Package: rubias (via r-universe)

August 20, 2024

Type Package

Title Bayesian Inference from the Conditional Genetic Stock Identification Model

Version 0.3.4

Maintainer Eric C. Anderson <eric.anderson@noaa.gov>

Description Implements Bayesian inference for the conditional genetic stock identification model. It allows inference of mixed fisheries and also simulation of mixtures to predict accuracy. A full description of the underlying methods is available in a recently published article in the Canadian Journal of Fisheries and Aquatic Sciences: <doi:10.1139/cjfas-2018-0016>.

License CC0

LazyData TRUE

Depends R (>= 3.3.0)

Imports dplyr, gtools, magrittr, Rcpp (>= 0.12.5), readr, rlang, stringr, tibble, tidyr, RcppParallel

LinkingTo Rcpp, RcppParallel

SystemRequirements GNU make

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, ggplot2

VignetteBuilder knitr

Encoding UTF-8

Repository https://eriqande.r-universe.dev

RemoteUrl https://github.com/eriqande/rubias

RemoteRef HEAD

RemoteSha bbb1753cfad91f69958eddb0151fc657e1deb2d0

Contents

`alewife` *Microsat data from alewife herring reference populations*

Description

Standard two-column genetic data with lots of other columns preceding it. Can be fed directly into rubias because it has at least the columns sample_type, collection, repunit and indiv.

Format

A tibble.

Source

https://datadryad.org/stash/dataset/doi:10.5061/dryad.80f4f

`assess_pb_bias_correction` *Test the effects of the parametric bootstrap bias correction on a reference dataset through cross-validation*

Description

This is a rewrite of bias_comparison(). Eric didn't want the plotting to be wrapped up in a function, and wanted to return a more informative data frame.

Usage

```
assess_pb_bias_correction(
  reference,
  gen_start_col,
  seed = 5,
  nreps = 50,
  mixsize = 100,
  alle_freq_prior = list(const_scaled = 1)
)
```

Arguments

`reference`	a two-column format genetic dataset, with a "repunit" column specifying each individual's reporting unit of origin, a "collection" column specifying the collection (population or time of sampling) and "indiv" providing a unique name
`gen_start_col`	the first column containing genetic data in `reference`. All columns should be genetic format following this column, and gene copies from the same locus should be adjacent
`seed`	the random seed for simulations
`nreps`	The number of reps to do.
`mixsize`	The size of each simulated mixture sample.
`alle_freq_prior`	a one-element named list specifying the prior to be used when generating Dirichlet parameters for genotype likelihood calculations. Valid methods include `"const"`, `"scaled_const"`, and `"empirical"`. See `?list_diploid_params` for method details.

Details

Takes a reference two-column genetic dataset, pulls a series of random "mixture" datasets with varying reporting unit proportions from this reference, and compares the results of GSI through standard MCMC vs. parametric-bootstrap MCMC bias correction

The amount of bias in reporting unit proportion calculations increases with the rate of misassignment between reporting units (decreases with genetic differentiation), and increases as the number of collections within reporting units becomes more uneven.

Output from the standard Bayesian MCMC method demonstrates the level of bias to be expected for the input data set, and parametric bootstrapping is an empirical method for the removal of any existing bias.

Value

`bias_comparison` returns a list; the first element is a list of the relevant rho values generated on each iteration of the random "mixture" creation. This includes the true rho value, the standard result `rho_mcmc`, and the parametric bootstrapped `rho_pb`.

The second element is a dataframe listing summary statistics for each reporting unit and estimation method. `mse`, the mean squared error, summarizes the deviation of the rho estimates from their true value, including both bias and other variance. `mean_prop_bias` is the average ratio of residual to true value, which gives greater weight to deviations at smaller values. `mean_bias` is simply the average residual; unlike `mse`, this demonstrates the direction of the bias.

Examples

```
## Not run:
## This takes too long to run in R CMD CHECK
ale_bias <- assess_pb_bias_correction(alewife, 17)

## End(Not run)
```

---

assess_reference_loo *Simulate mixtures and estimate reporting group and collection proportions.*

Description

From a reference dataset, this creates a genotype-logL matrix based on simulation-by-individual with randomly drawn population proportions, then uses this in two different estimates of population mixture proportions: maximum likelihood via EM-algorithm and posterior mean from MCMC.

Usage

```
assess_reference_loo(
  reference,
  gen_start_col,
  reps = 50,
  mixsize = 100,
  seed = 5,
  alpha_repunit = 1.5,
  alpha_collection = 1.5,
  resampling_unit = "individual",
  alle_freq_prior = list(const_scaled = 1),
  printSummary = FALSE,
  return_indiv_posteriors = FALSE
)
```

Arguments

reference	a two-column format genetic dataset, with "repunit", "collection", and "indiv" columns, as well as a "sample_type" column that has some "reference" entries
gen_start_col	the first column of genetic data in reference
reps	number of reps of mixture simulation and MCMC to do
mixsize	the number of individuals in each simulated mixture
seed	a random seed for simulations
alpha_repunit	If a vector, this is the dirichlet parameter for simulating the proportions of reporting units. Gets recycled to the number of reporting units. Default is 1.5. Otherwise, this could be a two-column data frame. The first column must be named "repunit" and the second one must be one of "dirichlet", "ppn", or "cnt", according to whether you wish to specify dirichlet parameters, or proportions, or exact counts, respectively, for each population. If you want to make multiple simulations, pass in a list of data frames or of individual dirichlet parameters. For examples, see sim_spec_examples.
alpha_collection	The dirichlet parameter for simulating proportions of collections within reporting units. Default = 1.5. If this is a data frame then the first column must be "collection" and the second must be one of "dirichlet", "ppn", "cnt", "sub_dirichlet", "sub_ppn". If you want to provide multiple different scenarios. You can pass them in as a list. If alpha_repunit or alpha_collection is a list with length greater than 1, the shorter will be recycled. For examples, see sim_spec_examples.
resampling_unit	what unit should be resampled. Currently the choices are "individuals" (the default) and "gene_copies". Using "individuals" preserves missing data patterns available in the reference data set. We also have "gene_copies_with_missing" capability, but it is not yet linked into this function.
alle_freq_prior	a one-element named list specifying the prior to be used when generating Dirichlet parameters for genotype likelihood calculations. Valid methods include "const", "scaled_const", and "empirical". See ?list_diploid_params for method details.
printSummary	if TRUE a summary of the reference samples will be printed to stdout.
return_indiv_posteriors	if TRUE, output is a list of 2. The first entry, mixing_proportions, contains the true (simulated) and estimated mixture proportions for each scenario, iteration, and collection. The second, indiv_posteriors, contains the posterior probability of assignment to each collection for each scenario, iteration, and individual. If FALSE, output is a single data frame, mixing_proportions

Examples

```
# very small number of reps so it is quick enough for example
ale_dev <- assess_reference_loo(alewife, 17, reps = 5)
```

`assess_reference_mc` *Partition a reference dataset and estimate reporting group and collection proportions*

Description

From a reference dataset, this draws (without replacement) a simulated mixture dataset with randomly drawn population proportions, then uses this in two different estimates of population mixture proportions: maximum likelihood via EM-algorithm and posterior mean from MCMC.

Usage

```
assess_reference_mc(
  reference,
  gen_start_col,
  reps = 50,
  mixsize = 100,
  seed = 5,
  alpha_repunit = 1.5,
  alpha_collection = 1.5,
  min_remaining = 5,
  alle_freq_prior = list(const_scaled = 1)
)
```

Arguments

`reference`	a two-column format genetic dataset, with "repunit", "collection", and "indiv" columns, as well as a "sample_type" column that has some "reference" entries.
`gen_start_col`	the first column of genetic data in reference
`reps`	number of reps to do
`mixsize`	the number of individuals in each simulated mixture.
`seed`	a random seed for simulations
`alpha_repunit`	The dirichlet parameter for simulating the proportions of reporting units. Default = 1.5
`alpha_collection`	The dirichlet parameter for simulating proportions of collections within reporting units. Default = 1.5
`min_remaining`	the minimum number of individuals which should be conserved in each reference collection during sampling without replacement to form the simulated mixture
`alle_freq_prior`	a one-element named list specifying the prior to be used when generating Dirichlet parameters for genotype likelihood calculations. Valid methods include `"const"`, `"scaled_const"`, and `"empirical"`. See `?list_diploid_params` for method details.

Details

This method is referred to as "Monte Carlo cross-validation". The input parameters for `assess_reference_mc` are more restrictive than those of `assess_reference_loo`. Rather than allowing a *data.frame* to specify Dirichlet parameters, proportions, or counts for specific reporting units and collections, `assess_reference_mc` only allows vector input (default = 1.5) for `alpha_repunit` and `alpha_collection`. These inputs specify the uniform Dirichlet parameters for all reporting units and collections, respectively.

For mixture proportion generation, the rho values are first drawn using a stick-breaking model of the Dirichlet distribution, but with proportions capped by `min_remaining`. Stick-breaking is then used to subdivide each reporting unit into collections. In addition to the constraint that mixture sampling without replacement cannot deplete the number of individuals in each collection below `min_remaining`, a similar constraint is placed upon the number of individuals left in reporting units, determined as `min_remaining` * (# collections in reporting unit).

Note that this implies that the data are only truly Dirichlet distributed when no rejections based on `min_remaining` occur. This is a reasonable certainty with sufficient reference collection sizes relative to the desired mixture size.

Examples

```
# only 5 reps, so it doesn't take too long.  Typically you would
# do many more
ale_dev <- assess_reference_mc(alewife, 17, 5)
```

---

blueback *Microsat data from blueback herring reference populations*

Description

Standard two-column genetic data with lots of other columns preceding it. Can be fed directly into rubias because it has at least the columns sample_type, collection, repunit and indiv.

Format

A tibble.

Source

https://datadryad.org/stash/dataset/doi:10.5061/dryad.80f4f

chinook *SNP data from chinook reference populations*

Description

Chinook salmon baseline data similar to that which can be downloaded from https://datadryad.org/stash/dataset/doi:10.5061/dryad.574sv. This data set includes 91 SNPs and 7301 fish and is what the Dryad data became after we converted from TaqMan to SNPtype assays (being forced to toss some loci) and tossed out a bunch of lousy historical samples from Trinity River.

Format

A tbl_df-ed (from dplyr) data frame with 7,301 rows and 185 variables. The first three columns are

repunit (chr) the reporting unit that the individual is in

pop (chr) the population from which the individual was sampled

ID (chr) Unique identifier of the individual fish

The remaining columns are two columns for each locus. These columns are named like, "Locus.1" and "Locus.2" for the first and second gene copies at that locus. For example, "Ots_104569-86.1" and "Ots_104569-86.2". The locus columns are ints and missing data is denoted by NA.

Source

https://datadryad.org/stash/dataset/doi:10.5061/dryad.574sv

chinook_collection_levels *a vector that gives a desired sort order of the chinook collections*

Description

This is just an example of what one would use as levels in order to get the chinook collections in a desired sort order after analysis. The issue here is collection in the input data frame to most functions must be a character vector, not a factor. But, after analysis you can always make them a factor again and use a vector like this one to specify the levels.

Source

Made it up!

`chinook_mix` *SNP data from Chinook salmon taken in May/August 2015 from California fisheries*

Description

This has data from 91 SNP markers (a subset of the 95 markers in the `chinook` baseline data set).

Format

A tbl_df-ed (from dplyr) data frame with 2256 rows and 193 variables. The first four columns are meta data. The remaining columns are two columns for each locus. These columns are named like, "Locus.1" and "Locus.2" for the first and second gene copies at that locus. For example, "Ots_104569-86.1" and "Ots_104569-86.2". The locus columns are ints and missing data is denoted by NA.

Source

Southwest Fisheries Science Center, Santa Cruz, CA

`chinook_repunit_levels` *a vector that gives a desired sort order of the chinook repunits*

Description

This is just an example of what one would use as levels in order to get the `chinook` repunits in a desired sort order after analysis. The issue here is that repunit in the input data frame to most functions must be a character vector, not a factor. But, after analysis you can always make them a factor again and use a vector like this one to specify the levels.

Source

Made it up!

`close_matching_samples` *check for matching (or close to matching) genotypes in a data frame*

Description

Super simple function that looks at all pairs of fish from the data frame and returns a tibble that includes those which shared a fraction >= than min_frac_non_miss of the genotypes not missing in either fish, and which were matching at a fraction >= min_frac_matching of those non-missing pairs of genotypes.

Usage

```
close_matching_samples(
  D,
  gen_start_col,
  min_frac_non_miss = 0.7,
  min_frac_matching = 0.9
)
```

Arguments

`D`	a two-column format genetic dataset, with "repunit", "collection", and "indiv" columns, as well as a "sample_type" column that has entried either of "reference" or "mixture" or both.
`gen_start_col`	the first column of genetic data in `reference`
`min_frac_non_miss`	the fraction of loci that the pair must share non missing in order to be reported
`min_frac_matching`	the fraction of shared non-missing loci that must be shared between the indivduals to be reported as a matching pair.

Value

a tibble ...

Examples

```
# one pair found in the interal alewife data set:
close_matching_samples(alewife, 17)
```

`infer_mixture` *Estimate mixing proportions and origin probabilities from one or several mixtures*

Description

Takes a mixture and reference dataframe of two-column genetic data, and a desired method of estimation for the population mixture proportions (MCMC, PB, BR). Returns the output of the chosen estimation method

Usage

```
infer_mixture(
  reference,
  mixture,
  gen_start_col,
  method = "MCMC",
  alle_freq_prior = list(const_scaled = 1),
  pi_prior = NA,
  pi_init = NULL,
  reps = 2000,
  burn_in = 100,
  pb_iter = 100,
  prelim_reps = NULL,
  prelim_burn_in = NULL,
  sample_int_Pi = 1,
  sample_theta = TRUE,
  pi_prior_sum = 1
)
```

Arguments

`reference`	a dataframe of two-column genetic format data, proceeded by "repunit", "collection", and "indiv" columns. Does not need "sample_type" column, and will be overwritten if provided
`mixture`	a dataframe of two-column genetic format data. Must have the same structure as `reference` dataframe, but "collection" and "repunit" columns are ignored. Does not need "sample_type" column, and will be overwritten if provided
`gen_start_col`	the first column of genetic data in both data frames
`method`	a choice between "MCMC", "PB", "BR" methods for estimating mixture proportions
`alle_freq_prior`	a one-element named list specifying the prior to be used when generating Dirichlet parameters for genotype likelihood calculations. Valid methods include `"const"`, `"scaled_const"`, and `"empirical"`. See `?list_diploid_params` for method details.

pi_prior — The prior to be added to the collection allocations, in order to generate pseudo-count Dirichlet parameters for the simulation of new pi vectors in MCMC. Default value of NA leads to the calculation of a symmetrical prior based on pi_prior_sum. To provide other values to certain collections, you can pass in a data frame with two columns, "collection" listing the relevant collection, and "pi_param" listing the desired prior for that collection. Specific priors may be listed for as few as one collection. The special collection name "DEFAULT_PI" is used to set the prior for all collections not explicitly listed; if no "DEFAULT_PI" is given, it is taken to be 1/(# collections).

pi_init — The initial value to use for the mixing proportion of collections. This lets the user start the chain from a specific value of the mixing proportion vector. If pi_init is NULL (the default) then the mixing proportions are all initialized to be equal. Otherwise, you pass in a data frame with one column named "collection" and the other named "pi_init". Every value in the pi_init column must be strictly positive (> 0), and a value must be given for every collection. If they sum to more than one the values will be normalized to sum to one.

reps — the number of iterations to be performed in MCMC

burn_in — how many reps to discard in the beginning of MCMC when doing the mean calculation. They will still be returned in the traces if desired.

pb_iter — how many bootstrapped data sets to do for bootstrap correction using method PB. Default is 100.

prelim_reps — for method "BR", the number of reps of conditional MCMC (as in method "MCMC") to perform prior to MCMC with baseline resampling. The posterior mean of mixing proportions from this conditional MCMC is then used as pi_init in the baseline resampling MCMC.

prelim_burn_in — for method "BR", this sets the number of sweeps out of prelim_reps that should be discarded as burn in when preparing the posterior means of the mixing proportions to be set as pi_init in the baseline resampling MCMC.

sample_int_Pi — how many iterations between storing the mixing proportions trace. Default is 1. Can't be 0. Can't be so large that fewer than 10 samples are taken from the burn in and the sweeps.

sample_theta — for method "BR", whether or not the function should store the posterior mean of the updated allele frequences. Default is TRUE

pi_prior_sum — For pi_prior = NA, the prior on the mixing proportions is set as a Dirichlet vector of length C, with each element being W/C, where W is the pi_prior_sum and C is the number of collections. By default this is 1. If it is made much smaller than 1, things could start to mix more poorly.

Details

"MCMC" estimates mixing proportions and individual posterior probabilities of assignment through Markov-chain Monte Carlo conditional on the reference allele frequencies, while "PB" does the same with a parametric bootstrapping correction, and "BR" runs MCMC sweeps while simulating reference allele frequencies using the genotypes of mixture individuals and allocations from the previous sweep. All methods default to a uniform 1/(# collections or RUs) prior for the mixing proportions.

Value

Tidy data frames in a list with the following components: mixing_proportions: the estimated mixing proportions of the different collections. indiv_posteriors: the posterior probs of fish being from each of the collections. mix_prop_traces: the traces of the mixing proportions. Useful for computing credible intervals. bootstrapped_proportions: If using method "PB" this returns the bootstrap corrected reporting unit proportions.

Examples

```
mcmc <- infer_mixture(reference = small_chinook_ref,
                      mixture = small_chinook_mix,
                      gen_start_col = 5,
                      method = "MCMC",
                      reps  = 200)
```

---

perfect_chinook *perfect-assignment genetic data for chinook.*

Description

This is just like the chinook data, but only has 7 loci and all loci are fixed in fortuitous patterns so that every single collection is easily resolved. This is primarily useful for testing purposes.

Source

Made it up!

---

perfect_chinook_mix *perfect-assignment mixture genetic data for chinook.*

Description

This is similar to the chinook_mix data, but only has 7 loci and all loci are fixed in fortuitous patterns so that every single collection is easily resolved. This is primarily useful for testing purposes. The name of the individual has its collection inside the colons.

Source

Made it up!

`read_gsi_sim` *read a gsi_sim formatted input file into a tibble that rubias can use*

Description

Note that this relies on a system call to awk. It probably won't work on Windows.

Usage

```
read_gsi_sim(path, sample_type, repunits = NULL)
```

Arguments

`path`	path to the gsi_sim file
`sample_type`	should be "reference" or "mixture" depending on what kind of file it is
`repunits`	the gsi_sim reporting units file. Has no effect if sample_type is "mixture". If sample_type is "reference" and this is left as NULL, then all collections will be put in the the "default_repu".

`rubias` *rubias: Bayesian inference from the conditional genetic stock identification model*

Description

Read the "rubias-overview" vignette for information on data input formats and how to use the package

the `rubias` main high-level functions

The following functions are wrappers, designed for user-friendly input and useful output:

`infer_mixture` is used to perform genetic stock identification. Options include standard MCMC and the parametric bootstrap bias correction.

`self_assign` does simple self-assignment of individuals in a reference data set to the various collections in the reference data set.

`assess_reference_loo` does leave-one-out based simulations to predict how accurately GSI can be done.

`assess_reference_mc` uses Monte-Carlo cross-validation based simulations to predict how accurately GSI can be done.

`assess_pb_bias_correction` attempts to demonstrate how much (or little) improvement can be expected from the parametric bootstrap correction given a particular reference data set.

genetic data format

See the vignette.

example data

alewife, blueback, and chinook are genetic data sets that are useful for playing around with rubias and testing it out.

`self_assign` *Do leave-one-out self-assignment of individuals in a reference baseline*

Description

Returns a tidy data frame

Usage

```
self_assign(
  reference,
  gen_start_col,
  preCompiledParams = NULL,
  alle_freq_prior = list(const_scaled = 1)
)
```

Arguments

`reference`	a two-column format genetic dataset, with "repunit", "collection", and "indiv" columns, as well as a "sample_type" column that has some "reference" entries
`gen_start_col`	the first column of genetic data in `reference`
`preCompiledParams`	Users should never use this option. It is here only so that this function can be called on a precompiled set of parameters with infer_mixture. Don't use this, unless you are one of the package developers...
`alle_freq_prior`	a one-element named list specifying the prior to be used when generating Dirichlet parameters for genotype likelihood calculations. Valid methods include `"const"`, `"scaled_const"`, and `"empirical"`. See `?list_diploid_params` for method details.

Value

a tibble ...

Examples

```
ale_sa <- self_assign(alewife, 17)
```

sim_spec_examples *List of example ways of specifying repunit and collection quantities in simulations*

Description

This is just a list of tibbles that can be passed to the alpha_repunit or the alpha_collection parameters in, for example, assess_reference_loo.

Source

Made it up!

small_chinook_mix *Small sample of SNP data from Chinook salmon taken in May/August 2015 from California fisheries*

Description

This is simply a sample of 100 fish from chinook.

Source

Southwest Fisheries Science Center, Santa Cruz, CA

small_chinook_ref *SNP data from selected chinook reference populations*

Description

A small number of poulations from the Chinook salmon baseline data similar to that which can be downloaded from https://datadryad.org/stash/dataset/doi:10.5061/dryad.574sv. This data set includes 91 SNPs and 909 fish.

Format

A tbl_df-ed (from dplyr) data frame with 909 rows and 185 variables. The first three columns are

repunit (chr) the reporting unit that the individual is in

pop (chr) the population from which the individual was sampled

ID (chr) Unique identifier of the individual fish

The remaining columns are two columns for each locus. These columns are named like, "Locus.1" and "Locus.2" for the first and second gene copies at that locus. For example, "Ots_104569-86.1" and "Ots_104569-86.2". The locus columns are ints and missing data is denoted by NA.

Source

https://datadryad.org/stash/dataset/doi:10.5061/dryad.574sv

`write_gsi_sim_mixture` *Write a mixture data frame to gsi_sim format baseline and repunits file*

Description

Note, this is only intended to work with integer-valued alleles, at the moment. It was just written for testing and verifying that things are working correctly.

Usage

```
write_gsi_sim_mixture(mix, gen_start_col, mixprefix)
```

Arguments

`mix`	mixture data frame
`gen_start_col`	column in which the genetic data start
`mixprefix`	path to write the mixture file to. The mixture collection name + .txt will be appended to this. This path can include directories if they exist. An example would be "./my_gsi_data/mixture". This is a required argument.

Examples

```
# this writes to file prefix "mixfile" in a temporary directory
dd <- tempdir()
prefix <- file.path(dd, "mixfile")

# print that
prefix

# note that in practice you will probably want to specify
# your own directory...

# run the function
write_gsi_sim_mixture(chinook_mix, 5, prefix)

# see where those files live:
dir(dd, pattern = "mixfile*", full.names = TRUE)
```

write_gsi_sim_reference *Write a reference data frame to gsi_sim format baseline and repunits file*

Description

Note, this is only intended to work with integer-valued alleles, at the moment. It was just written for testing and verifying that things are working correctly.

Usage

```
write_gsi_sim_reference(
  ref,
  gen_start_col,
  baseout = "baseline.txt",
  repout = "repunits.txt"
)
```

Arguments

ref	reference data frame
gen_start_col	column in which the genetic data start
baseout	path to write the baseline file to. Required.
repout	path to write the repunits file to. Required.

Examples

```
# create a temp directory to put example outputs
dd <- tempdir()
basefile <- file.path(dd, "baseline.txt")
repunitsfile <- file.path(dd, "repunits.txt")

# print those
basefile
repunitsfile

# note that in practice you will probably want to specify
# your own filepaths...

# run the function
write_gsi_sim_reference(alewife, 17, basefile, repunitsfile)
```

Index