Package 'dartR.sim'

May 8, 2026

Type Package

Title Computer Simulations of 'SNP' Data

Version 1.2.2

Date 2026-02-17

Revision Elastic Elapid

Description Allows to simulate SNP data using genlight objects. For example, it is straight forward to simulate a simple drift scenario with exchange of individuals between two populations or create a new genlight object based on allele frequencies of an existing genlight object.

Encoding UTF-8

Depends R (>= 4.1.0), dartR.base, dartR.data, ggplot2, dartR.popgen

Imports adegenet (>= 2.0.0), shiny, fields, utils, methods, stringi, stringr, data.table, Rcpp, shinyBS, shinyjs, shinythemes, shinyWidgets, hierfstat, reshape2, foreach, ggrepel, dplyr, doParallel

License GPL (>= 3)

RoxygenNote 7.3.3

NeedsCompilation no

Author Jose L. Mijangos [aut, cre], Bernd Gruber [aut], Arthur Georges [aut], Carlo Pacioni [aut], Diana Robledo-Ruiz [aut], Peter J. Unmack [ctb], Oliver Berry [ctb]

URL https://green-striped-gecko.github.io/dartR/, https://github.com/green-striped-gecko/dartR.sim

BugReports https://github.com/green-striped-gecko/dartR.sim/issues

Maintainer Jose L. Mijangos <luis.mijangos@gmail.com>

Repository CRAN

Date/Publication 2026-03-17 11:40:02 UTC

Contents

`gl.diagnostics.sim` *Comparing simulations against theoretical expectations*

Description

Comparing simulations against theoretical expectations

Usage

```
gl.diagnostics.sim(
  x,
  Ne,
  iteration = 1,
  pop_he = 1,
  pops_fst = c(1, 2),
  plot_theme = theme_dartR(),
  plot.file = NULL,
  plot.dir = NULL,
  verbose = NULL
)
```

Arguments

`x`	Output from function `gl.sim.WF.run` [required].
`Ne`	Effective population size to use as input to compare theoretical expectations [required].
`iteration`	Iteration number to analyse [default 1].
`pop_he`	Population name in which the rate of loss of heterozygosity is going to be compared against theoretical expectations [default 1].

pops_fst	Pair of populations in which FST is going to be compared against theoretical expectations [default c(1,2)].
plot_theme	User specified theme [default theme_dartR()].
plot.file	Name for the RDS binary file to save (base name only, exclude extension) [default NULL]
plot.dir	Directory in which to save files [default = working directory]
verbose	Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log ; 3, progress and results summary; 5, full report [default NULL, unless specified using gl.set.verbosity].

Details

Two plots are presented comparing the simulations against theoretical expectations:

1. Expected heterozygosity under neutrality (Crow & Kimura, 1970, p. 329) is calculated as:
   Het = He0(1-(1/2Ne))^t,
   where Ne is effective population size, He0 is heterozygosity at generation 0 and t is the number of generations.
2. Expected FST under neutrality (Takahata, 1983) is calculated as:
   FST=1/(4Nem(n/(n-1))^2+1),
   where Ne is effective populations size of each individual subpopulation, m is dispersal rate and n the number of subpopulations (always 2).

Value

Returns plots comparing simulations against theoretical expectations

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

References

- Crow JF, Kimura M. An introduction to population genetics theory. An introduction to population genetics theory. 1970.
- Takahata N. Gene identity and genetic differentiation of populations in the finite island model. Genetics. 1983;104(3):497-512.

Examples

```
ref_table <- gl.sim.WF.table(file_var=system.file('extdata',
'ref_variables.csv', package = 'dartR.sim'),interactive_vars = FALSE)

res_sim <- gl.sim.WF.run(file_var = system.file('extdata',
 'sim_variables.csv', package ='dartR.sim'),ref_table=ref_table,
 interactive_vars = FALSE,number_pops_phase2=2,population_size_phase2="10 10")

 res <- gl.diagnostics.sim(x=res_sim, Ne=10)
```

`gl.report.nall` *Report allelic retention and simulate a rarefaction curve*

Description

This function reports per-population allele counts and simulates a rarefaction-style curve showing the proportion of the dataset's total allelic diversity captured as progressively more individuals are sampled.

Usage

```
gl.report.nall(
  x,
  simlevels = seq(1, nInd(x), 5),
  reps = 10,
  plot.colors.pop = gl.colors("dis"),
  ncores = 2,
  plot.display = TRUE,
  plot.theme = theme_dartR(),
  plot.dir = NULL,
  plot.file = NULL,
  verbose = NULL
)
```

Arguments

`x`	Name of the genlight/dartR object containing the SNP data. The object needs to have no missing data as subsampling from missing data is not possible. So we recommend to filter by callrate using a threshold of 1 [required].
`simlevels`	A vector that defines the different levels the combined population should be subsampled [default seq(1,nInd(x),5)].
`reps`	Number of replicate subsamples per sample size [default 10].
`plot.colors.pop`	A color palette for population plots or a list with as many colors as there are populations in the dataset [default gl.colors("dis")].
`ncores`	Number of cores to be used for parallel processing [default 10].
`plot.display`	Specify if plot is to be produced [default TRUE].
`plot.theme`	A 'ggplot2' theme object for styling the plot [default theme_dartR()].
`plot.dir`	Directory to save the plot RDS files [default as specified by the global working directory or tempdir()].
`plot.file`	Filename (minus extension) for the RDS plot file [Required for plot save].
`verbose`	Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log; 3, progress and results summary; 5, full report [default 2, unless specified using gl.set.verbosity].

Details

The function estimates how sampling effort affects observed allelic diversity by repeatedly subsampling individuals from the pooled set of all individuals at user-defined sample sizes (‘simlevels‘), with each subsample replicated (‘reps‘ times). The maximum attainable allele count is first determined by pooling all individuals into a single group; all simulation outputs and per-population observations are then normalized to this pooled maximum and expressed as a proportion of alleles retained.

For each target sample size, replicated subsamples are aggregated to yield the mean, minimum, and maximum proportions of alleles retained. A plot is produced showing (i) the mean rarefaction curve with an uncertainty ribbon (min–max across replicates) and (ii) points for each empirical population at its observed sample size and retained proportion.

How to use the output

- Assess genetic diversity and sampling sufficiency. The curve indicates how quickly allelic diversity accumulates with additional individuals, and where diminishing returns begin. - Interpret population points relative to the curve.

- Above the curve: population retains more allelic diversity than expected for its sample size (e.g., unusually high diversity or more private/low-frequency alleles).
- On/within the ribbon: diversity consistent with random sampling from the pooled dataset at that size.
- Below the curve: population retains fewer alleles than expected, suggesting reduced diversity (e.g., drift, bottleneck), uneven missingness, or data-quality issues.

Value

A list with three elements:

- ‘sim‘: ‘data.frame‘ with columns ‘Npop‘ (sample size), ‘mnall‘ (mean proportion retained), ‘low‘ (minimum), and ‘high‘ (maximum) across replicates.
- ‘points‘: ‘data.frame‘ with observed per-population values at their actual sample sizes (columns include ‘popname‘, ‘Npop‘, and scaled ‘N.all‘).
- ‘p1‘: a ‘ggplot‘ object showing the rarefaction curve, uncertainty ribbon, and per-population points.

Author(s)

Custodian: Bernd Gruber – Post to https://groups.google.com/d/forum/dartr

Examples

```
dummy <- gl.report.nall(possums.gl[c(1:5,31:35),], simlevels=seq(1,10,3),
reps=5, ncores=2)
```

gl.sim.create_dispersal

Creates a dispersal file as input for the function gl.sim.WF.run

Description

This function writes a csv file called "dispersal_table.csv" which contains the dispersal variables for each pair of populations to be used as input for the function gl.sim.WF.run.

The values of the variables can be modified using the columns "transfer_each_gen" and "number_transfers" of this file.

See documentation and tutorial for a complete description of the simulations. These documents can be accessed by typing in the R console: browseVignettes(package="dartR")

Usage

```
gl.sim.create_dispersal(
  number_pops,
  dispersal_type = "all_connected",
  number_transfers = 1,
  transfer_each_gen = 1,
  outpath = tempdir(),
  outfile = "dispersal_table.csv",
  verbose = NULL
)
```

Arguments

number_pops	Number of populations [required].
dispersal_type	One of: "all_connected", "circle" or "line" [default "all_connected"].
number_transfers	Number of dispersing individuals. This value can be . modified by hand after the file has been created [default 1].
transfer_each_gen	Interval of number of generations in which dispersal occur. This value can be modified by hand after the file has been created [default 1].
outpath	Path where to save the output file. Use outpath=getwd() or outpath='.' when calling this function to direct output files to your working directory [default tempdir(), mandated by CRAN].
outfile	File name of the output file [default 'dispersal_table.csv'].
verbose	Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log; 3, progress and results summary; 5, full report [default 2, unless specified using gl.set.verbosity].

Value

A csv file containing the dispersal variables for each pair of populations to be used as input for the function gl.sim.WF.run.

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

See Also

gl.sim.WF.run

Other simulation functions: gl.sim.WF.run(), gl.sim.WF.table()

Examples

```
gl.sim.create_dispersal(number_pops=10)
```

---

gl.sim.emigration *Simulates emigration between populations*

Description

A function that allows to exchange individuals of populations within a genlight object (=simulate emigration between populations).

There are two ways to specify emigration. If an emi.table is provided (a square matrix of dimension of the populations that specifies the emigration from column x to row y), then emigration is deterministic in terms of numbers of individuals as specified in the table. If perc.mig and emi.m are provided, then emigration is probabilistic. The number of emigrants is determined by the population size times the perc.mig and then the population where to migrate to is taken from the relative probability in the columns of the emi.m table.

Be aware if the diagonal is non zero then migration can occur into the same patch. So most often you want to set the diagonal of the emi.m matrix to zero. Which individuals is moved is random, but the order is in the order of populations. It is possible that an individual moves twice within an emigration call(as there is no check, so an individual moved from population 1 to 2 can move again from population 2 to 3).

Usage

```
gl.sim.emigration(x, perc.mig = NULL, emi.m = NULL, emi.table = NULL)
```

Arguments

x	A genlight or list of genlight objects [required].
perc.mig	Percentage of individuals that migrate (emigrates = nInd times perc.mig) [default NULL].
emi.m	Probabilistic emigration matrix (emigrate from=column to=row) [default NULL]
emi.table	If presented emi.m matrix is ignored. Deterministic emigration as specified in the matrix (a square matrix of dimension of the number of populations). e.g. an entry in the 'emi.table[2,1]<- 5' means that five individuals emigrate from population 1 to population 2 (from=columns and to=row) [default NULL].

Value

A list or a single [depends on the input] genlight object, where emigration between population has happened

Author(s)

Custodian: Bernd Gruber (Post to https://groups.google.com/d/forum/dartr)

Examples

```
x <- possums.gl
#one individual moves from every population to
#every other population
emi.tab <- matrix(1, nrow=nPop(x), ncol=nPop(x))
diag(emi.tab)<- 0
np <- gl.sim.emigration(x, emi.table=emi.tab)
np
```

---

gl.sim.ind *Simulates individuals based on allele frequencies*

Description

This function simulates individuals based on the allele frequencies of a genlight object. The output is a genlight object with the same number of loci as the input genlight object.

Usage

```
gl.sim.ind(x, n = 50, popname = "pop1")
```

Arguments

x	Name of the genlight object containing the SNP data [required].
n	Number of individuals that should be simulated [default 50].
popname	A population name for the simulated individuals objects [default "pop1"].

Details

The function can be used to simulate populations for sampling designs or for power analysis. Check the example below where the effect of drift is explored, by simply simulating several generation a genlight object and putting in the allele frequencies of the previous generation. The beauty of the function is, that it is lightning fast. Be aware this is a simulation and to avoid lengthy error checking the function crashes if there are loci that have just NAs. If such a case can occur during your simulation, those loci need to be removed, before the function is called.

Value

A genlight object with n individuals.

Author(s)

Bernd Gruber (bernd.gruber@canberra.edu.au)

Examples

```
glsim <- gl.sim.ind(testset.gl, n=10, popname='sims')
glsim
###Simulate drift over 10 generation
# assuming a bottleneck of only 10 individuals
# [ignoring effect of mating and mutation]
# Simulate 20 individuals with no structure and 50 SNP loci
founder <- glSim(n.ind = 20, n.snp.nonstruc = 50, ploidy=2)
#number of fixed loci in the first generation
res <- sum(colMeans(as.matrix(founder), na.rm=TRUE) %%2 ==0)
simgl <- founder
#49 generations of only 10 individuals
for (i in 2:50) {
   simgl <- gl.sim.ind(simgl, n=10, popname='sims')
   res[i]<- sum(colMeans(as.matrix(simgl), na.rm=TRUE) %%2 ==0)
}
plot(1:50, res, type='b', xlab='generation', ylab='# fixed loci')
```

---

gl.sim.ind.af *Simulate diploid genotypes from per-population allele frequencies*

Description

This function generates a diploid SNP dataset by sampling genotypes for a specified number of individuals per population from user-provided allele frequencies. The result is returned as an ‘adegenet::genlight‘ object with population and individual metadata.

Usage

```
gl.sim.ind.af(df, pop.sizes)
```

Arguments

df	A ‘data.frame‘ with **three** columns: (1) population name, (2) locus name, and (3) frequency of the first allele (numeric in \[0, 1\]). The function internally renames these to ‘popn‘, ‘locus‘, and ‘frequency‘.
pop.sizes	A numeric (integer) vector of population sizes, with one element **per unique population** in ‘df‘, in the same order as ‘unique(df$popn)‘.

Details

The input ‘df‘ must have three columns: population name, locus name, and the frequency of the first allele for that population–locus combination. For each population, the function simulates two haploid chromosomes per individual by independently drawing alleles at each locus according to the provided allele frequency, then merges the two chromosomes into diploid genotypes (0, 1, 2 copies of the first allele). The procedure assumes Hardy–Weinberg proportions and linkage equilibrium (i.e., loci are sampled independently and there is no within-population structure beyond the supplied allele frequencies).

Sex labels are assigned as "Male"/"Female" in alternating blocks (stored as factors ‘"m"‘/‘"f"‘ in the returned object), and a placeholder phenotype is set to ‘"control"‘ for all individuals. Locus allele labels are initialized to ‘"G/C"‘ as a placeholder. Computation of chromosomes and genotype strings is implemented with ‘Rcpp‘ for speed.

Value

A ‘genlight‘ object with:

- diploid SNP genotypes encoded as allele counts (0, 1, 2 for copies of the first allele),
- ‘pop()‘ set to population names,
- individual IDs of the form ‘"0_<popIndex>_<i>"‘,
- ‘other$ind.metrics‘ containing ‘sex‘ (‘"m"‘/‘"f"‘) and ‘phenotype‘ (‘"control"‘).

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

Examples

```
# if (isTRUE(getOption("dartR_fbm"))) platypus.gl <- gl.gen2fbm(platypus.gl)
t1 <- gl.filter.callrate(platypus.gl[1:20,1:200],threshold = 1)
r1 <- gl.allele.freq(t1, by='popxloc' )
r2 <- r1[,c("popn",'locus',"frequency")]
res <- gl.sim.ind.af(df = r2, pop.sizes= c(10,10,10))
```

---

gl.sim.mutate *Simulates mutations within a genlight object*

Description

This script is intended to be used within the simulation framework of dartR. It adds the ability to add a constant mutation rate across all loci. Only works currently for biallelic data sets (SNPs). Mutation rate is checking for all alleles position and mutations at loci with missing values are ignored and in principle 'double mutations' at the same loci can occur, but should be rare.

Usage

```
gl.sim.mutate(x, mut.rate = 1e-06)
```

Arguments

x	Name of the genlight object containing the SNP data [required].
mut.rate	Constant mutation rate over nInd*nLoc*2 possible locations [default 1e-6].

Value

Returns a genlight object with the applied mutations

Author(s)

Bernd Gruber (Post to https://groups.google.com/d/forum/dartr)

Examples

```
b2 <- gl.sim.mutate(bandicoot.gl,mut.rate=1e-4 )
#check the mutations that have occurred
table(as.matrix(bandicoot.gl), as.matrix(b2))
```

---

gl.sim.Neconst *Simulate a population with constant mutation rate*

Description

This function simulates a population with a constant mutation rate using a beta distribution for allele frequencies.

Usage

```
gl.sim.Neconst(ninds, nlocs, mutation_rate = 1e-08, verbose = 0)
```

Arguments

ninds	Number of individuals in the population [required].
nlocs	Number of loci in the population [required].
mutation_rate	Mutation rate per generation (default is 1e-8) [default 1e-8].
verbose	Verbosity level (default is 0).

Details

The function generates a genlight object with the specified number of individuals and loci, simulating allele frequencies based on a beta distribution. The mutation rate is used to calculate theta, which in turn is the parameter in the beta function.

Value

A genlight object representing the simulated population.

Author(s)

Bernd Gruber (Post to https://groups.google.com/d/forum/dartr)

Examples

```
# Simulate a population with 50 individuals and 4000 loci
gg <- gl.sim.Neconst(ninds = 50, nlocs = 4000, mutation_rate = 1e-8, verbose = 0)
dartR.popgen::gl.sfs(gg)
```

---

gl.sim.offspring *Simulates offspring based on alleles provided by parents*

Description

This takes a population (or a single individual) of fathers (provided as a genlight object) and mother(s) and simulates offspring based on 'random' mating. It can be used to simulate population dynamics and check the effect of those dynamics and allele frequencies, number of alleles. Another application is to simulate relatedness of siblings and compare it to actual relatedness found in the population to determine kinship.

Usage

```
gl.sim.offspring(
  fathers,
  mothers,
  noffpermother,
  sexratio = 0.5,
  popname = "offspring",
  verbose = NULL
)
```

Arguments

fathers	Genlight object of potential fathers [required].
mothers	Genlight object of potential mothers simulated [required].
noffpermother	Number of offspring per mother [required].
sexratio	The sex ratio of simulated offspring (females / females +males, 1 equals 100 percent females) [default 0.5.].
popname	population name of the returned genlight object [default "offspring"].
verbose	Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log; 3, progress and results summary; 5, full report [default 2, unless specified using gl.set.verbosity].

Value

A genlight object with n individuals.

Author(s)

Bernd Gruber (Post to https://groups.google.com/d/forum/dartr)

Examples

```
#Simulate 10 potential fathers
gl.fathers <- glSim(10, 20, ploidy=2)
#Simulate 10 potential mothers
gl.mothers <- glSim(10, 20, ploidy=2)
res <- gl.sim.offspring(gl.fathers, gl.mothers, 2, sexratio=0.5)
```

---

gl.sim.WF.run *Runs Wright-Fisher simulations*

Description

This function simulates populations made up of diploid organisms that reproduce in non-overlapping generations. Each individual has a pair of homologous chromosomes that contains interspersed selected and neutral loci. For the initial generation, the genotype for each individual's chromosomes is randomly drawn from distributions at linkage equilibrium and in Hardy-Weinberg equilibrium.

See documentation and tutorial for a complete description of the simulations. These documents can be accessed at https://github.com/green-striped-gecko/dartR/wiki/Simulations-tutorial

Take into account that the simulations will take a little longer the first time you use the function gl.sim.WF.run() because C++ functions must be compiled.

Usage

```
gl.sim.WF.run(
  file_var,
  ref_table,
  x = NULL,
  file_dispersal = NULL,
  number_iterations = 1,
  every_gen = 10,
  sample_percent = 50,
  store_phase1 = FALSE,
  interactive_vars = TRUE,
  seed = NULL,
  verbose = NULL,
  ...
)
```

Arguments

file_var	Path of the variables file 'sim_variables.csv' (see details) [required if interactive_vars = FALSE].
ref_table	Reference table created by the function gl.sim.WF.table [required].
x	Name of the genlight object containing the SNP data to extract values for some simulation variables (see details) [default NULL].
file_dispersal	Path of the file with the dispersal table created with the function gl.sim.create_dispersal [default NULL].
number_iterations	Number of iterations of the simulations [default 1].
every_gen	Generation interval at which simulations should be stored in a genlight object [default 10].
sample_percent	Percentage of individuals, from the total population, to sample and save in the genlight object every generation [default 50].
store_phase1	Whether to store simulations of phase 1 in genlight objects [default FALSE].
interactive_vars	Run a shiny app to input interactively the values of simulations variables [default TRUE].
seed	Set the seed for the simulations [default NULL].
verbose	Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log; 3, progress and results summary; 5, full report [default 2, unless specified using gl.set.verbosity].
...	Any variable and its value can be added separately within the function, will be changed over the input value supplied by the csv file. See tutorial.

Value

Returns genlight objects with simulated data.

Author(s)

Custodian: Luis Mijangos

See Also

gl.sim.WF.table

Other simulation functions: gl.sim.WF.table(), gl.sim.create_dispersal()

Examples

```
ref_table <- gl.sim.WF.table(file_var=system.file("extdata",
"ref_variables.csv", package = "dartR.sim"),interactive_vars = FALSE)

res_sim <- gl.sim.WF.run(file_var = system.file("extdata",
 "sim_variables.csv", package ="dartR.sim"),ref_table=ref_table,
 interactive_vars = FALSE)
```

`gl.sim.WF.table` *Creates the reference table for running gl.sim.WF.run*

Description

This function creates a reference table to be used as input for the function `gl.sim.WF.run`. The created table has eight columns with the following information for each locus to be simulated:

- q - initial frequency.
- h - dominance coefficient.
- s - selection coefficient.
- c - recombination rate.
- loc_bp - chromosome location in base pairs.
- loc_cM - chromosome location in centiMorgans.
- chr_name - chromosome name.
- type - SNP type.

The reference table can be further modified as required.

See documentation and tutorial for a complete description of the simulations. These documents can be accessed at http://georges.biomatix.org/dartR

Usage

```
gl.sim.WF.table(
  file_var,
  x = NULL,
  file_targets_sel = NULL,
  file_r_map = NULL,
  interactive_vars = TRUE,
  seed = NULL,
  verbose = NULL,
  ...
)
```

Arguments

`file_var`	Path of the variables file 'ref_variables.csv' (see details) [required if interactive_vars = FALSE].
`x`	Name of the genlight object containing the SNP data to extract values for some simulation variables (see details) [default NULL].
`file_targets_sel`	Path of the file with the targets for selection (see details) [default NULL].
`file_r_map`	Path of the file with the recombination map (see details) [default NULL].

`interactive_vars`
: Run a shiny app to input interactively the values of simulation variables [default TRUE].

`seed`
: Set the seed for the simulations [default NULL].

`verbose`
: Verbosity: 0, silent or fatal errors; 1, begin and end; 2, progress log; 3, progress and results summary; 5, full report [default 2, unless specified using gl.set.verbosity].

`...`
: Any variable and its value can be added separately within the function, will be changed over the input value supplied by the csv file. See tutorial.

Details

Values for the variables to create the reference table can be submitted into the function interactively through a Shiny app if interactive_vars = TRUE. Optionally, if interactive_vars = FALSE, values for variables can be submitted by using the csv file 'ref_variables.csv' which can be found by typing in the R console: system.file('extdata', 'ref_variables.csv', package ='dartR.data').

The values of the variables can be modified using the third column ("value") of this file.

If a genlight object is used as input for some of the simulation variables, this function access the information stored in the slots x$position and x$chromosome.

Examples of the format required for the recombination map file and the targets for selection file can be found by typing in the R console:

- system.file('extdata', 'fly_recom_map.csv', package ='dartR.data')
- system.file('extdata', 'fly_targets_of_selection.csv', package ='dartR.data')

To show further information of the variables in interactive mode, it might be necessary to call first: 'library(shinyBS)' for the information to be displayed.

Value

Returns a list with the reference table used as input for the function `gl.sim.WF.run` and a table with the values variables used to create the reference table.

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

See Also

`gl.sim.WF.run`

Other simulation functions: `gl.sim.WF.run()`, `gl.sim.create_dispersal()`

Examples

```
ref_table <- gl.sim.WF.table(file_var=system.file("extdata",
"ref_variables.csv", package = "dartR.sim"),interactive_vars = FALSE)

res_sim <- gl.sim.WF.run(file_var = system.file("extdata",
 "sim_variables.csv", package ="dartR.sim"),ref_table=ref_table,
```

```
  interactive_vars = FALSE)
```

`interactive_reference` *Shiny app for the input of the reference table for the simulations*

Description

Shiny app for the input of the reference table for the simulations

Usage

```
interactive_reference()
```

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

---

`interactive_sim_run` *Shiny app for the input of the simulations variables*

Description

Shiny app for the input of the simulations variables

Usage

```
interactive_sim_run()
```

Author(s)

Custodian: Luis Mijangos – Post to https://groups.google.com/d/forum/dartr

Index