Package: MALECOT (via r-universe)

June 3, 2024

Type Package

Title Joint estimation of COI and population structure for malaria genetic data

Version 0.1.1

Maintainer Bob Verity <r.verity@imperial.ac.uk>

Description Carries out joint estimation of complexity of infection (COI) and population structure on malaria genetic data. Assumes a simple model in which individuals have genotypes sampled from one or more subpopulations, and the number of genotypes in an individual is equal to the COI, which is also unknown. All unknown parameters are inferred using MCMC.

License MIT + file LICENSE

BugReports https://github.com/bobverity/malecot/issues

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

LinkingTo Rcpp

Imports Rcpp (>= 0.12.14), parallel, coda, ggplot2, gridExtra, RColorBrewer

SystemRequirements C++11

Suggests testthat, covr, knitr, rmarkdown, tidyr, plotly, gridExtra

VignetteBuilder knitr

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

RemoteUrl https://github.com/bobverity/MALECOT

RemoteRef HEAD

RemoteSha 713d220d14f282dc63618ca10c83e2a902a83266

Contents

active_set — *Change the active set of a MALECOT project*

Description

Change the active set of a MALECOT project

Usage

```
active_set(project, set)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`set`	the new active set

Examples

```
# TODO
```

bind_data_biallelic — *Bind bi-allelic data to project*

Description

Bind data in bi-allelic format to MALECOT project. Data should be formatted as a dataframe with samples in rows and loci in columns. Genetic data should be coded as 1 (homozygote REF allele), 0 (homozygote ALT allele), or 0.5 (heterozygote). Additional meta-data columns can be specified, including a column for sample IDs and a column for sampling population.

Usage

```
bind_data_biallelic(project, df, ID_col = 1, pop_col = NULL,
  data_cols = NULL, ID = NULL, pop = NULL, missing_data = -9,
  name = NULL, check_delete_output = TRUE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`df`	a dataframe containing genetic information and optional meta-data
`ID_col`	which column of the input data contains the sample IDs. If NULL then IDs must be defined seperately through the `ID` argument
`pop_col`	which column of the input data contains the ostensible population of the samples. If NULL then populations must be defined seperately through the `pop` argument

`data_cols`	which columns of the input data contain genetic information. Defaults to all remaining columns of the data once meta-data columns have been accounted for
`ID`	sample IDs. Ignored if using the `ID_col` option
`pop`	ostensible populations. Ignored if using the `pop_col` option
`missing_data`	what value represents missing data. Defaults to -9. Must be a positive or negative integer, and cannot equal 0 or 1 as these are reserved for genetic data.
`name`	optional name of the data set to aid in record keeping
`check_delete_output`	whether to prompt the user before overwriting existing data

Examples

```
# TODO
```

`bind_data_multiallelic`

Bind multi-allelic format data to project

Description

Bind data in multi-allelic format to MALECOT project. Data should be formatted as a dataframe with three columns: "sample_ID", "locus" and "haplotype". Each row of this dataframe specifies a haplotype that was observed at that locus in that individual. Haplotypes should be coded as positive integers.

Usage

```
bind_data_multiallelic(project, df, pop = NULL, missing_data = -9,
  alleles = NULL, name = NULL, check_delete_output = TRUE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`df`	a dataframe with three columns, as decribed above
`pop`	ostensible populations of the samples
`missing_data`	what value represents missing data. Defaults to -9. Must be a positive or negative integer
`alleles`	the number of alleles at each locus. If scalar then the same number of alleles is assumed at all loci. If NULL then the number of alleles is inferred directly from data as the maximum observed value per locus
`name`	optional name of the data set to aid in record keeping
`check_delete_output`	whether to prompt the user before overwriting existing data

Examples

```
# TODO
```

`check_MALECOT_loaded` *Check that MALECOT package has loaded successfully*

Description

Simple function to check that MALECOT package has loaded successfully. Prints "MALECOT loaded successfully!" if so.

Usage

```
check_MALECOT_loaded()
```

`delete_set` *Delete parameter set*

Description

Delete a given parameter set from a MALECOT project.

Usage

```
delete_set(project, set = NULL, check_delete_output = TRUE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`set`	which set to delete. Defaults to the current active set
`check_delete_output`	whether to prompt the user before deleting any existing output

Examples

```
# TODO
```

get_ESS *Get ESS*

Description

Returns effective sample size (ESS) of chosen model run.

Usage

```
get_ESS(project, K = NULL)
```

Arguments

`project`	a MALCOT project, as produced by the function `malecot_project()`
`K`	get ESS for this value of K

Examples

```
# TODO
```

get_group_order *Match grouping against q-matrix*

Description

Compares qmatrix output for a chosen value of K against a `target_group` vector. Returns the order of `target_group` groups, such that there is the best possible alignment against the qmatrix. For example, if the vector returned is `c(2,3,1)` then the second group in the target vector should be matched against the first group in the qmatrix, followed by the third group in the target vector against the second group in the qmatrix, followed by the first group in the target vector against the third group in the qmatrix.

Usage

```
get_group_order(project, K, target_group)
```

Arguments

`project`	a MALCOT project, as produced by the function `malecot_project()`
`K`	compare against qmatrix output for this value of K
`target_group`	the target group to be aligned against the qmatrix

Examples

```
# TODO
```

`is.malecot_project` *Determine if object is of class malecot_project*

Description

Determine if object is of class malecot_project.

Usage

```
is.malecot_project(x)
```

Arguments

`x`	TODO

Details

TODO

Examples

```
# TODO
```

`MALECOT` *MALECOT package*

Description

MALECOT package

`malecot_file` *Import file*

Description

Import file from the inst/extdata folder of this package

Usage

```
malecot_file(name)
```

Arguments

`name`	name of file

`malecot_project` *Define empty malecot_project object*

Description

Define empty malecot_project object

Usage

```
malecot_project()
```

Details

TODO

Examples

```
# TODO
```

`more_colours` *Expand series of colours by interpolation*

Description

Expand a series of colours by interpolation to produce any number of colours from a given series. The pattern of interpolation is designed so that (n+1)th value contains the nth value plus one more colour, rather than being a completely different series. For example, running `more_colours(5)` and `more_colours(4)`, the first 4 colours will be shared between the two series.

Usage

```
more_colours(n = 5, raw_cols = col_hot_cold())
```

Arguments

`n`	how many colours to return
`raw_cols`	vector of colours to interpolate

`new_set` *Create new MALECOT parameter set*

Description

TODO

Usage

```
new_set(project, name = "(no name)", lambda = 1,
  COI_model = "poisson", COI_max = 20, COI_manual = NULL,
  estimate_COI_mean = TRUE, COI_mean = 3, COI_dispersion = 2,
  estimate_error = FALSE, e1 = 0, e2 = 0, e1_max = 0.2,
  e2_max = 0.2)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`name`	the name of the parameter set
`lambda`	the shape parameter(s) of the prior on allele frequencies. This prior is Beta in the bi-allelic case, and Dirichlet in the multi-allelic case. `lambda` can be: • a single scalar value, in which case the same value is used for every allele and every locus (i.e. the prior is symmetric) • a vector of values, in which case the same vector is used for every locus. Only works if the same number of alleles applies at every locus • a list of vectors specifying the shape parameter separately for each allele of each locus. The list must of length `L`, and must contain vectors of length equal to the number of alleles at that locus
`COI_model`	the type of prior on COI. Must be one of "uniform", "poisson", or "nb" (negative binomial)
`COI_max`	the maximum COI allowed for any given sample
`COI_manual`	A vector of length n (where n is the number of samples) allowing the COI to be specified manually. Positive values indicate fixed COIs that should not be updated as part of the MCMC, while -1 values indicate that COIs should be estimated. Defaults to `rep(-1,n)`, meaning all COIs will be esimated
`estimate_COI_mean`	whether the mean COI should be estimated for each subpopulation as part of the MCMC, otherwise the value `COI_mean` is used for all subpopulations. Defaults to `TRUE`. Note that mean COI estimation is only possible under the Poisson and negative binomial models (see `COI_model`)
`COI_mean`	single scalar value specifying the mean COI for all subpopulations (see `estimate_COI_mean` above)
`COI_dispersion`	the ratio of the variance to the mean of the prior on COI. Only applies under the negative binomial model. Must be >1, as a ratio of 1 can be achieved by using the Poisson distribution

`estimate_error`	whether to estimate error probabilities `e1` and `e2`
`e1`	the probability of a true homozygote being incorrectly called as a heterozygote
`e2`	the probability of a true heterozygote being incorrectly called as a homozygote
`e1_max`	the maximum possible value of `e1`
`e2_max`	the maximum possible value of `e2`

Details

TODO

Examples

```
# TODO
```

`plot_acf` *Produce MCMC autocorrelation plot*

Description

Produce MCMC autocorrelation plot of the log-likelihood

Usage

```
plot_acf(project, K = NULL, rung = NULL, col = "black")
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`rung`	which rung to plot. Defaults to the cold chain
`col`	colour of the trace

`plot_COI` *Plot COI 95% credible intervals*

Description

Plot COI 95% credible intervals of current active set

Usage

```
plot_COI(project, K = NULL)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot

Details

TODO

Examples

```
# TODO
```

`plot_COI_mean` *Plot COI_mean 95% credible intervals*

Description

Plot COI_mean 95% credible intervals of current active set

Usage

```
plot_COI_mean(project, K = NULL, deme_order = NULL)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`deme_order`	the order in which to plot demes. Defaults to increasing order

Details

TODO

Examples

```
# TODO
```

`plot_coupling` *Plot Metropolis-coupling acceptance rates*

Description

Plot Metropolis-coupling acceptance rates

Usage

```
plot_coupling(project, K = NULL)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot

`plot_density` *Produce MCMC density plot*

Description

Produce MCMC density plot of the log-likelihood

Usage

```
plot_density(project, K = NULL, rung = NULL, col = "black")
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`rung`	which rung to plot. Defaults to the cold chain
`col`	colour of the trace

plot_e *Plot error rate 95% credible intervals*

Description

Plot error rate 95% credible intervals of current active set

Usage

```
plot_e(project, K = NULL)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot

Details

TODO

Examples

```
# TODO
```

plot_GTI_path *Plot GTI path of current active set*

Description

Plot GTI path of current active set

Usage

```
plot_GTI_path(project, K = NULL, axis_type = 1)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`axis_type`	how to format the x-axis. 1 = integer rungs, 2 = values of beta

`plot_logevidence_K` *Plot log-evidence estimates over K*

Description

Plot log-evidence estimates over K

Usage

```
plot_logevidence_K(project)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`

`plot_logevidence_model` *Plot log-evidence estimates over parameter sets*

Description

Plot log-evidence estimates over parameter sets

Usage

```
plot_logevidence_model(project)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`

`plot_loglike` *Plot loglikelihood 95% credible intervals*

Description

Plot loglikelihood 95% credible intervals of current active set

Usage

```
plot_loglike(project, K = NULL, axis_type = 1,
  connect_points = FALSE, connect_whiskers = FALSE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`axis_type`	how to format the x-axis. 1 = integer rungs, 2 = values of beta, 3 = values of beta raised to the GTI power
`connect_points`	whether to connect points in the middle of intervals
`connect_whiskers`	whether to connect points at the ends of the whiskers

`plot_loglike_dignostic` *Produce diagnostic plots of log-likelihood*

Description

Produce diagnostic plots of the log-likelihood.

Usage

```
plot_loglike_dignostic(project, K = NULL, rung = NULL, col = "black")
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`rung`	which rung to plot. Defaults to the cold chain
`col`	colour of the trace

`plot_p` *Plot allele frequency 95% credible intervals*

Description

Plot allele frequency 95% credible intervals of current active set

Usage

```
plot_p(project, K = NULL, deme = 1)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`deme`	TODO

Details

TODO

Examples

```
# TODO
```

`plot_posterior_K` *Plot posterior K*

Description

Plot posterior K

Usage

```
plot_posterior_K(project)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`

`plot_posterior_model` *Plot posterior model*

Description

Plot posterior model

Usage

```
plot_posterior_model(project)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`

`plot_prior_COI` *Plot prior on COI*

Description

Produce plot of the prior on COI for given parameters. Options include the uniform distribution, and a modified form of Poisson and negative binomial distribution (see details).

Usage

```
plot_prior_COI(COI_model = "poisson", COI_mean = 3,
  COI_dispersion = 2, COI_max = 20)
```

Arguments

`COI_model`	the type of prior on COI. Must be one of "uniform", "poisson", or "nb" (negative binomial)
`COI_mean`	the prior mean (before truncating at `COI_max`). Note that this parameter only applies under the "poisson" and "nb" models
`COI_dispersion`	the ratio of the variance to the mean of the prior on COI. Only applies under the negative binomial model. Must be >1
`COI_max`	the maximum COI allowed. Distributions are truncated at this value

Details

The prior on COI can be uniform, Poisson, or negative binomial. In the uniform case there is an equal chance of any given sample having a COI between 1 and `COI_max` (inclusive). In the Poisson and negative binomial cases it is important to note that the distribution is over (COI-1), rather than over COI. This is because both Poisson and negative binomial distributions allow for 0 values, which cannot be the case here because observed samples must contain at least 1 genotype. Poisson and negative binomial distributions are also truncated at `COI_max`.

The full probability mass distribution for the Poisson case with `COI_mean`$= \mu$ and `COI_max`$= M$ can be written

$$Pr(COI = n) = z(\mu - 1)^(n - 1)exp(-(\mu - 1))/(n - 1)!$$

where z is a normalising constant that ensures the distribution sums to unity, and is defined as:

$$1/z = \sum_{i=1}^{M} (\mu - 1)^(i - 1)exp(-(\mu - 1))/(i - 1)!$$

The mean of this distribution will generally be very close to μ, and the variance will be close to $\mu - 1$ (strictly it will approach these values as M tends to infinity).

The full probability mass distribution for the negative binomial case with `COI_mean`$= \mu$, `COI_dispersion`$=$ v/μ and `COI_max`$= M$ can be written

$$Pr(COI = n) = z\Gamma(n - 1 + N)/(\Gamma(N)(n - 1)!)p^N(1 - p)^(n - 1)$$

where $N = (\mu - 1)^2/(v - \mu + 1)$, $p = (\mu - 1)/v$, and z is a normalising constant that ensures the distribution sums to unity, and is defined as:

$$1/z = \sum_{i=1}^{M} \Gamma(i - 1 + N)/(\Gamma(N)(i - 1)!)p^N(1 - p)^(i - 1)$$

The mean of this distribution will generally be very close to μ and the variance will be close to v (strictly it will approach these values as M tends to infinity).

`plot_prior_p` *Plot prior on allele frequencies*

Description

Produce plot of the prior on COI for given parameters. This prior is Beta in the bi-allelic case, and Dirichlet in the multi-allelic case.

Usage

```
plot_prior_p(lambda = 1, alleles = NULL)
```

Arguments

`lambda`	shape parameter(s) of the Beta or Dirichlet distribution. Can be a single scalar value, in which case the dimensionality is given by the number of `alleles`, or a vector of values specifying the shape parameter for each allele
`alleles`	the dimensionality of the prior. Defaults to the length of `lambda`, or to 2 of `lambda` is a scalar

---

`plot_structure` *Posterior allocation plot*

Description

Produce posterior allocation plot of current active set.

Usage

```
plot_structure(project, K = NULL, base_colours = col_hot_cold(),
  divide_ind_on = FALSE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`base_colours`	colours from which final plotting colours are taken. These will be interpolated to produce final colours
`divide_ind_on`	whether to add dividing lines between bars

`plot_trace` *Produce MCMC trace plot*

Description

Produce MCMC trace plot of the log-likelihood at each iteration.

Usage

```
plot_trace(project, K = NULL, rung = NULL, col = "black")
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	which value of K to plot
`rung`	which rung to plot. Defaults to the cold chain
`col`	colour of the trace

`print.malecot_project` *Custom print function for class malecot_project*

Description

Custom print function for class malecot_project, printing a summary of the key elements (also equivalent to `summary(x)`). To do an ordinary `print()` of all elements of the project, use the `print_full()` function.

Usage

```
## S3 method for class 'malecot_project'
print(x, ...)
```

Arguments

`x`	object of class `malecot_project`
`...`	other arguments (ignored)

`print_full` *Ordinary print function for class malecot_project*

Description

Calling `print()` on an object of class malecot_project results in custom output. This function therefore stands in for the base `print()` function, and is equivalent to running `print(unclass(x))`.

Usage

```
print_full(x, ...)
```

Arguments

`x`	object of class `malecot_project`
`...`	other arguments passed to `print()`

`recalculate_evidence` *Recalculate evidence and posterior estimates*

Description

When a new value of K is added in to the analysis it affects all downstream evidence estimates that depend on this K - for example the overall model evidence integrated over K. This function therefore looks through all values of K in the active set and recalculates all downstream elements as needed.

Usage

```
recalculate_evidence(project)
```

Arguments

`project`	a MALCOT project, as produced by the function `malecot_project()`

`run_mcmc` *Run main MCMC*

Description

Run the main MALECOT MCMC. Model parameters are taken from the current active parameter set, and MCMC parameters are passed in as arguments. All output is stored within the project.

Usage

```
run_mcmc(project, K = NULL, precision = 0.01, burnin = 1000,
  samples = 1000, rungs = 1, GTI_pow = 3, auto_converge = TRUE,
  converge_test = 100, solve_label_switching_on = TRUE,
  coupling_on = TRUE, cluster = NULL, pb_markdown = FALSE,
  store_acceptance = TRUE, store_raw = TRUE, silent = FALSE)
```

Arguments

`project`	a MALECOT project, as produced by the function `malecot_project()`
`K`	the values of K that the MCMC will explore
`precision`	the level of precision at which allele frequencies are represented in the bi-allelic case. This allows the use of look-up tables for the likelihood, which significantly speeds up the MCMC. Set to 0 to use exact values (up to C++ "double" precision) rather than using look-up tables
`burnin`	the number of burn-in iterations
`samples`	the number of sampling iterations
`rungs`	the number of temperature rungs
`GTI_pow`	the power used in the generalised thermodynamic integration method. Must be greater than 1.1
`auto_converge`	whether convergence should be assessed automatically every `converge_test` iterations, leading to termination of the burn-in phase. If `FALSE` then the full `burnin` iterations are used
`converge_test`	test for convergence every `convergence_test` iterations if `auto_converge` is being used
`solve_label_switching_on`	whether to implement the Stevens' solution to the label-switching problem. If turned off then Q-matrix output will no longer be correct, although evidence estimates will be unaffected.
`coupling_on`	whether to implement Metropolis-coupling over temperature rungs
`cluster`	option to pass in a cluster environment (see package "parallel")
`pb_markdown`	whether to run progress bars in markdown mode, in which case they are updated once at the end to avoid large amounts of output

`store_acceptance`	whether to store acceptance rates for all parameters updated by Metropolis-Hastings. Proposal distributions are tuned adaptively with a target acceptance rate of 23%
`store_raw`	whether to store raw MCMC output in addition to summary output. Setting to FALSE can considerably reduce output size in memory
`silent`	whether to suppress all console output

Examples

```
# TODO
```

`sim_data` *Simulate genetic data*

Description

Simulate genetic data from the same model used in the MALECOT inference step.

Usage

```
sim_data(n = 100, L = 24, K = 3, data_format = "biallelic",
  pop_col_on = TRUE, alleles = 2, lambda = 1,
  COI_model = "poisson", COI_max = 20, COI_manual = rep(-1, n),
  COI_mean = 3, COI_dispersion = 2, e1 = 0, e2 = 0,
  prop_missing = 0)
```

Arguments

`n`	the number of samples
`L`	the number of loci per sample
`K`	the number of subpopulations
`data_format`	whether to produce data in "biallelic" or "multiallelic" format. Note that if biallelic format is chosen then `alleles` is always set to 2
`pop_col_on`	TODO
`alleles`	the number of alleles at each locus. Can be a vector of length `L` specifying the number of alleles at each locus, or a single scalar value specifying the number of alleles at all loci
`lambda`	the shape parameter(s) of the prior on allele frequencies. This prior is Beta in the bi-allelic case, and Dirichlet in the multi-allelic case. `lambda` can be: • a single scalar value, in which case the same value is used for every allele and every locus (i.e. the prior is symmetric) • a vector of values, in which case the same vector is used for every locus. Only works if the same number of alleles applies at every locus

	• a list of vectors specifying the shape parameter separately for each allele of each locus. The list must of length `L`, and must contain vectors of length equal to the number of alleles at that locus
`COI_model`	the distribution from which COIs are drawn. Options include a uniform distribution (`"uniform"`), a Poisson distribution (`"poisson"`), or a negative binomial distribution (`"nb"`)
`COI_max`	the maximum allowed COI. Any COIs that are initially drawn larger than this value are set down to this value
`COI_manual`	option to override the MCMC and set the COI of one or more samples manually, in which case they are not updated. Vector of length `n` specifing the integer valued COI of each sample, with -1 indicating that a sample should be estimated
`COI_mean`	the mean of the distribution from which COIs are drawn. Only applies under the Poisson and negative binomial models (under the uniform model the mean is `(COI_max+1)/2` by definition)
`COI_dispersion`	Only used under the negative binomial model. Defines how much larger the variance is than the mean. Must be > 1
`e1`	the probability of a true homozygote being incorrectly called as a heterozygote
`e2`	the probability of a true heterozygote being incorrectly called as a homozygote
`prop_missing`	the proportion of the data that is missing. Note that data are masked out at random, meaning in some rare cases (and when the proportion of missing data is large) an entire sample or locus can end up being masked out, which will throw an error when loaded into a project

Details

TODO

Examples

```
# TODO
```

`sim_data_safe` *Simulate genetic data subject to constraints*

Description

TODO - text

Usage

```
sim_data_safe(..., data_format = "biallelic", no_invariant_loci = TRUE,
  no_missing_samples = TRUE, no_missing_loci = TRUE,
  max_attempts = 1000)
```

Arguments

`...`	TODO
`data_format`	TODO
`no_invariant_loci`	TODO
`no_missing_samples`	TODO
`no_missing_loci`	TODO
`max_attempts`	TODO

Details

TODO

Examples

```
# TODO
```

`summary.malecot_project` *Print summary for class malecot_project*

Description

Overload summary function for class malecot_project

Usage

```
## S3 method for class 'malecot_project'
summary(object, ...)
```

Arguments

`object`	object of class `malecot_project`
`...`	other arguments (ignored)

Index