one of drawers for pile.list

transp: numeric for transparency (default:adjusted, min:0, max=0)

transp: numeric for transparency (default:adjusted, min:0, max=0)

more arguments to feed the core drawer, depending on the object

more arguments to feed the core drawer, depending on the object

Details

Large Co0 are sampled, both in terms of the number of shapes and of points to drawn.

Note

A variation of this plot was called stack before Momocs 1.2.5

See Also

Other grindr: drawers, layers_morphospace, layers, mosaic_engine(), papers, plot_LDA(), plot_NMDS(), plot_PCA()

Examples

# all Co0 are supported with sensible defaults
pile(bot) # outlines
pile(olea, ~var, pal=pal_qual_Dark2, paper_fun=paper_grid) # curves
pile(wings) # landmarks

# you can continue the pipe with compatible drawers
pile(bot, trans=0.9) %>% draw_centroid

# if you are not happy with this, build your own!
# eg see Momocs::pile.Out (no quotes)

my_pile <- function(x, col_labels="red", transp=0.5){
x %>% paper_chess(n=100) %>%
    draw_landmarks(transp=transp) %>%
    draw_labels(col=col_labels)
}

# using it
wings %>% my_pile(transp=3/4)

# and as gridr functions propagate, you can even continue:
wings %>% my_pile() %>% draw_centroid(col="blue", cex=5)

# method on lists
bot$coo %>% pile
# it can be tuned when we have a list of landmarks with:
  wings$coo %>% pile(draw_fun=draw_landmarks)
# or on arrays (turn for draw_landmarks)
wings$coo %>% l2a %>% # we now have an array
  pile

### pix2chc

*Convert (x; y) coordinates to chaincoded coordinates*

**Description**

Useful to convert (x; y) coordinates to chain-coded coordinates.

**Usage**

```r
pix2chc(coo)
chc2pix(chc)
```

**Arguments**

- `coo`  
  (x; y) coordinates passed as a matrix
- `chc`  
  chain coordinates

**Note**

Note this function will be deprecated from Momocs when Momacs and Momit will be fully operational.

**References**


**See Also**

- `chc2pix`

Other import functions: `import_Conte()`, `import_StereoMorph_curve1()`, `import_jpg1()`, `import_jpg()`, `import_tps()`, `import_txt()`

Other import functions: `import_Conte()`, `import_StereoMorph_curve1()`, `import_jpg1()`, `import_jpg()`, `import_tps()`, `import_txt()`

**Examples**

```r
pix2chc(shapes[1]) %>% print %>% # from pix to chc
chc2pix() # and back
```
**Description**

The Momocs’ `LDA` plotter with many graphical options.

**Usage**

```r
## S3 method for class 'LDA'
plot(
  x,
  fac = x$fac,
  xax = 1,
  yax = 2,
  points = TRUE,
  col = "#000000",
  pch = 20,
  cex = 0.5,
  palette = col_solarized,
  center.origin = FALSE,
  zoom = 1,
  xlim = NULL,
  ylim = NULL,
  bg = par("bg"),
  grid = TRUE,
  nb.grids = 3,
  morphospace = FALSE,
  pos.shp = c("range", "full", "circle", "xy", "range_axes", "full_axes")[1],
  amp.shp = 1,
  size.shp = 1,
  nb.shp = 12,
  nr.shp = 6,
  nc.shp = 5,
  rotate.shp = 0,
  flipx.shp = FALSE,
  flipy.shp = FALSE,
  pts.shp = 60,
  border.shp = col_alpha("#000000", 0.5),
  lwd.shp = 1,
  col.shp = col_alpha("#000000", 0.95),
  stars = FALSE,
  ellipses = FALSE,
  conf.ellipses = 0.5,
  ellipsesax = TRUE,
  conf.ellipsesax = c(0.5, 0.9),
  lty.ellipsesax = 1,
)```
lwd.ellipsesax = sqrt(2),
chull = FALSE,
chull.lty = 1,
chull.filled = FALSE,
chull.filled.alpha = 0.92,
density = FALSE,
lev.density = 20,
contour = FALSE,
lev.contour = 3,
n.kde2d = 100,
delaunay = FALSE,
loadings = FALSE,
labelspoints = FALSE,
col.labelspoints = par("fg"),
cex.labelspoints = 0.6,
abbreviate.labelspoints = TRUE,
labelsgroups = TRUE,
cex.labelsgroups = 0.8,
rect.labelsgroups = FALSE,
abbreviate.labelsgroups = FALSE,
color.legend = FALSE,
axisnames = TRUE,
axisvar = TRUE,
unit = FALSE,
eigen = TRUE,
rug = TRUE,
title = substitute(x),
box = TRUE,
old.par = TRUE,
...)

Arguments

x an object of class "LDA", typically obtained with LDA
fac name or the column id from the $fac slot, or a formula combining column names from the $fac slot (cf. examples). A factor or a numeric of the same length can also be passed on the fly.
xax the first PC axis
yax the second PC axis
points logical whether to plot points
col a color for the points (either global, for every level of the fac or for every individual, see examples)
pch a pch for the points (either global, for every level of the fac or for every individual, see examples)
cex the size of the points
palette a palette
center.origin logical whether to center the plot onto the origin
zoom to keep your distances
xlim numeric of length two; if provided along with ylim, the x and y lims to use
ylim numeric of length two; if provided along with xlim, the x and y lims to use
bg color for the background
grid logical whether to draw a grid
nb.grids and how many of them
morphospace logical whether to add the morphological space
pos.shp passed to morphospace_positions, one of "range", "full", "circle", "xy", "range_axes", "full_axes"
Or directly a matrix of positions. See morphospace_positions
amp.shp amplification factor for shape deformation
size.shp the size of the shapes
nb.shp (pos.shp="circle") the number of shapes on the compass
nr.shp (pos.shp="full" or "range") the number of shapes per row
nc.shp (pos.shp="full" or "range") the number of shapes per column
rotate.shp angle in radians to rotate shapes (if several methods, a vector of angles)
flipx.shp same as above, whether to apply coo_flipx
flipy.shp same as above, whether to apply coo_flipy
pts.shp the number of points for drawing shapes
border.shp the border color of the shapes
lwd.shp the line width for these shapes
col.shp the color of the shapes
stars logical whether to draw "stars"
ellipses logical whether to draw confidence ellipses
conf.ellipses numeric the quantile for the (bivariate gaussian) confidence ellipses
ellipsesax logical whether to draw ellipse axes
c.conf.ellipsesax one or more numeric, the quantiles for the (bivariate gaussian) ellipses axes
lty.ellipsesax if yes, the lty with which to draw these axes
lwd.ellipsesax if yes, one or more numeric for the line widths
chull logical whether to draw a convex hull
chull.lty if yes, its linetype
chull.filled logical whether to add filled convex hulls
chull.filled.alpha numeric alpha transparency
density whether to add a 2d density kernel estimation (based on kde2d)
lev.density if yes, the number of levels to plot (through image)
contour  whether to add contour lines based on 2d density kernel
lev.contour if yes, the (approximate) number of lines to draw
n.kde2d the number of bins for kde2d, i.e. the 'smoothness' of density kernel
delaunay logical whether to add a delaunay 'mesh' between points
loadings logical whether to add loadings for every variables
labelspoints if TRUE rownames are used as labels, a colname from $fac can also be passed
col.labelspoints a color for these labels, otherwise inherited from fac
cex.labelspoints a cex for these labels
abbreviate.labelspoints logical whether to abbreviate
labels.groups logical whether to add labels for groups
cex.labels.groups if yes, a numeric for the size of the labels
rect.labels.groups logical whether to add a rectangle behind groups names
abbreviate.labels.groups logical, whether to abbreviate group names
color.legend logical whether to add a (cheap) color legend for numeric fac
axisnames logical whether to add PC names
axisvar logical whether to draw the variance they explain
unit logical whether to add plane unit
eigen logical whether to draw a plot of the eigen values
rug logical whether to add rug to margins
title character a name for the plot
box whether to draw a box around the plotting region
old.par whether to restore the old par. Set it to FALSE if you want to reuse the graphical window.
... useless here, just to fit the generic plot

Details

Widely inspired by the "layers" philosophy behind graphical functions of the ade4 R package.

Note

Morphospaces are deprecated so far. 99% of the code is shared with plot.PCA waiting for a general rewriting of a multivariate plotter. See https://github.com/vbonhomme/Momocs/issues/121

See Also

LDA, plot.CV, plot.CV2, plot.PCA.
Description

The Momocs’ PCA plotter with morphospaces and many graphical options.

Usage

```r
## S3 method for class 'PCA'
plot(
    x,
    fac,
    xax = 1,
    yax = 2,
    points = TRUE,
    col = "#000000",
    pch = 20,
    cex = 0.5,
    palette = col_solarized,
    center.origin = FALSE,
    zoom = 1,
    xlim = NULL,
    ylim = NULL,
    bg = par("bg"),
    grid = TRUE,
    nb.grids = 3,
    morphospace = TRUE,
    pos.shp = c("range", "full", "circle", "xy", "range_axes", "full_axes")[1],
    amp.shp = 1,
    size.shp = 1,
    nb.shp = 12,
    nr.shp = 6,
    nc.shp = 5,
    rotate.shp = 0,
    flipx.shp = FALSE,
    flipy.shp = FALSE,
    pts.shp = 60,
    border.shp = col_alpha("#000000", 0.5),
    lwd.shp = 1,
    col.shp = col_alpha("#000000", 0.95),
    stars = FALSE,
    ellipses = FALSE,
    conf.ellipses = 0.5,
    ellipsesax = FALSE,
    conf.ellipsesax = c(0.5, 0.9),
    lty.ellipsesax = 1,
)
lwd.ellipsesax = sqrt(2),
chull = FALSE,
chull.lty = 1,
chull.filled = TRUE,
chull.filled.alpha = 0.92,
density = FALSE,
lev.density = 20,
contour = FALSE,
lev.contour = 3,
n.kde2d = 100,
delaunay = FALSE,
loadings = FALSE,
labelspoints = FALSE,
col.labelspoints = par("fg"),
cex.labelspoints = 0.6,
abbreviate.labelspoints = TRUE,
labelsgroups = TRUE,
cex.labelsgroups = 0.8,
rect.labelsgroups = FALSE,
abbreviate.labelsgroups = FALSE,
col.legend = FALSE,
axisnames = TRUE,
axisvar = TRUE,
unit = FALSE,
eigen = TRUE,
rug = TRUE,
title = substitute(x),
box = TRUE,
old.par = TRUE,
...
)

Arguments

x PCA, typically obtained with PCA
fac name or the column id from the $fac slot, or a formula combining column names from the $fac slot (cf. examples). A factor or a numeric of the same length can also be passed on the fly.
xax the first PC axis
yax the second PC axis
points logical whether to plot points
col a color for the points (either global, for every level of the fac or for every individual, see examples)
pch a pch for the points (either global, for every level of the fac or for every individual, see examples)
cex the size of the points
palette  a palette
center.origin  logical whether to center the plot onto the origin
zoom  to keep your distances
xlim  numeric of length two; if provided along with ylim, the x and y lims to use
ylim  numeric of length two; if provided along with xlim, the x and y lims to use
bg  color for the background
grid  logical whether to draw a grid
nb.grids  and how many of them
morphospace  logical whether to add the morphological space
pos.shp  passed to morphospace_positions, one of "range", "full", "circle", "xy", "range_axes", "full_axes". Or directly a matrix of positions. See morphospace_positions
amp.shp  amplification factor for shape deformation
size.shp  the size of the shapes
nb.shp  (pos.shp="circle") the number of shapes on the compass
nr.shp  (pos.shp="full" or "range") the number of shapes per row
nc.shp  (pos.shp="full" or "range") the number of shapes per column
rotate.shp  angle in radians to rotate shapes (if several methods, a vector of angles)
flipx.shp  same as above, whether to apply coo_flipx
flipy.shp  same as above, whether to apply coo_flipy
pts.shp  the number of points for drawing shapes
border.shp  the border color of the shapes
lwd.shp  the line width for these shapes
col.shp  the color of the shapes
stars  logical whether to draw "stars"
ellipses  logical whether to draw confidence ellipses
conf.ellipses  numeric the quantile for the (bivariate gaussian) confidence ellipses
ellipsesax  logical whether to draw ellipse axes
conf.ellipsesax  one or more numeric, the quantiles for the (bivariate gaussian) ellipses axes
lty.ellipsesax  if yes, the lty with which to draw these axes
lwd.ellipsesax  if yes, one or more numeric for the line widths
chull  logical whether to draw a convex hull
chull.lty  if yes, its linetype
chull.filled  logical whether to add filled convex hulls
chull.filled.alpha  numeric alpha transparency
density  whether to add a 2d density kernel estimation (based on kde2d)
lev.density  if yes, the number of levels to plot (through image)
contour whether to add contour lines based on 2d density kernel
lev.contour if yes, the (approximate) number of lines to draw
n.kde2d the number of bins for \textit{kde2d}, ie the 'smoothness' of density kernel
delaunay logical whether to add a delaunay 'mesh' between points
loadings logical whether to add loadings for every variables
labelspoints if TRUE rownames are used as labels, a colname from $fac$ can also be passed
col.labelspoints a color for these labels, otherwise inherited from fac
cex.labelspoints a cex for these labels
abbreviate.labelspoints logical whether to abbreviate
labelsgroups logical whether to add labels for groups
cex.labelsgroups if yes, a numeric for the size of the labels
rect.labelsgroups logical whether to add a rectangle behind groups names
abbreviate.labelsgroups logical, whether to abbreviate group names
color.legend logical whether to add a (cheap) color legend for numeric fac
axisnames logical whether to add PC names
axisvar logical whether to draw the variance they explain
unit logical whether to add plane unit
eigen logical whether to draw a plot of the eigen values
rug logical whether to add rug to margins	
title character a name for the plot
box whether to draw a box around the plotting region
old.par whether to restore the old \textit{par}. Set it to FALSE if you want to reuse the graphical window.
...

Details
Widely inspired by the "layers" philosophy behind graphical functions of the ade4 R package.

Note
NAs in $fac$ are handled quite experimentally. More importantly, as of early 2018, I plan I complete rewrite of \textit{plot.PCA} and other multivariate plotters.

See Also
\textit{plot.LDA}
Examples

## Not run:
bot.f <- efourier(bot, 12)
bot.p <- PCA(bot.f)

### Morphospace options
plot(bot.p, pos.shp="full")
plot(bot.p, pos.shp="range")
plot(bot.p, pos.shp="xy")
plot(bot.p, pos.shp="circle")
plot(bot.p, pos.shp="range_axes")
plot(bot.p, pos.shp="full_axes")
plot(bot.p, morpho=FALSE)

### Passing factors to plot.PCA
# 3 equivalent methods
plot(bot.p, "type")
plot(bot.p, 1)
plot(bot.p, ~type)

# let's create a dummy factor of the correct length
# and another added to the $fac with mutate
# and a numeric of the correct length
f <- factor(rep(letters[1:2], 20))
z <- factor(rep(LETTERS[1:2], 20))
bot %<>% mutate(cs=coo_centsize(.), z=z)
bp <- bot %>% efourier %>% PCA
# so bp contains type, cs (numeric) and z; not f
# yet f can be passed on the fly
plot(bp, f)
# numeric fac are allowed
plot(bp, "cs", cex=3, color.legend=TRUE)
# formula allows combinations of factors
plot(bp, ~type+z)

### other morphometric approaches works the same
# open curves
op <- npoly(olea, 5)
op.p <- PCA(op)
op.p
plot(op.p, ~ domes + var, morpho=TRUE) # use of formula

# landmarks
wp <- fgProcrustes(wings, tol=1e-4)
wp <- PCA(wp)
wp
plot(wp, 1)

# traditionnal measurements
flower %>% PCA %>% plot(1)
# plot.PCA can be used after a PCA
PCA(iris[, 1:4], fac=iris$Species) %>% plot(1)

### Cosmetic options
#### window
plot(bp, 1, zoom=2)
plot(bp, zoom=0.5)
plot(bp, center.origin=FALSE, grid=FALSE)

#### colors
plot(bp, col="red") # globally
plot(bp, 1, col=c("#00FF00", "#0000FF")) # for every level
# a color vector of the right length
plot(bp, 1, col=rep(c("#00FF00", "#0000FF"), each=20))
# a color vector of the right length, mixign Rcolor names (not a good idea though)
plot(bp, 1, col=rep(c("#00FF00", "forestgreen"), each=20))

#### ellipses
plot(bp, 1, conf.ellipsesax=2/3)
plot(bp, 1, ellipsesax=FALSE)
plot(bp, 1, ellipsesax=TRUE, ellipses=TRUE)

#### stars
plot(bp, 1, stars=TRUE, ellipsesax=FALSE)

#### convex hulls
plot(bp, 1, chull=TRUE)
plot(bp, 1, chull.lty=3)

#### filled convex hulls
plot(bp, 1, chull.filled=TRUE)
plot(bp, 1, chull.filled.alpha = 0.8, chull.lty =1) # you can omit chull.filled=TRUE

#### density kernel
plot(bp, 1, density=TRUE, contour=TRUE, lev.contour=10)

#### delaunay
plot(bp, 1, delaunay=TRUE)

#### loadings
flower %>% PCA %>% plot(1, loadings=TRUE)

#### point/group labelling
plot(bp, 1, labelspoint=TRUE) # see options for abbreviations
plot(bp, 1, labelsgroup=TRUE) # see options for abbreviations

#### clean axes, no rug, no border, random title
plot(bp, axisvar=FALSE, axisnames=FALSE, rug=FALSE, box=FALSE, title="random")

# no eigen
plot(bp, eigen=FALSE) # eigen cause troubles to graphical window
# eigen may causes troubles to the graphical window. you can try old.par = TRUE
plot_CV

Plots a cross-validation table as an heatmap

Description
Either with frequencies (or percentages) plus marginal sums, and values as heatmaps. Used in Momocs for plotting cross-validation tables but may be used for any table (likely with freq=FALSE).

Usage
plot_CV(
  x,
  freq = FALSE,
  rm0 = FALSE,
  pc = FALSE,
  fill = TRUE,
  labels = TRUE,
  axis.size = 10,
  axis.x.angle = 45,
  cell.size = 2.5,
  signif = 2,
  ...
)

## Default S3 method: plot_CV(
  x,
  freq = FALSE,
  rm0 = FALSE,
  pc = FALSE,
  fill = TRUE,
  labels = TRUE,
  axis.size = 10,
  axis.x.angle = 45,
  cell.size = 2.5,
  signif = 2,
  ...
)

## S3 method for class 'LDA'
plot_CV(
  x,
  freq = TRUE,
  rm0 = TRUE,
pc = TRUE,
fill = TRUE,
labels = TRUE,
axis.size = 10,
axis.x.angle = 45,
cell.size = 2.5,
signif = 2,
...)

Arguments

x a (cross-validation table) or an LDA object
freq logical whether to display frequencies (within an actual class) or counts
rm0 logical whether to remove zeros
pc logical whether to multiply proportion by 100, ie display percentages
fill logical whether to fill cell according to count/freq
labels logical whether to add text labels on cells
axis.size numeric to adjust axis labels
axis.x.angle numeric to rotate x-axis labels
cell.size numeric to adjust text labels on cells
signif numeric to round frequencies using signif
... useless here

Value

a ggplot object

See Also

LDA, plot.LDA, and (pretty much the same) plot_table.

Examples

h <- hearts %>%
  fgProcrustes(0.01) %>% coo_slide(ldk=2) %>% stack %>%
efourier(6, norm=FALSE) %>% LDA(~aut)

h %>% plot_CV()

h %>% plot_CV(freq=FALSE, rm0=FALSE, fill=FALSE)
# you can customize the returned gg with some ggplot2 functions
h %>% plot_CV(labels=FALSE, fill=TRUE, axis.size=5) + ggplot2::ggtitle("A confusion matrix")

# or build your own using the prepared data_frame:
df <- h %>% plot_CV() %>% data
df
# you can even use it as a cross-table plotter
bot$fac %>% table %>% plot_CV()

---

## plot_CV2

**Plots a cross-correlation table**

### Description

Or any contingency/confusion table. A simple graphic representation based on variable width and/or color for arrows or segments, based on the relative frequencies.

### Usage

```r
plot_CV2(x, 
## S3 method for class 'LDA'
plot_CV2(x, 
## S3 method for class 'table'
plot_CV2(
  x,
  links.FUN = arrows,
  col = TRUE,
  col0 = "black",
  col.breaks = 5,
  palette = col_heat,
  lwd = TRUE,
  lwd0 = 5,
  gap.dots = 0.2,
  pch.dots = 20,
  gap.names = 0.25,
  cex.names = 1,
  legend = TRUE,
  ...
)
```

### Arguments

- **x**: an LDA object, a table or a squared matrix
- **...**: useless here.
- **links.FUN**: a function to draw the links: eg segments (by default), arrows, etc.
- **col**: logical whether to vary the color of the links
- **col0**: a color for the default link (when col = FALSE)
- **col.breaks**: the number of different colors
- **palette**: a color palette, eg col_summer, col_hot, etc.
lwd logical whether to vary the width of the links
lwd0 a width for the default link (when lwd = FALSE)
gap.dots numeric to set space between the dots and the links
pch.dots a pch for the dots
gap.names numeric to set the space between the dots and the group names
cex.names a cex for the names
legend logical whether to add a legend

Note
When freq=FALSE, the fill colors are not weighted within actual classes and should not be displayed if classes sizes are not balanced.

See Also
LDA, plot.LDA, plot.CV.

Examples
# Below various table that you can try. We will use the last one for the examples.
## Not run:
# pure random
a <- sample(rep(letters[1:4], each=10))
b <- sample(rep(letters[1:4], each=10))
tab <- table(a, b)

# veryhuge + some structure
a <- sample(rep(letters[1:10], each=10))
b <- sample(rep(letters[1:10], each=10))
tab <- table(a, b)
diag(tab) <- round(runif(10, 10, 20))
tab <- matrix(c(8, 3, 1, 0, 0,
            2, 7, 1, 2, 3,
            3, 5, 9, 1, 1,
            1, 1, 2, 7, 1,
            0, 9, 1, 4, 5), 5, 5, byrow=TRUE)
tab <- as.table(tab)

## End(Not run)
# good prediction
# tab <- matrix(c(8, 1, 1, 0, 0,
#                 1, 7, 1, 0, 0,
#                 1, 2, 9, 1, 0,
#                 1, 1, 1, 7, 1,
#                 0, 0, 0, 1, 8), 5, 5, byrow=TRUE)
tab <- as.table(tab)

plot.CV2(tab)
plot_devsegments

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

plot_devsegments(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.

See Also

Other plotting functions: coo_arrows(), coo_draw(), coo_listpanel(), coo_lolli(), coo_plot(), coo_ruban(), ldk_chull(), ldk_confell(), ldk_contour(), ldk_labels(), ldk_links(), plot_table()
Examples

```r
# we load some data
guinness <- coo_sample(bot[9], 100)

# we calculate the diff between 48 harm and one with 6 harm.
out.6 <- efourier_i(efourier(guinness, nb.h=6), nb.pts=120)

# we calculate deviations, you can also try 'edm'
dev <- edm_nearest(out.6, guinness) / coo_centsize(out.6)

# 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(guinness, main='Guiness fitted with 6 harm.', points=FALSE)
par(xpd=NA)
plot_devsegments(out.6, cols=cols, lwd=4)
coo_draw(out.6, lty=2, points=FALSE, col=NA)
par(xpd=FALSE)
```

plot_LDA

Quickly visualize LDA objects and build customs plots using the layers. See examples.

Usage

```r
plot_LDA(
x,
axes = c(1, 2),
palette = pal_qual,
points = TRUE,
points_transp = 1/4,
morphospace = FALSE,
morphospace_position = "range",
chull = TRUE,
chullfilled = FALSE,
labelgroups = FALSE,
legend = TRUE,
title = "",
center_origin = TRUE,
zoom = 0.9,
eigen = TRUE,
box = TRUE,
```
```r
plot_LDA

  iftwo_layer = layer_histogram_2,
  iftwo_split = FALSE,
  axesnames = TRUE,
  axesvar = TRUE

Arguments

x LDA object
axes numeric of length two to select PCs to use (c(1,2) by default)
palette color palette to use col_summer by default
points logical whether to draw this with layer_points
points_transp numeric to feed layer_points (default:0.25)
morphospace logical whether to draw this using layer_morphospace_PCA
morphospace_position to feed layer_morphospace_PCA (default: "range")
chull logical whether to draw this with layer_chull
chullfilled logical whether to draw this with layer_chullfilled
labelgroups logical whether to draw this with layer_labelgroups
legend logical whether to draw this with layer_legend
title character if specified, fee layer_title (default to "")
center_origin logical whether to center origin
zoom numeric zoom level for the frame (default: 0.9)
eigen logical whether to draw this using layer_eigen
box logical whether to draw this using layer_box
iftwo_layer function (no quotes) for drawing LD1 when there are two levels. So far, one of
  layer_histogram_2 (default) or layer_density_2
iftwo_split to feed split argument in layer_histogram_2 or layer_density_2
axesnames logical whether to draw this using layer_axesnames
axesvar logical whether to draw this using layer_axesvar

Note

This approach will replace plot.LDA. This is part of grindr approach that may be packaged at some
point. All comments are welcome.

See Also

Other grindr: drawers, layers_morphospace, layers, mosaic_engine(), papers, pile(), plot_NMDS(),
plot_PCA()
### First prepare an LDA object

```r
bl <- bot %>%
  # cheap alignment before efourier
  coo_align() %>%
  coo_center %>%
  coo_slidedirection("left") %>%
  # add a fake column
  mutate(fake=sample(letters[1:5], 40, replace=TRUE)) %>%
  # EFT
  efourier(6, norm=FALSE) %>%
  # LDA
  LDA(~fake)
```

```r
bl %>% plot_LDA %>% layer_morphospace_LDA
```

# Below inherited from plot_PCA and to adapt here.
#plot_PCA(bp)
#plot_PCA(bp, ~type)
#plot_PCA(bp, ~fake)

# Some curves with olea
#op <- olea %>%
#mutate(s=coo_area(.)) %>%
#filter(var !="Cypre") %>%
#chop(~view) %>% lapply(opoly, 5, nb.pts=90) %>%
#combine %>% PCA
#op$fac$s %<>% as.character() %>% as.numeric()

```r
#op %>% plot_PCA(title="hi there!")
```

### Now we can play with layers
# and for instance build a custom plot
# it should start with plot_PCA()

```r
#my_plot <- function(x, ...){

  #x %>%
  #  plot_PCA(...) %>%
  #  layer_points %>%
  #  layer_ellipsesaxes %>%
  #  layer_rug
  #}

  # and even continue after this function
  #op %>% my_plot(~var, axes=c(1, 3)) %>%
  #  layer_title("hi there!") %>%
  #  layer_stars()

  # You get the idea.
```
**plot_MSHAPES**

*Pairwise comparison of a list of shapes*

**Description**

"Confusion matrix" of a list of shapes. See examples.

**Usage**

```r
plot_MSHAPES(x, draw_fun, size, palette)
```

**Arguments**

- `x` a list of shapes (eg as returned by `MSHAPES`)
- `draw_fun` one of `draw_outline`, `draw_curves`, `draw_landmarks`. When the result of `MSHAPES` is passed, detected based on $Coe$, otherwise default to `draw_curves`.
- `size` numeric shrinking factor for shapes (and `coo_template`; 3/4 by default)
- `palette` on of `palettes`

**Note**

Directly inspired by Chitwood et al. (2016) in *New Phytologist*

**Examples**

```r
x <- bot %>% efourier(6) %>% MSHAPES(~type)

# custom colors
x %>% plot_MSHAPES(palette=pal_manual(c("darkgreen", "orange")))

# also works on list of shapes, eg:
leaves <- shapes %>% slice(grep("leaf", names(shapes))) %>% coo
class(leaves)
leaves %>% plot_MSHAPES()

# or
shapes %>%
# subset and degrade
slice(1:12) %>% coo_sample(60) %>% # grab the coo
coo %>%
plot_MSHAPES()
```
plot_NMDS

NMDS plot using grindr layers

Description

Quickly visualize MDS and NMDS objects and build custom plots using the layers. See examples.

Usage

```r
plot_NMDS(
x,  
f = NULL,  
axes = c(1, 2),  
points = TRUE,  
points_transp = 1/4,  
chull = TRUE,  
chullfilled = FALSE,  
labelgroups = FALSE,  
legend = TRUE,  
title = "",  
box = TRUE,  
axesnames = TRUE,  
palette = pal_qual
)
```

```r
plot_MDS(
x,  
f = NULL,  
axes = c(1, 2),  
points = TRUE,  
points_transp = 1/4,  
chull = TRUE,  
chullfilled = FALSE,  
labelgroups = FALSE,  
legend = TRUE,  
title = "",  
box = TRUE,  
axesnames = TRUE,  
palette = pal_qual
)
```

Arguments

- `x`: the result of MDS or NMDS
- `f`: factor specification to feed fac_dispatcher
- `axes`: numeric of length two to select PCs to use (c(1, 2) by default)
plot_PCA

PCA plot using grindr layers

Description

Quickly visualise PCA objects and friends and build custom plots using the layers. See examples.

Usage

plot_PCA(  
x,  
f = NULL, 
axes = c(1, 2),  
palette = NULL,  
points = TRUE,
points_transp = 1/4,
morphospace = TRUE,
morphospace_position = "range",
chull = TRUE,
chullfilled = FALSE,
labelpoints = FALSE,
labelgroups = FALSE,
legend = TRUE,
title = "",
center_origin = TRUE,
zoom = 0.9,
eigen = TRUE,
box = TRUE,
axesnames = TRUE,
axesvar = TRUE
)

Arguments

x a PCA object
f factor specification to feed fac_dispatcher
axes numeric of length two to select PCs to use (c(1,2) by default)
palette color palette to use col_summer by default
points logical whether to draw this with layer_points
points_transp numeric to feed layer_points (default: 0.25)
morphospace logical whether to draw this using layer_morphospace_PCA
morphospace_position numeric to feed layer_morphospace_PCA (default: "range")
chull logical whether to draw this with layer_chull
chullfilled logical whether to draw this with layer_chullfilled
labelpoints logical whether to draw this with layer_labelpoints
labelgroups logical whether to draw this with layer_labelgroups
legend logical whether to draw this with layer_legend
title character if specified, fee layer_title (default to "")
center_origin logical whether to center origin
zoom numeric zoom level for the frame (default: 0.9)
eigen logical whether to draw this using layer_eigen
box logical whether to draw this using layer_box
axesnames logical whether to draw this using layer_axesnames
axesvar logical whether to draw this using layer_axesvar

Note

This approach will replace plot.PCA (and plot.lda in further versions. This is part of grindr approach that may be packaged at some point. All comments are welcome.
See Also

Other grindr: `drawers, layers_morphospace, layers, mosaic_engine(), papers, pile(), plot_LDA(), plot_NMDS()`

Examples

```r
### First prepare two PCA objects.

# Some outlines with bot
bp <- bot %>% mutate(fake=sample(letters[1:5], 40, replace=TRUE)) %>%
  efourier(6) %>% PCA
plot_PCA(bp)
plot_PCA(bp, ~type)
plot_PCA(bp, ~fake)

# Some curves with olea
op <- olea %>%
  mutate(s=coo_area(.)) %>%
  filter(var != "Cypre") %>%
  chop(~view) %>%
  opoly(5, nb.pts=90) %>%
  combine %>%
  op$fac$s %<>% as.character() %>% as.numeric()

op %>% plot_PCA(title="hi there!")

### Now we can play with layers
# and for instance build a custom plot
# it should start with plot_PCA()

my_plot <- function(x, ...){
  x %>%
    plot_PCA(...) %>%
    layer_points %>%
    layer_ellipses %>%
    layer_rug
}

# and even continue after this function
op %>% my_plot(~var, axes=c(1, 3)) %>%
  layer_title("hi there!")

# grindr allows (almost nice) tricks like highlighting:

# bp %>% .layerize_PCA(~fake) %>%
# layer_frame %>% layer_axes() %>%
# layer_morphospace_PCA() -> x

# highlight <- function(x, ..., col_F="#CCCCCC", col_T="#FC8D62FF"){
#  args <- list(...)
#  x$colors_groups <- c(col_F, col_T)
#  x$colors_rows <- c(col_F, col_T)[(x$f %in% args)+1]
```
# x
# }
# x %>% highlight("a", "b") %>% layer_points()
# You get the idea.

---

**plot_silhouette**  
*Silhouette plot*

**Description**

Only used, so far, after **KMEDOIDS**.

**Usage**

```r
plot_silhouette(x, palette = pal_qual)
```

**Arguments**

- `x`  
  object returned by **KMEDOIDS**

- `palette`  
  one of **palettes**

**Value**

a ggplot plot

**Examples**

```r
olea %>% opoly(5) %>%
  KMEDOIDS(4) %>%
  plot_silhouette(pal_qual_solarized)
```

---

**plot_table**  
*Plots confusion matrix of sample sizes within $fac*

**Description**

An utility that plots a confusion matrix of sample size (or a barplot) for every object with a $fac. Useful to visually how large are sample sizes, how (un)balanced are designs, etc.

**Usage**

```r
plot_table(x, fac1, fac2 = fac1, rm0 = FALSE)
```
pProcrustes

Partial Procrustes alignment between two shapes

Description

Directly borrowed from Claude (2008), and called \texttt{pPsup} there.

Usage

\texttt{pProcrustes(coo1, coo2)}

Arguments

\begin{itemize}
  \item \texttt{coo1} Configuration matrix to be superimposed onto the centered preshape of \texttt{coo2}.
  \item \texttt{coo2} Reference configuration matrix.
\end{itemize}
Value

a list with components

- coo1 superimposed centered preshape of coo1 onto the centered preshape of coo2
- coo2 centered preshape of coo2
- rotation rotation matrix
- DP partial Procrustes distance between coo1 and coo2
- rho trigonometric Procrustes distance.

References


See Also

Other procrustes functions: `fProcrustes()`, `fgProcrustes()`, `fgsProcrustes()`

---

Ptolemy

Ptolemaic ellipses and illustration of efourier

Description

Calculate and display Ptolemaic ellipses which illustrates intuitively the principle behind elliptical Fourier analysis.

Usage

```r
Ptolemy(coo,
  t = seq(0, 2 * pi, length = 7)[-1],
  nb.h = 3,
  nb.pts = 360,
  palette = col_heat,
  zoom = 5/4,
  legend = TRUE,
  ...
)
```

Arguments

- **coo**: a matrix of (x; y) coordinates
- **t**: A vector of 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
rearrange_ldk

zoom numeric a zoom factor for coo_plot
legend logical. Whether to plot the legend box
...
additional parameters to feed coo_plot

References


See Also

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

exemplifying functions

Examples

cat <- shapes[4]
Ptolomy(cat, main="An EFT cat")

rearrange_ldk Rearrange, (select and reorder) landmarks to retain

Description

Helps reorder and retain landmarks by simply changing the order in which they are recorded in the Coo objects. Note that for Out and Opn objects, this rearranges the $ldk component. For Ldk, it rearranges the $coo directly.

Usage

rearrange_ldk(Coo, new_ldk_ids)

Arguments

Coo any appropriate Coo object (typically an Ldk) with landmarks inside
new_ldk_ids a vector of numeric with the ldk to retain and in the right order (see below)

See Also

Other ldk/slidings methods: add_ldk(), def_ldk(), def_slidings(), get_ldk(), get_slidings(), slidings_scheme()
reLDA

"Redo" a LDA on new data

### Description

Basically a wrapper around `predict.lda` from the package MASS. Uses a LDA model to classify new data.

### Usage

reLDA(newdata, LDA)

```r
## Default S3 method:
reLDA(newdata, LDA)

## S3 method for class 'PCA'
reLDA(newdata, LDA)

## S3 method for class 'Coe'
reLDA(newdata, LDA)
```

### Arguments

- `newdata`: to use, a PCA or any Coe object
- `LDA`: a LDA object

### Value

A list with components (from `?predict.lda`):

- `class`: factor of classification
- `posterior`: posterior probabilities for the classes
- `x`: the scores of test cases
- `res`: data.frame of the results
- `CV.tab`: a confusion matrix of the results
- `CV.correct`: proportion of the diagonal of CV.tab
- `newdata`: the data used to calculate passed to predict.lda
**Note**

Uses the same number of PC axis as the LDA object provided. You should probably use `rePCA` in conjunction with `reLDA` to get 'homologous' scores.

**Examples**

```r
# We select the first 10 individuals in bot, # for whisky and beer bottles. It will be our referential.
bot1 <- slice(bot, c(1:10, 21:30))
# Same thing for the other 10 individuals.
# It will be our unknown dataset on which we want # to calculate classes.
bot2 <- slice(bot, c(11:20, 31:40))

# We calculate efourier on these two datasets
bot1.f <- efourier(bot1, 8)
bot2.f <- efourier(bot2, 8)

# Here we obtain our LDA model: first, a PCA, then a LDA
bot1.p <- PCA(bot1.f)
bot1.l <- LDA(bot1.p, "type")

# we redo the same PCA since we worked with scores
bot2.p <- rePCA(bot1.p, bot2.f)

# we finally "predict" with the model obtained before
bot2.l <- reLDA(bot2.p, bot1.l)
bot2.l
```

---

### rename

**Rename columns by name**

**Description**

Rename variables, from the `$fac`. See examples and `dplyr::rename`.

**Usage**

```r
rename(.data, ...)
```

**Arguments**

- `.data` a Coo, Coe, PCA object
- `...` comma separated list of unquoted expressions

**Details**

dplyr verbs are maintained.
Value

a Momocs object of the same class.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice(), subsizeze()

Examples

olea
rename(olea, variety=var, domesticated=domes) # rename var column

```
# Redo a PCA on a new Coe

Description

Basically reapply rotation to a new Coe object.

Usage

rePCA(PCA, Coe)

Arguments

PCA a PCA object
Coe a Coe object

Note

Quite experimental. Dimensions of the matrices and methods must match.

Examples

b <- filter(bot, type="beer")
w <- filter(bot, type="whisky")

bf <- efourier(b, 8)
bp <- PCA(bf)

wf <- efourier(w, 8)

# and we use the "beer" PCA on the whisky coefficients
wp <- rePCA(bp, wf)

plot(wp)
```
rescale

Rescale coordinates from pixels to real length units

Description

Most of the time, (x, y) coordinates are recorded in pixels. If we want to have them in mm, cm, etc., we need to convert them and to rescale them. This function does the job for the two cases: i) either an homogeneous rescaling factor, e.g. if all pictures were taken using the very same magnification or ii) with various magnifications. More in the Details section.

Usage

rescale(x, scaling_factor, scale_mapping, magnification_col, ...)

Arguments

- x: any Coo object
- scaling_factor: numeric an homogeneous scaling factor. If all your (x, y) coordinates have the same scale
- scale_mapping: either a data.frame or a path to read such a data.frame. It MUST contain three columns in that order: magnification found in $fac, column "magnification_col", pixels, real length unit. Column names do not matter but must be specified, as read.table reads with header=TRUE. Every different magnification level found in $fac, column "magnification_col" must have its row.
- magnification_col: the name or id of the $fac column to look for magnification levels for every image
- ...: additional arguments (besides header=TRUE) to pass to read.table if `scale_mapping` is a path

Details

The i) case above is straightforward, if 1 cm is 500 pix long on all your pictures, just call rescale(your_Coo, scaling_factor=1/500) and all coordinates will be in cm.

The ii) second case is more subtle. First you need to code in your /linkCoo object, in the fac slot, a column named, say "mag", for magnification. Imagine you have 4 magnifications: 0.5, 1, 2 and 5, we have to indicate for each magnification, how many pixels stands for how many units in the real world.

This information is passed as a data.frame, built externally or in R, that must look like this:
We have to do that because, for optical reasons, the ratio pix/real_unit, is not a linear function of
the magnification.

All shapes will be centered to apply (the single or the different) scaling_factor.

Note

This function is simple but quite complex to detail. Feel free to contact me should you have any
problem with it. You can just access its code (type rescale) and reply it yourself.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), fac_dispatcher(),
filter(), mutate(), rename(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(),
sample_n(), select(), slice(), subsetize()

rfourier

Radii variation Fourier transform (equally spaced radii)

Description

rfourier computes radii variation Fourier analysis from a matrix or a list of coordinates where
points are equally spaced radii.

Usage

rfourier(x, ...)

## Default S3 method:
rfourier(x, nb.h, smooth.it = 0, norm = FALSE, ...)

## S3 method for class 'Out'
rfourier(x, nb.h = 40, smooth.it = 0, norm = TRUE, thres = pi/90, ...)

## S3 method for class 'list'
rfourier(x, ...)
Arguments

- **x**: A list or matrix of coordinates or an `Out` object
- **nb.h**: integer. The number of harmonics to use. If missing, 12 is used on shapes; 99 percent of harmonic power on `Out` objects, both with messages.
- **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.
- **thres**: numeric a tolerance to feed `is_equallyspacedradii`

Details

see the JSS paper for the maths behind. The methods for `Out` objects tests if coordinates have equally spaced radii using `is_equallyspacedradii`. A message is printed if this is not the case.

Value

A list with following components:

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

Note

Silent message and progress bars (if any) with `options("verbose"=FALSE)`.

Directly borrowed for Claude (2008), and called `fourier1` there.

References


See Also

Other `rfourier`: `rfourier_i()`, `rfourier_shape()`

Examples

data(bot)
coo <- coo_center(bot[1]) # centering is almost mandatory for `rfourier` family
coo_plot(coo)
rf <- rfourier(coo, 12)
rf
rfi <- rfourier_i(rf)
coo_draw(rfi, border='red', col=NA)

# Out method
bot %>% rfourier()
rfourier_i  
*Inverse radii variation Fourier transform*

### Description

rfourier_i uses the inverse radii variation (equally spaced radii) 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 = 120)
```

### 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 the JSS paper for the maths behind.

### 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.

### Note

Directly borrowed for Claude (2008), and called ifourier1 there.

### References


### See Also

Other rfourier: `rfourier_shape()`, `rfourier()`
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 $\text{harmonicrank}^{\alpha}$. If alpha is lower than 1, consecutive coefficients will thus increase. See rfourier for the mathematical background.

Value

A matrix of (x; y) coordinates.

References

See Also

Other rfourier: `rfourier_i()`, `rfourier()`

Examples

```r
data(bot)
rf <- rfourier(bot[1], 24)
rfourier_shape(rf$san, rf$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)

# Butterflies of the vignette's cover
panel(acl(replicate(100,
rfourier_shape(nb.h=6, alpha=0.4, nb.pts=200, plot=FALSE)))))
```

---

**rm_asym**

Removes asymmetric and symmetric variation on OutCoe objects

Description

Only for those obtained with `efourier`, otherwise a message is returned. `rm_asym` sets all B and C coefficients to 0; `rm_sym` sets all A and D coefficients to 0.

Usage

`rm_asym(OutCoe)`

```r
## Default S3 method:
rm_asym(OutCoe)

## S3 method for class 'OutCoe'
rm_asym(OutCoe)
```

`rm_sym(OutCoe)`

```r
## Default S3 method:
rm_sym(OutCoe)

## S3 method for class 'OutCoe'
rm_sym(OutCoe)
```

Arguments

- `OutCoe`: an OutCoe object
rm_harm

Value

an OutCoe object

References

Below: the first mention, and two applications.


See Also

symmetry and the note there.

Examples

```r
botf <- efourier(bot, 12)
botSym <- rm_asym(botf)
boxplot(botSym)
botSymp <- PCA(botSym)
plot(botSymp)
plot(botSymp, amp.shp=5)

# Asymmetric only
botAsym <- rm_sym(botf)
boxplot(botAsym)
botAsymp <- PCA(botAsym)
plot(botAsymp)
# strange shapes because the original shape was mainly symmetric and would need its # symmetric (eg its average) for a proper reconstruction. Should only be used like that:
plot(botAsymp, morpho=FALSE)
```

rm_harm

Removes harmonics from Coe objects

Description

Useful to drop harmonics on Coe objects. Should only work for Fourier-based approached since it looks for [A-D][1-drop] pattern.
Usage

```
rm_harm(x, drop = 1)
```

Arguments

- **x**: Coe object
- **drop**: numeric number of harmonics to drop

See Also

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_uncomplete()`, `rw_fac()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`

Examples

```
data(bot)
bf <- efourier(bot)
colnames(rm_harm(bf, 1)$coe)
```

---

**rm_missing**  
Remove shapes with missing data in fac

Description

Any row (or within a given column if by is specified) containing NA in $fac and the corresponding shapes in $coo, lines in $coe or other objects will also be dropped.

Usage

```
rm_missing(x, by)
```

Arguments

- **x**: the object on which to NA
- **by**: which column of the $fac should objects have complete views

See Also

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_uncomplete()`, `rw_fac()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`
Examples

```r
bot$fac$type[3] <- NA
bot$fac$fake[9] <- NA

bot %>% length()
bot %>% rm_missing() %>% length
bot %>% rm_missing("fake") %>% length()
```

---

**rm_uncomplete**

Remove shapes with incomplete slices

**Description**

Imagine you take three views of every object you study. Then, you can slice, filter or chop your entire dataset, do morphometrics on it, then want to combine it. But if you have forgotten one view, or if it was impossible to obtain, for one or more objects, combine will not work. This function helps you to remove those ugly ducklings. See examples

**Usage**

```r
rm_uncomplete(x, id, by)
```

**Arguments**

- **x**: the object on which to remove uncomplete "by"
- **id**: of the objects, within the $fac slot
- **by**: which column of the $fac should objects have complete views

**See Also**

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rw_fac()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`

**Examples**

```r
# we load olea
data(olea)
# we select the var Aglan since it is the only one complete
ol <- filter(olea, var == "Aglan")
# everything seems fine
table(ol$view, ol$ind)
# indeed
rm_uncomplete(ol, id="ind", by="view")

# we mess the ol object by removing a single shape
ol.pb <- slice(ol, -1)
table(ol.pb$view, ol.pb$ind)
```
# the counterpart has been removed with a notice
ol.ok <- rm_uncomplete(ol.pb, "ind", "view")
# now you can combine them
table(ol.ok$view, ol.ok$ind)

**rw_fac**

* Renames levels on Momocs objects *

**Description**

*rw_fac* stands for 'rewriting rule'. Typically useful to correct typos at the import, or merge some levels within covariates. Drops levels silently.

**Usage**

```
rw_fac(x, fac, from, to)
```

**Arguments**

- *x*: any Momocs object
- *fac*: the id of the name of the $fac column to look for (*fac_dispatcher* not yet supported)
- *from*: which level(s) should be renamed; passed as a single or several characters
- *to*: which name should be used to rename this/these levels

**Value**

a Momocs object of the same type

**See Also**

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rm_uncomplete()`, `sample_frac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`

**Examples**

```
# single renaming
rw_fac(bot, "type", "whisky", "agua_de_fuego")$type # 1 instead of "type" is fine too

# several renaming
bot2 <- mutate(bot, fake=factor(rep(letters[1:4], 10)))
rw_fac(bot2, "fake", c("a", "e"), "ae")$fake
```
Sample a fraction of shapes from a Momocs object. See examples and ?dplyr::sample_n.

Usage

```r
sample_frac(tbl, size, replace, fac, 
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>tbl</td>
<td>a Momocs object (Coo, Coe)</td>
</tr>
<tr>
<td>size</td>
<td>numeric (0 &lt; numeric &lt;= 1) the fraction of shapes to select</td>
</tr>
<tr>
<td>replace</td>
<td>logical whether sample should be done with or without replacement</td>
</tr>
<tr>
<td>fac</td>
<td>a column name if a $fac$ is defined; size is then applied within levels of this factor</td>
</tr>
<tr>
<td>...</td>
<td>additional arguments to dplyr::sample_frac and to maintain generic compatibility</td>
</tr>
</tbody>
</table>

Note

the resulting fraction is rounded with `ceiling`.

See Also

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rm_uncomplete()`, `rw_fac()`, `sample_n()`, `select()`, `slice()`, `subsetize()`

Examples

```r
# samples 50% of the bottles no matter their type
sample_frac(bot, 0.5)
# 80% bottles of beer and of whisky
table(sample_frac(bot, 0.8, fac="type")$fac)
# bootstrap the same number of bottles of each type but with replacement
table(names(sample_frac(bot, 1, replace=TRUE)))
```
Description

Sample n shapes from a Momocs object. See examples and ?dplyr::sample_n.

Usage

```r
sample_n(tbl, size, replace, fac, ...)
```

Arguments

tbl a Momocs object (Coo, Coe)
size numeric how many shapes should we sample
replace logical whether sample should be done with or without replacement
fac a column name if a $fac is defined; size is then applied within levels of this factor
... additional arguments to dplyr::sample_n and to maintain generic compatibility

See Also

Other handling functions: `arrange()`, `at_least()`, `chop()`, `combine()`, `dissolve()`, `fac_dispatcher()`, `filter()`, `mutate()`, `rename()`, `rescale()`, `rm_harm()`, `rm_missing()`, `rm_uncomplete()`, `rw_fac()`, `sample_frac()`, `select()`, `slice()`, `subsetize()`

Examples

```r
# samples 5 bottles no matter their type
sample_n(bot, 5)

# 5 bottles of beer and of whisky
table(sample_n(bot, 5, fac="type")$type)

# many repetitions
table(names(sample_n(bot, 400, replace=TRUE)))
```
scree

How many axes to retain this much of variance or trace?

Description

A set of functions around PCA/LDA eigen/trace. scree calculates their proportion and cumulated proportion; scree_min returns the minimal number of axis to use to retain a given proportion; scree_plot displays a screeplot.

Usage

scree(x, nax)
## S3 method for class 'PCA'
scree(x, nax)
## S3 method for class 'LDA'
scree(x, nax)

scree_min(x, prop)

scree_plot(x, nax)

Arguments

x a PCA object
nax numeric range of axes to consider. All by default for scree_min, display until 0.99 for scree_plot
prop numeric how many axes are enough to gather this proportion of variance. Default to 1, all axes are returned default to 1: all axis are returned

Value

scree returns a data.frame, scree_min a numeric, scree_plot a ggplot.

Examples

# On PCA
bp <- PCA(efourier(bot))
scree(bp)
scree_min(bp, 0.99)
scree_min(bp, 1)

scree_plot(bp)
scree_plot(bp, 1:5)

# on LDA, it uses svd
bl <- LDA(PCA(opoly(olea)), "var")
select

Select columns by name

Description

Select variables by name, from the $fac$. Selected variables can also be renamed on the fly. See examples and ?dplyr::select.

Usage

select(.data, ...)

Arguments

.data
a Coo, Coe, PCA object

...comma separated list of unquoted expressions

Details
dplyr verbs are maintained.

Value

a Momocs object of the same class.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), slice(), subsetize()

Examples

olea
select(olea, var, view) # drops domes and ind
select(olea, variety=var, domesticated_status=domes, view) # combine with filter with magrittr pipes
# only dorsal views, and 'var' and 'domes' columns
filter(olea, view="VD") %>% select(var, domes)
head(olea$fac) # select some columns
select(olea, domes, view) # remove some columns
select(olea, -ind) # rename on the fly and select some columns
select(olea, foo=domes)
sfourier

Radii variation Fourier transform (equally spaced curvilinear abscissa)

Description

sfourier computes radii variation Fourier analysis from a matrix or a list of coordinates where points are equally spaced along the curvilinear abscissa.

Usage

sfourier(x, nb.h)

## Default S3 method:
sfourier(x, nb.h)

## S3 method for class 'Out'
sfourier(x, nb.h)

## S3 method for class 'list'
sfourier(x, nb.h)

Arguments

x A list or matrix of coordinates or an Out object

nb.h integer. The number of harmonics to use. If missing, 12 is used on shapes; 99 percent of harmonic power on Out objects, both with messages.

Value

A list with following 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

Note

The implementation is still quite experimental (as of Dec. 2016)

References

Inverse radii variation Fourier transform

Description

sfourier_i uses the inverse radii variation (equally spaced curvilinear abscissa) transformation to calculate a shape, when given a list with Fourier coefficients, typically obtained computed with sfourier.

Usage

sfourier_i(rf, nb.h, nb.pts = 120, dtheta = FALSE)

Arguments

- **rf**: A list with ao, an and bn components, typically as returned by sfourier.
- **nb.h**: integer. The number of harmonics to calculate/use.
- **nb.pts**: integer. The number of points to calculate.
- **dtheta**: logical. Whether to use the dtheta correction method. FALSE by default. When TRUE, tries to correct the angular difference between reconstructed points; otherwise equal angles are used.

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.

See Also

Other sfourier: sfourier_i(), sfourier_shape()
sfourier_shape

References


See Also

Other sfourier: sfourier_shape(), sfourier()

Examples

coo <- coo_center(bot[1]) # centering is almost mandatory for sfourier family
coo_plot(coo)
rf <- sfourier(coo, 12)
rfi <- sfourier_i(rf)
coo_draw(rfi, border='red', col=NA)

sfourier_shape                     Calculates and draw 'sfourier' shapes.

Description

sfourier_shape calculates a 'Fourier radii variation shape' given Fourier coefficients (see Details) or can generate some 'sfourier' shapes.

Usage

sfourier_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

sfourier_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$^{\alpha}$. If alpha is lower than 1, consecutive coefficients will thus increase. See sfourier for the mathematical background.
Value

A matrix of (x; y) coordinates.

References


See Also

Other sfourier: `sfourier_i()`, `sfourier()`

Examples

```r
rf <- sfourier(bot[1], 24)
sfourier_shape(rf$san, rf$sbn) # equivalent to sfourier_i(rf)
sfourier_shape() # not very interesting

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

# Butterflies of the vignette cover
panel(Out(a2l(replicate(100, sfourier_shape(nb.h=6, alpha=0.4, nb.pts=200, plot=FALSE)))))
```

---

**shapes**

*Data: Outline coordinates of various shapes*

Description

Data: Outline coordinates of various shapes

Format

An `Out` object with the outline coordinates of some various shapes.

Source

Borrowed default shapes from (c) Adobe Photoshop. Do not send me to jail.

See Also

Other datasets: `apodemus`, `bot`, `chaff`, `charring`, `flower`, `hearts`, `molars`, `mosquito`, `mouse`, `nsfishes`, `oak`, `olea`, `trilo`, `wings`
Description

Select rows by position, based on $fac$. See examples and ?dplyr::slice.

Usage

slice(.data, ...)

Arguments

.data a Coo, Coe, PCA object
... logical conditions

Details

dplyr verbs are maintained.

Value

a Momocs object of the same class.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), subsetize()

Examples

olea
slice(olea, 1) # if you only want the coordinates, try bot[1]
slice(olea, 1:20)
slice(olea, 21:30)
slidings_scheme  
*Extracts partitions of sliding coordinates*

**Description**

Helper function that deduces (likely to be a reminder) partition scheme from `$slidings` of `Ldk` objects.

**Usage**

```r
slidings_scheme(Coo)
```

**Arguments**

- `Coo`  
an `Ldk` object

**Value**

A list with two components: `n` the number of partition; `id` their position. Or a `NULL` if no slidings are defined.

**See Also**

Other `ldk/slidings` methods: `add_ldk()`, `def_ldk()`, `def_slidings()`, `get_ldk()`, `get_slidings()`, `rearrange_ldk()`

**Examples**

```r
# no slidings defined a NULL is returned with a message
slidings_scheme(wings)

# slidings defined
slidings_scheme(chaff)
```

---

**stack**  
*Family picture of shapes*

**Description**

Plots all the outlines, on the same graph, from a `Coo` (`Out`, `Opn` or `Ldk`) object.
Usage

```r
## S3 method for class 'Coo'
stack(
x, 
cols, 
borders, 
fac, 
palette = col_summer, 
coo_sample = 120, 
points = FALSE, 
first.point = TRUE, 
centroid = TRUE, 
ldk = TRUE, 
ldk_pch = 3, 
ldk_col = "#FFFF00", 
ldk_cex = 0.5, 
ldk_links = FALSE, 
ldk_cone = FALSE, 
ldk_contour = FALSE, 
ldk_chull = FALSE, 
ldk_labels = FALSE, 
xy.axis = TRUE, 
title = substitute(x), 
...
)
```

```r
## S3 method for class 'Ldk'
stack(
x, 
cols, 
borders, 
first.point = TRUE, 
centroid = TRUE, 
ldk = TRUE, 
ldk_pch = 20, 
ldk_col = col_alpha("#000000", 0.5), 
ldk_cex = 0.3, 
meanshape = FALSE, 
meanshape_col = "#FFFF00", 
ldk_links = FALSE, 
ldk_cone = FALSE, 
ldk_contour = FALSE, 
ldk_chull = FALSE, 
ldk_labels = FALSE, 
slidings = TRUE, 
slidings_pch = "", 
xy.axis = TRUE, 
title = substitute(x),
```

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.

fac  a factor within the $fac slot for colors

palette  a color palette to use when fac is provided

coo_sample  if not NULL the number of point per shape to display (to plot quickly)

points  logical whether to draw or not points

first.point  logical whether to draw or not the first point

centroid  logical whether to draw or not the centroid

ldk  logical. Whether to display landmarks (if any).  

ldk_pch  pch for these landmarks

ldk_col  color for these landmarks

ldk_cex  cex for these landmarks

ldk_links  logical whether to draw links (of the mean shape)

ldk_confell  logical whether to draw conf ellipses

ldk_contour  logical whether to draw contour lines

ldk_chull  logical whether to draw convex hull

ldk_labels  logical whether to draw landmark labels

xy.axis  whether to draw or not the x and y axes

title  a title for the plot. The name of the Coo by default

... further arguments to be passed to coo_plot

meanshape  logical whether to add meanshape related stuff (below)

meanshape_col  a color for everything meanshape

slidings  logical whether to draw slidings semi landmarks

slidings_pch  pch for semi landmarks

See Also

Other Coo_graphics: inspect(), panel()
subsetize

Subsetize various Momocs objects

Description

Subsetize is a wrapper around dplyr’s verbs and should NOT be used directly.

Usage

subsetize(x, subset, ...)

Arguments

x a Coo or a Coe object.
subset logical taken from the $fac slot, or indices. See examples.
... useless here but maintains consistence with the generic subset.

See Also

Other handling functions: arrange(), at_least(), chop(), combine(), dissolve(), fac_dispatcher(), filter(), mutate(), rename(), rescale(), rm_harm(), rm_missing(), rm_uncomplete(), rw_fac(), sample_frac(), sample_n(), select(), slice()

Examples

# Do not use subset directly
Description
For OutCoe objects obtained with efourier, calculates several indices on the matrix of coefficients: AD, the sum of absolute values of harmonic coefficients A and D; BC same thing for B and C; amp the sum of the absolute value of all harmonic coefficients and sym which is the ratio of AD over amp. See references below for more details.

Usage
symmetry(OutCoe)

Arguments
OutCoe  efourier objects

Value
a matrix with 4 columns described above.

Note
What we call symmetry here is bilateral symmetry. By comparing coefficients resulting from efourier, with AD responsible for amplitude of the Fourier functions, and BC for their phase, it results in the plane and for fitted/reconstructed shapes that symmetry. As long as your shapes are aligned along their bilateral symmetry axis, you can use the approach coined by Iwata et al., and here implemented in Momocs.

References
Below: the first mention, and two applications.
#'

See Also
rm_asym and rm_sym.
tfourier

Examples

```r
bot.f <- efourier(bot, 12)
res <- symmetry(bot.f)
hist(res[, 'sym'])
```

tfourier

Tangent angle Fourier transform

Description

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

Usage

tfourier(x, ...)

## Default S3 method:
tfourier(x, nb.h, smooth.it = 0, norm = FALSE, ...)

## S3 method for class 'Out'
tfourier(x, nb.h = 40, smooth.it = 0, norm = TRUE, ...)

## S3 method for class 'list'
tfourier(x, ...)

Arguments

x         A list or matrix of coordinates or an Out
...
useless here
nb.h      integer. The number of harmonics to use. If missing, 12 is used on shapes; 99 percent of harmonic power on Out objects, both with messages.
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.

Value

A list with the following components:

- `ao` harmonic coefficient
- `an` vector of $a_{1\rightarrow n}$ harmonic coefficients
- `bn` vector of $b_{1\rightarrow n}$ 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.

Note
Silent message and progress bars (if any) with options("verbose"=FALSE).
Directly borrowed for Claude (2008), and called fourier2 there.

References

See Also
Other tfourier: *tfourier_i()*,*tfourier_shape()*

Examples
```r
coo <- bot[1]
coo_plot(coo)
tf <- tfourier(coo, 12)
tf
tfi <- tfourier_i(tf)
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

*Inverse tangent angle Fourier transform*

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 = 120,
  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).

Note

Directly borrowed for Claude (2008), and called `ifourier2` there.

References


See Also

Other `tfourier`: `tfourier_shape()`, `tfourier()`

Examples

`tfourier(bot[1], 24)`
`tfourier_shape()`
tfourier_shape

Calculates and draws '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.

Value

A matrix of (x; y) coordinates.

References


See Also

Other tfourier: tfourier_i(), tfourier()

Examples

tf <- tfourier(bot[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(Out(a2l(replicate(100,
   coo_force2close(tfourier_shape(nb.h=6, alpha=2, nb.pts=200, plot=FALSE)))))) # biological shapes
tie_jpg_txt  

Binds .jpg outlines from .txt landmarks taken on them

Description
Given a list of files (lf) that includes matching filenames with .jpg (black masks) and .txt (landmark positions on them as .txt), returns an Out with $ldk defined. Typically be useful if you use ImageJ to define landmarks on your outlines.

Usage
```
tie_jpg_txt(lf)
```

Arguments
- `lf`: a list of filenames

Note
Not optimized (images are read twice). Please do not hesitate to contact me should you have a particular case or need something.

See Also
Other babel functions: `lf_structure()`

tps2d  

Thin Plate Splines for 2D data

Description
tps2d is the core function for Thin Plate Splines. It is used internally for all TPS graphical functions. tps_apply is the very same function but with arguments properly named (I maintain tps2d as it is for historical reasons) when we want a apply a trasnformation grid.

Usage
```
tps2d(grid0, fr, to)
```
```
tps_apply(fr, to, new)
```

Arguments
- `grid0`: a matrix of coordinates on which to calculate deformations
- `fr`: the reference shape
- `to`: the target shape
- `new`: the shape on which to apply the shp1->shp2 calibrated tps trasnformation
Value

a shape.

See Also

Other thin plate splines: \texttt{tps\_arr()}, \texttt{tps\_grid()}, \texttt{tps\_iso()}, \texttt{tps\_raw()}

Examples

\begin{verbatim}
shapes <- shapes %>%
  coo_scale() %>% coo_center() %>%
  coo_slidedirection("up") %>%
  coo_sample(64)

leaf1 <- shapes[14]
leaf2 <- shapes[15]

# tps grid on the two leafs
tps_grid(leaf1, leaf2)
# apply the (leaf1 \rightarrow leaf2) tps trasnformation onto leaf1
# (that thus get closer to leaf2)
tps_apply(leaf1, leaf2, leaf1) %>% coo_draw(bor="purple")
\end{verbatim}

---

\texttt{tps\_arr} \hspace{1cm} Deformation 'vector field' using Thin Plate Splines

Description

\texttt{tps\_arr(ows)} calculates deformations between two configurations and illustrate them using arrows.

Usage

\begin{verbatim}
\texttt{tps\_arr(}
fr,
to,
amp = 1,
grid = TRUE,
over = 1.2,
palette = \texttt{col\_summer},
arr.nb = 200,
arr.levels = 100,
arr.len = 0.1,
arr.ang = 20,
arr.lwd = 0.75,
arr.col = "grey50",
poly = TRUE,
\texttt{)}
\end{verbatim}
tps_arr

  shp = TRUE,
  shp.col = rep(NA, 2),
  shp.border = col_qual(2),
  shp.lwd = c(2, 2),
  shp.lty = c(1, 1),
  legend = TRUE,
  legend.text,
  ...
)

**Arguments**

- **fr**
  - The reference \((x;y)\) coordinates
- **to**
  - The target \((x;y)\) coordinates
- **amp**
  - An amplification factor of differences between fr and to
- **grid**
  - Whether to calculate and plot changes across the graphical window TRUE or just within the starting shape (FALSE)
- **over**
  - Numeric that indicates how much the thin plate splines extends over the shapes
- **palette**
  - A color palette such those included in Momocs or produced with colorRampPalette
- **arr.nb**
  - Numeric. The number of arrows to calculate
- **arr.levels**
  - Numeric. The number of levels for the color of arrows
- **arr.len**
  - Numeric for the length of arrows
- **arr.ang**
  - Numeric for the angle for arrows’ heads
- **arr.lwd**
  - Numeric for the lwd for drawing arrows
- **arr.col**
  - If palette is not used the color for arrows
- **poly**
  - Whether to draw polygons (for outlines) or points (for landmarks)
- **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 for drawing the shapes
- **legend**
  - Logical. Whether to plot a legend
- **legend.text**
  - Some text for the legend
  ...
  - Additional arguments to feed coo_draw

**Value**

Nothing.

**See Also**

Other thin plate splines: tps2d(), tps_grid(), tps_iso(), tps_raw()
Examples

```r
top <- efourier(bot)
x <- MSHAPES(top, 'type', nb pts=80)$shp
fr <- x$beer
to <- x$whisky
tps_arr(fr, to, arr.nb=200, palette=col_sari, amp=3)
tps_arr(fr, to, arr.nb=200, palette=col_sari, amp=3, grid=FALSE)
```

tps_grid

Deformation grids using Thin Plate Splines

Description

tps_grid calculates and plots deformation grids between two configurations.

Usage

```r
tps_grid(
  fr,
  to,
  amp = 1,
  over = 1.2,
  grid.size = 15,
  grid.col = "grey80",
  poly = TRUE,
  shp = TRUE,
  shp.col = rep(NA, 2),
  shp.border = col.qual(2),
  shp.lwd = c(1, 1),
  shp.lty = c(1, 1),
  legend = TRUE,
  legend.text,
  ...
)
```

Arguments

- `fr`: the reference $(x; y)$ coordinates
- `to`: the target $(x; y)$ coordinates
- `amp`: an amplification factor of differences between `fr` and `to`
- `over`: numeric that indicates how much the thin plate splines extends over the shapes
- `grid.size`: numeric to specify the number of grid cells on the longer axis on the outlines
- `grid.col`: color for drawing the grid
- `poly`: whether to draw polygons (for outlines) or points (for landmarks)
- `shp`: logical. Whether to draw shapes
tps_iso

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 for drawing the shapes
legend  logical whether to plot a legend
legend.text  some text for the legend
...  additional arguments to feed coo_draw

Value

Nothing

See Also

Other thin plate splines: tps2d(), tps_arr(), tps_iso(), tps_raw()

Examples

botF <- efourier(bot)
x <- MSHAPES(botF, 'type', nb pts=80)$shp
fr <- x$beer
to <- x$whisky
tps_grid(fr, to, amp=3, grid.size=10)

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,
  grid = FALSE,
  over = 1.2,
  palette = col_spring,
  iso.nb = 1000,
  iso.levels = 12,
  cont = TRUE,
  cont.col = "black",
  poly = TRUE,
shp = TRUE,
    shp.border = col.qual(2),
    shp.lwd = c(2, 2),
    shp.lty = c(1, 1),
    legend = TRUE,
    legend.text,
    ...
)

Arguments

- **fr**: The reference \((x; y)\) coordinates
- **to**: The target \((x; y)\) coordinates
- **amp**: An amplification factor of differences between fr and to
- **grid**: Whether to calculate and plot changes across the graphical window TRUE or just within the starting shape (FALSE)
- **over**: A numeric that indicates how much the thin plate splines extends over the shapes
- **palette**: A color palette such those included in Momocs or produced with colorRampPalette
- **iso.nb**: A numeric. The number of points to use for the calculation of deformation
- **iso.levels**: numeric. The number of levels for mapping the deformations
- **cont**: logical. Whether to draw contour lines
- **cont.col**: A color for drawing the contour lines
- **poly**: whether to draw polygons (for outlines) or points (for landmarks)
- **shp**: logical. Whether to draw shapes
- **shp.border**: Two colors for drawing the borders
- **shp.lwd**: Two lwd for drawing shapes
- **shp.lty**: Two lty for drawing the shapes
- **legend**: logical whether to plot a legend
- **legend.text**: some text for the legend
- **...**: additional arguments to feed coo_draw

Value

No returned value

See Also

Other thin plate splines: tps2d(), tps_arr(), tps_grid(), tps_raw()
Examples

```r
botF <- efourier(bot)
x <- MSHAPES(botF, 'type', nb.pts=80)$shp
fr <- x$beer
to <- x$whisky
tps_iso(fr, to, iso.nb=200, amp=3)
tps_iso(fr, to, iso.nb=200, amp=3, grid=TRUE)
```

---

**tps_raw**

*Vanilla Thin Plate Splines*

**Description**

tps_raw calculates deformation grids and returns position of sampled points on it.

**Usage**

tps_raw(fr, to, amp = 1, over = 1.2, grid.size = 15)

**Arguments**

- `fr`: the reference \((x; y)\) coordinates
- `to`: the target \((x; y)\) coordinates
- `amp`: an amplification factor of differences between `fr` and `to`
- `over`: numeric that indicates how much the thin plate splines extends over the shapes
- `grid.size`: numeric to specify the number of grid cells on the longer axis on the outlines

**Value**

a list with two components: grid the \(xy\) coordinates of sampled points along the grid; dim the dimension of the grid.

**See Also**

Other thin plate splines: \texttt{tps2d()}, \texttt{tps_arr()}, \texttt{tps_grid()}, \texttt{tps_iso()}

**Examples**

```r
## Not run:
ms <- MSHAPES(efourier(bot, 10), "type")
b <- ms$shp$beer
w <- ms$shp$whisky
g <- tps_raw(b, w)
ldk_plot(g$grid)

# a wavy plot
ldk_plot(g$grid, pch=NA)
```
cols_ids <- 1:g$dim[1]
for (i in 1:g$dim[2]) lines(g$grid[cols_ids + (i-1)*g$dim[1], ])

## End(Not run)

### TraCoe

**Traditional morphometrics class**

#### Description

Defines the builder for traditional measurement class in Momocs. Is intended to ease calculations, data handling and multivariate statistics just ad the other Momocs' classes.

#### Usage

```r
TraCoe(coe = matrix(), fac = dplyr::data_frame())
```

#### Arguments

- **coe**: a matrix of measurements
- **fac**: a data.frame for covariates

#### See Also

Other classes: Coe(), Coo(), Ldk(), OpnCoe(), Opn(), OutCoe(), Out()

#### Examples

```r
# let's (more or less) rebuild the flower dataset
fl <- TraCoe(iris[, 1:4], dplyr::data_frame(sp=iris$Species))
fl %>% PCA() %>% plot("sp")
```

### trilo

**Data: Outline coordinates of cephalic outlines of trilobite**

#### Description

Data: Outline coordinates of cephalic outlines of trilobite

#### Format

A **Out** object 64 coordinates of 50 cephalic outlines from different ontogenetic stages of trilobite.

#### Source

Arranged from: [https://folk.universitetetetioslo.no/](https://folk.universitetetetioslo.no/) (used to be in ohammer website but seems to be deprecated now). The original data included 51 outlines and 5 ontogenetic stages, but one of them has just a single outline that has been removed.
verify

See Also
Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, wings

---

**verify** 

*Validates Coo objects*

---

**Description**
No validation for S3 objects, so this method is a (cheap) attempt at checking Coo objects, Out, Opn and Ldk objects.

**Usage**

```r
verify(Coo)
```

**Arguments**

- `Coo` any Coo object

**Details**
Implemented before all morphometric methods and handling verbs. To see what is checked, try eg `Momocs:::verify.Coo`

**Value**
a Coo object.

**Examples**

```r
## Not run:
verify(bot)
bot[12] <- NA
verify(bot)

verify(hearts)
hearts$ldk[[4]] <- c(1, 2)
verify(hearts)

## End(Not run)
```
which_out  Identify outliers

Description

A simple wrapper around dnorm that helps identify outliers. In particular, it may be useful on Coe object (in this case a PCA is first calculated) and also on Ldk for detecting possible outliers on freshly digitized/imported datasets.

Usage

which_out(x, conf, nax, ...)

Arguments

x  object, either Coe or a numeric on which to search for outliers
conf  confidence for dnorm (1e-3 by default)
nax  number of axes to retain (only for Coe), if <1 retain enough axes to retain this proportion of the variance
...  additional parameters to be passed to PCA (only for Coe)

Note

experimental. dnorm parameters used are median(x), sd(x)

Examples

# on a numeric
x <- rnorm(10)
which_out(x)

# on a Coe
bf <- bot %>% efourier(6)
bf$coe[c(1, 6), 1] <- 5
which_out(bf)

# on Ldk
w_no <- w_ok <- wings
w_no$coo[[2]][1, 1] <- 2
w_no$coo[[6]][2, 2] <- 2
which_out(w_ok, conf=1e-12) # with low conf, no outliers
which_out(w_no, conf=1e-12) # as expected

# a way to illustrate, filter outliers
# conf has been chosen deliberately low to show some outliers
x_f <- bot %>% efourier
x_p <- PCA(x_f)
## wings

**Data: Landmarks coordinates of mosquito wings**

### Description

Data: Landmarks coordinates of mosquito wings

### Format

A `Ldk` object containing 18 (x; y) landmarks from 127 mosquito wings, from

### Source


### See Also

Other datasets: apodemus, bot, chaff, charring, flower, hearts, molars, mosquito, mouse, nsfishes, oak, olea, shapes, trilo
Index

* Coe_graphics
  boxplot.OutCoe, 13
  hcontrib, 150
* Coo_graphics
  inspect, 159
  panel, 202
  stack, 262
* aligning functions
  coo_align, 41
  coo_aligncalliper, 42
  coo_alignminradius, 43
  coo_alignxax, 44
* babel functions
  lf_structure, 175
  tie_jpg_txt, 271
* baselining functions
  coo_baseline, 48
  coo_bookstein, 49
* bezier functions
  bezier, 11
  bezier_i, 12
* bridges functions
  as_df, 9
  bridges, 16
  complex, 38
  export, 138
* calibration
  calibrate_deviations, 17
  calibrate_harmonicpower, 20
  calibrate_r2, 22
  calibrate_reconstructions, 24
* calliper functions
  coo_calliper, 51
* centroid functions
  coo_centdist, 52
  coo_centpos, 54
  coo_centsize, 55
* classes
  Coe, 30
  Coo, 39
  Ldk, 170
  Opn, 194
  OpnCoe, 195
  Out, 198
  OutCoe, 199
  TraCoe, 278
* coo_descriptors
  coo_angle_edges, 45
  coo_angle_tangent, 46
  coo_area, 47
  coo_boundingbox, 50
  coo_chull, 56
  coo_circularity, 57
  coo_convexity, 60
  coo_eccentricity, 64
  coo_elongation, 65
  coo_length, 75
  coo_lw, 79
  coo_rectangularity, 86
  coo_rectilinearity, 87
  coo_scalars, 96
  coo軟體ility, 107
  coo_tac, 108
  coo_width, 117
* coo_intersect
  coo_intersect_angle, 70
  coo_intersect_segment, 71
* coo_utilities
  coo_align, 41
  coo_aligncalliper, 42
  coo_alignminradius, 43
  coo_alignxax, 44
  coo_baseline, 48
  coo_bookstein, 49
  coo_boundingbox, 50
  coo_calliper, 51
  coo_centdist, 52
  coo_center, 53
coo_centpos, 54
coo_close, 59
coo_down, 61
coo_dxy, 63
coo_extract, 66
coo_flipx, 67
coo_force2close, 68
coo_interpolate, 69
coo_is_closed, 72
coo_jitter, 73
coo_left, 74
coo_likely_clockwise, 76
coo_nb, 80
coo_perim, 82
coo_range, 85
coo_rev, 88
coo_right, 89
coo_rotate, 90
coo_rotatecenter, 91
coo_sample, 93
coo_sample_prop, 95
coo_samplerr, 94
coo_scale, 97
coo_shearx, 99
coo_slice, 100
coo_slide, 101
coo_slidedirection, 103
coo_slidegap, 104
coo_smooth, 105
coo_smoothcurve, 106
coo_template, 109
coo_trans, 110
coo_trim, 111
coo_trimbottom, 112
coo_trimtop, 113
coo_untiltx, 114
coo_up, 116
is_equallyspacedradii, 161

* coo_trimming functions
  coo_trim, 111
  coo_trimbottom, 112
  coo_trimtop, 113

* coo_utilities
  coo_centsize, 55

* datasets
  apodemus, 8
  bot, 13
  chaff, 26
  charring, 26
  flower, 145
  hearts, 151
  molars, 182
  mosquito, 187
  mouse, 187
  nsfishes, 192
  oak, 193
  olea, 193
  shapes, 260
  trilo, 278
  wings, 281

* dfourier
  dfourier, 122
  dfourier_i, 124
  dfourier_shape, 125

* efourier
  efourier, 134
  efourier_i, 136
  efourier_shape, 137

* exemplifying functions
  coo_dxy, 63

* farming
  breed, 15
  perm, 208

* grindr
  drawers, 127
  layers, 164
  layers_morphospace, 167
  mosaic_engine, 184
  papers, 204
  pile, 209
  plot_LDA, 228
  plot_NMDS, 232
  plot_PCA, 233

* handling functions
  arrange, 8
  at_least, 10
  chop, 27
  combine, 37
  dissolve, 126
  fac_dispatcher, 140
  filter, 143
  mutate, 189
  rename, 241
  rescale, 243
  rm_harm, 249
  rm_missing, 250
rm_uncomplete, 251
rw_fac, 252
sample_frac, 253
sample_n, 254
select, 256
slice, 261
subsetize, 265
* import functions
  import_Conte, 152
  import_jpg, 153
  import_jpg1, 154
  import_StereoMorph_curve1, 156
  import_tps, 157
  import_txt, 158
  pix2chc, 212
* ldk helpers
  def_links, 121
  ldk_check, 171
  links_all, 176
  links_delaunay, 177
* ldk plotters
  ldk_chull, 171
  ldk_confell, 172
  ldk_contour, 173
  ldk_labels, 174
  ldk_links, 174
* ldk/slidings methods
  add_ldk, 7
  def_ldk, 118
  def_slidings, 121
  get_ldk, 147
  get_slidings, 149
  rearrange_ldk, 239
  slidings_scheme, 262
* multivariate
  classification_metrics, 28
  CLUST, 29
  KMEANS, 162
  KMEDOIDS, 163
  LDA, 168
  MANOVA, 177
  MANOVA_PW, 179
  MDS, 180
  MSHAPES, 188
  NMDS, 190
  PCA, 205
* opening functions
  coo_down, 61
  coo_left, 74
  coo_right, 89
  coo_up, 116
* perimeter functions
  coo_perim, 82
* plotting functions
  coo_arrows, 48
  coo_draw, 62
  coo_listpanel, 77
  coo_lolli, 78
  coo_plot, 83
  coo_ruban, 92
  ldk_chull, 171
  ldk_confell, 172
  ldk_contour, 173
  ldk_labels, 174
  ldk_links, 174
  plot_devsegments, 227
  plot_table, 236
* polynomials
  npoly, 191
  opoly, 196
  opoly_i, 197
* premodern
  coo_truss, 113
  measure, 181
* procrustes functions
  fgsProcrustes, 141
  fgProcrustes, 142
  fProcrustes, 145
  pProcrustes, 237
* rfourier
  rfourier, 244
  rfourier_i, 246
  rfourier_shape, 247
* rotation functions
  coo_rotate, 90
  coo_rotatecenter, 91
* sampling functions
  coo_extract, 66
  coo.interpolate, 69
  coo_sample, 93
  coo_sample_prop, 95
  coo_samplerr, 94
* scaling functions
  coo_scale, 97
  coo_template, 109
* sfourier
INDEX

sfourier, 257
sfourier_i, 258
sfourier_shape, 259

* slicing functions
  coo_slice, 100

* sliding functions
  coo_slide, 101
  coo_slidedirection, 103
  coo_slidegap, 104

* smoothing functions
  coo_smooth, 105
  coo_smoothcurve, 106

* tfourier
  tfourier, 267
tfourier_i, 268
tfourier_shape, 270

* thin plate splines
  tps2d, 271
tps_arr, 272
tps_grid, 274
tps_iso, 275
tps_raw, 277

* transforming functions
  coo_flipx, 67
coo_sheax, 99

a2l (bridges), 16
a2m (bridges), 16
add_ldk, 7, 119, 122, 147, 149, 239, 262
apodemus, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
arrange, 8, 11, 27, 37, 77, 126, 140, 144, 189, 204, 242, 244, 250–254, 256, 261, 265
arrows, 48, 225
as_df, 9, 16, 38, 139
as_PCA (PCA), 205
at_least, 9, 10, 27, 37, 126, 140, 144, 189, 242, 244, 250–254, 256, 261, 265
bezier, 11, 12
bezier_i, 12, 12
bot, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
bboxplot, 195, 199
bboxplot.Coe (bboxplot.OutCoe), 13
bboxplot.OutCoe, 13, 151
bboxplot.PCA, 14
breed, 15, 208
bridges, 10, 16, 38, 139

calibrate_deviations, 17, 22, 23, 25
calibrate_deviations_dfourier
  (calibrate_deviations), 17
calibrate_deviations_efourier
  (calibrate_deviations), 17
calibrate_deviations_npoly
  (calibrate_deviations), 17
calibrate_deviations_opoly
  (calibrate_deviations), 17
calibrate_deviations_rfourier
  (calibrate_deviations), 17
calibrate_deviations_sfourier
  (calibrate_deviations), 17
calibrate_reconstructions_tfourier
  (calibrate_deviations), 17
calibrate_reconstructions
  (calibrate_deviations), 17
calibrate_reconstructions_tfourier
  (calibrate_deviations), 17

calibrate_harmonicpower, 20, 20, 23, 25
calibrate_harmonicpower_dfourier
  (calibrate_harmonicpower), 20
calibrate_harmonicpower_efourier
  (calibrate_harmonicpower), 20
calibrate_harmonicpower_rfourier
  (calibrate_harmonicpower), 20

calibrate_harmonicpower_sfourier
  (calibrate_harmonicpower), 20
calibrate_harmonicpower_tfourier
  (calibrate_harmonicpower), 20

calibrate_r2, 20, 22, 23, 25
calibrate_r2_npoly (calibrate_r2), 22
calibrate_r2_opoly (calibrate_r2), 22

calibrate_reconstructions, 20, 22, 23, 24

calibrate_reconstructions_dfourier
  (calibrate_reconstructions), 24
calibrate_reconstructions
  (calibrate_reconstructions), 24
calibrate_reconstructions_efourier
  (calibrate_reconstructions), 24

calibrate_reconstructions
  (calibrate_reconstructions), 24
calibrate_reconstructions_npoly
  (calibrate_reconstructions), 24

calibrate_reconstructions
  (calibrate_reconstructions), 24
calibrate_reconstructions_rfourier
  (calibrate_reconstructions), 24

calibrate_reconstructions
  (calibrate_reconstructions), 24

calibrate_reconstructions_sfourier
  (calibrate_reconstructions), 24

calibrate_reconstructions_tfourier
  (calibrate_reconstructions), 24

ceiling, 253
chaff, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
charring, 8, 13, 26, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281

chc2pix, 212

chc2pix (pix2chc), 212

chop, 9, 11, 27, 37, 126, 140, 144, 189, 244, 250–254, 256, 261, 265

chull, 56, 172

classification_metrics, 28, 30, 162, 163, 169, 178, 180, 181, 188, 190, 206

CLUST, 28, 29, 162, 163, 169, 178, 180, 181, 188, 190, 206

cluster::pam, 163


coeff_rearrange, 32

coeff_sel, 33

coeff_split, 34

col_alpha (col_transp), 36

col_autumn (color_palettes), 34

col_black (color_palettes), 34

col_bw (color_palettes), 34

col_cold (color_palettes), 34

col_gallus (color_palettes), 34

col_grey (color_palettes), 34

col_heat (color_palettes), 34

col_hot, 225

col_hot (color_palettes), 34

col_india (color_palettes), 34

col_qual (color_palettes), 34

col_sari (color_palettes), 34

col_solarized (color_palettes), 34

col_spring (color_palettes), 34

col_summer, 225

col_summer (color_palettes), 34

col_summer2 (color_palettes), 34

col_transp, 36

color_palettes, 34

colorRampPalette, 202, 273, 276

combine, 9, 11, 27, 37, 126, 140, 144, 189, 242, 244, 250–254, 256, 261, 265

complex, 10, 16, 38, 139


coo2cpx (complex), 38


coo_angle_edges, 45, 46, 47, 51, 57, 58, 61, 65, 66, 76, 79, 87, 88, 97, 107, 108, 117

coo_angle_tangent, 45, 46, 47, 51, 57, 58, 61, 65, 66, 76, 79, 87, 88, 97, 107, 108, 117


coo_arrows, 48, 62, 78, 79, 85, 92, 172–175, 227, 237


coo_centdist, 41–44, 49–51, 52, 53–55, 59, 60, 62, 64, 67–69, 72, 73, 75, 77, 80,
INDEX

coo_circularityonion (coo_circularity), 56
coo_circularityonion (coo_circularity), 57
coo_circularityonion (coo_circularity), 57
coo_diffrange (coo_range), 85
coo_down, 41–44, 49–54, 59, 60, 61, 64, 67–69, 72, 73, 75, 77, 80, 82, 86, 89–91, 93, 94, 96, 98–100, 102–106, 110–113, 115, 116, 161
coo_draw, 48, 62, 78, 79, 85, 92, 151, 172–175, 207, 227, 237, 273, 275, 276
coo_draw_rads, 63
coo_eccentricity, 45–47, 51, 57, 58, 61, 64, 66, 76, 79, 87, 88, 97, 107, 108, 117
coo_eccentricityonion (coo_eccentricity), 64
(coo_eccentricity), 64
coo_flipy (coo_flipx), 67
coo_intersect_angle, 70, 71, 119
coo_intersect_direction, 70, 119
coo_intersect_direction (coo_intersect_angle), 70
coo_intersect_segment, 70, 71
coo_lfd, 74
coo_likely_anticlockwise (coo_likely_clockwise), 76
coo_listpanel, 48, 62, 77, 79, 85, 92, 109,
INDEX

coo_nb, 48, 62, 78, 79, 85, 92, 172–175, 227, 237
coo_lolli, 48, 62, 78, 79, 85, 92, 172–175, 227, 237
coo_nb, 41–44, 49–54, 59, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 110–113, 115, 116, 161
coo_ruban, 48, 62, 78, 79, 85, 92, 172–175, 227, 237
coo_sample_prop, 41–44, 49–54, 59, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 95, 98–100, 102, 103, 105, 106, 110–113, 115, 116, 161
coo_sheary, 41–44, 49–54, 59, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 110–113, 115, 116, 161
coo_smooth, 41–44, 49–54, 60, 62, 64, 67–69,
INDEX

72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 106, 110–113, 115, 116, 161
coo_tangle (coo_angle_tangent), 46
coo_template_relatively, 186
coo_template_relatively (coo_template), 109
coo_trans, 41–44, 49–54, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 110, 110, 111–113, 115, 116, 161
coo_trimtop, 41–44, 49–54, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 110–113, 115, 116, 161
coo_truss, 113, 118, 181
coo_unclose, 59
coo_unclose (coo_close), 59
coo_ununtilt (coo_ununtiltx), 114
coo_ununtiltx, 41–44, 49–54, 60, 62, 64, 67–69, 72, 73, 75, 77, 80, 83, 86, 89–91, 93, 94, 96, 98–100, 102, 103, 105, 106, 110–113, 114, 116, 161

cpx2coo (complex), 38
d, 117, 181
d2m (bridges), 16
data.frame, 175
def_ldk, 7, 118, 122, 147, 149, 239, 262
def_ldk_angle, 119
def_ldk_direction (def_ldk_angle), 119
def_ldk_tips, 120, 120
def_links, 121, 171, 176, 177
def_slidings, 7, 119, 121, 147, 149, 239, 262
delaunayn, 177
density, 166
dfourier, 20, 122, 124, 125
dfourier_i, 123, 124, 125
dfourier_shape, 123, 124, 125
dissolve, 9, 11, 27, 37, 126, 140, 144, 189, 242, 244, 250–254, 256, 261, 265
dist, 29, 131, 133, 134

dnorm, 280
dplyr::data_frame, 10
dplyr::rename, 241
draw_axes (drawers), 127
draw_centroid (drawers), 127
draw_curve (drawers), 127
draw_curves, 231
draw_curves (drawers), 127
draw_firstpoint (drawers), 127
draw_labels (drawers), 127
draw_landmarks, 231
draw_landmarks (drawers), 127
draw_lines (drawers), 127
draw_links (drawers), 127
draw_outline, 231
draw_outline (drawers), 127
draw_outlines (drawers), 127
draw_points (drawers), 127
draw_polygon (drawers), 127
draw_ticks (drawers), 127
draw_title (drawers), 127
drawers, 127, 167, 168, 186, 204, 205, 211, 229, 233, 235
INDEX

layer_axesnames (layers), 164
layer_axesvar, 229, 234
layer_axesvar (layers), 164
layer_box, 229, 233, 234
layer_box (layers), 164
layer_chull, 229, 233, 234
layer_chull (layers), 164
layer_chullfilled, 229, 233, 234
layer_chullfilled (layers), 164
layer_delaunay (layers), 164
layer_density (layers), 164
layer_density_2, 229
layer_density_2 (layers), 164
layer_eigen, 229, 234
layer_eigen (layers), 164
layer_ellipses (layers), 164
layer_ellipsesaxes, 164
layer_frame (layers), 164
layer_fullframe (layers), 164
layer_grid (layers), 164
layer_histogram_2, 229
layer_histogram_2 (layers), 164
layer_labelgroups, 229, 233, 234
layer_labelgroups (layers), 164
layer_labelpoints, 234
layer_labelpoints (layers), 164
layer_legend, 229, 233, 234
layer_legend (layers), 164
layer_morphospace_LDA
  (layers_morphospace), 167
layer_morphospace_PCA, 229, 234
layer_morphospace_PCA
  (layers_morphospace), 167
layer_points, 229, 233, 234
layer_points (layers), 164
layer_rug (layers), 164
layer_stars (layers), 164
layer_ticks (layers), 164
layer_title, 229, 233, 234
layer_title (layers), 164
layers, 130, 164, 168, 186, 205, 211, 228, 229, 232, 233, 235
layers_morphospace, 130, 167, 167, 186, 205, 211, 229, 233, 235
LDA, 28, 30, 148, 162, 163, 167, 168, 178, 180, 181, 188, 190, 206, 213, 214, 216, 224–226, 228, 229, 240
lda, 168, 169
Ldk, 26, 31, 39, 93, 121, 122, 141, 142, 147, 149, 156, 158, 159, 170, 171, 193–195, 199, 202, 262, 278–281
ldk_check, 121, 171, 176, 177
ldk_chull, 48, 62, 78, 79, 85, 92, 171, 173–175, 227, 237
ldk_contour, 48, 62, 78, 79, 85, 92, 172, 173, 173, 174, 175, 227, 237
ldk_labels, 48, 62, 78, 79, 85, 92, 172, 173, 174, 175, 227, 237
ldk_plot (coo_plot), 83
LdkCoe (Ldk), 170
lf_structure, 39, 156, 175, 176, 271
lines, 84
links_all, 121, 171, 176, 177
links_delaunay, 121, 171, 176, 177
list.files, 155, 175
lm, 191, 195, 196
locator, 153, 155
m2a (bridges), 16
m2d (bridges), 16
m2l (bridges), 16
m2ll (bridges), 16
MANOVA, 28, 30, 162, 163, 169, 177, 179–181, 188, 190, 206
manova, 178–180
MANOVA_PW, 28, 30, 162, 163, 169, 178, 179, 181, 188, 190, 206
MASS: lda, 168
MDS, 28, 30, 162, 163, 169, 178, 180, 180, 188, 190, 206, 232
mean, 188
measure, 114, 117, 181
median, 188
molars, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
Momocs, 182
mosphospace_positions, 168, 183, 215, 219
mosaic (mosaic_engine), 184
mosaic_engine, 130, 167, 168, 184, 205, 211, 229, 233, 235
mosquito, 8, 13, 26, 145, 151, 182, 187, 187, 192, 193, 260, 279, 281
mouse, 8, 13, 26, 145, 151, 182, 187, 187, 192, 193, 260, 279, 281
MSHAPES, 28, 30, 121, 162, 163, 169, 178, 180, 181, 188, 190, 206, 231
mutate, 9, 11, 27, 37, 126, 140, 144, 189, 242, 244, 250–254, 256, 261, 265

NMDS, 28, 30, 162, 163, 169, 178, 180, 181, 188, 190, 206, 232
nply, 22, 191, 194, 197, 198
nply_i (nply_i), 197
nsfishes, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281

oak, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
olea, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281

Opn, 31, 39, 50, 61, 62, 75, 89, 93, 95, 96, 100, 116, 119, 120, 124, 141, 142, 147, 156, 158, 159, 170, 191, 193, 194, 194, 195, 196, 199, 202, 262, 278, 279

OpnCoe, 30, 31, 39, 170, 192, 194, 195, 197, 199, 278

opoly, 22, 192, 196, 197, 198
opoly_i, 192, 197, 197

Out, 8, 13, 26, 31, 39, 50, 61, 62, 75, 89, 93, 95, 96, 116, 119, 141, 142, 147, 151, 154, 158, 159, 170, 175, 182, 187, 192, 194, 195, 198, 199, 202, 260, 262, 278, 279

OutCoe, 30–32, 39, 170, 194, 195, 199, 199, 266, 278

pal (palettes), 200
pal_alpha (palettes), 200
pal_div (palettes), 200
pal_div_BrBG (palettes), 200
pal_div_PiYG (palettes), 200
pal_div_PRGn (palettes), 200
pal_div_PuOr (palettes), 200
pal_div_RdBu (palettes), 200
pal_div_RdYlBu (palettes), 200
pal_manual (palettes), 200
pal_qual (palettes), 200
pal_qual_Dark2 (palettes), 200
pal_qual_Paired (palettes), 200
pal_qual_Set2 (palettes), 200

pal_qual_solarized (palettes), 200
pal_seq (palettes), 200
pal_seq_Blues (palettes), 200
pal_seq_BuGn (palettes), 200
pal_seq_BuPu (palettes), 200
pal_seq_GnBu (palettes), 200
pal_seq_Greens (palettes), 200
pal_seq_grey (palettes), 200
pal_seq_Greys (palettes), 200
pal_seq_inferno (palettes), 200
pal_seq_magma (palettes), 200
pal_seq_Oranges (palettes), 200
pal_seq_orrd (palettes), 200
pal_seq_plasma (palettes), 200
pal_seq_PuBu (palettes), 200
pal_seq_PuBuGn (palettes), 200
pal_seq_PuRd (palettes), 200
pal_seq_Purples (palettes), 200
pal_seq_RdBu (palettes), 200
pal_seq_RdGy (palettes), 200
pal_seq_Viridis (palettes), 200
pal_seq_Y1Gn (palettes), 200
pal_seq_Y1GnBu (palettes), 200
pal_seq_Y1OrBr (palettes), 200
pal_seq_Y1OrRd (palettes), 200
palette (palettes), 200

Palettes (color_palettes), 34 palettes, 30, 186, 200, 211, 231, 236
panel, 159, 184, 202, 264
panel.Coo, 77
paper (papers), 204
paper_chess (papers), 204
paper_dots (papers), 204
paper_grid (papers), 204
paper_white (papers), 204
papers, 130, 167, 168, 186, 204, 211, 229, 233, 235
par, 216, 220

PCcontrib, 207
perm, 15, 208
pile, 130, 167, 168, 186, 205, 209, 229, 233, 235
pix2 chc, 153, 154, 156–158, 212
plot, 83
tps_apply (tps2d), 271
tps_arr, 272, 272, 275–277
tps_grid, 272, 273, 274, 276, 277
tps_iso, 272, 273, 275, 277
tps_raw, 272, 273, 275, 276, 277
TraCoe, 31, 39, 170, 181, 194, 195, 199, 278
trilo, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 278, 281

vegan::metaMDS, 180, 190
vegan::vegdist, 190
verify, 279

which_out, 280
wings, 8, 13, 26, 145, 151, 182, 187, 192, 193, 260, 279, 281
write.table, 139