Package 'CKMRpop'

July 17, 2021

Type Package

Title Forward-in-Time Simulation and Tallying of Pairwise Relationships

Version 0.1.3

Description Provides an R wrapper around the program 'spip' (<https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1471-8286.2005.00884.x>), a C program for the simulation of pedigrees within age-structured populations with user-specified life histories. Also includes a variety of functions to parse 'spip' output to compile information about related pairs amongst simulated, sampled individuals, to assess the feasibility and potential accuracy of close-kin mark-recapture (CKMR). Full documentation and vignettes are mirrored at <https://eriqande.github.io/CKMRpop/index.html> and can be read online there.

License CC0

Encoding UTF-8

LazyData true

LinkingTo Rcpp

Depends R (>= 3.5.0)

Imports dplyr, ggforce, ggplot2, ggraph, igraph, magrittr, purrr, Rcpp (>= 1.0.4), readr, stats, stringr, tibble, tidygraph, tidyr, vroom

Suggests knitr, rmarkdown, tidyverse

VignetteBuilder knitr

RoxygenNote 7.1.1

NeedsCompilation yes

Author Eric C. Anderson [aut, cre] (<https://orcid.org/0000-0003-1326-0840>)

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

Repository CRAN

Date/Publication 2021-07-17 10:00:06 UTC

R topics documented:

`amm2tibble` *convert an ancestry-matching matrix to a ggplot-able tibble*

Description

This merely converts a matrix to a tibble that can be plotted easily using ggplot.

Usage

```
amm2tibble(M)
```

Arguments

`M`	an ancestry-match matrix

Value

amm2tibble() returns a tibble with three columns:

- x: the 1-based index of the row of input matrix,
- y: the 1-based index of the column of the input matrix,
- amm: the logical value (TRUE/FALSE) of the (x,y)-th cell of the input matrix.

Examples

```
# convert one of the simple example AMMs to a tibble
amm2tibble(example_amms$Father_Offspring_2gen)
```

---

compile_related_pairs *compile pairwise relationships from the samples*

Description

Run this on some of the output from slurp_spip().

Usage

```
compile_related_pairs(S)
```

Arguments

S	a tibble. In the context of this package this tibble is typically going to often be the samples component of the output slurped up from spip with slurp_spip(). More generally, it is a tibble that must have the columns: • ID: the id of the sample • ancestors: a list column of the ancestor vectors of each individual • relatives: a list column of the vectors of individual samples (including self) that each individual is related to.

Value

a tibble with columns id_1 and id_2 for each pair. Any additional columns outside of relatives will be joined with _1 and _2 suffixes. In a typical run slurped up from spip this leads to the following columns:

- id_1: the id of the first sample of the pair,
- id_2: the id of the 2nd sample of the pair,
- conn_comp: the index of the connected component to which the pair belongs,
- dom_relat: the dominant relationship that the pair shares,
- max_hit: the number of shared ancestors at the level of the dominant relationship

- dr_hits: a list column of two-vectors—the number of shared ancestors at the level of the dominant relationship in the upper and lower quadrants, respectively of the ancestry match matrix. If the relationship is symmetrical, the two values are the same.
- upper_member: for non-symmetrical relationships, a 1 or a 2 indicating which member of the pair is the one that is typically older (i.e. the uncle in an uncle-nephew relationship), or NA if the relationship is symmetrical.
- times_encountered: the number of times this pair was encountered when processing the output of the depth first search algorithm that found these pairs. Not typically used for downstream analyses.
- primary_shared_ancestors: a list columns of two-vectors. The first element of each is the the position in the ancestry vector of id_1's primary shared ancestor. The second element is the same for id_2.
- psa_tibs: like primary_shared_ancestor but a list column of tibbles.
- pop_pre_1, pop_post_1, pop_dur_1: the population from which the id_1 individual was sampled during the prekill, postkill, or during-reproduction sampling episodes, respectively. NA for episodes in which the individual was not sampled
- pop_pre_2, pop_post_2, pop_dur_2: same as above for the id_2 individual.
- sex_1: sex of the id_1 individual,
- sex_2: sex of the id_2 individual,
- born_year_1: birth year of the id_1 individual,
- born_year_2: birth year of the id_2 individual,
- samp_years_list_pre_1: list column of years during which the id_1 individual was sampled during the prekill episode.
- samp_years_list_dur_1: list column of years during which the id_1 individual was sampled during reproduction.
- samp_years_list_post_1: list column of years during which the id_1 individual was sampled during the postkill episode.
- samp_years_list_1: by default this column is identical to samp_years_list_post_1 and is the column used in downstream plotting by some functions. If you want to use a different column, for example samp_years_list_pre_1 for the downstream plotting, then set the value of samp_years_list_1 to the same values,
- samp_years_list_pre_2, samp_years_list_dur_2, samp_years_list_post_2, samp_years_list_2: same as above but for individual with id_2,
- born_pop_1: index of population in which id_1 was born,
- ancestors_1: ancestry vector of id_1,
- born_pop_2: index of population in which id_2 was born,
- ancestors_2: ancestry vector of id_2,
- anc_match_matrix: the ancestry match matrix (a logical matrix) for the pair.

Examples

```
C <- compile_related_pairs(three_pops_with_mig_slurped_results$samples)
```

`count_and_plot_ancestry_matrices` *Count up the number of different kinds of related pairs and make a plot*

Description

This counts up all the different types of pairwise relationships and makes a plot (or multiple pages of them). For an example, see the Vignette: `vignette("species_1_simulation",package = "CKMRpop")`.

Usage

```
count_and_plot_ancestry_matrices(
  Pairs,
  nrow = 6,
  ncol = 5,
  white_ball_size = 1
)
```

Arguments

`Pairs`	a tibble like that returned by `compile_related_pairs()`.
`nrow`	the number of rows to plot per page
`ncol`	the number of columns to plot per page
`white_ball_size`	the size of the white dot plotted on top of the primary shared ancestors' cells.

Value

Returns a list with the following components:

- `highly_summarised`: a tibble reporting counts of different types of relationships with the columns:
  - `dom_relat`: The name of the dominant relationship (e.g. Si, PO, A, etc).
  - `max_hit`: The number of primary shared ancestors.
  - `n`: The number of occurrences of this type of relationship.
- `dr_counts`: A tibble with the counts of all the different, unique ancestry match matrices observed, with the following columns:
  - `dom_relat`: The dominant relationship type
  - `dr_hits`: list column with the number of primary shared ancestors in the upper and lower quadrants of the ancestry match matrix (values are repeated for symmetrical relationships).
  - `max_hit`: the number of primary shared ancestors.
  - `anc_match_matrix`: a list column of ancestry match matrices.

  - n: The number of pairs with this type of ancestry match matrix
  - tot_dom: The total number of pairs within the given dominant relationship category.
  - ID: A unique, properly sorting name for this category for placing a title on facets of plots of these ancestry match matrices.
- dr_plots: a list named by the dominant relationship. Each component is a ggplot object which is a plot of the ancestry match matrices, faceted by their ID's as given in dr_counts.
- anc_mat_counts: A tibble summarizing the counts of different ancestry match matrices without regard to dominant relationship type, etc. Rows are arranged in descending order of number of pairs observed with each ancestry match matrix. It has the columns:
  - anc_match_matrix: a list column of ancestry match matrices.
  - n: the number of such matrices observed.
- anc_mat_plots: a list of ggplot pages. These are plots faceted by ancestry match matrix of all ancestry match matrices observed, ordered by number of occurrences (as given in anc_mat_counts).

`count_and_plot_mate_distribution` *Count and plot the number of mates each individuals has produced offspring with*

Description

This just tallies up the information from the pedigree. It will plot things faceted by pop (over rows) and sexes (over columns).

Usage

```
count_and_plot_mate_distribution(P)
```

Arguments

P	the pedigree from the simulation, like that returned in the pedigree component of the list returned by slurp_spip().

Value

A list with two components with names:

- mate_counts: A tibble with information about the number of mates with which a parent produced offspring each year. It has the columns:
  - sex: the sex of this parent
  - year: the year during which the mating occurred
  - pop: the population this parent was in
  - parent: the ID of the parent
  - num_offs: the number of offspring this parent had in total

  - num_mates: the number of mates this parent had

- plot_mate_counts: a ggplot object, faceted on a grid by population in columns and sex in rows. The x-axis is the number of offspring (in a season), the y-axis is the number of mates in a season, and the fill color of the grid gives the number of parents with that number of offspring and mates.

Examples

```
result <- count_and_plot_mate_distribution(three_pops_no_mig_slurped_results$pedigree)

# have a look at the results:
result$mate_counts

result$plot_mate_counts
```

---

downsample_pairs *downsample the number of individuals sampled*

Description

This discards individuals from the sample, randomly, until the desired number of samples is achieved, then it returns only those pairs in which both members are part of the retained samples.

Usage

```
downsample_pairs(S, P, n)
```

Arguments

S	the tibble of samples with columns at least of ID and samp_years_list. Typically this will be what is returned in the samples component from slurp_spip().
P	the tibble of pairs. Typically this will be what has been returned from compile_related_pairs().
n	The desired number of individuals (or instances, really, see below) to retain in the sample.

Value

This returns a list with two components as follows:

- ds_samples: A tibble like S except having randomly removed individuals so as to only have n left.
- ds_pairs: A tibble like P except having removed any pairs that include individuals that were not retained in the sample.

Examples

```
# prepare some input
S <- three_pops_with_mig_slurped_results$samples
P <- compile_related_pairs(three_pops_with_mig_slurped_results$samples)
result <- downsample_pairs(S, P, n = 500)

# print the result
result
```

`example_amms` *a list of examples of ancestry-match matrices*

Description

This is a list of matrices with names that describe what they represent. The names are Relationship_Number-of-Generations, like: "Unrelated_2gen", "Self_2gen", "Unrelated_3gen", "Unrelated_4gen", "Unrelated_1gen", "Father_Offspring_2gen", "FullSibling_2gen", "PatHalfSibling_2gen", "FullCousin_2gen", "HalfAuntNiece_2gen".

Usage

```
example_amms
```

Source

I just made these for some illustrative figures in the manuscript about this R package.

`ggplot_census_by_year_age_sex` *Just a simple plot function*

Description

Easy to do, but quicker to have it wrapped up in a plot.

Usage

```
ggplot_census_by_year_age_sex(census)
```

Arguments

`census`	a tibble of census counts with columns `year` and `age`, and then the counts of the different sexes in columns named `male`, and `female`.

Value

ggplot_census_by_year_age_sex() returns a ggplot object which is a stacked barplot with year on the x-axis, counts on the y-axis with fill mapped to age. It is facet-gridded with sex in the columns and populations in the rows.

Examples

```
# A single population example
g <- ggplot_census_by_year_age_sex(species_1_slurped_results$census_postkill)

# a three-population example
g3 <- ggplot_census_by_year_age_sex(three_pops_with_mig_slurped_results$census_postkill)
```

---

half_first_cousin_amm *A half-first cousin ancestry match matrix*

Description

Just a simple AMM to use in some examples

Usage

```
half_first_cousin_amm
```

Source

Simply wrote this down.

---

install_spip *Download the spip binary and install it where CKMRpop expects it*

Description

This checks the operating system and installs the correct version (either Darwin or Linux for Mac or Linux, respectively.) To install the spip binary this function downloads it from its GitHub site. It also installs a windows implementation of awk.

Usage

```
install_spip(Dir = tempfile())
```

Arguments

Dir	the directory to install spip into. Because of restrictions on functions writing to the user's home filespace, this is set, by default, to a temporary directory. But to really use this function to install spip, this parameter must be set to system.file(package = "CKMRpop").

Value

No return value. Called for side effect of installing the 'spip' binary.

Examples

```
## Not run:
install_spip(Dir = system.file(package = "CKMRpop"))

## End(Not run)
```

`leslie_from_spip` *Return a Leslie-like matrix from the spip parameters*

Description

Here we take the survival rates for females and males and the sex ratio, as well as the annual new cohort size (assumed constant), and we make a leslie-like matrix to compute the stable age distribution.

Usage

```
leslie_from_spip(P, C)
```

Arguments

`P`	a named list of the spip parameters.
`C`	the constant size of the newborn cohort each year

Value

This function returns a list with the following components:

- `stable_age_distro_fem`: a vector of the expected number of females in each age group of 0 up to MaxAge-1 once the stable age distribution has been reached. Note that this corresponds to the PREKILL_CENSUS from spip. In this vector, the size of the MaxAge group is left out because this is how it is needed to be to insert into the `--initial-males` and `--initial-females` options in spip(). If you want the size of all age classes, use the output list component `stable_age_distro_fem_with_max_age_class`, described below.
- `stable_age_distro_male`: same as above, but for males.
- `stable_age_distro_fem_with_max_age_class`: The expected number of females from age 0 to MaxAge once the stable age distribution has been reached.
- `stable_age_distro_male_with_max_age_class`: same as above, but for females.
- `female_leslie_matrix`: The Leslie matrix implied by the spip parameters in P.

Examples

```
result <- leslie_from_spip(species_1_life_history, 300)

# print the result list:
result
```

`plot_amm_from_matrix` *plot an ancestry matrix (or multiple such matrices) from its (their) matrix form*

Description

For illustration purposes, if you want to simply plot an ancestry matrix (or several) to show particular values, then this is the handy function for you.

Usage

```
plot_amm_from_matrix(X)
```

Arguments

`X`	input tibble with a factor or character column `ID` that gives the "name" of the ancestry matrix that will be used if you want to facet over the values in `ID`. And also `X` must have a list column `anc_match_matrix` each element of which is a logical ancestry match matrix. `X` may have a list column of tibbles called `psa_tibs` that says which cells are the primary shared ancestors.

Value

`plot_amm_from_matrix()` returns a ggplot object: each facet is an image of the ancestry match matrix. It is facet-wrapped over the values in the ID column of `X`.

Examples

```
# get some input: all the 2-generation AMMs in `example_amms`
X <- example_amms[stringr::str_detect(names(example_amms), "2gen$")] %>%
    tibble::enframe(name = "ID", value = "anc_match_matrix")

# plot those
g <- plot_amm_from_matrix(X) +
    ggplot2::facet_wrap(~ ID)
```

plot_conn_comps *plot the graph showing the connected components*

Description

This is a simple wrapper for some tidygraph/ggraph functions that will let you find the connected components of a graph in which the related pairs are connected by edges. It also makes a plot of them.

Usage

```
plot_conn_comps(Pairs)
```

Arguments

Pairs	the tibble that comes out of compile_related_pairs(). For this function it must have, at least the columns id_1, id_2, and dom_relat.

Details

Note that it appears that the 'ggraph' package must be loaded for the plot output of this function to print correctly.

Value

This returns a list with two components:

- conn_comps: a tibble with three columns:
  - name: the name of the sample
  - cluster: the index of the connected component the sample belongs to
  - cluster_size: the number of samples belonging to that cluster
- plot: a ggraph/ggplot plot object showing the connected components as vertices with edges between them.

Examples

```
# get a Pairs tibble from the stored data
Pairs <- compile_related_pairs(species_1_slurped_results_1gen$samples)
PCC <-  plot_conn_comps(Pairs)

# look at the conn_comps:
head(PCC$conn_comps)

# if you want to print the plot, that seems to require
# loading the ggraph library
library(ggraph)
PCC$plot
```

`relationship_zone_names` *relationship zone names*

Description

A simple character vector of 15 relationship zones in the order they are encountered when traversing an ancestry match matrix out to four generations

Source

I simply defined these

`relationship_zone_perimeters` *Return the perimeters of all the relationship zones*

Description

This is primarily for plotting a figure in the paper about this package, showing where all the relationship zones are. It merely cycles over the possible relationships in `relationship_zone_names` and produces one or two rows in a tibble for each that has the corners of the rectangle of that zone in the columns xmin, xmax, ymin, and ymax. It is designed to be overlaid upon the ancestry_match_matrix plots. There are some additional columns that give us the midpoint of the area, etc.

Usage

```
relationship_zone_perimeters()
```

Value

Returns a tibble with the following columns:

- `which_matrix`: a column of values `M1` or `M2`. M1 denotes that the row's values are for the relationship zone found in or below the lower diagonal of the ancestry match matrix and M2 denotes that the row's value are of the zone found in the upper part of the ancestry match matrix. Symmetrical relationships are considered to be M1.
- `zone`: The abbreviation for the relationship (e.g., Se, PO, Si, etc.)
- `xmin`: The left-hand x value of the zone.
- `xmax`: The right-hand x value of the zone.
- `ymin`: The bottom y value of the zone.
- `ymax`: The top y value of the zone.
- `area`: The area in unit squares of the zone.
- `xmid`: The x midpoint of the zone.
- `ymid`: The y midpoint of the zone.

Examples

```
relationship_zone_perimeters()
```

`run_spip` *Run spip in a user-specified directory*

Description

This runs it in a directory and the output from stdout goes into a big file spip_out.txt in that directory. Currently this is pretty bare bones.

Usage

```
run_spip(
  pars,
  dir = tempfile(),
  spip_seeds = ceiling(runif(2, 1, 1e+09)),
  num_pops = 1,
  allele_freqs = list(c(0.5, 0.5))
)
```

Arguments

`pars`	A named list of parameter values.
`dir`	The directory to run it in. Defaults to a temp directory, which will be unique every time it is run.
`spip_seeds`	a vector of two positive integers. These get written to the file spip_seeds, which is used by spip to seed its random number generator. By default, R supplies these two integers from its own random number generator. This way reproducible results from spip can be obtained by calling `set.seed()` from within R before calling `run_spip()`. For the most part, the user should never really have to directly supply a value for spip_seeds.
`num_pops`	the number of demes that are being simulated. This is still being implemented...
`allele_freqs`	a list of allele frequencies provided if you want to simulate unlinked genotypes for the sampled individuals. The default is simply a single locus with two alleles at frequencies 0.5, 0.5, which is provided because spip has to be given some allele frequencies if sampling is to be carried out. Note that a user-specified value to this option should only be given if you want to actually simulate some genetic data from the sampled individuals. The length of the list should be the number of loci desired, and the length of each element should be the number of alleles. For examples for three loci with 2, 3, and 4 equifrequent alleles, respectively, you would provide `list(c(0.5,0.5),c(0.3333,0.3333,0.3333),c(0.25,0.25,0.25,0.25))`. Note that allele frequencies will be normalized to sum to one within each locus.

Details

This creates a temporary directory and runs spip in that directory, redirecting stdout and stderr to files. It then processes the output using awk to create a collection of files. If spip throws an error, the contents of stderr are written to the screen to notify the user of how to correct their input.

For a full example of its use see the Vignette: `vignette("species_1_simulation",package = "CKMRpop")`.

Value

Returns the path to the temporary directory were `spip` was run and where the processed output files can be found to be read in using `slurp_spip()`.

`slurp_spip` *Read in the pedigree, census, and sampling information from the spip run*

Description

This function is run after `run_spip()`. It assumes that `run_spip()` has left the files: `spip_pedigree.tsv`, `spip_prekill_census.tsv`, and `spip_samples.tsv`, `spip_postkill_census.tsv`, `spip_deaths.tsv`, `spip_genotypes.tsv`, and `spip_migrants.tsv` inside the directory where `run_spip()` was run.

Usage

```
slurp_spip(dir, num_generations)
```

Arguments

`dir`	the path to the directory where spip was run. This is returned by `run_spip()`.
`num_generations`	how many generations back do you wish to consider for find relatives of each sampled individual. 0 means just the individual themselves (so, not very interesting, and you likely wouldn't ever use it. 1 means up to and including the parents; 2 means up to and including the grandparents; 3 means up to and including the great grandparents; and so forth.

Details

For an example of its use, see the Vignette: `vignette("species_1_simulation",package = "CKMRpop")`.

Value

A list of tibbles. Each tibble is a named component of the return list. The names are as follows:

- pedigree
- census_prekill,
- census_postkill,
- samples,
- deaths,
- genotypes,
- migrants

You can inspect some example output in the package data object three_pops_with_mig_slurped_results

Examples

```
# see Vignette: vignette("species_1_simulation", package = "CKMRpop")
```

---

species_1_life_history

a list of life-history / life-table data for a hypothetical species

---

Description

species_1 for examples.

Usage

```
species_1_life_history
```

Source

Just values that might be typical of a fish.

`species_1_slurped_results` *The result of running spip in the species_1_simulation vignette.*

Description

This is stored as package data so that the vignette can be written even if spip is not installed on the system.

Usage

```
species_1_slurped_results
```

Source

Simulation results

`species_1_slurped_results_100_loci` *The result of running spip in the species_1_simulation vignette with 100 loci.*

Description

This is stored as package data so that the vignette can be written even if spip is not installed on the system. This particular version stores the results of running `run_spip()` calling for 100 loci segregating in the population, then slurping the results up with `slurp_spip()`.

Usage

```
species_1_slurped_results_100_loci
```

Source

Simulation results

`species_1_slurped_results_1gen`
The result of running spip in the species_1_simulation vignette and slurping out with num_generations = 1.

Description

This is stored as package data so that the vignette can be written even if spip is not installed on the system.

Usage

```
species_1_slurped_results_1gen
```

Source

Simulation results

`species_2_life_history`
a list of life-history / life-table data for another hypothetical species

Description

species_2 for examples. Note that I just set the male and female rates and parameters similar.

Usage

```
species_2_life_history
```

Source

This is something used for simulation testing

spip_exists *return TRUE if spip exists where it should be installed.*

Description

This just checks for the file where it gets installed with `install_spip()`. This is exported so it can be used to control the flow in the vignettes, so that vignettes can still be written using saved results when spip is not installed (i.e., at CRAN, etc.)

Usage

```
spip_exists()
```

Value

A single logical scalar, either TRUE or FALSE.

Examples

```
# will be FALSE unless spip has been externally installed
spip_exists()
```

spip_help *print the abbreviated usage information from spip*

Description

This simply calls spip with the `--help` option.

Usage

```
spip_help()
```

Value

This returns the exit status of `spip` if spip is installed, but the return value is of little use. Mainly this is run for the side effect of printing the `spip` help menu to the console.

Examples

```
## Not run: spip_help()
```

spip_help_full *print the full usage information from spip*

Description

This simply calls spip with the `--help-full` option.

Usage

```
spip_help_full()
```

Value

This returns the exit status of `spip` if spip is installed, but the return value is of little use. Mainly this is run for the side effect of printing the `spip` full help menu to the console.

Examples

```
## Not run: spip_help()
```

summarize_offspring_and_mate_numbers *Summarize the distribution of number of offspring and number of mates*

Description

More later

Usage

```
summarize_offspring_and_mate_numbers(
  census_postkill,
  pedigree,
  deaths,
  lifetime_hexbin_width = c(1, 1),
  contrib_bin_width = 0.01
)
```

Arguments

census_postkill
: a tibble with the postkill numbers of individuals. This is here so we know the total number of individuals that could have reproduced in a given year.

pedigree
: a tibble with columns of `year`, `kid`, `pa`, and `ma`. The IDs of the' individuals must by like MX_Y where M means male, X is the birth year, and Y is the unique ID number.

deaths
: a tibble with columns of `ID`, `year`, and `age`, giving the years and ages at which different individuals died.

lifetime_hexbin_width
: a vector of length two. The first element is the width in the age direction of each hexbin and the second is the width in the lifetime number of offspring direction for the `plot_lifetime_output_vs_age_at_death` output plot.

contrib_bin_width
: width of bins of histogram of contribution of parents of each age and sex to the offspring.

Value

A list with three components, each of them a ggplot object:

- `plot_age_specific_number_of_offspring`: a ggplot object that plots boxplots and jittered points. The x-axis are the ages of the individuals; the y-axis shows the number of offspring. Summarized over the entire spip simulation. This is faceted by sex.
- `plot_lifetime_output_vs_age_at_death`: a ggplot object. This is a hexbin plot. The x-axis are age-at-death bins, the y axis are bins of total number of offspring produced in a lifetime. The fill color of each bin gives the number of individuals with that age at death and number of offspring encountered over the whole simulation. Plot is faceted by sex.
- `plot_fraction_of_offspring_from_each_age_class`: a ggplot object. This shows the distribution over all years of the simulation, of the fraction of offspring produced each year that were produced by males or females of a given age (the plots are facet-wrapped by both age and sex). The blue vertical line gives the mean.

Examples

```
# get stored slurped output for an example
X <- species_1_slurped_results
g <- summarize_offspring_and_mate_numbers(
  X$census_postkill,
  X$pedigree,
  X$deaths
)

# Now g is a list holding three plots, accessible like this:

# g$plot_age_specific_number_of_offspring

# g$plot_lifetime_output_vs_age_at_death
```

```
# g$plot_fraction_of_offspring_from_each_age_class
```

`summarize_survival_from_census` *Summarize annual sex-and-age-specific survival rates from the census information*

Description

The prekill census in year t+1 is the post-kill census in year t, so we can use the prekill census to record the realized fraction of individuals of each age and sex that survived the death episode in each year. In the output survival in year t is the fraction of j-year olds in year t that survive to be j+1 year-olds in year t+1.

Usage

```
summarize_survival_from_census(
  census,
  fem_surv_probs = NULL,
  male_surv_probs = NULL,
  nbins = 10
)
```

Arguments

`census`	a tibble of census counts with columns `year` and `age`, and then the counts of the different sexes in columns named `male`, and `female`.
`fem_surv_probs`	a vector of the parameters used for the simulation. If present these are put on the histogram plots. If you provide one of these, you have to provide both.
`male_surv_probs`	a vector of the parameters used for the simulation. If present these are put on the histogram plots.
`nbins`	number of bins for the histograms

Details

This function does not track migrants. Another one is eventually in order that accounts for migrants out of the population. Also, the plots here might not play well with multiple populations.

Value

A list with components:

- `survival_tibble`: A tibble with the following columns:
  - `year`: The year
  - `pop`: The population whose census is being counted

  - age: The age of individuals
  - sex: The sex of individuals
  - n: The number of individuals alive and present of sex sex and age age in year year in pop pop.
  - cohort: The birth year of these individuals
  - surv_fract: The fraction of the n individuals that survive to have age age + 1 in year year + 1.

- plot_histos_by_age_and_sex: A ggplot object of histograms of observed survival fractions facet-wrapped by age and sex. Blue vertical lines are the observed means and dashed vertical red lines are the expected values given the simulation parameters.

Examples

```
result <- summarize_survival_from_census(
  species_1_slurped_results$census_prekill,
  species_1_life_history$`fem-surv-probs`,
  species_1_life_history$`male-surv-probs`
)

# print the results if you want
result$survival_tibble
result$plot_histos_by_age_and_sex
```

---

three_pops_no_mig_slurped_results

The result of running spip and slurping the output in the three population case with no migration

---

Description

This is stored as package data so that the vignette can be written even if spip is not installed on the system.

Usage

```
three_pops_no_mig_slurped_results
```

Source

Simulation results

three_pops_with_mig_slurped_results

The result of running spip and slurping the output in the three population case with migration

Description

This is stored as package data so that the vignette can be written even if spip is not installed on the system.

Usage

```
three_pops_with_mig_slurped_results
```

Source

Simulation results

uncooked_spaghetti

Summarise kin-pair information and use it to create uncooked spaghetti plots

Description

This gives a nice graphical summary of all the kin pairs along with when they were sampled and their age at the time of sampling and their sex. #' In order to visually summarize all the kin pairs that were found, with specific reference to their age, time of sampling, and sex, I find it helpful to use what I have named the "Uncooked Spaghetti Plot". There are multiple subpanels on this plot. Here is how to read/view these plots:

- Each row of subpanels is for a different dominant relationship, going from closer relationships near the top and more distant ones further down. You can find the abbreviation for the dominant relationship at the right edge of the panels.
- In each row, there are four subpanels: `F->F`, `F->M`, `M->F`, and `M->M`. These refer to the different possible combinations of sexes of the individuals in the pair.
  - For the non-symmetrical relationships these are naturally defined with the first letter (`F` for female or `M` for male) denoting the sex of the "upper_member" of the relationship. That is, if it is PO, then the sex of the parent is the first letter. The sex of the non-upper-member is the second letter. Thus a `PO` pair that consists of a father and a daughter would appear in a plot that is in the `PO` row in the `M->F` column.
  - For the symmetrical relationships, there isn't a comparably natural way of ordering the individuals' sexes for presentation. For these relationships, the first letter refers to the sex of the individual that was sampled in the earliest year. If both individuals were sampled in the same year, and they are of different sexes, then the female is considered the first one, so those all go on the `F->M` subpanel.

- On the subpanels, each straight line (i.e., each piece of uncooked spaghetti) represents a single kin pair. The two endpoints represent the year/time of sampling (on the x-axis) and the age of the individual when it was sampled (on the y-axis) of the two members of the pair.
  - If the relationship is non-symmetrical, then the line is drawn as an arrow pointing from the upper member to the lower member.
  - The color of the line gives the number of shared ancestors (`max_hit`) at the level of the dominant relationship. This is how you can distinguish full-sibs from half-sibs, etc.

Usage

```
uncooked_spaghetti(Pairs, Samples, jitter_age = 0.2, jitter_year = 0.2)
```

Arguments

`Pairs`	The tibble of kin pairs that comes out of `compile_related_pairs()`.
`Samples`	The tibble of samples that comes out of `slurp_spip()`.
`jitter_age`	half the width of the uniform jitter window around age
`jitter_year`	half the width of the uniform jitter window around sampling year

Value

`uncooked_spaghetti()` returns a list with two components: `input_data` and `plot`. `plot` is a ggplot object of the plot described above in "Description." `input_data` is, itself, another list with the following named components:

- `P5`: A tibble that is a processed version of the `Pairs` input. This is what goes into making the ggplot.
- `age_grid`: A tibble giving the coordinates for placing the alternating pink and white horizontal background rectangles on the plot.
- `year_grid`: A tibble giving the coordinates for placing the alternating pink and white vertical background rectangles on the plot.

Examples

```
# get the input variables
# only take the first 50 samples to reduce time for example
Samples <- species_1_slurped_results$samples[1:50, ]
Pairs <- compile_related_pairs(Samples)
result <- uncooked_spaghetti(Pairs, Samples)

# produce the plot with:
# result$plot
```

Index