Package 'MixSIAR'

April 13, 2018

Title Bayesian Mixing Models in R

Version 3.1.10

Description Creates and runs Bayesian mixing models to analyze biological tracer data (i.e. stable isotopes, fatty acids), which estimate the proportions of source (prey) contributions to a mixture (consumer). 'MixSIAR' is not one model, but a framework that allows a user to create a mixing model based on their data structure and research questions, via options for fixed/random effects, source data types, priors, and error terms. 'MixSIAR' incorporates several years of advances since 'MixSIR' and 'SIAR', and includes both GUI (graphical user interface) and script versions.

Depends R (>= 3.4.0)

Imports ggplot2 (>= 2.2.1), rjags (>= 4-6), R2jags (>= 0.5-7), MASS (>= 7.3), RColorBrewer (>= 1.1), reshape (>= 0.8.7), reshape2 (>= 1.4.3), lattice (>= 0.20-35), MCMCpack (>= 1.4-2), ggmcmc (>= 1.1), coda (>= 0.19-1), loo (>= 2.0.0), bayesplot (>= 1.4.0)

Suggests gWidgets (>= 0.0-54), gWidgetsRGtk2 (>= 0.0-86), splancs (>= 2.01-40), knitr, rmarkdown, testthat

SystemRequirements JAGS (>= 4.1) for the script version, both JAGS and GTK+ for the GUI version. For install instructions, see README file.

URL https://github.com/brianstock/MixSIAR

BugReports https://github.com/brianstock/MixSIAR/issues

License GPL-3

LazyData true

VignetteBuilder knitr

RoxygenNote 6.0.1

NeedsCompilation no

Author Brian Stock [cre, aut],
Brice Semmens [aut],
Eric Ward [ctb],

Andrew Parnell [ctb],
Andrew Jackson [ctb],
Donald Phillips [ctb]

Maintainer Brian Stock <b1stock@ucsd.edu>

Repository CRAN

Date/Publication 2018-04-13 03:09:50 UTC

R topics documented:

`build_mix_win` *Build the window to read-in mixture data.*

Description

This function is run when a user clicks the "Load mixture data" button in the main MixSIAR GUI, and creates a separate GUI window (`mix_win`) where the user loads the mixture data file. `build_mix_win` calls `load_mix_data` when the user clicks the "I'm finished" button.

Usage

```
build_mix_win()
```

Details

The "Load mix data" window contains:

1. "Load consumer data file" gbutton which loads the .csv file into `X`,
2. gtables where the user selects which columns of `X` are Isotopes and Fixed/Random/Continuous Effects.
3. (formerly) Individual Effect gcheckbox ("Include 'Individual' as a Random Effect"),
4. "I'm finished" gbutton that closes `mix_win` and calls `load_mix_data`.

If more than 2 Fixed/Random Effects are selected, a separate WARNING gwindow prompts the user to select 2, 1, or 0.

If more than 1 Continuous Effect is selected, a separate WARNING gwindow prompts the user to select 1 or 0.

If 2 Fixed/Random Effects are selected, a separate gwindow asks the user if the effects are hierarchical/nested.

Finally, the function adds a green check image if the data is successfully loaded, or a red_x image if not.

See Also

`load_mix_data`, which is run when the "Load MIX data" window is closed by clicking the "I'm finished" button.

`build_source_win` *Build the window to read-in source data*

Description

This function is run when a user clicks the "Load source data" button in the main MixSIAR GUI, and creates a separate GUI window (`source_win`) where the user loads the source data. `build_source_win` calls `load_source_data` when the user clicks the "I'm finished" button.

Usage

```
build_source_win()
```

Details

The "Load source data" window contains:

1. Questions the user if their source data is "by factor" (if they have Fixed or Random Effects),
2. Questions the user if they have Concentration Dependence data,
3. Buttons to either load "raw" or "means + SD" source data.

If `mix` does not yet exist, a WARNING appears.

A green check appears if the source data is successfully loaded, or a red_x if not.

See Also

load_source_data, which is run when the "Load source data" window is closed by clicking the "I'm finished" button.

---

calc_area *Calculate the normalized surface area of the source convex hull*

Description

calc_area calculates the normalized surface area of the SOURCE + TDF convex hull, only if there are exactly 2 biotracers.

Usage

```
calc_area(source, mix, discr)
```

Arguments

source	output from load_source_data
mix	output from load_mix_data
discr	output from load_discr_data

Details

Important detail is that, unlike in Brett (2014), calc_area uses the combined SOURCE + TDF variance to normalize the surface area:

$$\sqrt{\sigma_s^2 ource + \sigma_d^2 iscr}$$

This is the variance used in fitting the mixing model.

calc_area relies on the areapl function from splancs package, which is not automatically installed as a dependency by MixSIAR. If splancs is not installed, a WARNING message will appear.

Value

If source$by_factor = FALSE, calc_area returns a scalar, the normalized surface area of the SOURCE + TDF convex hull

If source$by_factor = TRUE, calc_area returns a vector, where the entries are the normalized surface areas of the convex hull of each source factor level (e.g. source data by 3 Regions, returns a 3-vector of the areas of the Region 1 convex hull, Region 2 convex hull, etc.)

See Also

Brett (2014): https://www.researchgate.net/profile/Michael_Brett/publication/269873625_Resource_polygon_geometry_predicts_Bayesian_stable_isotope_mixing_model_bias/links/549884090cf2519f5a1de635.pdf

combine_sources *Combine sources from a finished MixSIAR model (*a posteriori*)*

Description

combine_sources aggregates the proportions from multiple sources. Proportions are summed across posterior draws, since the source proportions are correlated.

Usage

```
combine_sources(jags.1, mix, source, alpha.prior = 1, groups)
```

Arguments

jags.1	rjags model object, output from run_model
mix	list, output from load_mix_data
source	list, output from load_source_data
alpha.prior	vector with length = n.sources, Dirichlet prior on p.global (default = 1, uninformative)
groups	list, which sources to combine, and what names to give the new combined sources. See example.

Details

Note: Aggregating sources after running the mixing model (a posteriori) effectively changes the prior weighting on the sources. Aggregating uneven numbers of sources will turn an 'uninformative'/generalist prior into an informative one. Because of this, combine_sources automatically generates a message describing this effect and a figure showing the original prior, the effective/aggregated prior, and what the 'uninformative'/generalist prior would be if sources were instead grouped before running the mixing model (a priori).

Value

combined, a list including:

- combined$post: matrix, posterior draws with new source groupings
- combined$source.new: list, original source list with modified entries for n.sources and source_names
- combined$groups: (input) list, shows original and combined sources
- combined$jags.1: (input) rjags model object
- combined$source.old: (input) list of original source data
- combined$mix: (input) list of original mix data
- combined$prior.old: (input) prior vector on original sources
- combined$prior.new: (output) prior vector on combined sources

See Also

summary_stat and plot_intervals

Examples

```
## Not run:
# first run mantis shrimp example
# combine 6 sources into 2 groups of interest (hard-shelled vs. soft-bodied)
#   'hard' = 'clam' + 'crab' + 'snail'           # group 1 = hard-shelled prey
#   'soft' = 'alphworm' + 'brittlestar' + 'fish' # group 2 = soft-bodied prey
combined <- combine_sources(jags.1, mix, source, alpha.prior=alpha,
             groups=list(hard=c("clam","crab","snail"), soft=c("alphworm","brittlestar","fish")))

# get posterior medians for new source groupings
apply(combined$post, 2, median)
summary_stat(combined, meanSD=FALSE, quantiles=c(.025,.5,.975), savetxt=FALSE)

## End(Not run)
```

---

compare_models *Compare the predictive accuracy of 2 or more MixSIAR models*

Description

compare_models uses the 'loo' package to compute LOO (leave-one-out cross-validation) or WAIC (widely applicable information criterion) for 2 of more fit MixSIAR models.

Usage

```
compare_models(x, loo = TRUE)
```

Arguments

x	list of two or more rjags model objects (output from run_model function)
loo	TRUE/FALSE: compute LOO if TRUE (preferred), compute WAIC if FALSE

Details

LOO and WAIC are "methods for estimating pointwise out-of-sample prediction accuracy from a fitted Bayesian model using the log-likelihood evaluated at the posterior simulations of the parameter values". See Vehtari, Gelman, & Gabry (2017). In brief:

- LOO and WAIC are preferred over AIC or DIC
- LOO is more robust than WAIC
- 'loo' estimates standard errors for the difference in LOO/WAIC between two models
- We can calculate the relative support for each model using LOO/WAIC weights

Value

Data frame with the following columns:

- `Model`: names of `x` (input list)
- `LOOic` / `WAIC`: LOO information criterion or WAIC
- `se_LOOic` / `se_WAIC`: standard error of LOOic / WAIC
- `dLOOic` / `dWAIC`: difference between each model and the model with lowest LOOic/WAIC. Best model has dLOOic = 0.
- `se_dLOOic` / `se_dWAIC`: standard error of the difference between each model and the model with lowest LOOic/WAIC
- `weight`: relative support for each model, calculated as Akaike weights (p.75 Burnham & Anderson 2002). Interpretation: "an estimate of the probability that the model will make the best predictions on new data, conditional on the set of models considered" (McElreath 2015).

See Also

'loo' package

Vehtari, A, A Gelman, and J Gabry. 2017. Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. Statistics and Computing.

Pages 75-88 in Burnham, KP and DR Anderson. 2002. Model selection and multimodel inference: a practical information-theoretic approach. Springer Science & Business Media.

Pages 188-201 in McElreath, R. 2016. Statistical rethinking: a Bayesian course with examples in R and Stan. CRC Press.

Examples

```
## Not run:
# Model 1 = wolf diet by Region + Pack
mix.1 <- load_mix_data(filename=mix.filename,
                       iso_names=c("d13C","d15N"),
                       factors=c("Region","Pack"),
                       fac_random=c(TRUE,TRUE),
                       fac_nested=c(FALSE,TRUE),
                       cont_effects=NULL)
source.1 <- load_source_data(filename=source.filename, source_factors="Region",
                             conc_dep=FALSE, data_type="means", mix.1)
discr.1 <- load_discr_data(filename=discr.filename, mix.1)

# Run Model 1
jags.1 <- run_model(run="test", mix.1, source.1, discr.1, model_filename,
                    alpha.prior = 1, resid_err=T, process_err=T)

# Model 2 = wolf diet by Region (no Pack)
mix.2 <- load_mix_data(filename=mix.filename,
                       iso_names=c("d13C","d15N"),
                       factors=c("Region"),
                       fac_random=c(TRUE),
                       fac_nested=c(FALSE),
```

```
                     cont_effects=NULL)
source.2 <- load_source_data(filename=source.filename, source_factors="Region",
                             conc_dep=FALSE, data_type="means", mix.2)
discr.2 <- load_discr_data(filename=discr.filename, mix.2)

# Run Model 2
jags.2 <- run_model(run="test", mix.2, source.2, discr.2, model_filename,
                    alpha.prior = 1, resid_err=T, process_err=T)

# Compare models 1 and 2 using LOO
compare_models(x=list(jags.1, jags.2), loo=TRUE)

# Compare models 1 and 2 using WAIC
compare_models(x=list(jags.1, jags.2), loo=FALSE)

# Get WAIC for model 1
compare_models(x=list(jags.1), loo=FALSE)

# Create named list of rjags objects to get model names in summary
x <- list(jags.1, jags.2)
names(x) <- c("Region + Pack", "Region")
compare_models(x)

## End(Not run)
```

`load_discr_data` *Load trophic discrimination factor (TDF) data*

Description

`load_discr_data` loads the trophic discrimination factor (TDF) data. TDF is the amount that a consumer's tissue biotracer values are modified (enriched/depleted) *after* consuming a source. If tracers are conservative, then set TDF = 0 (ex. essential fatty acids, fatty acid profile data, element concentrations).

Usage

```
load_discr_data(filename, mix)
```

Arguments

`filename`	csv file with the discrimination data
`mix`	output from `load_mix_data`

Value

discr, a list including:

- `discr$mu`, matrix of discrimination means
- `discr$sig2`, matrix of discrimination variances

`load_mix_data` *Load mixture data*

Description

`load_mix_data` loads the mixture data file and names the biotracers and any Fixed, Random, or Continuous Effects.

Usage

```
load_mix_data(filename, iso_names, factors, fac_random, fac_nested,
  cont_effects)
```

Arguments

`filename`	csv file with the mixture/consumer data
`iso_names`	vector of isotope column headings in 'filename'
`factors`	vector of random/fixed effect column headings in 'filename'. NULL if no factors.
`fac_random`	vector of TRUE/FALSE, TRUE if factor is random effect, FALSE if fixed effect. NULL if no factors.
`fac_nested`	vector of TRUE/FALSE, TRUE if factor is nested within the other. Only applies if 2 factors. NULL otherwise.
`cont_effects`	vector of continuous effect column headings in 'filename'

Value

`mix`, a list including:

- `mix$data`: dataframe, raw mix/consumer data (all columns in 'filename'),
- `mix$data_iso`: matrix, mix/consumer biotracer/isotope values (those specified in 'iso_names'),
- `mix$n.iso`: integer, number of biotracers/isotopes,
- `mix$n.re`: integer, number of random effects,
- `mix$n.ce`: integer, number of continuous effects,
- `mix$FAC`: list of fixed/random effect values, each of which contains:
  - `values`: factor, values of the effect for each mix/consumer point
  - `levels`: numeric vector, total number of values
  - `labels`: character vector, names for each factor level

  - lookup: numeric vector, if 2 factors and Factor.2 is nested within Factor.1, stores Factor.1 values for each level of Factor.2 (e.g. Wolf Ex has 8 Packs in 3 Regions, and mix$FAC[[2]]$lookup = c(1,1,1,2,2,2,2,3), the Regions each Pack belongs to).
  - re: T/F, is the factor a Random Effect? (FALSE = Fixed Effect)
  - name: character, name of the factor (e.g. "Region")
- mix$CE: list of length n.ce, contains the cont_effects values centered (subtract the mean) and scaled (divide by SD)
- mix$CE_orig: list of length n.ce, contains the original (unscaled) cont_effects values
- mix$CE_center: vector of length n.ce, means of each cont_effects
- mix$CE_scale: vector of length n.ce, SD of each cont_effects
- mix$cont_effects: vector of length n.ce, names of each cont_effects
- mix$MU_names: vector of biotracer/iso MEAN column headings to look for in the source and discrimination files (e.g. 'd13C' in iso_names, 'Meand13C' here)
- mix$SIG_names: vector of biotracer/iso SD column headings to look for in the source and discrimination files (e.g. 'd13C' in iso_names, 'SDd13C' here)
- mix$iso_names: vector of isotope column headings in 'filename' (same as input)
- mix$N: integer, number of mix/consumer data points
- mix$n.fe: integer, number of Fixed Effects
- mix$n.effects: integer, number of Fixed Effects + Random Effects
- mix$factors: vector of length n.effects, names of the Fixed and Random Effects
- mix$fac_random: T/F vector of length n.effects indicating which effects are Random (= TRUE) and Fixed (= FALSE)
- mix$fac_nested: T/F vector of length n.effects indicating which effects are nested within the other, if any
- mix$fere: TRUE if there are 2 Fixed Effects or 1 Fixed Effect and 1 Random Effect, FALSE otherwise. Used by write_JAGS_model.

If no biotracer/isotope columns are specified, a WARNING prompts the user to select 2, 1, or 0.

If more than 2 Fixed/Random Effects are selected, a WARNING prompts the user to select 2, 1, or 0.

If more than 1 Continuous Effect is selected, a WARNING prompts the user to select 1 or 0.

---

load_source_data *Load source data*

Description

load_source_data specifies the source data structure (factors, concentration dependence, data type) and loads the source data file. *Sources are sorted alphabetically.*

Usage

```
load_source_data(filename, source_factors = NULL, conc_dep, data_type, mix)
```

Arguments

filename	character, csv file with the source data.
source_factors	character, column heading in 'filename' that matches a Fixed or Random Effect from the mixture data (mixsiarmixfactors). Only used if you have source data by a factor (e.g. "Region"), otherwise NULL.
conc_dep	T/F, TRUE indicates you have concentration dependence data in 'filename'.
data_type	"raw" or "means". "Raw" source data are repeated source biotracer measurements, "means" data are source biotracer values as means, SDs, and sample size. See manual for formatting.
mix	list, output from load_mix_data.

Details

WARNING messages check for:

- More than one source factor selected
- Source factor not in mixture data
- Source sample sizes missing or entered incorrectly
- Source SD = 0

Value

source, a list including:

- source$n.sources: integer, number of sources
- source$source_names: vector, source names/labels
- source$S_MU: matrix, source means used for plotting - NOT passed to JAGS. If sources are by factor, then the third column of S_MU will be the factor values (e.g. for 4 sources and 3 Regions: 1 2 3 1 2 3 1 2 3 1 2 3)
- source$S_SIG: matrix, source SDs used for plotting - NOT passed to JAGS. Same structure as S_MU.
- source$S_factor1: factor or NULL, factor values if sources are by factor.
- source$S_factor_levels: scalar or NULL, number of S_factor1 levels if sources are by factor.
- source$conc: matrix or NULL, concentration dependence values for each isotope
- source$MU_array: array of source means, dim(src,iso,f1) or dim(src,iso) if data_type="means", NULL if data_type="raw".
- source$SIG2_array: array of source variances, dim(src,iso,f1) or dim(src,iso) if data_type="means", NULL if data_type="raw".
- source$n_array: vector/matrix of source sample sizes, dim(src,f1) or dim(src) if data_type="means", NULL if data_type="raw".

- source$SOURCE_array: array of source data, dim(src,iso,f1,replicate) or dim(src,iso,replicate) if data_type="raw", NULL if data_type="means".
- source$n.rep: vector/matrix of source sample sizes, dim(src,f1) or dim(src) if data_type="raw", NULL if data_type="means".
- source$by_factor: NA or factor number, are the source data by a Fixed or Random Effect?
- source$data_type: "raw" or "means", same as input.
- source$conc_dep: T/F, same as input.

See Also

load_mix_data and load_discr_data

---

mixsiar *mixsiar*

Description

mixsiar is an R environment the GUI version of MixSIAR uses to store objects and pass values through functions. Internal—only used in the event a user wants to extract/see MixSIAR objects (e.g. loaded mix data are accessed via mixsiar$mix, and the finished model output, rjags object, is mixsiar$jags.1)

Usage

```
mixsiar
```

Format

An object of class environment of length 0.

---

mixsiar_gui *Run the GUI version of MixSIAR*

Description

mixsiar_gui creates the GUI version of MixSIAR.

Usage

```
mixsiar_gui()
```

Details

Before running this function, a new user will need to install JAGS and GTK+. See Section 2 of the manual for install instructions: https://github.com/brianstock/MixSIAR/blob/master/inst/mixsiar_manual_small.pdf

`mixsiar_gui` calls most of the other `MixSIAR` functions:

- `load_mix_data`
- `load_source_data`
- `load_discr_data`
- `plot_data`
- `plot_prior`
- `write_JAGS_model`
- `run_model`
- `output_JAGS`

Value

`mixsiar_gui` itself returns nothing, but using the GUI creates R objects that you can access from the console:

- `mixsiar$mix`: mixture data, output of `load_mix_data`)
- `mixsiar$source`: source data, output of `load_source_data`)
- `mixsiar$discr`: discrimination (TDF) data, output of `load_discr_data`
- `mixsiar$jags.1`: rjags model object with MCMC chains, output of `run_model`

See Also

`load_mix_data` loads the mixture data file,

`load_source_data` loads the source data file,

`load_discr_data` loads the TDF data file,

`plot_data` creates an isospace plot,

`plot_prior` plots your prior and the uninformative prior,

`write_JAGS_model` creates a JAGS model file (.txt),

`run_model` sets up JAGS objects and calls JAGS to run the model,

`output_JAGS` processes the JAGS output (prints/saves diagnostics, summary statistics, and plots)

Examples

```
mixsiar_gui()
```

output_JAGS *Process mixing model output from JAGS*

Description

output_JAGS processes the mixing model output, prints and saves (in the working directory):

- diagnostics
- summary statistics
- posterior density plots
- pairs plot
- trace/XY plots

Usage

```
output_JAGS(jags.1, mix, source, output_options = list(summary_save = TRUE,
  summary_name = "summary_statistics", sup_post = FALSE, plot_post_save_pdf =
  TRUE, plot_post_name = "posterior_density", sup_pairs = FALSE,
  plot_pairs_save_pdf = TRUE, plot_pairs_name = "pairs_plot", sup_xy = TRUE,
  plot_xy_save_pdf = FALSE, plot_xy_name = "xy_plot", gelman = TRUE, heidel =
  FALSE, geweke = TRUE, diag_save = TRUE, diag_name = "diagnostics",
  indiv_effect = FALSE, plot_post_save_png = FALSE, plot_pairs_save_png = FALSE,
  plot_xy_save_png = FALSE))
```

Arguments

jags.1	rjags model object, output from run_model function
mix	output from load_mix_data
source	output from load_source_data
output_options	list containing options for plots and saving: • summary_save: Save the summary statistics as a txt file? • summary_name: Summary statistics file name (.txt will be appended) • sup_post: Suppress posterior density plot output in R? • plot_post_save_pdf: Save posterior density plots as pdfs? • plot_post_name: Posterior plot file name(s) (.pdf/.png will be appended) • sup_pairs: Suppress pairs plot output in R? • plot_pairs_save_pdf: Save pairs plot as pdf? • plot_pairs_name: Pairs plot file name (.pdf/.png will be appended) • sup_xy: Suppress xy/trace plot output in R? • plot_xy_save_pdf: Save xy/trace plot as pdf? • plot_xy_name: XY/trace plot file name (.pdf/.png will be appended) • gelman: Calculate Gelman-Rubin diagnostic test? • heidel: Calculate Heidelberg-Welch diagnostic test?

- geweke: Calculate Geweke diagnostic test?
- diag_save: Save the diagnostics as a .txt file?
- diag_name: Diagnostics file name (.txt will be appended)
- indiv_effect: artifact, set to FALSE
- plot_post_save_png: Save posterior density plots as pngs?
- plot_pairs_save_png: Save pairs plot as png?
- plot_xy_save_png: Save xy/trace plot as png?

Value

p.both – only if 2 fixed effects OR 1 fixed + 1 random, otherwise NULL).

p.both holds the MCMC chains for the estimated proportions at the different factor levels. Dimensions = [n.draws, f1.levels, f2.levels, n.sources].

Calculated by combining the ilr offsets from global intercept: ilr.both[,f1,f2,src] = ilr.global[,src] + ilr.fac1[,f1,src] + ilr.fac2[,f2,src] And then transforming from ilr- to proportion-space.

---

plot_continuous_var *Plot proportions by a continuous covariate*

Description

plot_continuous_var creates a plot of how the mixture proportions change according to a continuous covariate, as well as plots of the mixture proportions for the individuals with minimum, median, and maximum covariate values. Called by output_JAGS if any continuous effects are in the model.

Usage

```
plot_continuous_var(jags.1, mix, source, output_options)
```

Arguments

jags.1	output from run_model
mix	output from load_mix_data
source	output from load_source_data
output_options	list containing options for plots and saving, passed from output_JAGS

Details

MixSIAR fits a continuous covariate as a linear regression in ILR/transform-space. Two terms are fit for the proportion of each source: an intercept and a slope. The plotted line uses the posterior median estimates of the intercept and slope, and the lines are curved because of the ILR-transform back into proportion-space. The 95% credible intervals are shaded.

If the model contains both a continuous AND a categorical (factor) covariate, MixSIAR fits a different intercept term for each factor level and all levels share the same slope term.

See Also

Francis et al. 2011

`plot_data` *Plot biotracer data*

Description

`plot_data` creates plot(s) of the biotracer data and saves the plot(s) to file(s) in the working directory. All 3 required data files must have been loaded by `load_mix_data`, `load_source_data`, and `load_discr_data`. Behavior depends on the number of tracers:

- 1 tracer: calls `plot_data_one_iso` to create a 1-D plot.
- 2 tracers: calls `plot_data_two_iso` to create a biplot.
- >2 tracers: calls `plot_data_two_iso` in a loop to create biplots for each pairwise combination of biotracers.

Usage

```
plot_data(filename, plot_save_pdf, plot_save_png, mix, source, discr)
```

Arguments

`filename`	name of the plot file(s) to save (e.g. "isospace_plot")
`plot_save_pdf`	T/F, save the plot(s) as a pdf?
`plot_save_png`	T/F, save the plot(s) as a png?
`mix`	output from `load_mix_data`
`source`	output from `load_source_data`
`discr`	output from `load_discr_data`

Details

An important detail is that `plot_data_two_iso` and `plot_data_one_iso` plot the raw mix data and *add the TDF to the source data*, since this is the polygon that the mixing model uses to determine proportions. The plotted source means are:

$$\mu_source + \mu_discr$$

The source error bars are +/- 1 standard deviation, *calculated as a combination of source and TDF variances:*

$$\sqrt{\sigma_s^2 ource + \sigma_d^2 iscr}$$

`plot_data` looks for 'C', 'N', 'S', and 'O' in the biotracer column headers and assumes they are stable isotopes, labeling the axes with, e.g., expression(paste(delta^13, "C (u2030)",sep="")).

See Also

`plot_data_two_iso`, `plot_data_one_iso`

`plot_data_one_iso` *Plot biotracer data (1-D)*

Description

`plot_data_one_iso` creates a 1-D plot of mix and source tracer data and saves the plot to a file in the working directory

Usage

```
plot_data_one_iso(mix, source, discr, filename, plot_save_pdf, plot_save_png)
```

Arguments

`mix`	output from `load_mix_data`
`source`	output from `load_source_data`
`discr`	output from `load_discr_data`
`filename`	name of the plot file(s) to save (e.g. "isospace_plot")
`plot_save_pdf`	T/F, save the plot(s) as a pdf?
`plot_save_png`	T/F, save the plot(s) as a png?

Details

An important detail is that `plot_data_one_iso` plots the raw mix data and *adds the TDF to the source data*, since this is the polygon that the mixing model uses to determine proportions. The plotted source means are:

$$\mu_source + \mu_discr$$

The source error bars are +/- 1 standard deviation, *calculated as a combination of source and TDF variances:*

$$\sqrt{\sigma^2_source + \sigma^2_discr}$$

`plot_data_one_iso` looks for 'C', 'N', 'S', and 'O' in the biotracer column headers and assumes they are stable isotopes, labeling the axes with, e.g., expression(paste(delta^13, "C (u2030)",sep="")).

See Also

`plot_data`

plot_data_two_iso *Plot biotracer data (2-D)*

Description

plot_data_two_iso creates a 2-D plot of mix and source tracer data and saves the plot to a file in the working directory

Usage

```
plot_data_two_iso(isotopes, mix, source, discr, filename, plot_save_pdf,
  plot_save_png)
```

Arguments

isotopes	2-vector of biotracer indices to plot (e.g. c(1,2) or c(2,3))
mix	output from load_mix_data
source	output from load_source_data
discr	output from load_discr_data
filename	name of the plot file(s) to save (e.g. "isospace_plot")
plot_save_pdf	T/F, save the plot(s) as a pdf?
plot_save_png	T/F, save the plot(s) as a png?

Details

An important detail is that plot_data_two_iso plots the raw mix data and *adds the TDF to the source data*, since this is the polygon that the mixing model uses to determine proportions. The plotted source means are:

$$\mu_source + \mu_discr$$

The source error bars are +/- 1 standard deviation, *calculated as a combination of source and TDF variances:*

$$\sqrt{\sigma^2_source + \sigma^2_discr}$$

plot_data_two_iso looks for 'C', 'N', 'S', and 'O' in the biotracer column headers and assumes they are stable isotopes, labeling the axes with, e.g., expression(paste(delta^13, "C (u2030)",sep="")).

See Also

plot_data

plot_intervals *Plot posterior uncertainty intervals from a MixSIAR model*

Description

plot_intervals plots the posterior interval estimates (quantile-based) from the MCMC draws in a MixSIAR model. Calls bayesplot::mcmc_intervals.

Usage

```
plot_intervals(combined, toplot = "p", levels = NULL, groupby = "factor",
  savepdf = FALSE, filename = "post_intervals", ...)
```

Arguments

combined	list, output from combine_sources function
toplot	vector, which parameters to plot? Options are similar to summary_stat: • "p": plots all proportions (default) • "global": plots overall proportions • "fac1": plots factor 1 proportions • "fac2": plots factor 2 proportions • "epsilon": plots multiplicative error terms • "sd": plots random effect SD terms
levels	vector if toplot="fac1" or toplot="fac2", which level(s) to plot? Plots all levels if level=NULL (default). Specify levels as a vector, e.g. in wolves ex, levels=1 to plot Region 1, levels=c(1,2) to plot Regions 1 and 2.
groupby	character, group by "factor" or "source"? I.e. in wolves example, group proportions by Region 1, Region 2, Region 3 (groupby="factor") vs. Deer, Marine Mammals, Salmon (groupby="source"). Currently only "factor" is implemented.
savepdf	TRUE/FALSE, save plot as .pdf file (in working directory)?
filename	character, file name to save results as (.pdf will be appended automatically)
...	additional arguments to pass to bayesplot::mcmc_intervals. For example: • prob: sets inner (thick) interval (default = 50%) • prob_outer: sets outer (thin) interval (default = 90%) • point_est: what point estimate to use (dot), default = "median", can also use "mean" or "none"

See Also

combine_sources and summary_stat

Examples

```
## Not run:
# 1. run mantis shrimp example
original <- combine_sources(jags.1, mix, source, alpha,
                groups=list(alphworm="alphworm",brittlestar="brittlestar",clam="clam",
                            crab="crab",fish="fish",snail="snail"))
# 2. combine 6 sources into 2 groups of interest (hard-shelled vs. soft-bodied)
#   'hard' = 'clam' + 'crab' + 'snail'           # group 1 = hard-shelled prey
#   'soft' = 'alphworm' + 'brittlestar' + 'fish' # group 2 = soft-bodied prey
combined <- combine_sources(jags.1, mix, source, alpha.prior=alpha,
             groups=list(hard=c("clam","crab","snail"), soft=c("alphworm","brittlestar","fish")))

plot_intervals(combined,toplot="fac1")
plot_intervals(original,toplot="fac1")
plot_intervals(combined,toplot="fac1",levels=1)
plot_intervals(combined,toplot="fac1",levels=2)


## End(Not run)
```

`plot_prior` *Plot prior*

Description

`plot_prior` plots your prior on the global diet proportions (p.global) and the uninformative prior side-by-side. Your prior is in red, and the "uninformative"/generalist prior (alpha = 1) in dark grey.

Usage

```
plot_prior(alpha.prior = 1, source, plot_save_pdf = TRUE,
  plot_save_png = FALSE, filename = "prior_plot")
```

Arguments

`alpha.prior`	vector of alpha (dirichlet hyperparameters, none can be = 0)
`source`	output from `load_source_data`
`plot_save_pdf`	T/F, save the plot as a pdf?
`plot_save_png`	T/F, save the plot as a png?
`filename`	name of the file to save (e.g. "prior_plot")

run_model *Run the JAGS model*

Description

run_model calls JAGS to run the mixing model created by write_JAGS_model. This happens when the "RUN MODEL" button is clicked in the GUI.

Usage

```
run_model(run, mix, source, discr, model_filename, alpha.prior = 1, resid_err,
  process_err)
```

Arguments

run	list of MCMC parameters (chainLength, burn, thin, chains, calcDIC). Alternatively, a user can use a pre-defined parameter set by specifying a valid string: • "test": chainLength=1000, burn=500, thin=1, chains=3 • "very short": chainLength=10000, burn=5000, thin=5, chains=3 • "short": chainLength=50000, burn=25000, thin=25, chains=3 • "normal": chainLength=100000, burn=50000, thin=50, chains=3 • "long": chainLength=300000, burn=200000, thin=100, chains=3 • "very long": chainLength=1000000, burn=500000, thin=500, chains=3 • "extreme": chainLength=3000000, burn=1500000, thin=500, chains=3
mix	output from load_mix_data
source	output from load_source_data
discr	output from load_discr_data
model_filename	name of JAGS model file (usually should match filename input to write_JAGS_model).
alpha.prior	Dirichlet prior on p.global (default = 1, uninformative)
resid_err	include residual error in the model?
process_err	include process error in the model?

Value

jags.1, a rjags model object

Note: Tracer values are normalized before running the JAGS model. This allows the same priors to be used regardless of scale of the tracer data, without using the data to select the prior (i.e. by setting the prior mean equal to the sample mean). Normalizing the tracer data does not affect the proportion estimates (p_k), but does affect users seeking to plot the posterior predictive distribution for their data. For each tracer, we calculate the pooled mean and standard deviation of the mix and source data, then subtract the pooled mean and divide by the pooled standard deviation from the mix and source data. For details, see lines 226-269.

summary_stat *Summary statistics from posterior of MixSIAR model*

Description

summary_stat prints and saves summary statistics

Usage

```
summary_stat(combined, toprint = "all", groupby = "factor", meanSD = TRUE,
  quantiles = c(0.025, 0.25, 0.5, 0.75, 0.975), savetxt = TRUE,
  filename = "summary_statistics")
```

Arguments

combined	list, output from combine_sources function
toprint	vector, which parameters to print? Options are: "p" to print stats for proportions only (all factors), "global" to only print overall proportions, "fac1" to only print factor 1 proportions, "fac2" to print factor 2 proportions. Set = "epsilon" to print multiplicative error term only. Default = "all", prints stats for all model parameters.
groupby	character, group stats by "factor" or "source"? I.e. in wolves example, group proportions by Region 1, Region 2, Region 3 (groupby="factor") vs. Deer, Marine Mammals, Salmon (groupby="source"). Currently only "factor" is implemented.
meanSD	TRUE/FALSE, print mean and SD for the parameters?
quantiles	vector, which quantiles to print. Default = c(0.025, 0.25, 0.5, 0.75, 0.975).
savetxt	TRUE/FALSE, save results as .txt file (in working directory)?
filename	character, file name to save results as (.txt will be appended automatically)

See Also

combine_sources and plot_intervals

Examples

```
## Not run:
# first run mantis shrimp example
# combine 6 sources into 2 groups of interest (hard-shelled vs. soft-bodied)
#   'hard' = 'clam' + 'crab' + 'snail'           # group 1 = hard-shelled prey
#   'soft' = 'alphworm' + 'brittlestar' + 'fish' # group 2 = soft-bodied prey
combined <- combine_sources(jags.1, mix, source, alpha.prior=alpha,
             groups=list(hard=c("clam","crab","snail"), soft=c("alphworm","brittlestar","fish")))

summary_stat(combined)
summary_stat(combined, savetxt=FALSE)
```

```
summary_stat(combined, meanSD=FALSE)
summary_stat(combined, quantiles=c(.05,.5,.95))
summary_stat(combined, toprint="fac1")
summary_stat(combined, toprint="p")
summary_stat(combined, toprint="global")

## End(Not run)
```

`write_JAGS_model` *Write the JAGS model file*

Description

`write_JAGS_model` creates "MixSIAR_model.txt", which is passed to JAGS by `run_model` when the "RUN MODEL" button is clicked in the GUI. Several model options will have already been specified when loading the mix and source data, but here is where the error term options are selected:

1. Residual * Process (resid_err = TRUE, process_err = TRUE)
2. Residual only (resid_err = TRUE, process_err = FALSE)
3. Process only (resid_err = FALSE, process_err = TRUE)

Usage

```
write_JAGS_model(filename = "MixSIAR_model.txt", resid_err = TRUE,
  process_err = TRUE, mix, source)
```

Arguments

`filename`	the JAGS model file is saved in the working directory as 'filename' (default is "MixSIAR_model.txt", but user can specify).
`resid_err`	T/F: include residual error in the model?
`process_err`	T/F: include process error in the model?
`mix`	output from `load_mix_data`
`source`	output from `load_source_data`

Details

WARNING messages are displayed if:

- resid_err = FALSE and process_err = FALSE are both selected.
- N=1 mix data point and did not choose "Process only" error model (MixSIR)
- Fitting each individual mix data point separately as a Fixed Effect, but did not choose "Process only" error model (MixSIR).

Index