Package: Pv3Rs (via r-universe)

June 28, 2024

Title What the Package Does (One Line, Title Case)

Version 0.0.0.9000

Description What the package does (one paragraph).

License `use_mit_license()`, `use_gpl3_license()` or friends to pick a license

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports dplyr, fields, grDevices, gtools, igraph, matrixStats, MCMCpack, multicool, partitions, RColorBrewer

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Config/testthat/edition 3

VignetteBuilder knitr

Depends R (>= 2.10)

LazyData true

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

RemoteUrl https://github.com/aimeertaylor/Pv3Rs

RemoteRef HEAD

RemoteSha 5fd5ff35d12f2fd9fd2749b21925d12efa40687b

Contents

`compatible_rstrs` *Find all vectors of recurrence states compatible with relationship graph*

Description

Finds all possible recurrence states for each recurrence compatible with the relationship graph, then takes the Cartesian product to get all vectors of recurrence states. For a recurrence to be a recrudescence, all edges between the recurrent infection and the immediately preceding infection must be clonal edges. For a recurrence to be a reinfection, all edges between the recurrent infection and any preceding infection must be stranger edges. All recurrences may possibly be relapses.

Usage

```
compatible_rstrs(RG, gs_per_ts)
```

Arguments

`RG`	Relationship graph; see `enumerate_RGs`.
`gs_per_ts`	List of vectors of genotypes for each infection.

Value

Vector of strings (consisting of "C", "L", "I" for recrudescence, relapse, reinfection respectively) compatible with relationship graph.

Examples

```
MOIs <- c(2, 2, 1)
RG <- enumerate_RGs(MOIs, igraph = T)[[175]]
gs_per_ts <- split(paste0("g", 1:sum(MOIs)), rep(1:length(MOIs), MOIs))
# 1st recurrence can't be recrudescence, 2nd recurrence can't be reinfection
plot_RG(RG, edge.curved = 0.2)
compatible_rstrs(RG, gs_per_ts) # "LL" "IL" "LC" "IC"
```

`compute_posterior` *Compute the posterior distribution of recurrence states*

Description

Entry point to Bayesian inference for *P. vivax* recurrence states based on genetic data. Specifically, this function finds the posterior probabilities of relapse, reinfection and recrudescence from genetic data. Please note that the progress bar does not necessarily increment at a uniform rate, and may sometimes appear to be stuck while the code is still running.

Usage

```
compute_posterior(y, fs, prior = NULL, return.RG = FALSE, return.logp = FALSE)
```

Arguments

`y`	Observed data in the form of a list of lists. Alleles should be provided as a set of distinct alleles for each infection (per marker). The number of entries is the number of episodes in chronological order. Episode names can be specified, but they are not used. Each episode is in turn a list of observed alleles for each marker, which must be named, or `NA` if not observed. For a given marker, alleles are modeled as categorical random variables. As such, allele names are arbitrary, but must correspond with frequency names (see examples below). The same names can be used for alleles belonging to different markers. As such, frequencies must be specified per named allele per named marker.
`fs`	List of allele frequencies as vectors. Names of the list must match with the marker names in `y`. Within lists (i.e., for each marker), frequencies must be specified per allele name.
`prior`	Matrix of prior probabilities of the recurrence states for each recurrent episode. Each row corresponds to an episode in chronological order. The column names must be C, L, and I for recrudescence, relapse and reinfection respectively. Row names can be specified by they are not used. If prior is not provided, a uniform prior will be used.
`return.RG`	Boolean for whether to return the relationship graphs, defaults to `FALSE`.
`return.logp`	Boolean for whether to return the log-likelihood for each relationship graph, defaults to `FALSE`. Setting this to `FALSE` allows for permutation symmetries to be exploited to save computational time, see `enumerate_alleles`. Setting this to `TRUE` will result in longer runtimes, especially in the case of a larger multiplicity of infection.

Details

We enumerate all possible relationship graphs between genotypes, where each pair of genotypes may be clones, siblings, or strangers, each with a different level of expected genetic relatedness. The likelihood of a sequence of recurrence states can be determined from the likelihood of all relationship graphs compatible with said sequence. More details on the enumeration and likelihood calculation of relationship graphs can be found in `enumerate_RGs` and `RG_inference` respectively.

Model assumptions:

- No within-host mutations, genotyping errors, undetected alleles
- Parasites are outbred
- Relationship graphs are equally likely given recurrence states

`compute_posterior` supports missing data, which should be encoded as `NA`s. However, to avoid estimates based entirely on the weakly informative nature of multiple per-marker allele calls (see example below and `vignette("missing_data", "Pv3Rs")` for more details), we recommend against generating estimates for recurrences that have no paired data due to missingness (see Microsatellite data example in `vignette("demo", "Pv3Rs")`).

The data input expects each list of alleles (for an infection) to consist of a set of distinct alleles. Providing the function with a multiset of alleles may lead to undefined behaviour. Note that the function will automatically produce all assignments with repeated alleles.

Value

List containing:

marg Matrix of marginal posterior probabilities of the possible recurrence states for each recurrent episode, one row per reinfection.

joint Vector of joint posterior probabilities of each possible string of recurrence states.

RGs List of relationship graphs with their log-likelihoods stored. Only returned if `return.RG` is `TRUE`. See `enumerate_RGs`.

Examples

```
# ============================================================================
# Example where alleles are named arbitrarily
# ============================================================================
# Data on an enrolment episode and a recurrence:
y <- list(episode0 = list(marker1 = c("Tinky Winky", "Dipsy"),
                          marker2 = c("Laa-Laa", "Po")),
          episode1 = list(marker1 = "Tinky Winky",
                          marker2 = "Laa-Laa"))

# Allele frequencies:
fs <- list(
  marker1 = setNames(c(0.4, 0.6), c("Tinky Winky", "Dipsy")),
  marker2 = setNames(c(0.2, 0.8), c("Laa-Laa", "Po"))
)

# Compute posterior probabilities using default uniform prior, note that
```

```
# since there is only one recurrence, the marginal probabilities are the same
# as the joint probabilities:
compute_posterior(y, fs)



# ============================================================================
# Example where alleles are given numeric names: 1, 2, 3, 4, 5
# ============================================================================
# Allele frequencies with numeric names
fs <- list(m1=c('1'=0.78, '2'=0.14, '3'=0.07, '4'=0.005, '5' = 0.005),
           m2=c('1'=0.27, '2'=0.35, '3'=0.38),
           m3=c('1'=0.55, '2'=0.45))

# Data:
y <- list(enrol = list(m1=c('3','2'), m2=c('1','3'), m3=c('1','2')),
          recur1 = list(m1=c('1','4'), m2=c('1','2'), m3=c('1','2')),
          recur2 = list(m1=c('1','5'), m2=c('2','3'), m3=c('1')))

compute_posterior(y, fs)



# ============================================================================
# Example demonstrating the cosmetic-only nature of episode names. Due to the
# cosmetic-only nature of episode names, input info (episode data and prior)
# and output results should be chronologically ordered and interpreted.
# ============================================================================
y <- list(enrol = list(m1 = NA),
          recur2 = list(m1 = NA),
          recur1 = list(m1 = NA))
prior <- array(c(0.2,0.7,0.2,0.3,0.6,0), dim = c(2,3),
               dimnames = list(c("recur1", "recur2"), c("C", "L", "I")))

# The first prior row named "recur1" is returned for the first recurrence
# despite it being named "recur2" and the the second prior row named "recur2"
# is returned for the second recurrence despite it being named "recur1":
compute_posterior(y, fs, prior)
prior



#============================================================================
# compute_posterior() returns the prior when there are no data. However, the
# prior will be re-weighted if NAs encode MOIs that are incompatible with
# recrudescence. (Recrudescing parasites are clones of parasites in the
# preceding blood-stage infection. The Pv3R model assumes no within-host
# mutations, genotyping errors or undetected alleles. As such, recrudescence
# is incompatible with an MOI increase relative to the preceding infection.)
#============================================================================
# Allele frequencies:
fs <- list(m1 = setNames(c(0.25, 0.75), c("A", "Other")))
```

```
# Data on an enrolment episode and two recurrences:
y_no_data_MOIs111 <- list(enrol = list(m1 = NA),
                          recur1 = list(m1 = NA),
                          recur2 = list(m1 = NA))

y_no_data_MOIs121 <- list(enrol = list(m1 = NA),
                          recur1 = list(m1 = c(NA, NA)),
                          recur2 = list(m1 = NA))

# Compute posterior probabilities:
post3Rs_no_data_MOIs111 <- compute_posterior(y_no_data_MOIs111, fs)
post3Rs_no_data_MOIs121 <- compute_posterior(y_no_data_MOIs121, fs)

# Returns the default prior since MOIs 111 are compatible with all 3R
# sequences:
post3Rs_no_data_MOIs111

# Returns the default prior re-weighted to the exclusion of 3R sequences with
# a recrudescence at the first recurrence:
post3Rs_no_data_MOIs121



#==============================================================================
# Example demonstrating the weakly informative nature of multiple per-marker
# allele calls; see vignette("missing_data", "Pv3Rs") for more details.
#==============================================================================
fs = list(m1 = c('1' = 0.5, '2' = 0.5))
y <- list(enrol = list(m1 = c('1', '2')), recur = list(m1 = NA))

# compute_posterior() does not return the prior despite there being no
# recurrent data:
compute_posterior(y, fs)



#==============================================================================
# Example of the small but undesirable effect on the posterior of prior on
# graphs: the marginal probability that the first recurrence is a
# recrudescence increases as the number of recurrences increases even though
# only the first recurrence has data (also see vignette [to-do - base on
# MyDevFiles/Graph_prior_bias_examples.R])
#==============================================================================
# Allele frequencies:
fs <- list(m1 = setNames(c(0.25, 1-0.25), c("A", "Other")))

# Data for different scenarios; scenarios where the number of recurrences
# increases but only the first recurrence has data
ys <- list(y1 = list(enrol = list(m1 = "A"), recur1 = list(m1 = "A")),
           y2 = list(enrol = list(m1 = "A"), recur1 = list(m1 = "A"), recur2 = list(m1 = NA)),
          y3 = list(enrol = list(m1 = "A"), recur1 = list(m1 = "A"), recur2 = list(m1 = NA), recur3 = list(m1 = NA)),
          y4 = list(enrol = list(m1 = "A"), recur1 = list(m1 = "A"), recur2 = list(m1 = NA), recur3 = list(m1 = NA), rec
```

```
# Compute posterior probabilities and extract marginal probabilities:
results <- lapply(ys, function(y) compute_posterior(y, fs)$marg)

# Extract results for the first recurrence only:
first_recur <- sapply(results, function(result) result[1,])

# Plot 2D simplex
n_recur <- max(sapply(ys, length)-1)
pardefault <- par()
par(mar = c(0,0,0,0))
plot_simplex(v_labels = rownames(first_recur))

# Project probabilities onto 2D simplex coordinates
xy <- apply(first_recur, 2, project2D)

# Plot divergence from one recurrence to four:
arrows(x0 = xy["x", 1], x1 = xy["x", n_recur],
       y0 = xy["y", 1], y1 = xy["y", n_recur],
       length = 0.05, col = "red")

# Plot a point for each recurrence from one to four:
points(x = xy["x", ], y = xy["y", ], pch = ".")

# Restore plotting margins
par(mar = pardefault$mar)
```

`determine_MOIs` *Determine MOIs from unphased data*

Description

Determine MOIs from unphased data

Usage

```
determine_MOIs(y)
```

Arguments

`y`	A sequence of lists, where each list contains the genetic data at each marker as a vector. Each list corresponds to one infection.

Value

Returns a vector of the (minimum) multiplicity of infection (MOI) for each infection.

Examples

```
# two infections, three markers
y <- list(
  list(m1 = c("A", "B"), m2 = c("A"), m3 = c("C")),
  list(m1 = c("B"), m2 = c("B", "C"), m3 = c("A", "B", "C"))
)
determine_MOIs(y) # should be c(2, 3)
```

`enumerate_alleles` *Find all allele assignments for genotypes within the same infection*

Description

Note that this function is not tested for input with repeated alleles.

Usage

```
enumerate_alleles(y.inf, gs.inf, use.sym = TRUE)
```

Arguments

`y.inf`	List of unique alleles observed across markers for genotypes within one infection. Repeated alleles will lead to overcounting the assignments.
`gs.inf`	Vector of genotype names for genotypes within one infection.
`use.sym`	Boolean for permutation symmetry is exploited as a computational shortcut. Due to permutation symmetry of intra-infection genotypes, we can fix a single assignment for one of the markers whose number of alleles observed is equal to the MOI (consider it the anchor) and permute the rest, discarding combinations that under-represent the observed marker diversity. The default behaviour is to use this symmetry such that less assignments have to be considered.

Value

List of dataframes, one for each marker. The columns correspond to genotypes, while the rows correspond to allele assignments for a marker.

Examples

```
# 3 markers
y.inf <- list(m1 = c("A", "B"), m2 = c("B", "C", "D"), m3 = c("C"))
enumerate_alleles(y.inf, c("g1", "g2", "g3"))
# 6 assignments for m1 (BAA, ABA, BBA, AAB, BAB, ABB)
# 1 assignment for m2 (accounting for permutation symmetry)
# 1 assignment for m3 (CCC)
```

enumerate_CPs *Enumerate partitions induced by clonal relationships*

Description

A clonal partition is a partition of genotypes where a pair of genotypes of the same partition cell have a clonal relationship. Genotypes from the same infection cannot be clones. This code enumerates all clonal partitions, accounting for this intra-infection restriction.

Usage

```
enumerate_CPs(MOIs)
```

Arguments

MOIs	A numeric vector specifying, for each infection, the number of distinct parasite genotypes, a.k.a. the multiplicity of infection (MOI).

Value

A list of all possible partitions, where each partition is encoded as a membership vector, which indices (genotype names) with the same entry corresponding to genotypes being int he same partition cell.

Examples

```
enumerate_CPs(c(2, 2))
```

enumerate_halfsib_alleles *List all allelic draws for three half siblings*

Description

Given a specified number of alleles for a single marker, `generate_halfsib_alleles()` enumerates all the ways three half siblings can draw alleles from their respective parents by firstly enumerating all allelic combinations for the parents, and by secondly enumerating all the inheritable combinations for the children.

Usage

```
enumerate_halfsib_alleles(n_alleles)
```

Arguments

n_alleles	Positive whole number specifying a per-marker number of alleles, otherwise known as marker cardinality.

Value

A character matrix. Each column is an individual. Each row is a possible allelic draw. Alleles are represented by the first n_alleles letters of the latin alphabet.

Examples

```
generate_halfsib_alleles(3)
```

---

enumerate_RGs *Enumerate transitive relationship graphs, alternate version*

Description

A relationship graph is a complete graph on all genotypes, where each edge is annotated as a clone, sibling, or stranger edge. The enumerated relationship graphs satisfy the following constraints:

- The subgraph induced by the clone edges is a cluster graph.
- The subgraph induced by the clone edges and sibling edges is a cluster graph.
- Clone edges are only allowed for two genotypes from different infections.

Usage

```
enumerate_RGs(MOIs, igraph = TRUE)
```

Arguments

MOIs	A numeric vector specifying, for each infection, the number of distinct parasite genotypes, a.k.a. the multiplicity of infection (MOI).
igraph	Logical for whether to return igraph objects.

Details

This alternate version of enumerate_RGs_prev is based on generating set partitions. Since the clone edges induce a cluster graph, the information encoded by clonal relationships is equivalent to a partition of the genotypes. Note that genotypes from the same infection cannot belong to the same partition cell. Subsequent information encoded by sibling relationships is equivalent to further partitioning the clonal partition. There are no constraints when enumerating the sibling partitions. The data structure returned encodes each graph as a nested set partition. Each partition is represented in the form of a list of vectors (clone and sib) and as a membership vector (clone.vec and sib.vec), where each entry identifies the partition cell the corresponding index belongs to.

Value

A list of relationship graphs. If `igraph` is `FALSE`, each element is a list of four attributes:

clone A list of groups of genotypes that make up the clonal cells.

clone.vec A numeric vector indicating the clonal membership of each genotype.

sib A list of groups of clonal cells that make up the sibling cells.

sib.vec A numeric vector indicating the sibling membership of each clonal cell.

Otherwise, each element is an `igraph` object (see `enumerate_RGs_prev`) along with these four attributes. Note that the weight matrix contains information equivalent to that of the four attributes.

Examples

```
graphs <- enumerate_RGs(c(2, 1, 2), igraph=T) # 250 graphs
```

---

fs_VHX_BPD *Allele frequencies computed from example Plasmodium vivax data*

Description

The posterior mean of a multinomial-Dirichlet model with uniform prior fit to data on allele prevalence in initial episodes. Because the model is fit to allele prevalence (observed) not allele frequency (would requires integrating-out unknown multiplicities of infection) it is liable to underestimate the frequencies of common alleles and overestimate those of rare but detected alleles.

Usage

```
fs_VHX_BPD
```

Format

A list of nine markers; for each marker a named vector of allele frequencies that sum to one.

Source

- https://github.com/jwatowatson/RecurrentVivax/blob/master/RData/GeneticModel/MS_data_PooledAnalysis.RData
- https://github.com/aimeertaylor/Pv3Rs/blob/main/data-raw/fs_VHX_BPD.R

hash.IP *Convert IBD partition to a unique string for hashing*

Description

This is used for building a hash table for p(y at marker mlIBD).

Usage

```
hash.IP(IP, gs)
```

Arguments

IP	List containing vectors of genotype names with each vector corresponding to an IBD cell.
gs	Vector containing all genotype names.

Value

String where the integers in the IBD membership vector have been converted to ASCII characters.

Examples

```
gs <- paste0("g", 1:3)
IP1 <- list(c("g1", "g3"), c("g2"))
IP2 <- list(c("g2"), c("g3", "g1"))
hash1 <- hash.IP(IP1, gs)
hash2 <- hash.IP(IP2, gs)
hash1 == hash2 # TRUE, even though the order is different
```

locus_type_summary *Summarise locus-wise data types*

Description

Summarise locus-wise data types

Usage

```
locus_type_summary(y)
```

Arguments

y	A list of lists for two episodes; see compute_posterior() for more details.
m	A positive whole number indexing a marker.

Details

A function to summarise the data at each locus as one of four types:

- All match (all genotypes have the same allele).
- All diff. (all genotypes have a different allele).
- Intra-match (some intra-episode genotypes have the same allele)
- Inter-match (some inter-episode genotypes have the same allele).

  The number of apparent genotypes is the sum of the per-episode MOIs. When the total number of apparent genotypes exceeds 3, "Intra-match" excludes any "Inter-match" (i.e., it is equivalent to Inter-matches only), whereas an "Inter-match does not exclude an "Intra-match

Value

A vector of strings summarising the data at each locus.

Examples

```
# example code
y <- list(
  list(m1 = c("A", "B"), m2 = c("A"), m3 = c("C")),
  list(m1 = c("B"), m2 = c("B", "C"), m3 = c("A", "B", "C", "D"))
)

locus_type_summary(y)
```

---

msg_progress_bar-class

Message progress bar

Description

A simple progress bar to use in R packages where messages are preferred to console output. See https://gist.github.com/MansMeg/1ec56b54e1d9d238b4fd.

Fields

`iter` Total number of iterations

`i` Current iteration

`width` Width of the R console

`width_bar` Width of the progress bar

`progress` The number of character printed (continous)

`progress_step` Addition to progress per iteration

Methods

`increment()` A messagebar object.

`initialize(iter)` Initialize a messagebar object

Author(s)

Mans Magnusson (MansMeg @ github)

Examples

```
test_bar <- function(i = 10){
 bar <- msg_progress_bar(i)
 for(j in 1:i){
   bar$increment()
   Sys.sleep(4/i)
   }
 }
 test_bar(100)
```

---

plot_data *Plots the data*

Description

Plots the alleles (colours), which are observed in different episodes (rows), on different markers (columns), where episodes are grouped by patient. The patients and per-patient episodes are plotted from bottom to top. If more than one allele is detected per episode per marker, the corresponding row-column entry is subdivided into different colours. The legend depicts the alleles of the markers as the markers appear from left to right in the main plot. Otherwise stated, the legend is ordered by the order of markers stated on on the horizontal axis of the main plot. The colour scheme is adaptive. It is designed to visually differentiate the alleles as much as possible: the maximum range of qualitative scheme, with contrast of hue between adjacent colours, is always used; the adjacent colours are interpolated only if a given marker has more than 12 alleles. The names of the alleles are printed on top of their colours if marker_annotate.

Usage

```
plot_data(ys, fs = NULL, marker_annotate = TRUE)
```

Arguments

`ys`	A nested list of per-patient, per-episode, per-marker allelic data. Specifically, a per-patient list of a per-episode list of a per-marker list of character vectors of observed alleles.

fs
: A per-marker list of numeric vectors of allele frequencies. If NULL (default), for a given marker, only the alleles present in the data are represented in the legend, and each allele is represented equally. Because the colour scheme is adaptive (see introduction), the same allele will have a different colour in a plot of an alternative data list if more or fewer alleles are observed at the given marker across the alternative data list. If fs is specified, all possible alleles are represented and legend areas are proportional to allele frequencies; i.e., common alleles have relatively large legend areas, and rare alleles have relatively small legend areas. Specify fs to fix the colour of a given allele across plots of different data lists, thereby facilitating cross-comparison.

marker_annotate
: Logical. If true (default), the names of the alleles are printed on top of their colours in the legend.

Examples

```
# Examples

# Plot example Plasmodium vivax data set
plot_data(ys = ys_VHX_BPD)
plot_data(ys = ys_VHX_BPD, fs = fs_VHX_BPD)
plot_data(ys = ys_VHX_BPD, fs = fs_VHX_BPD, marker_annotate = F)

# Demonstrating the adaptive nature of the colour scheme:
ys <- ys_VHX_BPD["VHX_52"] # A single patient with
plot_data(ys) # Legend adapts to alleles detected in VHX_52 only
plot_data(ys, fs = fs_VHX_BPD)
```

---

plot_RG *Plot a relationship graph (RG)*

Description

This function is a wrapper around plot.igraph, written to group parasite genotypes by infection, both spatially and using vertex colour. Specifically, parasite genotypes within infections are vertically distributed with some horizontal jitter when layout_by_group is TRUE (default), and coloured the same. It also makes sure clonal and sibling edges are plotted differently using different line types.

Usage

```
plot_RG(
  RG,
  layout_by_group = TRUE,
  vertex_palette = "Set2",
  edge_lty = c(`0.5` = "dashed", `1` = "solid"),
```

```
  edge_col = c(`0.5` = "black", `1` = "black"),
  edge.width = 1.5,
  ...
)
```

Arguments

RG	A relationship graph, which is an igraph graph; see enumerate_RGs_prev.
layout_by_group	A logical argument which if TRUE (default) overrides the default layout of plot.igraph so that vertices that represent parasite genotypes from different infections are distributed horizontally and vertices that represent genotypes within infections are distributed vertically.
vertex_palette	A character string specifying an RColorBrewer palette. Overrides the default palette of plot.igraph.
edge_lty	A vector of edge line types corresponding to different relationships, where 0.5 represents a sibling and 1 represents a clone.
edge_col	A vector of edge colours corresponding to different relationships, where 0.5 represents a sibling and 1 represents a clone.
edge.width	Overrides the default edge.width of plot.igraph.
...	Additional arguments to pass to plot.igraph, e.g. edge.curved.

Provenance

This function was adapted from plot_Vivax_model at https://github.com/jwatowatson/RecurrentVivax/blob/master/Genetic_Model/iGraph_functions.R.

Examples

```
RGs <- enumerate_RGs_prev(c(2, 1, 1))
cpar <- par() # record current par before changing
par(mfrow = c(3, 4), mar = c(0.1, 0.1, 0.1, 0.1))
for (i in 12:23) {
  plot_RG(RGs[[i]], edge.curved = 0.1)
  box()
}
par(cpar) # reset par
```

---

plot_simplex *Plots a 2D simplex*

Description

Plots a 2D simplex, a triangle with unit sides centered at the origin, to which marginal posterior probabilities of the 3Rs can be added; see project2D() and examples below.

Usage

```
plot_simplex(v_labels = NULL)
```

Arguments

v_labels	A vector of labels with which to annotate vertices anticlockwise from the top vertex. If NULL (default), vertices are not annotated.

Examples

```
# Plot 2D simplex
plot_simplex()

xy <- project2D(v = c("C" = 1, "L" = 0, "I" = 0))
points(x = xy["x"], xy["y"], pch = "C")
text(x = xy["x"], xy["y"], labels = "(1,0,0)")

xy <- project2D(v = c("C" = 0, "L" = 1, "I" = 0))
points(x = xy["x"], xy["y"], pch = "L")
text(x = xy["x"], xy["y"], labels = "(0,1,0)")

xy <- project2D(v = c("C" = 0, "L" = 0, "I" = 1))
points(x = xy["x"], xy["y"], pch = "I")
text(x = xy["x"], xy["y"], labels = "(0,0,1)")

# ============================================================================
# Given data on an enrollment episode and a recurrence,
# compute the posterior probabilities of the 3Rs and plot the deviation of the
# posterior from the prior
# ============================================================================

# Some data:
y <- list(list(m1 = c("A", "C"), m2 = c("G", "T")), # Enrollment episode
          list(m1 = c("A"), m2 = c("G"))) # Recurrent episode

# Some allele frequencies:
fs <- list(m1 = setNames(c(0.4, 0.6), c("A", "C")),
           m2 = setNames(c(0.2, 0.8), c("G", "T")))

# A vector of prior probabilities:
prior <- array(c(0.2, 0.3, 0.5), dim = c(1,3),
               dimnames = list(NULL, c("C", "L", "I")))

# Compute posterior probabilities
post <- compute_posterior(y, fs, prior)

# Project marginal prior probabilities onto x and y coordinates:
xy_prior <- project2D(as.vector(prior))

# Project marginal posterior probabilities onto x and y coordinates:
xy_post <- project2D(as.vector(post$marg))
```

```
# Plot simplex
plot_simplex(colnames(post$marg))

# Plot the deviation of the posterior from the prior
arrows(x0 = xy_prior["x"], x1 = xy_post["x"],
       y0 = xy_prior["y"], y1 = xy_post["y"], length = 0.1)
```

`project2D` *Project 3D probability coordinates onto 2D simplex coordinates*

Description

Project three probabilities that sum to one (e.g., the marginal probabilities of the 3Rs) onto the coordinates of a 2D simplex centered at the origin (i.e., a triangle centered at (0,0) with unit sides).

Usage

```
project2D(v)
```

Arguments

v	A numeric vector of three probabilities that sum to one.

Value

A numeric vector of two coordinates that can be used to plot the probability vector v on the origin-centered 2D simplex (see `plot_simplex()`), where the top, left, and right vertices of the simplex correspond with the first, second and third entries of v respectively.

Examples

```
project2D(v = c(0.75,0.20,0.05))
```

`recombine_parent_ids` *Generate parental allocations for a meiotic tetrad*

Description

For a given set of per chromosome marker counts, generate per marker parental ids for a meiotic tetrad generated by sexual recombination (chromosomal crossovers followed by independent assortment). We assume

1. For all chromosomes, both pairs of non-sister chromatids cross over
2. One crossover per non-sister chromatid pair
3. Equal-length chromosomes (follows from above; in reality, chromosomes have different lengths, and the number of crossovers increases with length)
4. Equi-distributed markers (implicit)

Usage

```
recombine_parent_ids(per_chr_marker_counts)
```

Arguments

per_chr_marker_counts	A numeric vector of marker counts per chromosome

Examples

```
n_chrs <- 14 # P. vivax has 14 chromosomes
n_markers <- 100 # For 100 markers
chr_per_marker <- round(seq(0.51, n_chrs + 0.5, length.out = n_markers))
n_markers_per_chr <- table(chrs_per_marker) # Per chromosome marker counts
recombine_parent_ids(n_markers_per_chr)
```

RG_inference *Computes the likelihood of all relationship graphs*

Description

The likelihood of a relationship graph (RG) can be decomposed over markers, as we assume that marker data is conditionally independent across markers given the RG. The likelihood of a RG for one marker is then found by integrating over all possible IBD (identity-by-descent) partitions that are consistent with the RG. The probability distribution of these IBD partitions are always uniform. Since different RGs may use the same IBD, the likelihood of IPs for each marker are stored in hash tables for improved computational efficiency.

Usage

```
RG_inference(MOIs, fs, alleles_per_m)
```

Arguments

MOIs	A numeric vector specifying, for each infection, the number of distinct parasite genotypes, a.k.a. the multiplicity of infection (MOI).
fs	List of allele frequencies as vectors. Names of the list must match with the marker names in alleles_per_m.
alleles_per_m	List of allele assignments as dataframes for each marker. Each column corresponds to a genotype and each row corresponds to an allele assignment.

Value

List of all relationship graph objects, including their log-likelihoods as a variable under each object.

Examples

```
# 1 marker, 2 infections, MOIs = 2, 1
MOIs <- c(2, 1)
fs <- list(
  m1 = setNames(c(0.4, 0.6), c("A", "B")),
  m2 = setNames(c(0.2, 0.8), c("C", "D"))
)
al_df1 <- as.data.frame(matrix(c("A", "B", "B"), nrow = 1))
al_df2 <- as.data.frame(matrix(c(
  "C", "D", "C",
  "D", "C", "C"
), nrow = 2, byrow = T))
colnames(al_df1) <- colnames(al_df2) <- paste0("g", 1:3)
alleles_per_m <- list(m1 = al_df1, m2 = al_df2)
RGs <- RG_inference(MOIs, fs, alleles_per_m)
```

RG_to_igraph *Converts relationship graph to an* igraph *object*

Description

Makes a relationship graph object output by enumerate_RGs compatible with the igraph output of enumerate_RGs_prev.

Usage

```
RG_to_igraph(RG, gs, ts_per_gs)
```

Arguments

RG	Relationship graph output by enumerate_RGs with igraph=FALSE.
gs	Vector of genotype names.
ts_per_gs	Vector of infection numbers for each genotype. This can be inferred from the data y using rep(1:length(y), determine_MOIs(y)).

Value

An igraph object along with the original variables in RG.

Examples

```
set.seed(20)
RG <- sample_RG(c(2, 2))
RG <- RG_to_igraph(RG, c("g1", "g2"), c(1, 1, 2, 2))
```

sample_RG — *Sample a transitive relationship graph, alternate version*

Description

Uses the techniques in enumerate_RGs to uniformly sample a relationship graph. All clonal partitions are generated, and the number of sibling partitions consistent with each clonal partition is determined. A clonal partition is randomly selected with probability proportional to the corresponding number of sibling partitions, and a sibling partition is then uniformly sampled. The nested partition is equivalent to a relationship graph. See enumerate_RGs for details on the nested partition representation of a relationship graph.

Usage

```
sample_RG(MOIs, igraph = T)
```

Arguments

MOIs	A numeric vector specifying, for each infection, the number of distinct parasite genotypes, a.k.a. the multiplicity of infection (MOI).
igraph	Logical for whether to return an igraph object.

Value

A relationship graph, i.e. one entry of the list returned by enumerate_RGs.

Examples

```
set.seed(20)
RG <- sample_RG(c(2, 2))
```

split_two — *Partition a vector into at most two subvectors*

Description

Partition a vector into at most two subvectors

Usage

```
split_two(s)
```

Value

Given a vector with no repeats, returns a list consisting of

- a list that contains the original vector as its only element
- other lists where each list contains two disjoint vectors whose union covers the vector. All possible unordered pairs are included.

Examples

```
gs <- paste0("g", 1:3)
# 4 possibilities in total
# either all genotypes in one vector (1 possibility)
# or 2 genotypes in one vector and the last in one vector (3 possibilties)
split_two(gs)
```

---

ys_VHX_BPD *Example Plasmodium vivax data*

Description

Previously-published microsatellite data on Plasmodium vivax parasites extracted from patients enrolled in the Best Primaquine Dose (BPD) and Vivax History (VHX) trials; see https://www.nature.com/articles/s41467-019-13412-x.

Usage

```
ys_VHX_BPD
```

Format

A list of 217 patients; for each patient, a list of one or more episodes; for each episode, a list of three or more microsatellite markers; for each marker, a vector of observed alleles (repeat lengths). For example:

BPD_103 Patient identifier: patient 103 in the BPD trial

BPD_103_1 Episode identifier: episode one of patient 103 in the BPD trial

PV.3.27 Marker identifier: Plasmodium vivax 3.27

18 Repeat length: 18

Source

- https://github.com/jwatowatson/RecurrentVivax/blob/master/RData/GeneticModel/MS_data_PooledAnalysis.RData
- https://github.com/aimeertaylor/Pv3Rs/blob/main/data-raw/ys_VHX_BPD.R

Index