Package 'SIBER'

October 19, 2023

Type Package

Title Stable Isotope Bayesian Ellipses in R

Version 2.1.9

Depends R (>= 4.0.0)

SystemRequirements JAGS (>= 4.1)

Imports hdrcde, graphics, grDevices, mnormt, rjags, spatstat.geom, spatstat.utils, stats, tidyr, dplyr, ggplot2, magrittr, purrr

Suggests coda, ellipse, knitr, rmarkdown, viridis

Description Fits bi-variate ellipses to stable isotope data using Bayesian inference with the aim being to describe and compare their isotopic niche.

License GPL (>= 2)

Language en-GB

LazyLoad yes

LazyData true

Encoding UTF-8

NeedsCompilation yes

VignetteBuilder knitr

RoxygenNote 7.2.3

Author Andrew Jackson [aut, cre] (<https://orcid.org/0000-0001-7334-0434>), Andrew Parnell [aut] (<https://orcid.org/0000-0001-7956-7939>)

Maintainer Andrew Jackson <jacksoan@tcd.ie>

Repository CRAN

Date/Publication 2023-10-19 09:30:02 UTC

R topics documented:

addEllipse *Adds an ellipse around some data to an existing plot*

Description

This function adds an ellipse based on means and covariance to an existing plot. The ellipse can be scaled so as to represent any prediction interval of the data you wish, or alternatively any confidence interval of the bivariate means.

Usage

```
addEllipse(
  mu,
  sigma,
  m = NULL,
  n = 100,
  p.interval = NULL,
  ci.mean = FALSE,
  small.sample = FALSE,
  do.plot = TRUE,
  ...
)
```

Arguments

mu	a vector of length two specifying the bivariate means
sigma	a 2x2 covariance matrix for the data
m	is the sample size of the dataset on which the ellipse is to be plotted. This is only informative if calculating the confidence interval of the bivariate mean, which requires a correction of `1/sqrt(m)`. Defaults to NULL and has no effect.
n	the number of data points to be used to plot the ellipse. More points makes for a smoother ellipse, especially if it has high eccentricity. Defaults to `n = 100`.
p.interval	the quantile to be used to construct a prediction ellipse that contains p.interval proportion of the data. By default, `p.interval = NULL` and the Standard Ellipse is drawn which contains approximately 40% of the data. Setting `p.interval = 0.95` will result in an ellipse that contains approximately 95% of the data.
ci.mean	a logical that determines whether the ellipse drawn is a prediction ellipse of the entire data, or a confidence interval of the bivariate means. Defaults to `FALSE`. If set to `TRUE`, then `p.interval` can be used to generate an appropriate % confidence interval of the bivariate means.
small.sample	a logical that determines whether or not the small sample size correction is to be applied (TRUE) or not (FALSE). Defaults to FALSE. This allows SEAc rather than SEA to be plotted, but the correction can be applied to any percentile ellipse.

do.plot	A logical that determines whether plotting should occur (TRUE and default) or not (FALSE). Setting to false is useful if you want to access the coordinates of the ellipse in order to calculate overlap between ellipses for example.
...	additional arguments as a list to be passed to graphics::plot().

Value

A n x 2 matrix comprising the x and y coordinates of the ellipse.

Examples

```
#-- NOT RUN --
# data(demo.siber.data)
# my.siber.data <- createSiberObject(demo.siber.data)
# communityMetricsML(my.siber.data)
# -- END --
```

---

allCentroidVectors *Plot the pairwise distances and angles describing the difference between centroids of all groups*

Description

Plots the posterior densities

Usage

```
allCentroidVectors(centroids, upper = TRUE, do.plot = TRUE, ...)
```

Arguments

centroids	the list containing distance and angle matrices as returned by siberCentroids().
upper	a logical determining whether to plot the upper or lower triangle of angles. Defaults to TRUE which is the upper triangle and returns the angle from the second ellipse to the first by centering on the first centroid.
do.plot	a logical indicating whether plotting should be done or not. Defaults to TRUE.
...	additional arguments to pass onwards, not currently implemented.

Value

A nice plot. You can get the corresponding matrices used to generate the plots if you ask for it nicely: the_data <- plotCentroidVectors(centroids)

allgroups *The entire set of Geese isotope data*

Description

A 5 column matrix containing isotopic estimates for 251 geese collected at 8 different time points. The first column indicates the time point group, the second and third are d15N (Nitrogen) and d13C (Carbon) isotopic values for the Geese plasma, the third and fourth are d15N and d13C values for the Geese cells. Note that these are raw values; they have not undergone fractionation correction.

Usage

```
data(allgroups)
```

Format

A data frame with 251 observations on the following 5 variables.

`Group` Group number / time point

`d15NPl` d15N plasma

`d13CPl` d13C plasma

`d15NCe` d15N cells

`d13CCe` d13C cells

Examples

```
#see siarmenu() and option 9 for a demo using part of this data
```

bayesianLayman *Calculate Layman metrics on Bayesian posterior samples of a community*

Description

This function loops over the posterior distribution of group means within each community and generates the corresponding Bayesian estimate of the 6 Layman metrics.

Usage

```
bayesianLayman(mu.post)
```

Arguments

`mu.post`	a list of length n.communities, with each list element containing the estimated means of the groups comprising that community. The typical workflow to generate mu.post follows. The Bayesian ellipses are fitted using `siberEllipses()`, then the posterior means (centre of mass of each group) is extracted using `extractPosteriorMeans()`. See the example below.

Value

A list of length n.communities, with each element containing a matrix of 6 columns, each representing the Bayesian posterior distribution of the 6 Layman metrics for each of the posterior draws recorded by the fitting process (i.e. which determines the number of rows in this matrix).

`bayesianOverlap` *Calculate the overlap between two ellipses based on their posterior distributions.*

Description

This function loops over the posterior distribution of the means and covariances matrices of two specified groups.

Usage

```
bayesianOverlap(
  ellipse1,
  ellipse2,
  ellipses.posterior,
  draws = 10,
  p.interval = 0.95,
  n = 100,
  do.plot = FALSE
)
```

Arguments

`ellipse1`	character code of the form `"x.y"` where `x` is an integer indexing the community, and `y` an integer indexing the group within that community. This specifies the first of two ellipses whose overlap will be compared.
`ellipse2`	same as `ellipse1` specifying a second ellipse.
`ellipses.posterior`	a list of posterior means and covariances fitted using `siberEllipses()`.
`draws`	an integer specifying how many of the posterior draws are to be used to estimate the posterior overlap. Defaults to `10` which uses the first 10 draws. In all cases, the selection will be `1:draws` so independence of the posterior draws is assumed. Setting to `NULL` will use all the draws (WARNING - like to be very slow).

p.interval	the prediction interval used to scale the ellipse as per addEllipse().
n	the number of points on the edge of the ellipse used to define it. Defaults to 100 as per addEllipse().
do.plot	logical switch to determine whether the corresponding ellipses should be plotted or not. A use-case would be in conjunction with a low numbered draws so as to visualise a relatively small number of the posterior ellipses. Defaults to FALSE.

Value

A data.frame comprising three columns: the area of overlap, the area of the first ellipse and the area of the second ellipse and as many rows as specified by draws.

---

communityMetricsML *Calculate the point estimates of the Layman metrics for each community*

Description

This function loops over each community, determines the centre of mass (centroid) of each of the groups comprising the community using the basic base::mean() function independently on the marginal x and y vectors, and calculates the corresponding 6 Layman metrics based on these points.

Usage

```
communityMetricsML(siber)
```

Arguments

siber	a siber object as created by createSiberObject().

Value

A 6 x m matrix of the 6 Layman metrics of dX_range, dY_range, TA, CD, MNND and SDNND in rows, for each community by column

Examples

```
data(demo.siber.data)
my.siber.data <- createSiberObject(demo.siber.data)
communityMetricsML(my.siber.data)
```

concdepdemo *A set of concentration dependence values for stable isotope analysis*

Description

A dataset of concentration dependent corrections for 4 food sources of brent geese. Intended for use in a Stable Isotope Mixing Model.

Usage

```
data(concdepdemo)
```

Format

A 5 column, 4 row data.frame object containing the concentration dependence data for the geese1demo and geese2demo datasets. The first column Source is a factor determining the name of the source. The second and third columns are the mean d13C and mean d15N concentration values for each source respectively. Columns 3 and 5 are the standard deviations but these are not currently implemented in either simmr or MixSIAR stable isotope mixing models. Note that the order of the isotope data has been swapped since siar in order to present d13C as the first isotope and hence on the x-axis by default.

Author(s)

Rich Inger

correctionsdemo *A set of trophic discrimination factors for brent geese feeding on their sources.*

Description

A dataset of estimated trophic discrimination factors for brent geese. The data assume the same TDF for each food source. Intended for use in a Stable Isotope Mixing Model.

Usage

```
data(correctionsdemo)
```

Format

A 5 column, 4 row data.frame object containing the trophic discrimination factors for brent geese consumers relative to 4 of their food sources (in Ireland). The first column Source is a factor determining the name of the source. The second and third columns are the mean d13C and mean d15N TDF values for each source respectively. Columns 3 and 5 are the standard deviations of the d13C and d15N TDF values respectively. Note that the order of the isotope data has been swapped since siar in order to present d13C as the first isotope and hence on the x-axis by default.

Author(s)

Rich Inger

`createSiberObject` *Read in SIBER format data and generate the SIBER object*

Description

This function takes raw isotope data and creates a SIBER object which contains information in a structured manner that enables other functions to loop over groups and communities, fit Bayesian ellipses, and afterwards, generate various plots, and additional analyses on the posterior distributions.

Arguments

`data.in`	Specified In a basic R data.frame or matrix comprising 4 columns. The first two of which are typically isotope tracers, then the third is a column that indicates the group membership, and the fourth column indicates the community membership of an observation. Communities labels should be entered as sequential numbers. As of v2.0.1 group labels can be entered as strings and/or numbers and need not be sequential.

Value

A siber list object, that contains data that helps with various model fitting and plotting.

- `original.data` The original data as passed into this function
- `iso.summary` The max, min, mean and median of the isotope data useful for plotting
- `sample.sizes` The number of observations tabulated by group and community
- `raw.data` A list object of length equal to the number of communities

Examples

```
data(demo.siber.data)
my.siber.data <- createSiberObject(demo.siber.data)
names(my.siber.data)
```

demo.siber.data *Simulated d13C and d15N isotope-space data*

Description

Data for two communities, created by `generateSiberData()` used to generate the vignette and illustrates the main functionality of SIBER.

Usage

```
data(demo.siber.data)
```

Format

An object of class `"data.frame"` containing four variables. The first and second variables are generic isotopes called `iso1` and `iso2`. The third variable `group` identifies which group within a community an observation belongs. Group are required to be integers in sequential order starting at `1` and numbering should restart within each community. The fourth variable `community` identifies which community an observation belongs, and again is required to be an integer in sequential order staring at `1`.

Author(s)

Andrew Jackson

demo.siber.data.2 *Simulated d13C and d15N isotope-space data*

Description

Data for two communities, created by `generateSiberData()` used to generate the vignette and illustrates the main functionality of SIBER.

Usage

```
data(demo.siber.data.2)
```

Format

An object of class `"data.frame"` containing four variables. The first and second variables are generic isotopes called `iso1` and `iso2`. The third variable `group` identifies which group within a community an observation belongs. Group are required to be integers in sequential order starting at `1` and numbering should restart within each community. The fourth variable `community` identifies which community an observation belongs, and again is required to be an integer in sequential order staring at `1`.

Author(s)

Andrew Jackson

`ellipseBackTransform` *Back-transform a z-score siber ellipse to original location and scale.*

Description

Back-transforms a bivariate siber ellipse fitted to z-scored data to the original location and scale. Not intended for direct call by users.

Usage

```
ellipseBackTransform(jags.output, siber, idx.community, idx.group)
```

Arguments

`jags.output`	a mcmc.list object of posterior samples created by `rjags::rjags()`. In siber this is created typically by `fitEllipse()`
`siber`	a siber object as created by createSiberObject.
`idx.community`	an integer specifying which community to back-transform.
`idx.group`	an integer specifying which group to back-transform.

Value

A 6 x n matrix representing the back-transformed posterior distributions of the bivariate normal distribution for a specified group within a specified community, where n is the number of posterior draws in the saved sample. The first four columns are the covariance matrix Sigma in vector format. This vector converts to the covariance matrix as `matrix(v[1:4], nrow = 2, ncol = 2)`. The remaining two columns are the back-transformed means.

`ellipseInOut` *Test whether a set of points are inside or outside a defined circle*

Description

Takes a

Usage

```
ellipseInOut(Z, p = 0.95, r = NULL)
```

Arguments

Z	the i x d matrix of data points to be tested.
p	the percentile of the ellipse to be tested.
r	a manually defined radius of the circle to be used. Setting r to be anything other than NULL will override the choice of p.

Value

A logical vector indicating whether the point is inside or outside the circle

---

ellipsoidTransform *Apply a normalisation transformation to vectors of data onto ellipsoids*

Description

Takes a vector x and transforms the points onto the same geometry of a normalised ellipse given by the inverse of the covariance matrix SigSqrt and the location mu.

Usage

```
ellipsoidTransform(x, SigSqrt, mu)
```

Arguments

x	the vector of data points to be transformed
SigSqrt	the inverse of the covariance matrix
mu	the vector of means of the ellipse

Value

A vector of transformed data points

extractPosteriorMeans *Extract posterior means from call to* siberMVN

Description

This function extracts the posterior means from a call to siberMVN() which can then be used to calculate Bayesian layman metrics. This function is designed to create an array of posterior means that is more easily interrogated for plotting and summary statistics.

Usage

```
extractPosteriorMeans(siber, post)
```

Arguments

siber	a siber object as created by createSiberObject().
post	a list containing the posterior estimated parameters fitted to each group within every community. See siberMVN() which creates this object for details.

Value

A list of length n.communities with each entry representing a n.draws * 2 * n.groups array of rows equal to the number of posterior samples, 2 columns representing the two means of the multivariate data and n.groups the number of groups within the focal community.

fitEllipse *Fit a multivariate normal distribution to x and y data using jags*

Description

This function contains and defines the jags model script used to fit a bivariate normal distribution to a vector of x and y data. Although not intended for direct calling by users, it presents a quick way to fit a model to a single group of data. Advanced users should be able to manipulate the contained jags model to fit more complex models using different likelihoods, such as multivariate lognormal distributions, multivariate gamma distributions etc...

Usage

```
fitEllipse(x, y, parms, priors, id = NULL)
```

Arguments

x	a vector of data representing the x-axis
y	a vector of data representing the y-axis
parms	a list containing four items providing details of the rjags::rjags() run to be sampled. • n.iter The number of iterations to sample • n.burnin The number of iterations to discard as a burnin from the start of sampling. • n.thin The number of samples to thin by. • n.chains The number of chains to fit.
priors	a list of three items specifying the priors to be passed to the jags model. • R The scaling vector for the diagonal of Inverse Wishart distribution prior on the covariance matrix Sigma. Typically set to a 2x2 matrix matrix(c(1, 0, 0, 1), 2, 2). • k The degrees of freedom of the Inverse Wishart distribution for the covariance matrix Sigma. Typically set to the dimensionality of Sigma, which in this bivariate case is 2. • tau The precision on the normal prior on the means mu.
id	a character string to prepend to the raw saved jags model output. This is typically passed on from the calling function siberMVN() and identifies the community and group with an integer labelling system. Defaults to NULL which will prevent the output object being saved even if parms$save.output is set to TRUE. The file itself will be saved to the user-specified location via parms$save.dir.

Value

A mcmc.list object of posterior samples created by jags.

Examples

```
x <- stats::rnorm(50)
y <- stats::rnorm(50)
parms <- list()
parms$n.iter <- 2 * 10^3
parms$n.burnin <- 500
parms$n.thin <- 2
parms$n.chains <- 2
priors <- list()
priors$R <- 1 * diag(2)
priors$k <- 2
priors$tau.mu <- 1.0E-3
fitEllipse(x, y, parms, priors)
```

geese1demo *A single group of the geese data*

Description

A dataset for a single group of geese (as consumers) for two isotope tracers. Intended for use in a Stable Isotope Mixing Model.

Usage

```
data(geese1demo)
```

Format

A 2 column, 9 row matrix containing the plasma data for the first group of geese. Columns are in the order d13C and d15N. Retained here as legacy from now defunct package siar. Note that the order of the data has been swapped since siar in order to present d13C as the first isotope and hence on the x-axis by default.

Author(s)

Rich Inger

geese2demo *A single group of the geese data*

Description

A dataset for a single group of geese (as consumers) for two isotope tracers. Intended for use in a Stable Isotope Mixing Model.

Usage

```
data(geese2demo)
```

Format

A 3 column, 251 row matrix containing the plasma data for the 8 groups of gees as consumers. Columns are in the order Group which is an integer that determines which of the 8 groups the observation belongs. The second and third columns are d13C and d15N values derived from the blood plasma for each observation. Retained here as legacy from now defunct package siar. Note that the order of the isotope data has been swapped since siar in order to present d13C as the first isotope and hence on the x-axis by default.

Author(s)

Rich Inger

genCircle *Create a sequence of points on a circle*

Description

This is a helper function that creates a sequence of points on a circle of radius `r` as a resolution determined by `n`. It is not intended for direct calling, and is used by the ellipse plotting function `addEllipse()`. NB not an exported function.

Usage

```
genCircle(n = 100, r)
```

Arguments

`n`	the number of points to create around the circle. Defaults to 100.
`r`	the radius of the circle to create.

Value

A 2 x n matrix of x and y coordinates of points on a circle.

generateSiberCommunity *A utility function to simulate a single community comprised of groups*

Description

This function simulates data for a single community by sampling from a normal distribution with different means for each group within some specified boundaries.

Usage

```
generateSiberCommunity(
  n.groups = 3,
  community.id = 1,
  n.obs = 30,
  mu.range = c(-1, 1, -1, 1),
  wishSigmaScale = 1
)
```

Arguments

n.groups	the an integer specifying the number of groups to simulate. Defaults to 3.
community.id	an integer identifying the community's ID number. Defaults to 1.
n.obs	the number of observations to draw per group.
mu.range	a vector of length 4, specifying the mix and max x and y values to sample means from. Group means are sampled from a uniform distribution within this range. The first two entries are the min and max of the x-axis, and the second two the min and max of the y-axis. Defaults to `c(-1, 1, -1, 1)`.
wishSigmaScale	is a simple multiplier for the call to `stats::rWishart()` which scales the diagonal sigma matrix using `wishSigmaScale * diag(2)` that is ultimately passed on to `generateSiberGroup`.

Value

A data.frame object comprising a column of x and y data, a group identifying column and a community identifying column, all of which are numeric.

---

generateSiberData *A utility function to simulate isotope data for several communities*

Description

This function simulates data for a specified number of communities. It is a wrapper function for `generateSiberCommunity()`.

Usage

```
generateSiberData(
  n.groups = 3,
  n.communities = 2,
  n.obs = 30,
  mu.range = c(-1, 1, -1, 1),
  wishSigmaScale = 1
)
```

Arguments

n.groups	the an integer specifying the number of groups per community to simulate. Defaults to 3.
n.communities	the number of communities to simulate data for. Defaults to 2.
n.obs	the number of observations to draw per group.
mu.range	a vector of length 4, specifying the mix and max x and y values to sample means from. Group means are sampled from a uniform distribution within this range. The first two entries are the min and max of the x-axis, and the second two the min and max of the y-axis. Defaults to `c(-1, 1, -1, 1)`.

wishSigmaScale	is a simple multiplier for the call to stats::rWishart() which scales the diagonal sigma matrix using wishSigmaScale * diag(2) that is ultimately passed on to generateSiberGroup.

Value

A data.frame object comprising a column of x and y data, a group identifying column and a community identifying column, all of which are numeric.

Examples

```
generateSiberData()
```

---

generateSiberGroup *A utility function to simulate a single group of data*

Description

This function simulates data for a single group by sampling from a normal distribution with different means for each group within some specified boundaries.

Usage

```
generateSiberGroup(mu.range = c(-1, 1, -1, 1), n.obs = 30, wishSigmaScale = 1)
```

Arguments

mu.range	a vector of length 4, specifying the mix and max x and y values to sample means from. Group means are sampled from a uniform distribution within this range. The first two entries are the min and max of the x-axis, and the second two the min and max of the y-axis.
n.obs	the number of observations to draw per group. Defaults to 30.
wishSigmaScale	is a simple multiplier for the call to stats::rWishart() which scales the diagonal sigma matrix using wishSigmaScale * diag(2).

Value

A data.frame object comprising a column of x and y data, a group identifying column and a community identifying column, all of which are numeric.

Examples

```
# generateSiberGroup()
```

groupMetricsML — *Calculate maximum likelihood based measures of dispersion of bivariate data*

Description

This function loops over each group within each community and calculates the convex hull total area, Standard Ellipse Area (SEA) and its corresponding small sample size corrected version SEAc based on the maximum likelihood estimates of the means and covariance matrices of each group.

Usage

```
groupMetricsML(siber)
```

Arguments

siber	a siber object as created by createSiberObject.

Value

A 3 x m matrix of the 6 Layman metrics of dX_range, dY_range, TA, CD, MNND and SDNND in rows, where each column is a different group nested within a community.

Examples

```
data(demo.siber.data)
my.siber.data <- createSiberObject(demo.siber.data)
groupMetricsML(my.siber.data)
```

hullArea — *Calculate the area of a convex hull given its coordinates*

Description

Given the coordinates of a convex hull (i.e. a polygon), this function calculates its area. Not intended for direct use outside of siberConvexhull().

Usage

```
hullArea(x, y)
```

Arguments

x	a vector of x-axis data
y	a vector of y-axis data

Value

a scalar representing the area of the convex hull in units of `x * y`; i.e. most commonly in permille squared for isotope data.

---

kapow *KAPOW!*

Description

This function packs a punch and makes a pretty figure.

Usage

```
kapow(cd = 7, ng = 25, n = 50, sc = 10, do.plot = TRUE)
```

Arguments

`cd`	sets the random seed to this
`ng`	the number of ellipses to draw
`n`	the number of data points to simulate per group, but never displayed
`sc`	the scaling factor the rwishart sigma called by `stats::rWishart()`
`do.plot`	a logical indicating whether the plot should be printed (defaults to TRUE).

Value

A ggplot object

---

laymanMetrics *Calculates the 6 Layman metrics on a vector of x and y data*

Description

This function takes two x and y vectors, and calculates the corresponding 6 Layman metrics based on these points. Note that for generality, the original metrics of dC_range and dN_range have been renamed dX_range and dY_range respectively. These modified names represent the x and y axes in terms of the order in which the data have been entered, and relate typically to how one plots the data. These x and y vectors could represent the means of the group members comprising a community as is preferred under the SIBER model framework. However, one could use them to calculate the point estimates of the 6 Layman metrics for an entire group of data. In fact, you are free to pass this function any set of x and y data you wish.

Usage

```
laymanMetrics(x, y)
```

Arguments

x	a vector of locations in the x-axis direction.
y	a vector of locations in the y-axis direction.

Value

A vector of the 6 Layman metrics of dX_range, dY_range, TA, CD, MNND and SDNND

Examples

```
x <- stats::runif(10)
y <- stats::runif(10)
laymanMetrics(x, y)
```

---

maxLikOverlap *Calculate the overlap between two ellipses based on the maximum likelihood fitted ellipses.*

Description

This function uses the ML estimated means and covariances matrices of two specified groups to calculate the area of overlap.

Usage

```
maxLikOverlap(
  ellipse1,
  ellipse2,
  siber.object,
  p.interval = 0.95,
  n = 100,
  do.plot = FALSE
)
```

Arguments

ellipse1	character code of the form "x.y" where x is an integer indexing the community, and y an integer indexing the group within that community. This specifies the first of two ellipses whose overlap will be compared.
ellipse2	same as ellipse1 specifying a second ellipse.
siber.object	an object created by createSiberObject() which contains the ML estimates for the means and covariance matrices for each group.
p.interval	the prediction interval used to scale the ellipse as per addEllipse().
n	the number of points on the edge of the ellipse used to define it. Defaults to 100 as per addEllipse().

do.plot — logical switch to determine whether the corresponding ellipses should be plotted or not. A use-case would be in conjunction with a low numbered draws so as to visualise a relatively small number of the posterior ellipses. Defaults to FALSE.

Value

A vector comprising three columns: the area of overlap, the area of the first ellipse and the area of the second ellipse and as many rows as specified by draws.

Examples

```
# load in the included demonstration dataset data("demo.siber.data")
siber.example <- createSiberObject(demo.siber.data)

# The first ellipse is referenced using a character string representation
# where in "x.y", "x" is the community, and "y" is the group within that
# community.
ellipse1 <- "1.2"

# Ellipse two is similarly defined: community 1, group3
ellipse2 <- "1.3"

# the overlap betweeen the corresponding 95% prediction ellipses is given by:
ellipse95.overlap <- maxLikOverlap(ellipse1, ellipse2, siber.example,
p.interval = 0.95, n = 100)
```

---

mongoose *A set of isotope observations for mongooses nested within packs*

Description

A dataset of multiple isotopes per individual mongooses nested within packs where the goal is to understand isotopic niche occupancy of individuals with respect to their own pack.

Usage

```
data(mongoose)
```

Format

A 4 column, 783 row data.frame object containing unique individual mongoose identifiers in the first column "indiv.id"; an integer identifier for the pack to which each individual belongs in "pack"; Delta 13 Carbon values "c13; and Delta 15 Nitrogen values in "n15". See the paper Sheppard et al 2018 doi:10.1111/ele.12933 for more details, although N.B. the data here are provided as an example, not as a reproducible analysis of that paper.

Author(s)

Harry Marshall

`plotCommunityHulls` *Adds convex hulls to each community to an existing plot*

Description

This function loops over each community and plots the convex hull based on the centres of each of the groups that make up the community. See the demonstration scripts for example implementation.

Usage

```
plotCommunityHulls(
  siber,
  plot.args = list(col = 1, lty = 2),
  iso.order = c(1, 2),
  ...
)
```

Arguments

`siber`	a siber object as created by createSiberObject.R
`plot.args`	a list of plotting arguments with the following suggested, but non-exhaustive inputs. Additional plotting arguments for passing to the internal call to `graphics::plot()` can either be specified here, or as additional arguments under the `...` method. • `col` the color of the lines of the convex hull. See `graphics::lines()` for more details. • `lty` the line type of the convex hull.See `graphics::lines()` for more details. • `lwd` the line width of the convex hulls. See `graphics::lines()` for more details.
`iso.order`	a vector of length 2, either c(1,2) or c(2,1). The order determines which of the columns of raw data are plotted on the x (1) or y (2) axis. N.B. this will be deprecated in a future release, and plotting order will be achieved at point of data-entry.
`...`	additional arguments for passing to `graphics::plot()`.

Value

Convex hulls, drawn as lines on an existing figure.

plotGroupEllipses *Adds ellipses to an existing plot for each of your groups*

Description

This function loops over each community and group within, and plots an ellipse around the data. See demonstration scripts for more examples.

Usage

```
plotGroupEllipses(siber, plot.args = list(), iso.order = c(1, 2), ...)
```

Arguments

siber	a siber object as created by createSiberObject
plot.args	a list of plotting arguments for passing to addEllipse(). See addEllipse() for details of the options, and you can also pass additional arguments such as line widths and styles. See also the demonstration scripts for examples of use.
iso.order	a vector of length 2, either c(1,2) or c(2,1). The order determines which of the columns of raw data are plotted on the x (1) or y (2) axis. N.B. this will be deprecated in a future release, and plotting order will be achieved at point of data-entry.
...	additional arguments to be passed to addEllipse().

Value

Ellipses, drawn as lines on an existing figure.

plotGroupHulls *Plots illustrative convex hulls for each group within all communities*

Description

This function loops over each community and group within, and plots a convex hull around the data. N.B. use of convex hulls to compare isotopic niche width among groups within or between communities is not recommended owing to strong sample size bias. Use of ellipse area is recommended instead. This feature is provided for illustrative purposes only, and because some people have expressed a desire for this feature for figure generation. See demonstration scripts for more examples.

Usage

```
plotGroupHulls(siber, plot.args = NULL, iso.order = c(1, 2), ...)
```

Arguments

siber	a siber object as created by createSiberObject
plot.args	a list of plotting arguments for passing to graphics::lines(). See graphics::lines() for details of the options. See also the demonstration scripts for examples of use.
iso.order	a vector of length 2, either c(1,2) orc(2,1). The order determines which of the columns of raw data are plotted on the x (1) or y (2) axis. N.B. this will be deprecated in a future release, and plotting order will be achieved at point of data-entry.
...	additional arguments to be passed to addEllipse().

Value

A series of convex hulls added to an existing plot.

---

plotSiberObject *Creates an isotope scatterplot and provides a wrapper to ellipse and hull plotting*

Description

This function takes a SIBER object as created by createSiberObject, and loops over communities and their groups, creating a scatterplot, and adding ellipses and hulls as desired. Ellipses can be added to groups, while convex hulls can be added at both the group and community level (the former for illustrative purposes only, with no analytical tools in SIBER to fit Bayesian hulls to individual groups. This is not mathematically possible in a Bayesian framework.).

Usage

```
plotSiberObject(
  siber,
  iso.order = c(1, 2),
  ax.pad = 1,
  hulls = TRUE,
  community.hulls.args = NULL,
  ellipses = TRUE,
  group.ellipses.args = NULL,
  group.hulls = FALSE,
  group.hulls.args = NULL,
  bty = "L",
  xlab = "Isotope 1",
  ylab = "Isotope 2",
  las = 1,
  x.limits = NULL,
  y.limits = NULL,
  points.order = 1:25,
  ...
)
```

Arguments

siber	a siber object as created by createSiberObject().
iso.order	a vector of length 2, either c(1,2) or c(2,1). The order determines which of the columns of raw data are plotted on the x (1) or y (2) axis. N.B. this will be deprecated in a future release, and plotting order will be achieved at point of data-entry.
ax.pad	a padding amount to apply to the x-axis either side of the extremes of the data. Defaults to 1.
hulls	a logical defaulting to TRUE determining whether or not hulls based on the means of groups within communities should be drawn. That is, a community-level convex hull.
community.hulls.args	a list of plotting arguments to pass to plotCommunityHulls(). See plotCommunityHulls() for further details.
ellipses	a logical defaulting to TRUE determining whether or not an ellipse should be drawn around each group within each community.
group.ellipses.args	a list of plotting arguments to pass to plotGroupEllipses(). See plotGroupEllipses() for further details.
group.hulls	a logical defaulting to FALSE determining whether or not convex hulls should be drawn around each group within each community.
group.hulls.args	a list of plotting options to pass to plotGroupHulls(). See plotGroupHulls() for further details.
bty	a string specifying the box type for the plot. See graphics::par() for details.
xlab	a string for the x-axis label.
ylab	a string for the y-axis label.
las	a scalar determining the rotation of the y-axis labels. Defaults to horizontal with las = 1. See graphics::par() for more details.
x.limits	allows you to specify a two-element vector of lower and upper x-axis limits. Specifying this argument over-rides the automatic plotting and ax.pad option. Defaults to NULL.
y.limits	allows you to specify a two-element vector of lower and upper y-axis limits. Specifying this argument over-rides the automatic plotting and ax.pad option. Defaults to NULL.
points.order	a vector of integers specifying the order of point types to use. See graphics::points() for how integers map onto point types. Defaults to the sequence 1:15 as per graphics::points(). It must have at least as many entries as there are communities to plot, else a warning will be issued, and the order will default to the sequence 1:25.
...	additional arguments to be passed to graphics::plot().

Value

An isotope scatterplot.

pointsToEllipsoid *Test whether a set of points are inside or outside a defined ellipse*

Description

Takes a `i x d` matrix of points where `d` is the dimension of the space considered, and `i` is the number of points and returns `TRUE` or `FALSE` for whether each point is inside or outside a d-dimensional ellipsoid defined by a covariance matrix `Sigma` and vector of means `mu`.

Usage

```
pointsToEllipsoid(X, Sigma, mu)
```

Arguments

`X`	the `i x d` matrix of data points to be transformed
`Sigma`	the `d x d` covariance matrix of the ellipsoid
`mu`	the vector of means of the ellipse of length `d`

Value

A matrix of transformed data points corresponding to `X`

Examples

```
X <- matrix(runif(200,-2.5, 2.5), ncol = 2, nrow = 100)
SIG <- matrix(c(1,0,0,1), ncol = 2, nrow = 2)
mu <- c(0, 0)
Z <- pointsToEllipsoid(X, SIG, mu)
test <- ellipseInOut(Z, p = 0.95)
plot(X, col = test + 1, xlim = c(-3, 3), ylim = c(-3, 3), asp = 1)
addEllipse(mu, SIG, p.interval = 0.95)
```

posteriorSEA *Calculate the SEA based on a posterior distribution of Sigma*

Description

This function loops over each posterior draw of a single group's Bayesian bivariate ellipse and calculates the Standard Ellipse Area (SEA) for each draw, thereby generating a distribution of SEA estimates. Not intended for direct calling outside of `siberEllipses()`.

Usage

```
posteriorSEA(post)
```

Arguments

post	a matrix of posterior covariance matrices and mean estimates for a bivariate ellipse. In SIBER, this is typically one list element of the object returned by siberMVN().

Value

A vector of posterior Bayesian Standard Ellipse Areas (SEA_B)

---

siberCentroids *Calculate the polar form of the vector between pairs of ellipse centroids*

Description

This function loops over each group within each community and calculates the vector in polar form between the estimated centroids of each ellipse to each other ellipse.

Usage

```
siberCentroids(corrected.posteriors)
```

Arguments

corrected.posteriors	the Bayesian ellipses as returned by siberMVN().

Value

A list containing two arrays, one `r` contains the pairwise distances between ellipse centroids in as the first two dimensions, with the third dimension containing the same for each posterior draw defining the ellipse. The second array `theta` has the same structure and contains the angle in radians (from 0 to 2*pi) between the pairs. A third object `labels` refers to which community.group combination is in each of the first two dimensions of the arrays.

---

siberConvexhull *Calculate metrics and plotting information for convex hulls*

Description

This function calculates the area of the convex hull describing a set of bivariate points, and returns other information useful for plotting the hull.

Usage

```
siberConvexhull(x, y)
```

Arguments

x	a vector of x-axis data
y	a vector of y-axis data

Value

A list of length four comprising:

- TA the area of the convex hull.
- hullX the x-coordinates of the points describing the convex hull.
- hullY the y-coordinates of the points describing the convex hull.
- ind the indices of the original data in x and y that form the boundaries of the convex hull.

Examples

```
x <- stats::rnorm(15)
y <- stats::rnorm(15)
siberConvexhull(x, y)
```

---

siberDensityPlot *Plot credible intervals as shaded boxplots using* hdr.boxplot

---

Description

This function is essentially hdrcde::hdr.boxplot() but it more easily works with matrices of data, where each column is a different variable of interest. It has some limitations though....

Usage

```
siberDensityPlot(
  dat,
  probs = c(95, 75, 50),
  xlab = "Group",
  ylab = "Value",
  xticklabels = NULL,
  yticklabels = NULL,
 clr = matrix(rep(grDevices::gray((9:1)/10), ncol(dat)), nrow = 9, ncol = ncol(dat)),
  scl = 1,
  xspc = 0.5,
  prn = F,
  ct = "mode",
  ylims = NULL,
  lbound = -Inf,
  ubound = Inf,
  main = "",
```

```
  ylab.line = 2,
  ...
)
```

Arguments

dat	a matrix of data for which density region boxplots will be constructed and plotted for each column.
probs	a vector of credible intervals to represent as box edges. Defaults to `c(95, 75, 50`.
xlab	a string for the x-axis label. Defaults to `"Group"`.
ylab	a string of the y-axis label. Defaults to ‘"Value".
xticklabels	a vector of strings to override the x-axis tick labels.
yticklabels	a vector of strings to override the y-axis tick labels.
clr	a matrix of colours to use for shading each of the box regions. Defaults to greyscale `grDevices::gray((9:1)/10)` replicated for as many columns as there are in `dat`. When specified by the user, rows contain the colours of each of the confidence regions specified in `probs` and columns represent each of the columns of data in `dat`. In this way, one could have shades of blue, red and yellow for each of the groups.
scl	a scalar multiplier to scale the box widths. Defaults to 1.
xspc	a scalar determining the amount of spacing between each box. Defaults to 0.5.
prn	a logical value determining whether summary statistics of each column should be printed to screen `prn = TRUE` or suppressed as per default `prn = FALSE`.
ct	a string of either `c("mode", "mean", "median")` which determines which measure of central tendency will be plotted as a point in the middle of the boxes. Defaults to `"mode"`.
ylims	a vector of length two, specifying the lower and upper limits for the y-axis. Defaults to `NULL` which inspects the data for appropriate limits.
lbound	a lower boundary to specify on the distribution to avoid the density kernel estimating values beyond that which can be expected a priori. Useful for example when plotting dietary proportions which must lie in the interval `0 <= Y <= 1`. Defaults to `-Inf`
ubound	an upper boundary to specify on the distribution to avoid the density kernel estimating values beyond that which can be expected a priori. Useful for example when plotting dietary proportions which must lie in the interval `0 <= Y <= 1`. Defaults to `+Inf`.
main	a title for the figure. Defaults to blank.
ylab.line	a postive scalar indicating the line spacing for rendering the y-axis label. This is included as using the permille symbol has a tendency to push the axis label off the plotting window margins. See the `line` option in `graphics::axis()` for more details as ylab.line passes to this.
...	further graphical parameters for passing to `graphics::plot()`

Value

A new figure window.

Warning

: This function will not currently recognise and plot multimodal distributions, unlike `hdrcde::hdr.boxplot()`. You should take care, and plot basic histograms of each variable (column in the object you are passing) and check that they are indeed unimodal as expected.

Examples

```
# A basic default greyscale density plot
Y <- matrix(stats::rnorm(1000), 250, 4)
siberDensityPlot(Y)

# A more colourful example
my_clrs <- matrix(c("lightblue", "blue", "darkblue",
"red1", "red3", "red4",
"yellow1", "yellow3", "yellow4",
"turquoise", "turquoise3", "turquoise4"), nrow = 3, ncol = 4)
siberDensityPlot(Y, clr = my_clrs)
```

---

siberEllipses *Calculate the Bayesian Standard Ellipse Area for all groups*

Description

This function loops over each group within each community and calculates the posterior distribution describing the corresponding Standard Ellipse Area.

Usage

```
siberEllipses(corrected.posteriors)
```

Arguments

`corrected.posteriors`
: the Bayesian ellipses as returned by `siberMVN()`.

Value

A matrix of with each column containing the posterior estimates of the SEA.

siberKapow *Calculates the boundary of a union of ellipses*

Description

Intended to calculate the area of an ellipse as a proportion of a group of ellipses represented by their union, i.e. the total area encompassed by all ellipses superimposed.

Usage

```
siberKapow(
  dtf,
  isoNames = c("iso1", "iso2"),
  group = "group",
  pEll = stats::pchisq(1, df = 2)
)
```

Arguments

dtf	a data.frame object comprising bivariate data as a requirement, and possibly other variables too but these are currently ignored.
isoNames	a character vector of length 2 providing the names of the variables containing the x and y data respectively.
group	a character vector of length 1 providing the name of the grouping variable on which to calculate the KAPOW ellipse.
pEll	the probability ellipse to draw for each group. Defaults to the Standard Ellipse with `pEll = stats::pchisq(1, df = 2)`.

Value

an object of class `spatstat.geom::owin` containing the numerically calculated ellipses and their union along with the raw ellipse boundaries in both raw and `spatstat.geom::owin` format.

siberMVN *Fit Bayesian bivariate normal distributions to each group in each community*

Description

This function loops over each community and then loops over each group member, fitting a Bayesian multivariate (bivariate in this case) normal distribution to each group of data. Not intended for direct calling by users.

Usage

```
siberMVN(siber, parms, priors)
```

Arguments

siber	a siber object as created by `createSiberObject()`
parms	a list containing four items providing details of the `rjags::rjags()` run to be sampled. • `n.iter` The number of iterations to sample • `n.burnin` The number of iterations to discard as a burnin from the start of sampling. • `n.thin` The number of samples to thin by. • `n.chains` The number of chains to fit.
priors	a list of three items specifying the priors to be passed to the jags model. • `R` The scaling vector for the diagonal of Inverse Wishart distribution prior on the covariance matrix Sigma. Typically set to a 2x2 matrix `matrix(c(1, 0, 0, 1), 2, 2)`. • `k` The degrees of freedom of the Inverse Wishart distribution for the covariance matrix Sigma. Typically set to the dimensionality of Sigma, which in this bivariate case is 2. • `tau` The precision on the normal prior on the means mu.

Value

A list of length equal to the total number of groups in all communities. Each entry is named 1.1 1.2... 2.1.. with the first number designating the community, and the second number the group within that community. So, 2.3 would be the third group within the second community. Each list entry is a 6 x n matrix representing the back-transformed posterior distributions of the bivariate normal distribution, where n is the number of posterior draws in the saved sample. The first two columns are the back- transformed means, and the remaining four columns are the covariance matrix Sigma in vector format. This vector converts to the covariance matrix as `matrix(v[1:4], nrow = 2, ncol = 2)`.

---

sigmaSEA *Calculate metrics corresponding to the Standard Ellipse based on a covariance matrix*

Description

This function takes a covariance 2x2 matrix Sigma and returns various metrics relating to the corresponding Standard Ellipse. The function is limited to the 2-dimensional case, as many of the ancillary summary statistics are not defined for higher dimensions (e.g. eccentricity).

Usage

```
sigmaSEA(sigma)
```

Arguments

sigma	a 2x2 covariance ellipse.

Value

A list comprising the following metrics for summarising the Standard Ellipse

- SEA the Standard Ellipse Area (not sample size corrected).
- eccentricity a measure of the elongation of the ellipse.
- a the length of the semi-major axis.
- b the length of the semi-minor axis.

Note

This function is currently based on the eigenvalue and eigenvector approach which is more flexible for higher dimensional problems method for calculating the standard ellipse, and replaces the parametric method used previously in siar and siber.

Examples

```
# A perfect circle
sigma <- matrix( c(1, 0, 0, 1), 2, 2)
sigmaSEA(sigma)
```

---

sourcesdemo *A set of isotope observations on food sources of brent geese*

Description

A dataset of isotope observations on 4 food sources of brent geese comprising their mean and standard deviations. Intended for use in a Stable Isotope Mixing Model.

Usage

```
data(sourcesdemo)
```

Format

A 5 column, 4 row data.frame object containing 4 different plants and their measurements on 2 different isotopes. The first column Sources is a factor determining the name of the source. The second and third columns are the mean d13C and mean d15N values for each source respectively. Columns 3 and 5 are the standard deviations of the d13C and d15N values respectively. Note that the order of the isotope data has been swapped since siar in order to present d13C as the first isotope and hence on the x-axis by default.

Author(s)

Rich Inger

specificCentroidVectors *Calculate the pairwise distances and angles describing the difference between centroids of paired groups*

Description

Plots the posterior densities

Usage

```
specificCentroidVectors(centroids, do.these, upper = TRUE, do.plot = TRUE, ...)
```

Arguments

centroids	the list containing distance and angle matrices as returned by siberCentroids().
do.these	a character vector of the pattern used to find paired matches in the matrix of all comparisons. Usually the group names within any of the communities.
upper	a logical determining whether to plot the upper or lower triangle of angles. Defaults to TRUE which is the upper triangle and returns the angle from the second ellipse to the first by centering on the first centroid.
do.plot	a logical indicating whether plotting should be done or not. Defaults to TRUE.
...	additional arguments to pass onwards, not currently implemented.

Value

A nice plot. You can get the corresponding matrices used to generate the plots if you ask for it nicely: thedata <- plotCentroidVectors(centroids)

Index