Package ‘retistruct’

April 4, 2020

License CC BY-NC-SA 4.0

Title Retinal Reconstruction Program

Description Reconstructs retinae by morphing a flat surface with cuts (a dissected flat-mount retina) onto a curvilinear surface (the standard retinal shape). It can estimate the position of a point on the intact adult retina to within 8 degrees of arc (3.6% of nasotemporal axis). The coordinates in reconstructed retinae can be transformed to visuotopic coordinates.

Version 0.6.3

URL http://davidcsterratt.github.io/retistruct/

BugReports https://github.com/davidcsterratt/retistruct/issues

Date 2020-04-03

Depends R (>= 3.5.0)

Imports foreign, RImageJROI, png, ttutils, sp, geometry (>= 0.4.3), RTriangle (>= 1.6-0.9), rgl, R.matlab, R6

Suggests spelling, testthat, gWidgets2RGtk2 (>= 1.0.6), gWidgets2 (>= 1.0.6), cairoDevice, RGtk2

Language en-GB

RoxygenNote 7.1.0

NeedsCompilation yes

Author David C. Sterratt [aut, cre, cph], Daniel Lyngholm [aut, cph]

Maintainer David C. Sterratt <david.c.sterratt@ed.ac.uk>

Repository CRAN

Date/Publication 2020-04-04 10:10:02 UTC

R topics documented:

`AnnotatedOutline` *Class containing functions and data relating to annotating outlines*

Description

An AnnotatedOutline contains a function to annotate tears on the outline.

Value

AnnotatedOutline object, with extra fields for tears latitude of rim `phi0` and index of fixed point `i0`.

Super classes

`retistruct::OutlineCommon` -> `retistruct::Outline` -> `retistruct::PathOutline` -> `AnnotatedOutline`

Public fields

`tears` Matrix in which each row represents a tear by the indices into the outline points of the apex (`V0`) and backward (`VB`) and forward (`VF`) points

`phi0` rim angle in radians

`lambda0` longitude of fixed point

`i0` index of fixed point

Methods

Public methods:

- `AnnotatedOutline$new()`
- `AnnotatedOutline$labelTearPoints()`
- `AnnotatedOutline$whichTear()`
- `AnnotatedOutline$getTear()`
- `AnnotatedOutline$getTears()`
- `AnnotatedOutline$computeTearRelationships()`
- `AnnotatedOutline$addTear()`
- `AnnotatedOutline$removeTear()`
- `AnnotatedOutline$checkTears()`
- `AnnotatedOutline$setFixedPoint()`
- `AnnotatedOutline$getFixedPoint()`
- `AnnotatedOutline$getRimSet()`
- `AnnotatedOutline$ensureFixedPointInRim()`
- `AnnotatedOutline$getRimLengths()`
- `AnnotatedOutline$clone()`

Method `new()`**:** Constructor

Usage:

```
AnnotatedOutline$new(...)
```

Arguments:

`...` Parameters to `PathOutline`

Method `labelTearPoints()`**:** Label a set of three unlabelled points supposed to refer to the apex and vertices of a cut and tear with the `V0` (Apex), `VF` (forward vertex) and `VB` (backward vertex) labels.

Usage:

```
AnnotatedOutline$labelTearPoints(pids)
```

Arguments:

`pids` the vector of three indices

Returns: Vector of indices labelled with `V0`, `VF` and `VB`

Method `whichTear()`**:** Return index of tear in an AnnotatedOutline in which a point appears

Usage:

`AnnotatedOutline$whichTear(pid)`

Arguments:

`pid` ID of point

Returns: ID of tear

Method `getTear()`**:** Return indices of tear in AnnotatedOutline

Usage:

`AnnotatedOutline$getTear(tid)`

Arguments:

`tid` Tear ID, which can be returned from `whichTear()`

Returns: Vector of three point IDs, labelled with `V0`, `VF` and `VB`

Method `getTears()`**:** Get tears

Usage:

`AnnotatedOutline$getTears()`

Returns: Matrix of tears

Method `computeTearRelationships()`**:** Compute the parent relationships for a potential set of tears. The function throws an error if tears overlap.

Usage:

`AnnotatedOutline$computeTearRelationships(tears = NULL)`

Arguments:

`tears` Matrix containing columns `V0` (Apices of tears) `VB` (Backward vertices of tears) and `VF` (Forward vertices of tears)

Returns: List containing

- `Rset`the set of points on the rim
- `TFset`list containing indices of points in each forward tear
- `TBset`list containing indices of points in each backward tear
- `h`correspondence mapping
- `hf`correspondence mapping in forward direction for points on boundary
- `hb`correspondence mapping in backward direction for points on boundary

Method `addTear()`**:** Add tear to an AnnotatedOutline

Usage:

`AnnotatedOutline$addTear(pids)`

Arguments:

`pids` Vector of three point IDs to be added

Method `removeTear()`**:** Remove tear from an AnnotatedOutline

Usage:

`AnnotatedOutline$removeTear(tid)`

Arguments:

`tid` Tear ID, which can be returned from `whichTear()`

Method `checkTears()`**:** Check that all tears are correct.

Usage:

`AnnotatedOutline$checkTears()`

Returns: If all is OK, returns empty vector. If not, returns indices of problematic tears.

Method `setFixedPoint()`**:** Set fixed point

Usage:

`AnnotatedOutline$setFixedPoint(i0, name)`

Arguments:

`i0` Index of fixed point

`name` Name of fixed point

Method `getFixedPoint()`**:** Get point ID of fixed point

Usage:

`AnnotatedOutline$getFixedPoint()`

Returns: Point ID of fixed point

Method `getRimSet()`**:** Get point IDs of points on rim

Usage:

`AnnotatedOutline$getRimSet()`

Returns: Point IDs of points on rim

Method `ensureFixedPointInRim()`**:** Ensure that the fixed point `i0` is in the rim, not a tear. Alters object in which `i0` may have been changed.

Usage:

`AnnotatedOutline$ensureFixedPointInRim()`

Method `getRimLengths()`**:** Get lengths of edges on rim

Usage:

`AnnotatedOutline$getRimLengths()`

Returns: Vector of rim lengths

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`AnnotatedOutline$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

Examples

```
P <- rbind(c(1,1),   c(2,1),  c(2,-1),
           c(1,-1),  c(1,-2), c(-1,-2),
           c(-1,-1), c(-2,-1),c(-2,1),
           c(-1,1),  c(-1,2), c(1,2))
o <- TriangulatedOutline$new(P)
o$addTear(c(3, 4, 5))
o$addTear(c(6, 7, 8))
o$addTear(c(9, 10, 11))
o$addTear(c(12, 1, 2))
flatplot(o)
```

---

azel.to.sphere.colatitude

Convert azimuth-elevation coordinates to spherical coordinates

---

Description

Convert azimuth-elevation coordinates to spherical coordinates

Usage

```
azel.to.sphere.colatitude(r, r0)
```

Arguments

`r`	Coordinates of points in azimuth-elevation coordinates represented as 2 column matrix with column names `alpha` (elevation) and `theta` (azimuth).
`r0`	Direction of the axis of the sphere on which to project represented as a 2 column matrix of with column names `alpha` (elevation) and `theta` (azimuth).

Value

2-column matrix of spherical coordinates of points with column names `psi` (colatitude) and `lambda` (longitude).

Author(s)

David Sterratt

Examples

```
r0 <- cbind(alpha=0, theta=0)
r <- rbind(r0, r0+c(1,0), r0-c(1,0), r0+c(0,1), r0-c(0,1))
azel.to.sphere.colatitude(r, r0)
```

`azimuthal.conformal` *Azimuthal conformal or stereographic or Wulff projection*

Description

Azimuthal conformal or stereographic or Wulff projection

Usage

```
azimuthal.conformal(r, ...)
```

Arguments

`r`	2-column Matrix of spherical coordinates of points on sphere. Column names are `phi` and `lambda`.
`...`	Arguments not used by this projection.

Value

2-column Matrix of Cartesian coordinates of points on polar projection. Column names should be `x` and `y`.

Note

This is a special case with the point centred on the projection being the South Pole. The MathWorld equations are for the more general case.

Author(s)

David Sterratt

References

http://en.wikipedia.org/wiki/Map_projection, http://mathworld.wolfram.com/StereographicProjection.html Fisher, N. I., Lewis, T., and Embleton, B. J. J. (1987). Statistical analysis of spherical data. Cambridge University Press, Cambridge, UK.

`azimuthal.equalarea` *Lambert azimuthal equal area projection*

Description

Lambert azimuthal equal area projection

Usage

```
azimuthal.equalarea(r, ...)
```

Arguments

`r`	2-column Matrix of spherical coordinates of points on sphere. Column names are `phi` and `lambda`.
`...`	Arguments not used by this projection.

Value

2-column Matrix of Cartesian coordinates of points on polar projection. Column names should be `x` and `y`.

Note

This is a special case with the point centred on the projection being the South Pole. The MathWorld equations are for the more general case.

Author(s)

David Sterratt

References

http://en.wikipedia.org/wiki/Map_projection, http://mathworld.wolfram.com/LambertAzimuthalEqual-Area html Fisher, N. I., Lewis, T., and Embleton, B. J. J. (1987). Statistical analysis of spherical data. Cambridge University Press, Cambridge, UK.

`azimuthal.equidistant` *Azimuthal equidistant projection*

Description

Azimuthal equidistant projection

Usage

```
azimuthal.equidistant(r, ...)
```

Arguments

`r`	2-column Matrix of spherical coordinates of points on sphere. Column names are `phi` and `lambda`.
`...`	Arguments not used by this projection.

Value

2-column Matrix of Cartesian coordinates of points on polar projection. Column names should be `x` and `y`.

Note

This is a special case with the point centred on the projection being the South Pole. The MathWorld equations are for the more general case.

Author(s)

David Sterratt

References

http://en.wikipedia.org/wiki/Map_projection, http://mathworld.wolfram.com/AzimuthalEquidistantProjec
html

bary.to.sphere.cart — *Convert barycentric coordinates of points in mesh on sphere to cartesian coordinates*

Description

Given a triangular mesh on a sphere described by mesh locations (`phi`, `lambda`), a radius `R` and a triangulation `Tt`, determine the Cartesian coordinates of points `cb` given in barycentric coordinates with respect to the mesh.

Usage

```
bary.to.sphere.cart(phi, lambda, R, Tt, cb)
```

Arguments

`phi`	Latitudes of mesh points
`lambda`	Longitudes of mesh points
`R`	Radius of sphere
`Tt`	Triangulation
`cb`	Object returned by tsearch containing information on the triangle in which a point occurs and the barycentric coordinates within that triangle

Value

An N-by-3 matrix of the Cartesian coordinates of the points

Author(s)

David Sterratt

central.angle — *Central angle between two points on a sphere*

Description

On a sphere the central angle between two points is defined as the angle whose vertex is the centre of the sphere and that subtends the arc formed by the great circle between the points. This function computes the central angle for two points (ϕ_1, λ_1) and (ϕ_2, λ_2).

Usage

```
central.angle(phi1, lambda1, phi2, lambda2)
```

Arguments

`phi1`	Latitude of first point
`lambda1`	Longitude of first point
`phi2`	Latitude of second point
`lambda2`	Longitude of second point

Value

Central angle

Author(s)

David Sterratt

Source

Wikipedia http://en.wikipedia.org/wiki/Central_angle

`checkDatadir` *Check the whether directory contains valid data*

Description

Check the whether directory contains valid data

Usage

```
checkDatadir(dir = NULL)
```

Arguments

`dir`	Directory to check.

Value

`TRUE` if `dir` contains valid data; `FALSE` otherwise.

Author(s)

David Sterratt

circle — *Return points on the unit circle*

Description

Return points on the unit circle in an anti-clockwise direction. If `L` is not specified `n` points are returned. If `L` is specified, the same number of points are returned as there are elements in `L`, the interval between successive points being proportional to `L`.

Usage

```
circle(n = 12, L = NULL)
```

Arguments

`n`	Number of points
`L`	Intervals between points

Value

The cartesian coordinates of the points

Author(s)

David Sterratt

compute.intersections.sphere — *Find the intersection of a plane with edges of triangles on a sphere*

Description

Find the intersections of the plane defined by the normal `n` and the distance `d` expressed as a fractional distance along the side of each triangle.

Usage

```
compute.intersections.sphere(phi, lambda, T, n, d)
```

Arguments

`phi`	Latitude of grid points on sphere centred on origin.
`lambda`	Longitude of grid points on sphere centred on origin.
`T`	Triangulation
`n`	Normal of plane
`d`	Distance of plane along normal from origin.

Value

Matrix with same dimensions as T. Each row gives the intersection of the plane with the corresponding triangle in T. Column 1 gives the fractional distance from vertex 2 to vertex 3. Column 2 gives the fractional distance from vertex 3 to vertex 1. Column 2 gives the fractional distance from vertex 1 to vertex 2. A value of NaN indicates that the corresponding edge lies in the plane. A value of Inf indicates that the edge lies parallel to the plane but outside it.

Author(s)

David Sterratt

---

compute.kernel.estimate

Kernel estimate over grid

Description

Compute a kernel estimate over a grid and do a contour analysis of this estimate. The contour heights the determined by finding heights that exclude a certain fraction of the probability. For example, the 95 and it should enclose about 5 are specified by the contour.levels option; by default they are c(5,25,50,75,95).

Usage

```
compute.kernel.estimate(Dss, phi0, fhat, compute.conc)
```

Arguments

Dss	List of datasets. The first two columns of each datasets are coordinates of points on the sphere in spherical polar (latitude, phi, and longitude, lambda) coordinates. In the case kernel smoothing, there is a third column of values of dependent variables at those points.
phi0	Rim angle in radians
fhat	Function such as kde.fhat or kr.yhat to compute the density given data and a value of the concentration parameter kappa of the Fisher density.
compute.conc	Function to return the optimal value of the concentration parameter kappa given the data.

Value

A list containing

kappa	The concentration parameter
h	A pseudo-bandwidth parameter, the inverse of the square root of kappa. Units of degrees.
flevels	Contour levels.

`labels`	Labels of the contours.
`g`	Raw density estimate drawn on non-area-preserving projection. Comprises locations of gridlines in Cartesian coordinates (`xs` and `ys`), density estimates at these points, `f` and location of maximum in Cartesian coordinates (`max`).
`gpa`	Raw density estimate drawn on area-preserving projection. Comprises same elements as above.
`contour.areas`	Area of each individual contour. One level may have more than one contour; this shows the areas of all such contours.
`tot.contour.areas`	Data frame containing the total area within the contours at each level.

Author(s)

David Sterratt

`CountSet` *Subclass of* `FeatureSet` *to represent counts centred on points*

Description

A CountSet contains information about points located on `Outline`s. Each CountSet contains a list of matrices, each of which has columns labelled `X` and `Y` describing the cartesian coordinates (in the unscaled coordinate frame) of the centres of boxes in the Outline, and a column `C` representing the counts in those boxes.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::FeatureSet` -> `CountSet`

Methods

Public methods:

- `CountSet$new()`
- `CountSet$reconstruct()`
- `CountSet$clone()`

Method `new()`**:** Constructor

Usage:

```
CountSet$new(data = NULL, cols = NULL)
```

Arguments:

`data` List of matrices describing data. Each matrix should have columns named `X`, `Y` and `C`

`cols` Named vector of colours for each data set. The name is used as the ID (label) for the data set. The colours should be names present in the output of the `colors` function

Method `reconstruct()`**:** Map the CountSet to a `ReconstructedOutline`

Usage:

```
CountSet$reconstruct(ro)
```

Arguments:

`ro` The `ReconstructedOutline`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
CountSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`create.polar.cart.grid` *Create grid on projection of hemisphere onto plane*

Description

Create grid on projection of hemisphere onto plane

Usage

```
create.polar.cart.grid(pa, res, phi0)
```

Arguments

`pa`	If `TRUE`, make this an area-preserving projection
`res`	Resolution of grid
`phi0`	Value of `phi0` at edge of grid

Value

List containing:

`s`	Grid locations in spherical coordinates
`c`	Grid locations in Cartesian coordinates on plane
`xs`	X grid line locations in Cartesian coordinates on plane
`ys`	Y grid line locations in Cartesian coordinates on plane

Author(s)

David Sterratt

csv.read.dataset *Read a retinal dataset in CSV format*

Description

Read a retinal dataset in CSV format. Each dataset is a folder containing a file called outline.csv that specifies the outline in X-Y coordinates. It may also contain a file `datapoints.csv`, containing the locations of data points and a file `datacounts.csv`, containing the locations of data counts; see `read.datapoints` and `read.datacounts` for the formats of these files. The folder may also contain a file `od.csv` specifying the coordinates of the optic disc.

Usage

```
csv.read.dataset(dataset, report = message)
```

Arguments

`dataset`	Path to directory containing `outline.csv`
`report`	Function to report progress

Value

A `RetinalOutline` object

Author(s)

David Sterratt

dE *The deformation energy gradient function*

Description

The function that computes the gradient of the energy (or error) of the deformation of the mesh from the flat outline to the sphere. This depends on the locations of the points given in spherical coordinates. The function is designed to take these as a vector that is received from the `optim` function.

Usage

```
dE(
  p,
  Cu,
  C,
  L,
  B,
  T,
  A,
  R,
  Rset,
  i0,
  phi0,
  lambda0,
  Nphi,
  N,
  alpha = 1,
  x0,
  nu = 1,
  verbose = FALSE
)
```

Arguments

`p`	Parameter vector of `phi` and `lambda`
`Cu`	The upper part of the connectivity matrix
`C`	The connectivity matrix
`L`	Length of each edge in the flattened outline
`B`	Connectivity matrix
`T`	Triangulation in the flattened outline
`A`	Area of each triangle in the flattened outline
`R`	Radius of the sphere
`Rset`	Indices of points on the rim
`i0`	Index of fixed point on rim
`phi0`	Latitude at which sphere curtailed
`lambda0`	Longitude of fixed points
`Nphi`	Number of free values of `phi`
`N`	Number of points in sphere
`alpha`	Area penalty scaling coefficient
`x0`	Area penalty cut-off coefficient
`nu`	Power to which to raise area
`verbose`	How much information to report

Value

A vector representing the derivative of the energy of this particular configuration with respect to the parameter vector

Author(s)

David Sterratt

`depthplot3D` *Draw the "flat" outline in 3D with depth information*

Description

Draw the "flat" outline in 3D with depth information

Usage

```
depthplot3D(r, ...)
```

Arguments

`r`	`TriangulatedOutline` object
`...`	Parameters depending on class of `r`

Author(s)

David Sterratt

`E` *The deformation energy function*

Description

The function that computes the energy (or error) of the deformation of the mesh from the flat outline to the sphere. This depends on the locations of the points given in spherical coordinates. The function is designed to take these as a vector that is received from the `optim` function.

Usage

```
E(
  p,
  Cu,
  C,
  L,
  B,
  T,
  A,
  R,
  Rset,
  i0,
  phi0,
  lambda0,
  Nphi,
  N,
  alpha = 1,
  x0,
  nu = 1,
  verbose = FALSE
)
```

Arguments

p	Parameter vector of phi and lambda
Cu	The upper part of the connectivity matrix
C	The connectivity matrix
L	Length of each edge in the flattened outline
B	Connectivity matrix
T	Triangulation in the flattened outline
A	Area of each triangle in the flattened outline
R	Radius of the sphere
Rset	Indices of points on the rim
i0	Index of fixed point on rim
phi0	Latitude at which sphere curtailed
lambda0	Longitude of fixed points
Nphi	Number of free values of phi
N	Number of points in sphere
alpha	Area scaling coefficient
x0	Area cut-off coefficient
nu	Power to which to raise area
verbose	How much information to report

Value

A single value, representing the energy of this particular configuration

Author(s)

David Sterratt

Ecart *The deformation energy function*

Description

The function that computes the energy (or error) of the deformation of the mesh from the flat outline to the sphere. This depends on the locations of the points given in spherical coordinates. The function is designed to take these as a vector that is received from the `optim` function.

Usage

```
Ecart(P, Cu, L, T, A, R, alpha = 1, x0, nu = 1, verbose = FALSE)
```

Arguments

`P`	N-by-3 matrix of point coordinates
`Cu`	The upper part of the connectivity matrix
`L`	Length of each edge in the flattened outline
`T`	Triangulation in the flattened outline
`A`	Area of each triangle in the flattened outline
`R`	Radius of sphere
`alpha`	Area penalty scaling coefficient
`x0`	Area penalty cut-off coefficient
`nu`	Power to which to raise area
`verbose`	How much information to report

Value

A single value, representing the energy of this particular configuration

Author(s)

David Sterratt

f — *Piecewise smooth function used in area penalty*

Description

Piecewise, smooth function that increases linearly with negative arguments.

$$f(x) = \begin{cases} -(x - x_0/2) & x < 0 \\ \frac{1}{2x_0}(x - x_0)^2 & 0 < x < x_0 \\ 0 & x \geq x_0 \end{cases}$$

Usage

```
f(x, x0)
```

Arguments

x	Main argument
x0	The cut-off parameter. Above this value the function is zero.

Value

The value of the function.

Author(s)

David Sterratt

Fcart — *The deformation energy gradient function*

Description

The function that computes the gradient of the energy (or error) of the deformation of the mesh from the flat outline to the sphere. This depends on the locations of the points given in spherical coordinates. The function is designed to take these as a vector that is received from the `optim` function.

Usage

```
Fcart(P, C, L, T, A, R, alpha = 1, x0, nu = 1, verbose = FALSE)
```

Arguments

P	N-by-3 matrix of point coordinates
C	The connectivity matrix
L	Length of each edge in the flattened outline
T	Triangulation in the flattened outline
A	Area of each triangle in the flattened outline
R	Radius of sphere
alpha	Area penalty scaling coefficient
x0	Area penalty cut-off coefficient
nu	Power to which to raise area
verbose	How much information to report

Value

A vector representing the derivative of the energy of this particular configuration with respect to the parameter vector

Author(s)

David Sterratt

FeatureSet — *Superclass containing functions and data relating to sets of features in flat* `Outlines`

Description

A FeatureSet contains information about features located on `Outlines`. Each FeatureSet contains a list of matrices, each of which has columns labelled `X` and `Y` describing the cartesian coordinates of points on the Outline, in the unscaled coordinate frame. Derived classes, e.g. a `CountSet`, may have extra columns. Each matrix in the list has an associated label and colour, which is used by plotting functions.

Super class

`retistruct::FeatureSetCommon` -> `FeatureSet`

Methods

Public methods:

- `FeatureSet$new()`
- `FeatureSet$clone()`

Method `new()`**:** Constructor

Usage:

```
FeatureSet$new(data = NULL, cols = NULL, type = NULL)
```

Arguments:

`data` List of matrices describing data. Each matrix should have columns named `X` and `Y`

`cols` Named vector of colours for each data set. The name is used as the ID (label) for the data set. The colours should be names present in the output of the `colors` function

`type` String

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
FeatureSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

FeatureSetCommon — *Class containing functionality common to `FeatureSet`s and `ReconstructedFeatureSet`s*

Description

An FeatureSetCommon has functionality for retrieving sets of features (e.g. points or landmarks associated with an outline)

Public fields

`data` List of matrices describing data

`cols` Vector of colours for each data set

`type` String giving type of feature set

Methods

Public methods:

- `FeatureSetCommon$getIndex()`
- `FeatureSetCommon$getIDs()`
- `FeatureSetCommon$setID()`
- `FeatureSetCommon$getFeature()`
- `FeatureSetCommon$getFeatures()`
- `FeatureSetCommon$getCol()`
- `FeatureSetCommon$clone()`

Method `getIndex()`**:** Get numeric index of features

Usage:

`FeatureSetCommon$getIndex(fid)`

Arguments:

`fid` Feature ID (string)

Method `getIDs()`**:** Get IDs of features

Usage:

`FeatureSetCommon$getIDs()`

Returns: Vector of IDs of features

Method `setID()`**:** Set name

Usage:

`FeatureSetCommon$setID(i, fid)`

Arguments:

`i` Numeric index of feature

`fid` Feature ID (string)

Method `getFeature()`**:** Get feature by feature ID

Usage:

`FeatureSetCommon$getFeature(fid)`

Arguments:

`fid` Feature ID string

Returns: Matrix describing feature

Method `getFeatures()`**:** Get all features

Usage:

`FeatureSetCommon$getFeatures()`

Method `getCol()`**:** Get colour in which to plot feature ID

Usage:

`FeatureSetCommon$getCol(fid)`

Arguments:

`fid` Feature ID string

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`FeatureSetCommon$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

fire — *The FIRE algorithm*

Description

This is an implementation of the FIRE algorithm for structural relaxation put forward by Bitzek et al. (2006)

Usage

```
fire(
  r,
  force,
  restraint,
  m = 1,
  dt = 0.1,
  maxmove = 100,
  dtmax = 1,
  Nmin = 5,
  finc = 1.1,
  fdec = 0.5,
  astart = 0.1,
  fa = 0.99,
  a = 0.1,
  nstep = 100,
  tol = 1e-05,
  verbose = FALSE,
  report = message
)
```

Arguments

`r`	Initial locations of particles
`force`	Force function
`restraint`	Restraint function
`m`	Masses of points
`dt`	Initial time step
`maxmove`	Maximum distance to move in any time step
`dtmax`	Maximum time step
`Nmin`	Number of steps after which to start increasing `dt`
`finc`	Fractional increase in `dt` per time step
`fdec`	Fractional decrease in `dt` after a stop
`astart`	Starting value of `a` after a stop
`fa`	Fraction of `a` to retain after each step

a	Initial value of a
nstep	Maximum number of steps
tol	Tolerance - if RMS force is below this value, stop and report convergence
verbose	If TRUE report progress verbosely
report	Function to report progress when verbose is TRUE

Value

List containing x, the positions of the points, conv, which is 0 if convergence as occurred and 1 otherwise, and frms, the root mean square of the forces on the particles.

Author(s)

David Sterratt

References

Bitzek, E., Koskinen, P., G\"ahler, F., Moseler, M., and Gumbsch, P. (2006). Structural relaxation made simple. Phys. Rev. Lett., 97:170201.

---

flatplot *Plot "flat" (unreconstructed) representation of outline*

Description

Plot "flat" (unreconstructed) representation of outline

Usage

```
flatplot(x, axt = "n", xlim = NULL, ylim = NULL, ...)
```

Arguments

x	Outline, AnnotatedOutline, StitchedOutline &c object
axt	whether to plot axes
xlim	x limits
ylim	y limits
...	Other plotting parameters

Author(s)

David Sterratt

`flatplot.AnnotatedOutline`
Flat plot of AnnotatedOutline

Description

Plot flat `AnnotatedOutline`. The user markup is displayed by default.

Usage

```
## S3 method for class 'AnnotatedOutline'
flatplot(x, axt = "n", xlim = NULL, ylim = NULL, markup = TRUE, ...)
```

Arguments

`x`	`AnnotatedOutline` object
`axt`	whether to plot axes
`xlim`	x-limits
`ylim`	y-limits
`markup`	If `TRUE`, plot markup
`...`	Other plotting parameters

Author(s)

David Sterratt

`flatplot.Outline` *Flat plot of outline*

Description

Plot flat `Outline`.

Usage

```
## S3 method for class 'Outline'
flatplot(
  x,
  axt = "n",
  xlim = NULL,
  ylim = NULL,
  add = FALSE,
  image = TRUE,
  scalebar = 1,
```

```
    rimset = FALSE,
    pids = FALSE,
    pid.joggle = 0,
    lwd.outline = 1,
    ...
)
```

Arguments

x	Outline object
axt	whether to plot axes
xlim	x limits
ylim	y limits
add	If TRUE, don't draw axes; add to existing plot.
image	If TRUE the image (if it is present) is displayed behind the outline
scalebar	If numeric and if the Outline has a scale field, a scale bar of length scalebar mm is plotted. If scalebar is FALSE or there is no scale information in the Outline x the scale bar is suppressed.
rimset	If TRUE, plot the points computed to be in the rim in the colour specified by the option rimset.col
pids	If TRUE, plot point IDs
pid.joggle	Amount to joggle point IDs by randomly
lwd.outline	Line width of outline
...	Other plotting parameters

Author(s)

David Sterratt

---

flatplot.ReconstructedOutline
Flat plot of reconstructed outline

Description

Plot ReconstructedOutline object. This adds a mesh of gridlines from the spherical retina (described by points phi, lambda and triangulation Tt and cut-off point phi0) onto a flattened retina (described by points P and triangulation T).

Usage

```
## S3 method for class 'ReconstructedOutline'
flatplot(
  x,
  axt = "n",
  xlim = NULL,
  ylim = NULL,
  grid = TRUE,
  strain = FALSE,
  ...
)
```

Arguments

x	ReconstructedOutline object
axt	whether to plot axes
xlim	x-limits
ylim	y-limits
grid	Whether or not to show the grid lines of latitude and longitude
strain	Whether or not to show the strain
...	Other plotting parameters

Author(s)

David Sterratt

---

flatplot.StitchedOutline

Flat plot of AnnotatedOutline

Description

Plot flat StitchedOutline. If the optional argument `stitch` is `TRUE` the user markup is displayed.

Usage

```
## S3 method for class 'StitchedOutline'
flatplot(x, axt = "n", xlim = NULL, ylim = NULL, stitch = TRUE, lwd = 1, ...)
```

Arguments

x	AnnotatedOutline object
axt	whether to plot axes
xlim	x-limits
ylim	y-limits
stitch	If TRUE, plot stitch
lwd	Line width
...	Other parameters

Author(s)

David Sterratt

---

flatplot.TriangulatedOutline
Plot flat TriangulatedOutline.

Description

Plot flat TriangulatedOutline.

Usage

```
## S3 method for class 'TriangulatedOutline'
flatplot(x, axt = "n", xlim = NULL, ylim = NULL, mesh = TRUE, ...)
```

Arguments

x	TriangulatedOutline object
axt	whether to plot axes
xlim	x-limits
ylim	y-limits
mesh	If TRUE, plot mesh
...	Other plotting parameters

Author(s)

David Sterratt

`flipped.triangles` *Determine indices of triangles that are flipped*

Description

In the projection of points onto the sphere, some triangles maybe flipped, i.e. in the wrong orientation. This functions determines which triangles are flipped by computing the vector pointing to the centre of each triangle and comparing this direction to vector product of two sides of the triangle.

Usage

```
flipped.triangles(Ps, Tt, R = 1)
```

Arguments

`Ps`	N-by-2 matrix with columns containing latitudes (`phi`) and longitudes (`lambda`) of N points
`Tt`	Triangulation of points
`R`	Radius of sphere

Value

List containing:

`flipped`	Indices of in rows of `Tt` of flipped triangles.
`cents`	Vectors of centres.
`areas`	Areas of triangles.

Author(s)

David Sterratt

`flipped.triangles.cart` *Determine indices of triangles that are flipped*

Description

In the projection of points onto the sphere, some triangles maybe flipped, i.e. in the wrong orientation. This function determines which triangles are flipped by computing the vector pointing to the centre of each triangle and comparing this direction to vector product of two sides of the triangle.

Usage

```
flipped.triangles.cart(P, Tt, R)
```

Arguments

`P`	Points in Cartesian coordinates
`Tt`	Triangulation of points
`R`	Radius of sphere

Value

List containing:

`flipped`	Indices of in rows of `Tt` of flipped triangles.
`cents`	Vectors of centres.
`areas`	Areas of triangles.

Author(s)

David Sterratt

`fp` *Piecewise smooth function used in area penalty*

Description

Derivative of `f`

Usage

```
fp(x, x0)
```

Arguments

`x`	Main argument
`x0`	The cut-off parameter. Above this value the function is zero.

Value

The value of the function.

Author(s)

David Sterratt

`Fragment` *Construct an outline object. This sanitises the input points `P`, as described below.*

Description

Construct an outline object. This sanitises the input points `P`, as described below.

Construct an outline object. This sanitises the input points `P`, as described below.

Public fields

`P` A N-by-2 matrix of points of the `Outline` arranged in anticlockwise order

`gf` For each row of `P`, the index of `P` that is next in the outline travelling anticlockwise (forwards)

`gb` For each row of `P`, the index of `P` that is next in the outline travelling clockwise (backwards)

`h` For each row of `P`, the correspondence of that point (which will be to itself initially)

`A.tot` Total area of the Fragment

Methods

Public methods:

- `Fragment$initializeFromPoints()`
- `Fragment$clone()`

Method `initializeFromPoints()`**:** Initialise a Fragment from a set of points

Usage:

```
Fragment$initializeFromPoints(P)
```

Arguments:

`P` An N-by-2 matrix of points of the `Outline`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Fragment$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`identity.transform` *The identity transformation*

Description

The identity transformation

Usage

```
identity.transform(r, ...)
```

Arguments

`r`	Coordinates of points in spherical coordinates represented as 2 column matrix with column names `phi` (latitude) and `lambda` (longitude).
`...`	Other arguments

Value

Identical matrix

Author(s)

David Sterratt

`idt.read.dataset` *Read one of the Thompson lab's retinal datasets*

Description

Read one of the Thompson lab's retinal datasets. Each dataset is a folder containing a SYS file in SYSTAT format and a MAP file in text format. The SYS file specifies the locations of the data points and the MAP file specifies the outline.

Usage

```
idt.read.dataset(dataset, report = message, d.close = 0.25)
```

Arguments

`dataset`	Path to directory containing as SYS and MAP file
`report`	Function to report progress
`d.close`	Maximum distance between points for them to count as the same point. This is expressed as a fraction of the width of the outline.

Details

The function returns the outline of the retina. In order to do so, it has to join up the segments of the MAP file. The tracings are not always precise; sometimes there are gaps between points that are actually the same point. The parameter `d.close` specifies how close points must be to count as the same point.

Value

`dataset`	The path to the directory given as an argument
`raw`	List containing `map` The raw MAP data `sys` The raw SYS data
`P`	The points of the outline
`gf`	Forward pointers along the outline
`gb`	Backward pointers along the outline
`Ds`	List of datapoints
`Ss`	List of landmark lines

Author(s)

David Sterratt

`ijroi.read.dataset` *Read a retinal dataset in IJROI format*

Description

Read a retinal dataset in IJROI format. Each dataset is a folder containing a file called `outline.roi` that specifies the outline in X-Y coordinates. It may also contain a file `datapoints.csv`, containing the locations of data points; see `read.datapoints` for the format of this file. The folder may also contain a file `od.roi` specifying the coordinates of the optic disc.

Usage

```
ijroi.read.dataset(dataset, report = report)
```

Arguments

`dataset`	Path to directory containing `outline.roi`
`report`	Function to report progress

Value

A `RetinalOutline` object

Author(s)

David Sterratt

`interpolate.image` *Interpolate values in image*

Description

Interpolate values in image

Usage

```
interpolate.image(im, P, invert.y = FALSE)
```

Arguments

`im`	image to interpolate
`P`	N by 2 matrix of x, y values at which to interpolate. x is in range `[0,ncol(im)]` and y is in range `[0,nrow(im)]`
`invert.y`	If `FALSE` (the default), the y coordinate is zero at the top of the image. `TRUE` the zero y coordinate is at the bottom.

Value

Vector of N interpolated values

Author(s)

David Sterratt

`invert.sphere` *Invert sphere about its centre*

Description

Invert sphere about its centre

Usage

```
invert.sphere(r, ...)
```

Arguments

`r`	Coordinates of points in spherical coordinates represented as 2 column matrix with column names `phi` (latitude) and `lambda` (longitude).
`...`	Other arguments

Value

Matrix in same format, but with `pi` added to lambda and `phi` negated.

Author(s)

David Sterratt

`invert.sphere.to.hemisphere`
Invert sphere to hemisphere

Description

Invert image of a partial sphere and scale the longitude so that points at latitude `phi0` is projected onto a longitude of 0 degrees (the equator).

Usage

```
invert.sphere.to.hemisphere(r, phi0, ...)
```

Arguments

`r`	Coordinates of points in spherical coordinates represented as 2 column matrix with column names `phi` (latitude) and `lambda` (longitude).
`phi0`	The latitude to map onto the equator
`...`	Other arguments

Value

Matrix in same format, but with `pi` added to lambda and `phi` negated and scaled so that the longitude `phi0` is projected to 0 degrees (the equator)

Author(s)

David Sterratt

karcher.mean.sphere *Karcher mean on the sphere*

Description

The Karcher mean of a set of points on a manifold is defined as the point whose sum of squared Riemann distances to the points is minimal. On a sphere using spherical coordinates this distance can be computed using the formula for central angle.

Usage

```
karcher.mean.sphere(x, na.rm = FALSE, var = FALSE)
```

Arguments

`x`	Matrix of points on sphere as N-by-2 matrix with labelled columns \codephi (latitude) and `lambda` (longitude)
`na.rm`	logical value indicating whether `NA` values should be stripped before the computation proceeds.
`var`	logical value indicating whether variance should be returned too.

Value

Vector of means with components named `phi` and `lambda`. If `var` is `TRUE`, a list containing mean and variance in elements `mean` and `var`.

Author(s)

David Sterratt

References

Heo, G. and Small, C. G. (2006). Form representations and means for landmarks: A survey and comparative study. *Computer Vision and Image Understanding*, 102:188-203.

See Also

`central.angle`

`kde.compute.concentration` *Find the optimal concentration for a set of data*

Description

Find the optimal concentration for a set of data

Usage

```
kde.compute.concentration(mu)
```

Arguments

`mu`	Data in spherical coordinates

Value

The optimal concentration

Author(s)

David Sterratt

`kde.fhat` *Kernel density estimate on sphere using Fisherian density with polar coordinates*

Description

Kernel density estimate on sphere using Fisherian density with polar coordinates

Usage

```
kde.fhat(r, mu, kappa)
```

Arguments

`r`	Locations at which to estimate density in polar coordinates
`mu`	Locations of data points in polar coordinates
`kappa`	Concentration parameter

Value

Vector of density estimates

Author(s)

David Sterratt

`kde.fhat.cart` *Kernel density estimate on sphere using Fisherian density with Cartesian coordinates*

Description

Kernel density estimate on sphere using Fisherian density with Cartesian coordinates

Usage

```
kde.fhat.cart(r, mu, kappa)
```

Arguments

`r`	Locations at which to estimate density in Cartesian coordinates on unit sphere
`mu`	Locations of data points in Cartesian coordinates on unit sphere
`kappa`	Concentration parameter

Value

Vector of density estimates

Author(s)

David Sterratt

`kde.L` *Estimate of the log likelihood of the points mu given a particular value of the concentration kappa*

Description

Estimate of the log likelihood of the points mu given a particular value of the concentration kappa

Usage

```
kde.L(mu, kappa)
```

Arguments

`mu`	Locations of data points in Cartesian coordinates on unit sphere
`kappa`	Concentration parameter

Value

Log likelihood of data

Author(s)

David Sterratt

`kr.compute.concentration` *Find the optimal concentration for a set of data*

Description

Find the optimal concentration for a set of data

Usage

```
kr.compute.concentration(mu, y)
```

Arguments

`mu`	Locations in Cartesian coordinates (independent variables)
`y`	Values at locations (dependent variables)

Value

The optimal concentration

Author(s)

David Sterratt

`kr.sscv` *Cross validation estimate of the least squares error of the points mu given a particular value of the concentration kappa*

Description

Cross validation estimate of the least squares error of the points mu given a particular value of the concentration kappa

Usage

```
kr.sscv(mu, y, kappa)
```

Arguments

`mu`	Locations in Cartesian coordinates (independent variables)
`y`	Values at locations (dependent variables)
`kappa`	Concentration parameter

Value

Least squares error

Author(s)

David Sterratt

`kr.yhat` *Kernel regression on sphere using Fisherian density with polar coordinates*

Description

Kernel regression on sphere using Fisherian density with polar coordinates

Usage

```
kr.yhat(r, mu, y, kappa)
```

Arguments

`r`	Locations at which to estimate dependent variables in polar coordinates
`mu`	Locations in polar coordinates (independent variables)
`y`	Values at data points (dependent variables)
`kappa`	Concentration parameter

Value

Estimates of dependent variables at locations `r`

Author(s)

David Sterratt

kr.yhat.cart — *Kernel regression on sphere using Fisherian density with Cartesian coordinates*

Description

Kernel regression on sphere using Fisherian density with Cartesian coordinates

Usage

```
kr.yhat.cart(r, mu, y, kappa)
```

Arguments

`r`	Locations at which to estimate dependent variables in Cartesian coordinates
`mu`	Locations in Cartesian coordinates (independent variables)
`y`	Values at locations (dependent variables)
`kappa`	Concentration parameter

Value

Estimates of dependent variables at locations `r`

Author(s)

David Sterratt

LandmarkSet — *Subclass of `FeatureSet` to represent points*

Description

A LandmarkSet contains information about points located on `Outline`s. Each LandmarkSet contains a list of matrices, each of which has columns labelled `X` and `Y` describing the cartesian coordinates (in the unscaled coordinate frame) of points in landmarks on the Outline.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::FeatureSet` -> `LandmarkSet`

Methods

Public methods:

- `LandmarkSet$new()`
- `LandmarkSet$reconstruct()`
- `LandmarkSet$clone()`

Method `new()`**:** Constructor

Usage:

```
LandmarkSet$new(data = NULL, cols = NULL)
```

Arguments:

`data` List of matrices describing data. Each matrix should have columns named `X` and `Y`

`cols` Named vector of colours for each data set. The name is used as the ID (label) for the data set. The colours should be names present in the output of the `colors` function

Method `reconstruct()`**:** Map the LandmarkSet to a `ReconstructedOutline`

Usage:

```
LandmarkSet$reconstruct(ro)
```

Arguments:

`ro` The `ReconstructedOutline`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LandmarkSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`line.line.intersection` *Determine intersection between two lines*

Description

Determine the intersection of two lines L1 and L2 in two dimensions, using the formula described by Weisstein.

Usage

```
line.line.intersection(P1, P2, P3, P4, interior.only = FALSE)
```

Arguments

P1	vector containing x,y coordinates of one end of L1
P2	vector containing x,y coordinates of other end of L1
P3	vector containing x,y coordinates of one end of L2
P4	vector containing x,y coordinates of other end of L2
interior.only	boolean flag indicating whether only intersections inside L1 and L2 should be returned.

Value

Vector containing x,y coordinates of intersection of L1 and L2. If L1 and L2 are parallel, this is infinite-valued. If `interior.only` is `TRUE`, then when the intersection does not occur between P1 and P2 and P3 and P4, a vector containing `NA`s is returned.

Author(s)

David Sterratt

Source

Weisstein, Eric W. "Line-Line Intersection." From MathWorld–A Wolfram Web Resource. http://mathworld.wolfram.com/Line-LineIntersection.html

Examples

```
## Intersection of two intersecting lines
line.line.intersection(c(0, 0), c(1, 1), c(0, 1), c(1, 0))

## Two lines that don't intersect
line.line.intersection(c(0, 0), c(0, 1), c(1, 0), c(1, 1))
```

---

list.datasets *List datasets underneath a directory*

Description

List valid datasets underneath a directory. This reports all directories that appear to be valid.

Usage

```
list.datasets(path = ".", verbose = FALSE)
```

Arguments

path	Directory path to start searching from
verbose	If `TRUE` report on progress

Value

A vector of directories containing datasets

Author(s)

David Sterratt

`list_to_R6` *Convert an list created by R6_to_list() into an R6 object.*

Description

Convert an list created by R6_to_list() into an R6 object.

Usage

```
list_to_R6(l)
```

Arguments

`l`	list created by R6_to_list()

Value

R6 object or list list

Author(s)

David Sterratt

`lvsLplot` *Plot the fractional change in length of mesh edges*

Description

Plot the fractional change in length of mesh edges. The length of each edge in the mesh in the reconstructed object is plotted against each edge in the spherical object. The points are colour-coded according to the amount of log strain each edge is under.

Usage

```
lvsLplot(r, ...)
```

Arguments

`r`	`ReconstructedOutline` object
`...`	Other plotting parameters

Author(s)

David Sterratt

`name.list` *Return a new version of the list in which any unnamed elements have been given standardised names*

Description

Return a new version of the list in which any unnamed elements have been given standardised names

Usage

```
name.list(l)
```

Arguments

`l`	the list with unnamed elements

Value

The list with standardised names

Author(s)

David Sterratt

`normalise.angle` *Bring angle into range*

Description

Bring angle into range

Usage

```
normalise.angle(theta)
```

Arguments

`theta`	Angle to bring into range `[-pi,pi]`

Value

Normalised angle

Author(s)

David Sterratt

`orthographic` *Orthographic projection*

Description

Orthographic projection

Usage

```
orthographic(r, proj.centre = cbind(phi = 0, lambda = 0), ...)
```

Arguments

`r`	Latitude-longitude coordinates in a matrix with columns labelled `phi` (latitude) and `lambda` (longitude)
`proj.centre`	Location of centre of projection as matrix with column names `phi` (elevation) and `lambda` (longitude).
`...`	Arguments not used by this projection.n

Value

Two-column matrix with columns labelled `x` and `y` of locations of projection of coordinates on plane

Author(s)

David Sterratt

References

http://en.wikipedia.org/wiki/Map_projection, http://mathworld.wolfram.com/OrthographicProjection.html

`Outline` *Class containing basic information about flat outlines*

Description

An Outline has contains the polygon describing the outline and an image associated with the outline.

Super class

`retistruct::OutlineCommon` -> `Outline`

Public fields

`P` A N-by-2 matrix of points of the `Outline` arranged in anticlockwise order

`scale` The length of one unit of `P` in arbitrary units

`units` String giving units of scaled P, e.g. "um"

`gf` For each row of `P`, the index of `P` that is next in the outline travelling anticlockwise (forwards)

`gb` For each row of `P`, the index of `P` that is next in the outline travelling clockwise (backwards)

`h` For each row of `P`, the correspondence of that point (which will be to itself initially)

`im` An image as a `raster` object

Methods

Public methods:

- `Outline$new()`
- `Outline$getImage()`
- `Outline$replaceImage()`
- `Outline$mapFragment()`
- `Outline$mapPids()`
- `Outline$addPoints()`
- `Outline$getPoints()`
- `Outline$getPointsScaled()`
- `Outline$getRimSet()`
- `Outline$getOutlineSet()`
- `Outline$getOutlineLengths()`
- `Outline$addFeatureSet()`
- `Outline$clone()`

Method `new()`**:** Construct an outline object. This sanitises the input points `P`.

Usage:

```
Outline$new(P = NULL, scale = NA, im = NULL, units = NA)
```

Arguments:

`P` An N-by-2 matrix of points of the `Outline`

`scale` The length of one unit of `P` in arbitrary units

`im` The image as a `raster` object

`units` String giving units of scaled P, e.g. "um"

Method `getImage()`**:** Image accessor

Usage:

```
Outline$getImage()
```

Returns: An image as a `raster` object

Method `replaceImage()`**:** Image setter

Usage:

```
Outline$replaceImage(im)
```

Arguments:

`im` An image as a `raster` object

Method `mapFragment()`**:** Map the point IDs of a Fragment on the point IDs of this Outline

Usage:

```
Outline$mapFragment(fragment, pids)
```

Arguments:

`fragment` Fragment to map

`pids` Point IDs in Outline of points in Fragment

Method `mapPids()`**:** Map references to points

Usage:

```
Outline$mapPids(x, y, pids)
```

Arguments:

`x` References to point indices in source

`y` References to existing point indices in target

`pids` IDs of points in point register

Returns: New references to point indices in target

Method `addPoints()`**:** Add points to the outline register of points

Usage:

```
Outline$addPoints(P)
```

Arguments:

`P` 2 column matrix of points to add

Returns: The ID of each added point in the register. If points already exist a point will not be created in the register, but an ID will be returned

Method `getPoints()`**:** Get unscaled mesh points

Usage:

`Outline$getPoints()`

Returns: Matrix with columns `X` and `Y`

Method `getPointsScaled()`**:** Get scaled mesh points

Usage:

`Outline$getPointsScaled()`

Returns: Matrix with columns `X` and `Y` which is exactly `scale` times the matrix returned by `getPoints`

Method `getRimSet()`**:** Get set of points on rim

Usage:

`Outline$getRimSet()`

Returns: Vector of point IDs, i.e. indices of the rows in the matrices returned by `getPoints` and `getPointsScaled`

Method `getOutlineSet()`**:** Get points on the edge of the outline

Usage:

`Outline$getOutlineSet()`

Returns: Vector of points IDs on outline

Method `getOutlineLengths()`**:** Get lengths of edges of the outline

Usage:

`Outline$getOutlineLengths()`

Returns: Vector of lengths of edges connecting neighbouring points

Method `addFeatureSet()`**:** Add a FeatureSet, e.g. a PointSet or LandmarkSet

Usage:

`Outline$addFeatureSet(fs)`

Arguments:

`fs` FeatureSet to add

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`Outline$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`OutlineCommon` *Class containing functionality common to flat and reconstructed outlines*

Description

An OutlineCommon has functionality for retrieving sets of features (e.g. points or landmarks associated with an outline)

Public fields

`version` Version of reconstruction file data format

`featureSets` List of feature sets associated with the outline, which may be of various types, e.g. a PointSet or LandmarkSet

Methods

Public methods:

- `OutlineCommon$getFeatureSets()`
- `OutlineCommon$getFeatureSet()`
- `OutlineCommon$clearFeatureSets()`
- `OutlineCommon$getIDs()`
- `OutlineCommon$getFeatureSetTypes()`
- `OutlineCommon$clone()`

Method `getFeatureSets()`**:** Get all the feature sets

Usage:

`OutlineCommon$getFeatureSets()`

Returns: List of FeatureSets associated with the outline

Method `getFeatureSet()`**:** Get all feature sets of a particular type, e.g. PointSet or LandmarkSet

Usage:

`OutlineCommon$getFeatureSet(type)`

Arguments:

`type` The type of the feature set as a string

Returns: All FeatureSets of that type

Method `clearFeatureSets()`**:** Clear all feature sets from the outline

Usage:

`OutlineCommon$clearFeatureSets()`

Method `getIDs()`**:** Get all the distinct IDs contained in the FeatureSets

Usage:

`OutlineCommon$getIDs()`

Returns: Vector of IDs

Method `getFeatureSetTypes()`**:** Get all the distinct types of FeatureSets

Usage:

`OutlineCommon$getFeatureSetTypes()`

Returns: Vector of types as strings, e.g. *PointSet*, *LandmarkSet*

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`OutlineCommon$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

`panlabel` *Ancillary function to place labels*

Description

Ancillary function to place labels

Usage

```
panlabel(panlabel, line = -0.7)
```

Arguments

`panlabel`	Label text
`line`	Line on which to appear

Author(s)

David Sterratt

parabola.arclength *Arc length of a parabola y=x^2/4f*

Description

Arc length of a parabola y=x^2/4f

Usage

```
parabola.arclength(x1, x2, f)
```

Arguments

x1	x co-ordinate of start of arc
x2	x co-ordinate of end of arc
f	focal length of parabola

Value

length of parabola arc

Author(s)

David Sterratt

parabola.invarclength *Inverse arc length of a parabola y=x^2/4f*

Description

Inverse arc length of a parabola y=x^2/4f

Usage

```
parabola.invarclength(x1, s, f)
```

Arguments

x1	co-ordinate of start of arc
s	length of parabola arc to follow
f	focal length of parabola

Value

x co-ordinate of end of arc

Author(s)

David Sterratt

`parse.dependencies` *Parse dependencies*

Description

Parse dependencies

Usage

```
parse.dependencies(deps)
```

Arguments

`deps`	Text produced by, e.g., `installed.packages()["packagename","Suggests"]`

Value

Table with package column, relationship column and version number

Author(s)

David Sterratt

`PathOutline` *Add point correspondences to the outline*

Description

Add point correspondences to the outline

Add point correspondences to the outline

Details

The member function `stitchSubpaths()` stitches together two subpaths of the outline. One subpath is stitched in the forward direction from the point indexed by `VF0` to the point indexed by `VF1`. The other is stitched in the backward direction from `VB0` to `VB1`. Each point in the subpath is linked to points in the opposing pathway at an equal or near-equal fraction along. If a point exists in the opposing pathway within a distance `epsilon` of the projection, this point is connected. If no point exists within this tolerance, a new point is created.

Value

To the `Outline` object this adds

`hf` point correspondence mapping in forward direction for points on boundary

`hb` point correspondence mapping in backward direction for points on boundary

Super classes

`retistruct::OutlineCommon` -> `retistruct::Outline` -> `PathOutline`

Public fields

`hf` Forward correspondences

`hb` Backward correspondences

Methods

Public methods:

- `PathOutline$addPoints()`
- `PathOutline$nextPoint()`
- `PathOutline$insertPoint()`
- `PathOutline$stitchSubpaths()`
- `PathOutline$clone()`

Method `addPoints()`**:** Add points to the outline register of points

Usage:

```
PathOutline$addPoints(P)
```

Arguments:

`P` 2 column matrix of points to add

Returns: The ID of each added point in the register. If points already exist a point will not be created in the register, but an ID will be returned

Method `nextPoint()`**:** Get next point in path for

Usage:

```
PathOutline$nextPoint(pids)
```

Arguments:

`pids` Point IDs of points to get next position

Method `insertPoint()`**:** Insert point at a fractional distance between points

Usage:

```
PathOutline$insertPoint(i0, i1, f)
```

Arguments:

`i0` Point ID of first point

`i1` Point ID of second point

`f` Fraction of distance between points `i0` and `i1` at which to insert point

Method `stitchSubpaths()`**:** Stitch subpaths

Usage:

```
PathOutline$stitchSubpaths(VF0, VF1, VB0, VB1, epsilon)
```

Arguments:

`VF0` First vertex of “forward” subpath

`VF1` Second vertex of “forward” subpath

`VB0` First vertex of “backward” subpath

`VB1` Second vertex of “backward” subpath

`epsilon` Minimum distance between points

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
PathOutline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`PointSet` *Subclass of* `FeatureSet` *to represent points*

Description

A PointSet contains information about points located on `Outline`s. Each PointSet contains a list of matrices, each of which has columns labelled `X` and `Y` describing the cartesian coordinates (in the unscaled coordinate frame) of points on the Outline.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::FeatureSet` -> `PointSet`

Methods

Public methods:

- `PointSet$new()`
- `PointSet$reconstruct()`
- `PointSet$clone()`

Method `new()`**:** Constructor

Usage:

```
PointSet$new(data = NULL, cols = NULL)
```

Arguments:

`data` List of matrices describing data. Each matrix should have columns named `X` and `Y` `cols` Named vector of colours for each data set. The name is used as the ID (label) for the data set. The colours should be names present in the output of the `colors` function

Method `reconstruct()`**:** Map the PointSet to a `ReconstructedOutline`

Usage:

`PointSet$reconstruct(ro)`

Arguments:

`ro` The `ReconstructedOutline`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`PointSet$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`polar.cart.to.sphere.spherical`
Convert polar projection in Cartesian coordinates to spherical coordinates on sphere

---

Description

This is the inverse of `sphere.spherical.to.polar.cart`

Usage

```
polar.cart.to.sphere.spherical(r, pa = FALSE, preserve = "latitude")
```

Arguments

`r`	2-column Matrix of Cartesian coordinates of points on polar projection. Column names should be `x` and `y`
`pa`	If `TRUE`, make this an area-preserving projection
`preserve`	Quantity to preserve locally in the projection. Options are `latitude`, `area` or `angle`

Value

2-column Matrix of spherical coordinates of points on sphere. Column names are `phi` and `lambda`.

Author(s)

David Sterratt

polartext *Put text on the polar plot*

Description

Place text at bottom right of `projection`

Usage

```
polartext(text)
```

Arguments

`text`	Test to place

Author(s)

David Sterratt

projection *Plot projection of a reconstructed outline*

Description

Plot projection of a reconstructed outline

Usage

```
projection(r, ...)
```

Arguments

`r`	Object such as a `ReconstructedOutline`
`...`	Other plotting parameters

Author(s)

David Sterratt

`projection.ReconstructedOutline` *Projection of a reconstructed outline*

Description

Draw a projection of a `ReconstructedOutline`. This method sets up the grid lines and the angular labels and draws the image.

Usage

```
## S3 method for class 'ReconstructedOutline'
projection(
  r,
  transform = identity.transform,
  axisdir = cbind(phi = 90, lambda = 0),
  projection = azimuthal.equalarea,
  proj.centre = cbind(phi = 0, lambda = 0),
  lambdalim = c(-180, 180),
  philim = c(-90, 90),
  labels = c(0, 90, 180, 270),
  mesh = FALSE,
  grid = TRUE,
  grid.bg = "transparent",
  grid.int.minor = 15,
  grid.int.major = 45,
  colatitude = TRUE,
  pole = FALSE,
  image = TRUE,
  markup = TRUE,
  add = FALSE,
  max.proj.dim = getOption("max.proj.dim"),
  ...
)
```

Arguments

`r`	`ReconstructedOutline` object
`transform`	Transform function to apply to spherical coordinates before rotation
`axisdir`	Direction of axis (North pole) of sphere in external space as matrix with column names `phi` (elevation) and `lambda` (longitude).
`projection`	Projection in which to display object, e.g. `azimuthal.equalarea` or `sinusoidal`
`proj.centre`	Location of centre of projection as matrix with column names `phi` (elevation) and `lambda` (longitude).
`lambdalim`	Limits of longitude (in degrees) to display

philim	Limits of latitude (in degrees) to display
labels	Vector of 4 labels to plot at 0, 90, 180 and 270 degrees
mesh	If TRUE, plot mesh
grid	Whether or not to show the grid lines of latitude and longitude
grid.bg	Background colour of the grid
grid.int.minor	Interval between minor grid lines in degrees
grid.int.major	Interval between major grid lines in degrees
colatitude	If TRUE have radial labels plotted with respect to colatitude rather than latitude
pole	If TRUE indicate the pole with a "*"
image	If TRUE, show the image
markup	If TRUE, plot markup, i.e. reconstructed tears
add	If TRUE, don't draw axes; add to existing plot.
max.proj.dim	Maximum width of the image created in pixels
...	Graphical parameters to pass to plotting functions

---

projection.RetinalReconstructedOutline
Plot projection of reconstructed dataset

Description

Plot projection of reconstructed dataset

Usage

```
## S3 method for class 'RetinalReconstructedOutline'
projection(
  r,
  transform = identity.transform,
  projection = azimuthal.equalarea,
  axisdir = cbind(phi = 90, lambda = 0),
  proj.centre = cbind(phi = 0, lambda = 0),
  lambdalim = c(-180, 180),
  datapoints = TRUE,
  datapoint.means = TRUE,
  datapoint.contours = FALSE,
  grouped = FALSE,
  grouped.contours = FALSE,
  landmarks = TRUE,
  mesh = FALSE,
  grid = TRUE,
  image = TRUE,
  ids = r$getIDs(),
  ...
)
```

Arguments

- `r` — RetinalReconstructedOutline object
- `transform` — Transform function to apply to spherical coordinates before rotation
- `projection` — Projection in which to display object, e.g. azimuthal.equalarea or sinusoidal
- `axisdir` — Direction of axis (North pole) of sphere in external space
- `proj.centre` — Location of centre of projection as matrix with column names phi (elevation) and lambda (longitude).
- `lambdalim` — Limits of longitude (in degrees) to display
- `datapoints` — If TRUE, display data points
- `datapoint.means` — If TRUE, display Karcher mean of data points.
- `datapoint.contours` — If TRUE, display contours around the data points generated using Kernel Density Estimation.
- `grouped` — If TRUE, display grouped data.
- `grouped.contours` — If TRUE, display contours around the grouped data generated using Kernel Regression.
- `landmarks` — If TRUE, display landmarks.
- `mesh` — If TRUE, display the triangular mesh used in reconstruction
- `grid` — If TRUE, show grid lines
- `image` — If TRUE, show the reconstructed image
- `ids` — IDs of groups of data within a dataset, returned using getIDs.
- `...` — Graphical parameters to pass to plotting functions

R6_to_list — *Convert an R6 object into a list, ignoring functions and environments*

Description

Convert an R6 object into a list, ignoring functions and environments

Usage

```
R6_to_list(r, path = "", envs = list())
```

Arguments

- `r` — R6 object or list
- `path` — root of the path to the list - no need to supply. Not used but could be developed for pretty-printing
- `envs` — list of environments already encountered - do not set

Value

List with structure mirroring the R6 object.

Author(s)

David Sterratt

Rcart — *Restore points to spherical manifold*

Description

Restore points to spherical manifold after an update of the Lagrange integration rule

Usage

```
Rcart(P, R, Rset, i0, phi0, lambda0)
```

Arguments

`P`	Point positions as N-by-3 matrix
`R`	Radius of sphere
`Rset`	Indices of points on rim
`i0`	Index of fixed point
`phi0`	Cut-off of curtailed sphere in radians
`lambda0`	Longitude of fixed point on rim

Value

Points projected back onto sphere

Author(s)

David Sterratt

read.datacounts — *Read data counts in CSV format*

Description

Read data counts from a file 'datacounts.csv' in the directory dataset. The CSV file should contain two columns for every dataset. Each pair of columns must contain a unique name in the first cell of the first row and a valid colour in the second cell of the first row. In the remaining rows, the X coordinates of data counts should be in the first column and the Y coordinates should be in the second column.

Usage

```
read.datacounts(dataset)
```

Arguments

dataset	Path to directory containing dataponts.csv

Value

List containing

Ds	List of sets of data counts. Each set comprises a 2-column matrix and each set is named.
cols	List of colours for each dataset. There is one element that corresponds to each element of Ds and which bears the same name.

Author(s)

David Sterratt

read.datapoints — *Read data points in CSV format*

Description

Read data points from a file dataponts.csv in the directory dataset. The CSV should contain two columns for every dataset. Each pair of columns must contain a unique name in the first cell of the first row and a valid colour in the second cell of the first row. In the remaining rows, the X coordinates of data points should be in the first column and the Y coordinates should be in the second column.

Usage

```
read.datapoints(dataset)
```

Arguments

`dataset` Path to directory containing `dataponts.csv`

Value

List containing

`Ds` List of sets of datapoints. Each set comprises a 2-column matrix and each set is named.

`cols` List of colours for each dataset. There is one element that corresponds to each element of `Ds` and which bears the same name.

Author(s)

David Sterratt

`ReconstructedCountSet` *Class containing functions and data to map CountSets to ReconstructedOutlines*

Description

A ReconstructedCountSet contains information about features located on `ReconstructedOutline`s. Each ReconstructedCountSet contains a list of matrices, each of which has columns labelled `phi` (latitude) and `lambda` (longitude) describing the spherical coordinates of points on the ReconstructedOutline, and a column `C` representing the counts at these points.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::ReconstructedFeatureSet` -> `ReconstructedCountSet`

Public fields

`KR` Kernel regression

Methods

Public methods:

- `ReconstructedCountSet$new()`
- `ReconstructedCountSet$getKR()`
- `ReconstructedCountSet$clone()`

Method `new()`**:** Constructor

Usage:

```
ReconstructedCountSet$new(fs = NULL, ro = NULL)
```

Arguments:

`fs` FeatureSet to reconstruct

`ro` ReconstructedOutline to which feature set should be mapped

Method `getKR()`**:** Get kernel regression estimate of grouped data points

Usage:

```
ReconstructedCountSet$getKR()
```

Returns: Kernel regression computed using `compute.kernel.estimate`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReconstructedCountSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`ReconstructedFeatureSet` *Class containing functions and data to map FeatureSets to ReconstructedOutlines*

Description

A ReconstructedFeatureSet contains information about features located on `ReconstructedOutline`s. Each ReconstructedFeatureSet contains a list of matrices, each of which has columns labelled `phi` (latitude) and `lambda` (longitude) describing the spherical coordinates of points on the ReconstructedOutline. Derived classes, e.g. a `ReconstructedCountSet`, may have extra columns. Each matrix in the list has an associated label and colour, which is used by plotting functions.

Super class

`retistruct::FeatureSetCommon` -> `ReconstructedFeatureSet`

Methods

Public methods:

- `ReconstructedFeatureSet$new()`
- `ReconstructedFeatureSet$clone()`

Method `new()`**:** Constructor

Usage:

```
ReconstructedFeatureSet$new(fs = NULL, ro = NULL)
```

Arguments:

`fs` FeatureSet to reconstruct

`ro` ReconstructedOutline to which feature set should be mapped

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReconstructedFeatureSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`ReconstructedLandmarkSet`

Class containing functions and data to map LandmarkSets to ReconstructedOutlines

---

Description

A ReconstructedLandmarkSet contains information about features located on `ReconstructedOutline`s. Each ReconstructedLandmarkSet contains a list of matrices, each of which has columns labelled `phi` (latitude) and `lambda` (longitude) describing the spherical coordinates of points on the ReconstructedOutline.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::ReconstructedFeatureSet` -> `ReconstructedLandmarkSet`

Methods

Public methods:

- `ReconstructedLandmarkSet$clone()`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReconstructedLandmarkSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`ReconstructedOutline` *Class containing functions to reconstruct StitchedOutlines and store the associated data*

Description

The function `reconstruct` reconstructs outline into spherical surface Reconstruct outline into spherical surface.

Super class

`retistruct::OutlineCommon` -> `ReconstructedOutline`

Public fields

- `ol` Annotated outline
- `ol0` Original Annotated outline
- `Pt` Transformed cartesian mesh points
- `Tt` Transformed triangulation
- `Ct` Transformed links
- `Cut` Transformed links
- `Bt` Transformed binary vector representation of edge indices onto a binary vector representation of the indices of the points linked by the edge
- `Lt` Transformed lengths
- `ht` Transformed correspondences
- `u` Indices of unique points in untransformed space
- `U` Transformed indices of unique points in untransformed space
- `Rsett` Transformed rim set
- `i0t` Transformed marker
- `H` mapping from edges onto corresponding edges
- `Ht` Transformed mapping from edges onto corresponding edges
- `phi0` Rim angle
- `R` Radius of spherical template
- `lambda0` Longitude of pole on rim
- `lambda` Longitudes of transformed mesh points
- `phi` Latitudes of transformed mesh points
- `Ps` Location of mesh point on sphere in spherical coordinates
- `n` Number of mesh points
- `alpha` Weighting of areas in energy function
- `x0` Area cut-off coefficient

`nflip0` Initial number flipped triangles

`nflip` Final number flipped triangles

`opt` Optimisation object

`E.tot` Energy function including area

`E.l` Energy function based on lengths alone

`mean.strain` Mean strain

`mean.logstrain` Mean log strain

`debug` Debug function

Methods

Public methods:

- `ReconstructedOutline$loadOutline()`
- `ReconstructedOutline$reconstruct()`
- `ReconstructedOutline$mergePointsEdges()`
- `ReconstructedOutline$projectToSphere()`
- `ReconstructedOutline$getStrains()`
- `ReconstructedOutline$optimiseMapping()`
- `ReconstructedOutline$optimiseMappingCart()`
- `ReconstructedOutline$transformImage()`
- `ReconstructedOutline$getIms()`
- `ReconstructedOutline$getTearCoords()`
- `ReconstructedOutline$getFeatureSet()`
- `ReconstructedOutline$reconstructFeatureSets()`
- `ReconstructedOutline$getPoints()`
- `ReconstructedOutline$mapFlatToSpherical()`
- `ReconstructedOutline$clone()`

Method `loadOutline()`**:** Load AnnotatedOutline into ReconstructedOutline object

Usage:

```
ReconstructedOutline$loadOutline(
  ol,
  n = 500,
  alpha = 8,
  x0 = 0.5,
  plot.3d = FALSE,
  dev.flat = NA,
  dev.polar = NA,
  report = retistruct::report,
  debug = FALSE
)
```

Arguments:

`ol` `AnnotatedOutline` object, containing the following information `n` Number of points in triangulation.

`alpha` Area scaling coefficient

`x0` Area cut-off coefficient

`plot.3d` Whether to show 3D picture during optimisation.

`dev.flat` Device to plot grid onto. Value of `NA` (default) means no plotting.

`dev.polar` Device display projection. Value of NA (default) means no plotting.

`report` Function to report progress.

`debug` If `TRUE` print extra debugging output

Method `reconstruct()`**:** Reconstruct Reconstruction proceeds in a number of stages:

1. The flat object is triangulated with at least `n` triangles. This can introduce new vertices in the rim.
2. The triangulated object is stitched.
3. The stitched object is triangulated again, but this time it is not permitted to add extra vertices to the rim.
4. The corresponding points determined by the stitching process are merged to form a new set of merged points and a new triangulation.
5. The merged points are projected roughly to a sphere.
6. The locations of the points on the sphere are moved so as to minimise the energy function.

Usage:

```
ReconstructedOutline$reconstruct(
  plot.3d = FALSE,
  dev.flat = NA,
  dev.polar = NA,
  report = getOption("retistruct.report")
)
```

Arguments:

`plot.3d` If `TRUE` make a 3D plot in an RGL window

`dev.flat` Device handle for plotting flatplot updates to. If `NA` don't make any flat plots

`dev.polar` Device handle for plotting polar plot updates to. If `NA` don't make any polar plots.

`report` Function to report progress.

`Control` argument to pass to `optim`

Method `mergePointsEdges()`**:** Merge stitched points and edges. Create merged and transformed versions (all suffixed with `t`) of a number of existing variables, as well as a matrix `Bt`, which maps a binary vector representation of edge indices onto a binary vector representation of the indices of the points linked by the edge. Sets following fields

- `Pt`Transformed point locations
- `Tt`Transformed triangulation
- `Ct`Transformed connection set
- `Cut`Transformed symmetric connection set
- `Bt`Transformed binary vector representation of edge indices onto a binary vector representation of the indices of the points linked by the edge
- `Lt`Transformed edge lengths

- `ht`Transformed correspondences
- `u`Indices of unique points in untransformed space
- `U`Transformed indices of unique points in untransformed space
- `Rset`The set of points on the rim (which has been reordered)
- `Rsett`Transformed set of points on rim
- `i0t`Transformed index of the landmark
- `H`mapping from edges onto corresponding edges
- `Ht`Transformed mapping from edges onto corresponding edges

Usage:

```
ReconstructedOutline$mergePointsEdges()
```

Method `projectToSphere()`**:** Project mesh points in the flat outline onto a sphere This takes the mesh points from the flat outline and maps them to the curtailed sphere. It uses the area of the flat outline and `phi0` to determine the radius `R` of the sphere. It tries to get a good first approximation by using the function `stretchMesh`. The following fields are set:

- `phi`Latitude of mesh points.
- `lmabda`Longitude of mesh points.
- `R`Radius of sphere.

Usage:

```
ReconstructedOutline$projectToSphere()
```

Method `getStrains()`**:** Return strains edges are under in spherical retina Set information about how edges on the sphere have been deformed from their flat state.

Usage:

```
ReconstructedOutline$getStrains()
```

Returns: A list containing two data frames `flat` and `spherical`. Each data frame contains for each edge in the flat or spherical meshes:

- `L`Length of the edge in the flat outline
- `l`Length of the corresponding edge on the sphere
- `strain`The strain of each connection
- `logstrain`The logarithmic strain of each connection

Method `optimiseMapping()`**:** Optimise the mapping from the flat outline to the sphere

Usage:

```
ReconstructedOutline$optimiseMapping(
  alpha = 4,
  x0 = 0.5,
  nu = 1,
  optim.method = "BFGS",
  plot.3d = FALSE,
  dev.flat = NA,
  dev.polar = NA,
  control = list()
)
```

Arguments:

`alpha` Area penalty scaling coefficient

`x0` Area penalty cut-off coefficient

`nu` Power to which to raise area

`optim.method` Method to pass to `optim`

`plot.3d` If `TRUE` make a 3D plot in an RGL window

`dev.flat` Device handle for plotting flatplot updates to. If `NA` don't make any flat plots

`dev.polar` Device handle for plotting polar plot updates to. If `NA` don't make any polar plots.

`control` Control argument to pass to `optim`

Method `optimiseMappingCart()`**:** Optimise the mapping from the flat outline to the sphere

Usage:

```
ReconstructedOutline$optimiseMappingCart(
  alpha = 4,
  x0 = 0.5,
  nu = 1,
  method = "BFGS",
  plot.3d = FALSE,
  dev.flat = NA,
  dev.polar = NA,
  ...
)
```

Arguments:

`alpha` Area penalty scaling coefficient

`x0` Area penalty cut-off coefficient

`nu` Power to which to raise area

`method` Method to pass to `optim`

`plot.3d` If `TRUE` make a 3D plot in an RGL window

`dev.flat` Device handle for plotting grid to

`dev.polar` Device handle for plotting polar plot to

`...` Extra arguments to pass to `fire`

Method `transformImage()`**:** Transform an image into the reconstructed space Transform an image into the reconstructed space. The four corner coordinates of each pixel are transformed into spherical coordinates and a mask matrix with the same dimensions as `im` is created. This has `TRUE` for pixels that should be displayed and `FALSE` for ones that should not. Sets the field

- `ims`Coordinates of corners of pixels in spherical coordinates

Usage:

`ReconstructedOutline$transformImage()`

Method `getIms()`**:** Get coordinates of corners of pixels of image in spherical coordinates

Usage:

`ReconstructedOutline$getIms()`

Returns: Coordinates of corners of pixels in spherical coordinates

Method `getTearCoords()`: Get location of tear coordinates in spherical coordinates

Usage:

```
ReconstructedOutline$getTearCoords()
```

Returns: Location of tear coordinates in spherical coordinates

Method `getFeatureSet()`: Get ReconstructedFeatureSet

Usage:

```
ReconstructedOutline$getFeatureSet(type)
```

Arguments:

`type` Base type of FeatureSet as string. E.g. `PointSet` returns a ReconstructedPointSet

Method `reconstructFeatureSets()`: Reconstruct any attached feature sets.

Usage:

```
ReconstructedOutline$reconstructFeatureSets()
```

Method `getPoints()`: Get mesh points in spherical coordinates

Usage:

```
ReconstructedOutline$getPoints()
```

Returns: Matrix with columns `phi` (latitude) and `lambda` (longitude)

Method `mapFlatToSpherical()`: Return location of point on sphere corresponding to point on the flat outline

Usage:

```
ReconstructedOutline$mapFlatToSpherical(P)
```

Arguments:

`P` Cartesian coordinates on flat outline as a matrix with `X` and `Y` columns

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ReconstructedOutline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

ReconstructedPointSet *Class containing functions and data to map PointSets to ReconstructedOutlines*

Description

A ReconstructedPointSet contains information about features located on ReconstructedOutlines. Each ReconstructedPointSet contains a list of matrices, each of which has columns labelled phi (latitude) and lambda (longitude) describing the spherical coordinates of points on the ReconstructedOutline.

Super classes

`retistruct::FeatureSetCommon` -> `retistruct::ReconstructedFeatureSet` -> `ReconstructedPointSet`

Public fields

`KDE` Kernel density estimate, computed using `compute.kernel.estimate` in `getKDE`

Methods

Public methods:

- `ReconstructedPointSet$getMean()`
- `ReconstructedPointSet$getHullarea()`
- `ReconstructedPointSet$getKDE()`
- `ReconstructedPointSet$clone()`

Method `getMean()`**:** Get Karcher mean of datapoints in spherical coordinates

Usage:

```
ReconstructedPointSet$getMean()
```

Returns: Karcher mean of datapoints in spherical coordinates

Method `getHullarea()`**:** Get area of convex hull around data points on sphere

Usage:

```
ReconstructedPointSet$getHullarea()
```

Returns: Area in degrees squared

Method `getKDE()`**:** Get kernel density estimate of data points

Usage:

```
ReconstructedPointSet$getKDE()
```

Returns: See `compute.kernel.estimate`

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ReconstructedPointSet$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`remove.identical.consecutive.rows` *Remove identical consecutive rows from a matrix*

Description

This is similar to unique(), but spares rows which are duplicated, but at different points in the matrix

Usage

```
remove.identical.consecutive.rows(P)
```

Arguments

`P`	Source matrix

Value

Matrix with identical consecutive rows removed.

Author(s)

David Sterratt

`remove.intersections` *Remove intersections between adjacent segments in a closed path*

Description

Suppose segments AB and CD intersect. Point B is replaced by the intersection point, defined B'. Point C is replaced by a point C' on the line B'D. The maximum distance of B'C' is given by the parameter d. If the distance l B'D is less than 2d, the distance B'C' is l/2.

Usage

```
remove.intersections(P, d = 50)
```

Arguments

`P`	The points, as a 2-column matrix
`d`	Criterion for maximum distance when points are inserted

Value

A new closed path without intersections

Author(s)

David Sterratt

`report` *Reporting utility function*

Description

Calls function specified by option `retistruct.report`

Usage

```
report(...)
```

Arguments

`...`	Arguments to reporting function

Author(s)

David Sterratt

`RetinalOutline` *Class containing functions and data relating to retinal outlines*

Description

In addition to fields inherited from StitchedOutline, a RetinalOutline contains a `dataset` field, describing the system path to dataset directory and metadata specific to retinae and some formats of retinae

An `retinalOutline` object. This contains the following fields:

Super classes

`retistruct::OutlineCommon` -> `retistruct::Outline` -> `retistruct::PathOutline` -> `retistruct::AnnotatedOutl` -> `retistruct::TriangulatedOutline` -> `retistruct::StitchedOutline` -> `RetinalOutline`

Public fields

`DVflip` `TRUE` if the raw data is flipped in the dorsoventral direction

`side` The side of the eye (“Left” or “Right”)

`dataset` File system path to dataset directory

Methods

Public methods:

- `RetinalOutline$new()`
- `RetinalOutline$clone()`

Method `new()`**:** Constructor

Usage:

```
RetinalOutline$new(..., dataset = NULL)
```

Arguments:

`...` Parameters to superclass constructors

`dataset` File system path to dataset directory

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RetinalOutline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`RetinalReconstructedOutline`
A version of ReconstructedOutline that is specific to retinal datasets

---

Description

A RetinalReconstructedOutline overrides methods of ReconstructedOutline so that they return data point and landmark coordinates that have been transformed according to the values of `DVflip` and `side`. When reconstructing, it also computes the “Optic disc displacement”, i.e. the number of degrees subtended between the optic disc and the pole.

Super classes

`retistruct::OutlineCommon` -> `retistruct::ReconstructedOutline` -> `RetinalReconstructedOutline`

Public fields

`EOD` Optic disc displacement in degrees

Methods

Public methods:

- `RetinalReconstructedOutline$getIms()`
- `RetinalReconstructedOutline$getTearCoords()`
- `RetinalReconstructedOutline$reconstruct()`
- `RetinalReconstructedOutline$getFeatureSet()`
- `RetinalReconstructedOutline$clone()`

Method `getIms()`**:** Get coordinates of corners of pixels of image in spherical coordinates, transformed according to the value of `DVflip`

Usage:

```
RetinalReconstructedOutline$getIms()
```

Returns: Coordinates of corners of pixels in spherical coordinates

Method `getTearCoords()`**:** Get location of tear coordinates in spherical coordinates, transformed according to the value of `DVflip`

Usage:

```
RetinalReconstructedOutline$getTearCoords()
```

Returns: Location of tear coordinates in spherical coordinates

Method `reconstruct()`**:**

Usage:

```
RetinalReconstructedOutline$reconstruct(...)
```

Arguments:

`...` Parameters to `ReconstructedOutline`

Method `getFeatureSet()`**:** Get ReconstructedFeatureSet, transformed according to the value of `DVflip`

Usage:

```
RetinalReconstructedOutline$getFeatureSet(type)
```

Arguments:

`type` Base type of FeatureSet as string. E.g. `PointSet` returns a ReconstructedPointSet

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RetinalReconstructedOutline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`retistruct` *Start the Retistruct GUI*

Description

Start the Retistruct GUI

Usage

```
retistruct()
```

Value

Object with `getData()` method to return reconstructed retina data and environment `this` which contains variables in object.

See Also

gWidgets2

`retistruct.batch` *Batch operation using the parallel package*

Description

This function reconstructs a number of datasets, using the R `parallel` package to distribute the reconstruction of multiple datasets across CPUs. If `datasets` is not specified the function recurses through a directory tree starting at `tldir`, determining whether the directory contains valid raw data and markup, and performing the reconstruction if it does.

Usage

```
retistruct.batch(
  tldir = ".",
  outputdir = tldir,
  datasets = NULL,
  device = "pdf",
  titrate = FALSE,
  cpu.time.limit = 3600,
  mc.cores = getOption("mc.cores", 2L)
)
```

Arguments

`tldir`	If datasets is not specified, the top level of the directory tree through which to recurse in order to find datasets.
`outputdir`	directory in which to dump a log file and images
`datasets`	Vector of dataset directories to reconstruct
`device`	string indicating what type of graphics output required. Options are `"pdf"` and `"png"`.
`titrate`	Whether to "titrate" the reconstruction for different values of `phi0`. See `titrate.reconstructedOutline`
`cpu.time.limit`	amount of CPU after which to terminate the process
`mc.cores`	The number of cores to use. Defaults to the value given by the option `mc.cores`

Author(s)

David Sterratt

`retistruct.batch.analyse.summaries`

Extract statistics from a directory containing reconstruction directories.

Description

Extract statistics from a directory containing reconstruction directories.

Usage

```
retistruct.batch.analyse.summaries(path)
```

Arguments

`path`	Directory containing reconstruction directories

Value

Data frame containing various statistics

Author(s)

David Sterratt

`retistruct.batch.analyse.summary`
Extract statistics from the retistruct-batch.csv summary file

---

Description

Extract statistics from the retistruct-batch.csv summary file

Usage

```
retistruct.batch.analyse.summary(path)
```

Arguments

`path`	The path to the retistruct-batch.csv

Value

list of various statistics

Author(s)

David Sterratt

`retistruct.batch.export.matlab`
Export data from reconstruction data files to MATLAB

Description

Recurse through a directory tree, determining whether the directory contains valid derived data and converting '`r.rData`' files to files in MATLAB format named '`r.mat`'

Usage

```
retistruct.batch.export.matlab(tldir = ".")
```

Arguments

`tldir`	The top level of the directory tree through which to recurse

Author(s)

David Sterratt

retistruct.batch.figures

Plot figures for a batch of reconstructions

Description

Recurse through a directory tree, determining whether the directory contains valid derived data and plotting graphs if it does.

Usage

```
retistruct.batch.figures(tldir = ".", outputdir = tldir, ...)
```

Arguments

`tldir`	The top level directory of the tree through which to recurse.
`outputdir`	Directory in which to dump a log file and images
`...`	Parameters passed to plotting functions

Author(s)

David Sterratt

retistruct.batch.get.titrations

Get titrations from a directory of reconstructions

Description

Get titrations from a directory of reconstructions

Usage

```
retistruct.batch.get.titrations(tldir = ".")
```

Arguments

`tldir`	The top level directory of the tree through which to recurse. The files have to have been reconstructed with the `titrate` option to `retistruct.batch`

`retistruct.batch.plot.titrations`
Plot titrations

Description

Plot titrations

Usage

```
retistruct.batch.plot.titrations(tdat)
```

Arguments

`tdat`	Output of `retistruct.batch.get.titrations`

`retistruct.batch.summary`
Extract summary data for a batch of reconstructions

Description

Recurse through a directory tree, determining whether the directory contains valid derived data and extracting summary data if it does.

Usage

```
retistruct.batch.summary(tldir = ".", cache = TRUE)
```

Arguments

`tldir`	The top level directory of the tree through which to recurse.
`cache`	If `TRUE` use the cached statistics rather than generate on the fly (which is slower).

Value

Data frame containing summary data

Author(s)

David Sterratt

`retistruct.check.markup` *Retistruct check markup*

Description

Check that markup such as tears and the nasal or dorsal points are present.

Usage

```
retistruct.check.markup(o)
```

Arguments

`o`	Outline object

Value

If all markup is present, return `TRUE`. Otherwise return `FALSE`.

Author(s)

David Sterratt

`retistruct.cli` *Process a dataset with a time limit*

Description

This calls `retistruct.cli.process` with a time limit specified by `cpu.time.limit`.

Usage

```
retistruct.cli(
  dataset,
  cpu.time.limit = Inf,
  outputdir = NA,
  device = "pdf",
  ...
)
```

Arguments

`dataset`	Path to dataset to process
`cpu.time.limit`	Time limit in seconds
`outputdir`	Directory in which to save any figures
`device`	String representing device to print figures to
`...`	Other arguments to pass to `retistruct.cli.process`

Value

A list comprising

`status`	0 for success, 1 for reaching `cpu.time.limit` and 2 for an unknown error
`time`	The time take in seconds
`mess`	Any error message

Author(s)

David Sterratt

`retistruct.cli.figure` *Print a figure to file*

Description

Print a figure to file

Usage

```
retistruct.cli.figure(
  dataset,
  outputdir,
  device = "pdf",
  width = 6,
  height = 6,
  res = 100
)
```

Arguments

`dataset`	Path to dataset to process
`outputdir`	Directory in which to save any figures
`device`	String representing device to print figures to
`width`	Width of figures in inches
`height`	Height of figures in inches
`res`	Resolution of figures in dpi (only applies to bitmap devices)

Author(s)

David Sterratt

---

retistruct.cli.process

Process a dataset, saving results to disk

Description

This function processes a `dataset`, saving the reconstruction data and MATLAB export data to the `dataset` directory and printing figures to `outputdir`.

Usage

```
retistruct.cli.process(dataset, outputdir = NA, device = "pdf")
```

Arguments

`dataset`	Path to dataset to process
`outputdir`	Directory in which to save any figures
`device`	String representing device to print figures to

Author(s)

David Sterratt

---

retistruct.export.matlab

Save reconstruction data in MATLAB format

Description

Save as a MATLAB object certain fields of an object `r` of class`RetinalReconstructedOutline` to a file called `r.mat` in the directory `r$dataset`.

Usage

```
retistruct.export.matlab(r, filename = NULL)
```

Arguments

`r`	`RetinalReconstructedOutline` object
`filename`	Filename of output file. If not specified, is `r.mat` in the same directory as the input files

Author(s)

David Sterratt

`retistruct.read.dataset`
Read a retinal dataset

Description

Read a retinal dataset in one of three formats; for information on formats see `idt.read.dataset`, `csv.read.dataset` and `ijroi.read.dataset`. The format is autodetected from the files in the directory.

Usage

```
retistruct.read.dataset(dataset, report = message, ...)
```

Arguments

`dataset`	Path to directory containing the files corresponding to each format.
`report`	Function to report progress. Set to `FALSE` for no reporting.
`...`	Parameters passed to the format-specific functions.

Value

A `RetinalOutline` object

Author(s)

David Sterratt

`retistruct.read.markup`
Read the markup data

Description

Read the markup data contained in the files '`markup.csv`', '`P.csv`' and '`T.csv`' in the directory '`dataset`', which is specified in the reconstruction object `r`.

Usage

```
retistruct.read.markup(a, error = stop)
```

Arguments

`a`	Dataset object, containing `dataset` path
`error`	Function to run on error, by default `stop()`

Details

The tear information is contained in the files 'P.csv' and 'T.csv'. The first file contains the locations of outline points that the tears were marked up on. The second file contains the indices of the apex and backward and forward vertices of each tear. It is necessary to have the file of points just in case the algorithm that determines P in `retistruct.read.dataset` has changed since the markup of the tears.

The remaining information is contained in the file 'markup.csv'.

If DVflip is specified, the locations of points P flipped in the y-direction. This operation also requires the swapping of gf and gb and VF and VB.

Value

o RetinalDataset object

V0	Indices in P of apices of tears
VB	Indices in P of backward vertices of tears
VF	Indices in P of backward vertices of tears
iN	Index in P of nasal point, or NA if not marked
iD	Index in P of dorsal point, or NA if not marked
iOD	Index in Ss of optic disc
phi0	Angle of rim in degrees
DVflip	Boolean variable indicating if dorsoventral (DV) axis has been flipped

Author(s)

David Sterratt

---

retistruct.read.recdata

Read the reconstruction data from file

Description

Given an outline object with a dataset field, read the reconstruction data from the file '*dataset*/r.Rdata'.

Usage

```
retistruct.read.recdata(o, check = TRUE)
```

Arguments

o	Outline object containing dataset field
check	If TRUE check that the base information in the reconstruction object is the same as the base data in source files.

Value

If the reconstruction data exists, return a reconstruction object, else return the outline object o.

Author(s)

David Sterratt

---

retistruct.reconstruct

Reconstruct a retina

Description

Reconstruct a retina

Usage

```
retistruct.reconstruct(
  a,
  report = NULL,
  plot.3d = FALSE,
  dev.flat = NA,
  dev.polar = NA,
  debug = FALSE,
  ...
)
```

Arguments

a	RetinalOutline object with tear and correspondence annotations
report	Function to report progress. Set to FALSE for no reporting or to NULL to inherit from the argument given to retistruct.read.dataset
plot.3d	If TRUE show progress in a 3D plot
dev.flat	The ID of the device to which to plot the flat representation
dev.polar	The ID of the device to which to plot the polar representation
debug	If TRUE print extra debugging output
...	Parameters to be passed to RetinalReconstructedOutline constructor

Value

A RetinalReconstructedOutline object

Author(s)

David Sterratt

retistruct.save.markup

Save markup

Description

Save the markup in the RetinalOutline a to a file called markup.csv in the directory a$dataset.

Usage

```
retistruct.save.markup(a)
```

Arguments

a	RetinalOutline object

Author(s)

David Sterratt

retistruct.save.recdata

Save reconstruction data

Description

Save the reconstruction data in an object r of class RetinalReconstructedOutline to a file called r.Rdata in the directory r$dataset.

Usage

```
retistruct.save.recdata(r)
```

Arguments

r	RetinalReconstructedOutline object

Author(s)

David Sterratt

rotate.axis *Rotate axis of sphere*

Description

This rotates points on sphere by specifying the direction its polar axis, i.e. the axis going through (90, 0), should point after (a) a rotation about an axis through the points (0, 0) and (0, 180) and (b) rotation about the original polar axis.

Usage

```
rotate.axis(r, r0)
```

Arguments

`r`	Coordinates of points in spherical coordinates represented as 2 column matrix with column names `phi` (latitude) and `lambda` (longitude).
`r0`	Direction of the polar axis of the sphere on which to project represented as a 2 column matrix of with column names `phi` (latitude) and `lambda` (longitude).

Value

2-column matrix of spherical coordinates of points with column names `phi` (latitude) and `lambda` (longitude).

Author(s)

David Sterratt

Examples

```
r0 <- cbind(phi=0, lambda=-pi/2)
r <- rbind(r0, r0+c(1,0), r0-c(1,0), r0+c(0,1), r0-c(0,1))
r <- cbind(phi=pi/2, lambda=0)
rotate.axis(r, r0)
```

simplifyFragment *Simplify an outline object by removing short edges*

Description

Simplify a fragment object by removing vertices bordering short edges while not encroaching on any of the outline. At present, this is done by finding concave vertices. It is safe to remove these, at the expense of increasing the area a bit.

Usage

```
simplifyFragment(P, min.frac.length = 0.001, plot = FALSE)
```

Arguments

`P`	points to simplify
`min.frac.length`	the minimum length as a fraction of the total length of the outline.
`plot`	whether to display plotting or not during simplification

Value

Simplified `outline` object

Author(s)

David Sterratt

---

simplifyOutline — *Simplify an outline object by removing short edges*

Description

Simplify a outline object by removing vertices bordering short edges while not encroaching on any of the outline. At present, this is done by finding concave vertices. It is safe to remove these, at the expense of increasing the area a bit.

Usage

```
simplifyOutline(P, min.frac.length = 0.001, plot = FALSE)
```

Arguments

`P`	points to simplify
`min.frac.length`	the minimum length as a fraction of the total length of the outline.
`plot`	whether to display plotting or not during simplification

Value

Simplified `outline` object

Author(s)

David Sterratt

`sinusoidal` *Sinusoidal projection*

Description

Sinusoidal projection

Usage

```
sinusoidal(
  r,
  proj.centre = cbind(phi = 0, lambda = 0),
  lambdalim = NULL,
  lines = FALSE,
  ...
)
```

Arguments

`r`	Latitude-longitude coordinates in a matrix with columns labelled `phi` (latitude) and `lambda` (longitude). Alternatively string "boundary", indicating that boundary of projection should be drawn.
`proj.centre`	Location of centre of projection as matrix with column names `phi` (elevation) and `lambda` (longitude). Currently only longitude is used by this function.
`lambdalim`	Limits of longitude to plot
`lines`	If this is `TRUE` create breaks of `NA`s when lines cross the limits of longitude. This prevents lines crossing the centre of the projection.
`...`	Arguments not used by this projection.

Value

Two-column matrix with columns labelled `x` and `y` of locations of projection of coordinates on plane

Author(s)

David Sterratt

References

http://en.wikipedia.org/wiki/Map_projection, http://mathworld.wolfram.com/SinusoidalProjection.html

sphere.cart.to.sphere.dualwedge

Convert from Cartesian to 'dual-wedge' coordinates

Description

Convert points in 3D cartesian space to locations of points on sphere in 'dual-wedge' coordinates (*fx*, *fy*). Wedges are defined by planes inclined at angle running through a line between poles on the rim above the x axis or the y-axis. *fx* and *fy* are the fractional distances along the circle defined by the intersection of this plane and the curtailed sphere.

Usage

```
sphere.cart.to.sphere.dualwedge(P, phi0, R = 1)
```

Arguments

`P`	locations of points on sphere as N-by-3 matrix with labelled columns `X`, `Y` and `Z`
`phi0`	rim angle as colatitude
`R`	radius of sphere

Value

2-column Matrix of 'wedge' coordinates of points on sphere. Column names are `phi` and `lambda`.

Author(s)

David Sterratt

sphere.cart.to.sphere.spherical

Convert from Cartesian to spherical coordinates

Description

Convert locations on the surface of a sphere in cartesian (X, Y, Z) coordinates to spherical (phi, lambda) coordinates.

Usage

```
sphere.cart.to.sphere.spherical(P, R = 1)
```

Arguments

`P`	locations of points on sphere as N-by-3 matrix with labelled columns "X", "Y" and "Z"
`R`	radius of sphere

Details

It is assumed that all points are lying on the surface of a sphere of radius R.

Value

N-by-2 Matrix with columns ("phi" and "lambda") of locations of points in spherical coordinates

Author(s)

David Sterratt

---

sphere.cart.to.sphere.wedge

Convert from Cartesian to 'wedge' coordinates

Description

Convert points in 3D cartesian space to locations of points on sphere in 'wedge' coordinates (*psi*, *f*). Wedges are defined by planes inclined at an angle *psi* running through a line between poles on the rim above the x axis. *f* is the fractional distance along the circle defined by the intersection of this plane and the curtailed sphere.

Usage

```
sphere.cart.to.sphere.wedge(P, phi0, R = 1)
```

Arguments

`P`	locations of points on sphere as N-by-3 matrix with labelled columns "X", "Y" and "Z"
`phi0`	rim angle as colatitude
`R`	radius of sphere

Value

2-column Matrix of 'wedge' coordinates of points on sphere. Column names are `phi` and `lambda`.

Author(s)

David Sterratt

sphere.spherical.to.polar.cart

Convert spherical coordinates on sphere to polar projection in Cartesian coordinates

Description

This is the inverse of `polar.cart.to.sphere.spherical`

Usage

```
sphere.spherical.to.polar.cart(r, pa = FALSE, preserve = "latitude")
```

Arguments

`r`	2-column Matrix of spherical coordinates of points on sphere. Column names are `phi` and `lambda`.
`pa`	If `TRUE`, make this an area-preserving projection
`preserve`	Quantity to preserve locally in the projection. Options are `latitude`, `area` or `angle`

Value

2-column Matrix of Cartesian coordinates of points on polar projection. Column names should be `x` and `y`

Author(s)

David Sterratt

sphere.spherical.to.sphere.cart

Convert from spherical to Cartesian coordinates

Description

Convert locations of points on sphere in spherical coordinates to points in 3D cartesian space

Usage

```
sphere.spherical.to.sphere.cart(Ps, R = 1)
```

Arguments

`Ps`	N-by-2 matrix with columns containing latitudes (`phi`) and longitudes (`lambda`) of N points
`R`	radius of sphere

Value

An N-by-3 matrix in which each row is the cartesian (X, Y, Z) coordinates of each point

Author(s)

David Sterratt

`sphere.tri.area` *Area of triangles on a sphere*

Description

This uses L'Hullier's theorem to compute the spherical excess and hence the area of the spherical triangle.

Usage

```
sphere.tri.area(P, Pt)
```

Arguments

`P`	2-column matrix of vertices of triangles given in spherical polar coordinates. Columns need to be labelled `phi` (latitude) and `lambda` (longitude).
`Pt`	3-column matrix of indices of rows of `P` giving triangulation

Value

Vectors of areas of triangles in units of steradians

Author(s)

David Sterratt

Source

Wolfram MathWorld http://mathworld.wolfram.com/SphericalTriangle.html and http://mathworld.wolfram.com/SphericalExcess.html

Examples

```
## Something that should be an eighth of a sphere, i.e. pi/2
P <- cbind(phi=c(0, 0, pi/2), lambda=c(0, pi/2, pi/2))
Pt <- cbind(1, 2, 3)
## The result of this should be 0.5
print(sphere.tri.area(P, Pt)/pi)

## Now a small triangle
P1 <- cbind(phi=c(0, 0, 0.01), lambda=c(0, 0.01, 0.01))
```

```
Pt1 <- cbind(1, 2, 3)
## The result of this should approximately 0.01^2/2
print(sphere.tri.area(P, Pt)/(0.01^2/2))

## Now check that it works for both
P <- rbind(P, P1)
Pt <- rbind(1:3, 4:6)
## Should have two components
print(sphere.tri.area(P, Pt))
```

---

sphere.wedge.to.sphere.cart

Convert from 'wedge' to Cartesian coordinates

Description

This in the inverse of `sphere.cart.to.sphere.wedge`

Usage

```
sphere.wedge.to.sphere.cart(psi, f, phi0, R = 1)
```

Arguments

psi	vector of slice angles of N points
f	vector of fractional distances of N points
phi0	rim angle as colatitude
R	radius of sphere

Value

An N-by-3 matrix in which each row is the cartesian (X, Y, Z) coordinates of each point

Author(s)

David Sterratt

`spherical.to.polar.area` *Convert latitude on sphere to radial variable in area-preserving projection*

Description

Project spherical coordinate system (ϕ, λ) to a polar coordinate system (ρ, λ) such that the area of each small region is preserved.

Usage

```
spherical.to.polar.area(phi, R = 1)
```

Arguments

`phi`	Latitude
`R`	Radius

Details

This requires

$$R^2 \delta\phi \cos\phi \delta\lambda = \rho\delta\rho\delta\lambda$$

. Hence

$$R^2 \int_{-\pi/2}^{\phi} \cos\phi' d\phi' = \int_0^{\rho} \rho' d\rho'$$

. Solving gives $\rho^2/2 = R^2(\sin\phi + 1)$ and hence

$$\rho = R\sqrt{2(\sin\phi + 1)}$$

.

As a check, consider that total area needs to be preserved. If ρ_0 is maximum value of new variable then $A = 2\pi R^2(\sin(\phi_0) + 1) = \pi\rho_0^2$. So $\rho_0 = R\sqrt{2(\sin\phi_0 + 1)}$, which agrees with the formula above.

Value

Coordinate `rho` that has the dimensions of length

Author(s)

David Sterratt

sphericalplot *Spherical plot of reconstructed outline*

Description

Spherical plot of reconstructed outline

Usage

```
sphericalplot(r, ...)
```

Arguments

`r`	Object inheriting `ReconstructedOutline`
`...`	Parameters depending on class of `r`

Author(s)

David Sterratt

sphericalplot.ReconstructedOutline *Spherical plot of reconstructed outline*

Description

Draw a spherical plot of reconstructed outline. This method just draws the mesh.

Usage

```
## S3 method for class 'ReconstructedOutline'
sphericalplot(r, strain = FALSE, surf = TRUE, ...)
```

Arguments

`r`	`ReconstructedOutline` object
`strain`	If `TRUE`, plot the strain
`surf`	If `TRUE`, plot the surface
`...`	Other graphics parameters – not used at present

Author(s)

David Sterratt

StitchedOutline *Class containing functions and data relating to Stitching outlines*

Description

A StitchedOutline contains a function to stitch the tears, setting the correspondences `hf`, `hb` and `h`

Super classes

`retistruct::OutlineCommon` -> `retistruct::Outline` -> `retistruct::PathOutline` -> `retistruct::AnnotatedOutl`
-> `retistruct::TriangulatedOutline` -> `StitchedOutline`

Public fields

`Rset` the set of points on the rim

`TFset` list containing indices of points in each forward tear

`epsilon` the minimum distance between points, set automatically

Methods

Public methods:

- `StitchedOutline$new()`
- `StitchedOutline$stitchTears()`
- `StitchedOutline$clone()`

Method `new()`**:** Constructor

Usage:

```
StitchedOutline$new(...)
```

Arguments:

`...` Parameters to superclass constructors

Method `stitchTears()`**:** Stitch together the incisions and tears by inserting new points in the tears and creating correspondences between new points.

Usage:

```
StitchedOutline$stitchTears()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StitchedOutline$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`strain.colours` *Generate colours for strain plots*

Description

Generate colours for strain plots

Usage

```
strain.colours(x)
```

Arguments

`x`	Vector of values of log strain

Value

Vector of colours corresponding to strains

Author(s)

David Sterratt

`stretchMesh` *Stretch mesh*

Description

Stretch the mesh in the flat retina to a circular outline

Usage

```
stretchMesh(Cu, L, i.fix, P.fix)
```

Arguments

`Cu`	Edge matrix
`L`	Lengths in flat outline
`i.fix`	Indices of fixed points
`P.fix`	Coordinates of fixed points

Value

New matrix of 2D point locations

Author(s)

David Sterratt

`tri.area` *Area of triangles on a plane*

Description

Area of triangles on a plane

Usage

```
tri.area(P, Pt)
```

Arguments

`P`	3-column matrix of vertices of triangles
`Pt`	3-column matrix of indices of rows of `P` giving triangulation

Value

Vectors of areas of triangles

Author(s)

David Sterratt

`tri.area.signed` *"Signed area" of triangles on a plane*

Description

"Signed area" of triangles on a plane

Usage

```
tri.area.signed(P, Pt)
```

Arguments

`P`	3-column matrix of vertices of triangles
`Pt`	3-column matrix of indices of rows of `P` giving triangulation

Value

Vectors of signed areas of triangles. Positive sign indicates points are anticlockwise direction; negative indicates clockwise.

Author(s)

David Sterratt

TriangulatedFragment *Class to triangulate Fragments*

Description

A TriangulatedFragment contains a function to create a triangulated mesh over an fragment, and fields to hold the mesh information.

Super class

`retistruct::Fragment` -> `TriangulatedFragment`

Public fields

`T` 3 column matrix in which each row contains IDs of points of each triangle

`A` Area of each triangle in the mesh - has same number of elements as there are rows of `T`

`Cu` 2 column matrix in which each row contains IDs of points of edge in mesh

`L` Length of each edge in the mesh - has same number of elements as there are rows of `Cu`

`A.signed` Signed area of each triangle generated using `tri.area.signed`. Positive sign indicates points are anticlockwise direction; negative indicates clockwise.

Methods

Public methods:

- `TriangulatedFragment$new()`
- `TriangulatedFragment$clone()`

Method `new()`**:** Constructor

Usage:

```
TriangulatedFragment$new(
  fragment,
  n = 200,
  suppress.external.steiner = FALSE,
  report = message
)
```

Arguments:

`fragment` Fragment to triangulate

`n` Minimum number of points in the triangulation

`suppress.external.steiner` If `TRUE` prevent the addition of points in the outline. This happens to maintain triangle quality.

`report` Function to report progress

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`TriangulatedFragment$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

`TriangulatedOutline` *Class containing functions and data relating to Triangulation*

Description

A TriangulatedOutline contains a function to create a triangulated mesh over an outline, and fields to hold the mesh information. Note that areas and lengths are all scaled using the value of the `scale` field.

Super classes

`retistruct::OutlineCommon` -> `retistruct::Outline` -> `retistruct::PathOutline` -> `retistruct::AnnotatedOutl`
-> `TriangulatedOutline`

Public fields

`T` 3 column matrix in which each row contains IDs of points of each triangle

`A` Area of each triangle in the mesh - has same number of elements as there are rows of `T`

`A.tot` Total area of the mesh

`Cu` 2 column matrix in which each row contains IDs of

`L` Length of each edge in the mesh - has same number of elements as there are rows of `Cu`

Methods

Public methods:

- `TriangulatedOutline$triangulate()`
- `TriangulatedOutline$mapTriangulatedFragment()`
- `TriangulatedOutline$clone()`

Method `triangulate()`**:** Triangulate (mesh) outline

Usage:

`TriangulatedOutline$triangulate(n = 200, suppress.external.steiner = FALSE)`

Arguments:

`n` Desired number of points in mesh

`suppress.external.steiner` Boolean variable describing whether to insert external Steiner points - see TriangulatedFragment

Method `mapTriangulatedFragment()`**:** Map the point IDs of a TriangulatedFragment on the point IDs of this Outline

Usage:

`TriangulatedOutline$mapTriangulatedFragment(fragment, pids)`

Arguments:

`fragment` TriangulatedFragment to map

`pids` Point IDs in TriangulatedOutline of points in TriangulatedFragment

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

`TriangulatedOutline$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

Author(s)

David Sterratt

Examples

```
P <- rbind(c(1,1),   c(2,1),  c(2,-1),
           c(1,-1),  c(1,-2), c(-1,-2),
           c(-1,-1), c(-2,-1),c(-2,1),
           c(-1,1),  c(-1,2), c(1,2))
o <- TriangulatedOutline$new(P)
o$addTear(c(3, 4, 5))
o$addTear(c(6, 7, 8))
o$addTear(c(9, 10, 11))
o$addTear(c(12, 1, 2))
flatplot(o)
```

`vecnorm` *Vector norm*

Description

Vector norm

Usage

```
vecnorm(X)
```

Arguments

`X`	Vector or matrix.

Value

If a vector, returns the 2-norm of the vector. If a matrix, returns the 2-norm of each row of the matrix

Author(s)

David Sterratt

Index