Package 'coala'

July 22, 2025

Version 0.7.2

License MIT + file LICENSE

Title A Framework for Coalescent Simulation

Author Paul Staab [aut],
Dirk Metzler [aut, ths, cre],
Jorge E. Amaya Romero [ctb]

BugReports https://github.com/statgenlmu/coala/issues

URL https://github.com/statgenlmu/coala

Description Coalescent simulators can rapidly simulate biological sequences evolving according to a given model of evolution.
You can use this package to specify such models, to conduct the simulations and to calculate additional statistics from the results (Staab, Metzler, 2016 <doi:10.1093/bioinformatics/btw098>).
It relies on existing simulators for doing the simulation, and currently supports the programs 'ms', 'msms' and 'scrm'. It also supports finite-sites mutation models by combining the simulators with the program 'seq-gen'.
Coala provides functions for calculating certain summary statistics, which can also be applied to actual biological data.
One possibility to import data is through the 'PopGenome' package (<https://github.com/pievos101/PopGenome>).

Depends R (>= 3.1.0)

Imports assertthat (>= 0.1), digest, methods, parallel, R6 (>= 2.0.1), Rcpp (>= 0.11.0), rehh (>= 3.0.0), scrm (>= 1.6.0-2), stats, utils

Suggests abc (>= 2.0), knitr, PopGenome (>= 2.1.0), phyclust (>= 0.1-16), rmarkdown, testthat (>= 0.11.0)

LinkingTo Rcpp, RcppArmadillo (>= 0.3.810.0)

VignetteBuilder knitr

Collate 'RcppExports.R' 'cache.R' 'coala.R' 'model.R' 'feature.R' 'feature_growth.R' 'feature_ignore_singletons.R' 'feature_migration.R' 'feature_mutation.R' 'feature_outgroup.R' 'feature_pop_merge.R' 'feature_recombination.R'

'feature_sample.R' 'feature_selection.R'
'feature_size_change.R' 'feature_sumstats.R'
'feature_unphased.R' 'import_popgenome.R' 'interface_abc.R'
'locus.R' 'model_build.R' 'model_check.R' 'model_examples.R'
'model_getters.R' 'model_print.R' 'model_scale.R'
'model_simulate.R' 'onLoad.R' 'parameter.R' 'parameter_prior.R'
'parameter_variation.R' 'parameter_zero_inflation.R'
'search_executable.R' 'segsites.R' 'simulation_tasks.R'
'simulator_class.R' 'simulator_ms.R' 'simulator_msms.R'
'simulator_scrm.R' 'simulator_seqgen.R' 'sumstat.R'
'sumstat_dna.R' 'sumstat_file.R' 'sumstat_four_gamete.R'
'sumstat_ihh.R' 'sumstat_jsfs.R' 'sumstat_mcmf.R'
'sumstat_nucleotide_div.R' 'sumstat_omega.R'
'sumstat_seg_sites.R' 'sumstat_sfs.R' 'sumstat_tajimas_d.R'
'sumstat_trees.R' 'tools.R'

Encoding UTF-8

RoxygenNote 7.3.1

NeedsCompilation yes

Maintainer Dirk Metzler <metzler@bio.lmu.de>

Repository CRAN

Date/Publication 2024-03-04 16:20:17 UTC

Contents

`coala-package` *A Framework for Coalescent Simulation in R*

Description

This package allows to specify and simulate coalescent models from within R. The `introduction` vignette is a good place to start.

Author(s)

Maintainer: Dirk Metzler <metzler@bio.lmu.de> [thesis advisor]

Authors:

- Paul Staab <develop@paulstaab.de>

Other contributors:

- Jorge E. Amaya Romero [contributor]

See Also

Useful links:

- https://github.com/statgenlmu/coala
- Report bugs at https://github.com/statgenlmu/coala/issues

as.segsites *Convert genetic data to coala's internal format*

Description

This function can be used to convert the genomic data formats used in other packages to calculate coala's segregating sites object. This is useful for calculating the summary statistics of biological data using the `calc_sumstats_from_data` function.

Usage

```
as.segsites(data, ...)
```

Arguments

`data`	The data object that is converted.
`...`	Additional arguments specific for the used package.

Details

Currently, only the package **PopGenome** is supported, see `as.segsites.GENOME` for details.

Value

A list of segregating sites objects.

See Also

Further instructions are provided in the `coala-data-import` vignette

For information about segsites: `segsites`

as.segsites.GENOME *Convert PopGenome Data into Coala's Format*

Description

Using this function, you can convert genetic data imported with the package **PopGenome** into coala's segsites format. See `as.segsites` for general information on converting genetic data for coala.

Usage

```
## S3 method for class 'GENOME'
as.segsites(data, only_synonymous = FALSE, ...)
```

Arguments

`data`	The `GENOME` data from **PopGenome**.
`only_synonymous`	Only import synonymous SNPs if set to `TRUE`. This requires that **PopGenome** knows where coding regions are., e.g. by using gff files.
`...`	Ignored.

Details

This function imports all loci from the `GENOME` object that have at least one valid site (`data@n.valid.sites`). The number of valid sites is used as length of a locus.

See Also

An example and additional instructions are provided in the `coala-data-import` vignette

---

`calc_sumstats_from_data`

Calculate summary statistics for biological data

Description

This function calculates a model's summary statistic from biological data. The data needs to be provided as a list of segregating sites objects. These objects can be create using the `create_segsites` function.

Usage

```
calc_sumstats_from_data(
  model,
  segsites_list = NULL,
  tree_list = NULL,
  trios = NULL,
  ...
)
```

Arguments

`model`	The coala model. The summary statistics present in this model will be calculated. The model should fit to the data, in particular regarding the number of loci and haploids.
`segsites_list`	Either a list of `segsites` objects, or an object that can be converted using `as.segsites`. It is possible to specify additional argument for the conversion using the `...` argument.
`tree_list`	Not yet implemented.

trios	If your model is using locus trios, then you can create these by combining individual loci. This is a list that defines which loci are combined to a trio. Each entry should consist of either one or three numbers. For one number, the locus used for calculating the summary statistics is locus in the provided data that corresponds to the number. If three numbers are provided, the locus for calculation is created by combining the corresponding three loci from the given data.
...	Additional arguments that will be passed to `as.segsites`.

Examples

```
segsites <- create_segsites(matrix(c(1, 0, 0,
                                     1, 1, 0,
                                     0, 0, 1), 3, 3, TRUE),
                            c(.1, .3, .5))
model <- coal_model(3, 1) +
  sumstat_sfs() +
  sumstat_nucleotide_div() +
  sumstat_mcmf()
sumstats <- calc_sumstats_from_data(model, list(segsites))
print(sumstats)
```

---

check_model *Check which simulator can simulate a model*

Description

This function checks which of the available simulators can simulate a given model. It also states the problems for the ones that are incompatible with the model.

Usage

```
check_model(model)
```

Arguments

model	The model which is checked

See Also

Do view the priority of the simulators: `list_simulators`

Examples

```
model <- coal_model(10, 1) +
  feat_mutation(5, fixed = TRUE)
check_model(model)
```

coal_model *Create a Coalescent Model*

Description

This creates a basic coalescent model to which more features, loci, parameters and summary statistics can be added later. Data under the model can be simulated using the `simulate` function.

Usage

```
coal_model(sample_size, loci_number = 0, loci_length = 1000, ploidy = 1)
```

Arguments

`sample_size`	Defines the number of populations and the number of individual sampled from each population. Given as an integer vector where each entry gives the number of individuals sampled from the corresponding population.
`loci_number`	You can optionally add a number of loci with equal length to the model. This gives to number of loci to add.
`loci_length`	This gives the length of the loci to add.
`ploidy`	The number of chromosomes that will be simulated per individual.

Value

The basic coalescent model which can be extended with features, parameters, loci and summary statistics.

See Also

The 'coala-intro' vignette for a general description on how to extend models.

For checking which simulators can be used for this model: `check_model`

For adding mutation or for a list of other features: `feat_mutation`

For adding loci: `locus_single`, `locus_averaged`, `locus_trio`

For a generating DNA sequences or for a list of summary statistics: `sumstat_dna`

Examples

```
# A model with one population and 20 unlinked loci:
model <- coal_model(10, 20) +
  feat_mutation(5) +
  sumstat_tajimas_d()
check_model(model)
simulate(model)

# A model with two populations:
model <- coal_model(c(13, 18), 5) +
```

```
  feat_migration(.5, symmetric = TRUE) +
  sumstat_trees()
check_model(model)
simulate(model)

# A model with 10 populations:
model <- coal_model(rep(2, 10), 5) +
  feat_migration(.5, symmetric = TRUE) +
  sumstat_trees()
check_model(model)
simulate(model)

# A model with recombination:
model <- coal_model(20, 1, 1000) +
  feat_recombination(10) +
  feat_mutation(5) +
  sumstat_four_gamete()
check_model(model)
simulate(model)
```

`create_abc_param` *Convert Simulation Results to abc's Parameter Format*

Description

This function creates an object compatible with the `param` argument of the `abc` function from coala's simulation results.

Usage

```
create_abc_param(sim_results, model)
```

Arguments

`sim_results`	The simulation results as returned from `simulate`.
`model`	The model used for the simulations.

Value

A data.frame that can be used as `param` argument of `abc`.

See Also

For generating abc's summary statistics format: `create_abc_sumstat`

Examples

```
model <- coal_model(10, 1) +
  feat_mutation(par_prior("theta", rnorm(1, 5, .5))) +
  sumstat_sfs()
sim_results <- simulate(model, nsim = 2)
abc_param <- create_abc_param(sim_results, model)
print(abc_param)
```

`create_abc_sumstat` *Convert Simulation Results to abc's Summary Statistic Format*

Description

This function creates an object compatible with the `sumstat` argument of the `abc` function from coala's simulation results. It converts all summary statistics that are in the simulation results and expects that each of them is a numeric vector. Use transformation functions to convert none vector-valued statistics (e.g. `sumstat_jsfs`, `sumstat_omega` or `sumstat_trees`) into a vector.

Usage

```
create_abc_sumstat(sim_results, model)
```

Arguments

`sim_results`	The simulation results as returned from `simulate`.
`model`	The model used for the simulations.

Value

A data.frame that can be used as `sumstat` argument of `abc`.

See Also

For generating abc's parameter format: `create_abc_param`

Examples

```
# Using the SFS:
model <- coal_model(10, 1) +
  feat_mutation(par_prior("theta", rnorm(1, 5, .5))) +
  sumstat_sfs()
sim_results <- simulate(model, nsim = 2)
abc_sumstat <- create_abc_sumstat(sim_results, model)
print(abc_sumstat)

# Using the JSFS and converting it into a vector:
model <- coal_model(c(10, 10), 1) +
  feat_mutation(par_prior("theta", rnorm(1, 5, .5))) +
```

```
  feat_migration(par_prior("m", rnorm(1, .5, .1)), symmetri = TRUE) +
  sumstat_jsfs(transformation = function(jsfs) {
    c(sum(jsfs[1, ]), sum(jsfs[, 1]), sum(jsfs[-1, -1]))
  })
sim_results <- simulate(model, nsim = 2)
abc_sumstat <- create_abc_sumstat(sim_results, model)
print(abc_sumstat)
```

`create_segsites` *Segregating Sites*

Description

These functions create and modify segregating sites objects, which are one of the basic intermediary statistics that is calculated in coala. Segregating sites consist primarily of a SNP matrix that contains all SNPs for one locus, with some additional information attached. The parts of the S3 class are detailed below.

Usage

```
create_segsites(snps, positions, trio_locus = numeric(0), check = TRUE)

get_snps(segsites)

get_positions(segsites)

set_positions(segsites, positions)

get_trio_locus(segsites)

set_trio_locus(segsites, trio_locus)

is_segsites(segsites)

create_locus_trio(left, middle, right)
```

Arguments

`snps`	The SNP Matrix (see Details).
`positions`	A numeric vector indicating the relative positions of each SNP on the locus (see Details).
`trio_locus`	If the locus consists of a locus trio (see Details).
`check`	Whether non-segregating sites are removed from the segregating sites (`TRUE`) or not (`FALSE`).
`segsites`	The segregating sites object
`left`	The segregating sites from the left locus
`middle`	The segregating sites from the middle locus
`right`	The segregating sites from the right locus

Details

A segregating sites object contains all SNPs for one genetic locus. Each object consists of three parts: A SNP matrix, a vector of SNP positions and a vector that states which transcript a SNP belong to, if the locus consists of multiple transscripts ('locus trio').

- In the **SNP** matrix, each row represents a haplotype and each column represents a SNP. An entry is either `1` if the haplotype carries the derived allele for the SNP, or `0` if it carries the ancestral one.
- In the **positions** vector, each entry gives the relative position of SNP in the corresponding column of the SNP matrix.
- The **trio_locus** vector contains the trio locus each SNP belongs to. Entry of `-1`,`0`, `1` represent the left, middle, and right locus, respectively. For normal loci, this just consists of `0`'s

Functions

- `create_segsites()`: Creates segregating sites
- `get_snps()`: Returns the SNP matrix from a segregating sites object.
- `get_positions()`: Returns the SNP's positions from a segregating sites object.
- `set_positions()`: Sets the SNP's positions in a segregating sites object.
- `get_trio_locus()`: Returns the trio locus positions from a segregating sites object.
- `set_trio_locus()`: Sets the trio locus in a segregating sites object.
- `is_segsites()`: Checks whether an object is a segsites object.
- `create_locus_trio()`: Combines three segregating sites to a locus trio

Author(s)

Paul Staab

See Also

For converting biological data to segsites: `as.segsites`

Examples

```
snps <- matrix(c(1, 0, 0,
                 1, 1, 0,
                 0, 0, 1), 3, 3, TRUE)
pos <- c(.1, .3, .5)
segsites <- create_segsites(snps, pos)
print(segsites)
get_snps(segsites)
get_positions(segsites)

# When subsetting individuals, sites that are not
# segregating in these are automatically removed:
segsites[1:2, 1:3]
```

feat_growth *Feature: Exponential population size growth/decline*

Description

This feature changes the growth factor of a population at given point in time. This factor applies to the time interval further into the past from this point.

Usage

```
feat_growth(rate, population = "all", time = "0", locus_group = "all")
```

Arguments

rate	The growth rate. Can be a numeric or a parameter. See Details for an explanation how the rate affects the population size.
population	The population which growths/declines. Can be "all" for all populations, or the number of one population.
time	The time at which the growth rate is changed. Can also be a parameter.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Details

The population size changes by a factor $exp(-\alpha * t)$, where α is the growth parameter and t is the time since the growth has started. For positive alpha, the population will decline backwards in time or grow forwards in time. For a negative value of α it will decline (forward in time).

Value

The feature, which can be added to a model created with coal_model using +.

See Also

For instantaneous population size changes: feat_size_change

For creating a model: coal_model

Other features: feat_ignore_singletons(), feat_migration(), feat_mutation(), feat_outgroup(), feat_pop_merge(), feat_recombination(), feat_selection(), feat_size_change(), feat_unphased()

Examples

```
# Simulate a haploid population that has been expanding for
# the last 2*Ne generations
model <- coal_model(10, 1) +
  feat_growth(5, time = 0) +
  feat_growth(0, time = 1) +
  feat_mutation(10) +
  sumstat_sfs()
simulate(model)
```

---

feat_ignore_singletons

Feature: Ignore Singletons

Description

Mutations that are observed in just one haplotype ('singletons') are often regarded as likely candidates for sequencing errors. Sometimes, it can be advantageous to exclude them from an analysis. This feature removes all singletons from the simulated data before the summary statistics are calculated.

Usage

```
feat_ignore_singletons(locus_group = "all")
```

Arguments

`locus_group`	The loci for which this features is used. Can either be `"all"` (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in `locus_` commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has `locus_single(10) + locus_averaged(10, 11) + locus_single(12)` and this argument is `c(2, 3)`, than the feature is used for all but the first locus (that is locus 2 - 12).

Details

This function assumes that a singleton is a mutation for which the derived allele is observed exactly once in all sequences, regardless of the population structure.

Value

The feature, which can be added to a model created with `coal_model` using `+`.

See Also

For creating a model: `coal_model`

Other features: `feat_growth()`, `feat_migration()`, `feat_mutation()`, `feat_outgroup()`, `feat_pop_merge()`, `feat_recombination()`, `feat_selection()`, `feat_size_change()`, `feat_unphased()`

Examples

```
model <- coal_model(2, 1) +
  feat_mutation(10) +
  feat_ignore_singletons() +
  sumstat_sfs("n_mut", transformation = sum)
# In this model, all mutations are singletons. Therefore,
# the number of mutations is 0 when removing singletons:
simulate(model)$n_mut
```

`feat_migration` *Feature: Migration/Gene Flow*

Description

This feature changes the migration rates at a given time point. Per default, no migration between the population occurs, which corresponds to a `rate` of `0`. Set it to a value greater than zero to enable migration from one population to another.

Usage

```
feat_migration(
  rate,
  pop_from = NULL,
  pop_to = NULL,
  symmetric = FALSE,
  time = "0",
  locus_group = "all"
)
```

Arguments

`rate`	The migration rate. Can be a numeric or a `parameter`. The rate is specified as $4 * N0 * m$, where m is the fraction of `pop_to` that is replaced by migrants from `pop_from` each generation (in forward time).
`pop_from`	The population from which the individuals leave.
`pop_to`	The population to which the individuals move.
`symmetric`	Use the rate for all pairs of populations.
`time`	The time point at which the migration with the migration rate is set. The rate applies to the time past warts of the time point, until it is changed again.
`locus_group`	The loci for which this features is used. Can either be `"all"` (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in `locus_` commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has `locus_single(10) + locus_averaged(10, 11) + locus_single(12)` and this argument is `c(2, 3)`, than the feature is used for all but the first locus (that is locus 2 - 12).

Details

When looking forward in time, a fraction of pop_to that is replaced by migrants from pop_from each generation (see rate). When looking backwards in time, ancestral lines in pop_to move to pop_from with the given rate.

Value

The feature, which can be added to a model created with coal_model using +.

See Also

For creating a model: coal_model

Other features: feat_growth(), feat_ignore_singletons(), feat_mutation(), feat_outgroup(), feat_pop_merge(), feat_recombination(), feat_selection(), feat_size_change(), feat_unphased()

Examples

```
# Asymmetric migration between two populations:
model <- coal_model(c(5, 5), 10) +
  feat_migration(0.5, 1, 2) +
  feat_migration(1.0, 2, 1) +
  feat_mutation(5) +
  sumstat_sfs()
simulate(model)

# Three populations that exchange migrations with equal
# rates at times more than 0.5 time units in the past:
model <- coal_model(c(3, 4, 5), 2) +
  feat_migration(1.2, symmetric = TRUE, time = 0.5) +
  feat_mutation(5) +
  sumstat_sfs()
simulate(model)
```

---

feat_mutation *Feature: Mutation*

Description

This feature adds mutations to a model. Mutations occur in the genomes of the individuals with a given rate. The rate is per locus for unlinked loci and per trio for linked locus trios. By default, the same mutation rate is used for all loci, but it is possible to change this with par_variation and par_zero_inflation.

Usage

```
feat_mutation(
  rate,
  model = "IFS",
  base_frequencies = NA,
  tstv_ratio = NA,
  gtr_rates = NA,
  fixed_number = FALSE,
  locus_group = "all"
)
```

Arguments

rate	The mutation rate. Can be a numeric or a parameter. The rate is specified as $4 * N0 * mu$, where mu is the mutation rate per locus.
model	The mutation model you want to use. Can be either 'IFS' (default), 'HKY' or 'GTR'. Refer to the mutation model section for detailed information.
base_frequencies	The equilibrium frequencies of the four bases used in the 'HKY' mutation model. Must be a numeric vector of length four, with the values for A, C, G and T, in that order.
tstv_ratio	The ratio of transitions to transversions used in the 'HKY' muation model.
gtr_rates	The rates for the six amino acid substitutions used in the 'GTR' model. Must be a numeric vector of length six. Order: A<->C, A<->G, A<->T, C<->G, C<->T, G<->T.
fixed_number	If set to TRUE, the number of mutations on each locus will always be exactly equal to the rate, rather than happening with a rate along the ancestral tree.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Value

The feature, which can be added to a model using +.

The feature, which can be added to a model created with coal_model using +.

Mutation Models

The infinite sites mutation (**IFS**) model is a frequently used simplification in population genetics. It assumes that each locus consists of infinitely many sites at which mutations can occur, and each mutation hits a new site. Consequently, there are no back-mutations with this model. It does not generate DNA sequences, but rather only 0/1 coded data, were 0 denotes the ancestral state of the site, and 1 the derived state created by a mutation.

The other mutation models are finite site models that generate more realistic sequences.

The Hasegawa, Kishino and Yano (**HKY**) model (Hasegawa et al., 1985) allows for a different rate of transitions and transversions (tstv_ratio) and unequal frequencies of the four nucleotides (base_frequencies).

The general reversible process (**GTR**) model (e.g. Yang, 1994) is more general than the HKY model and allows to define the rates for each type of substitution. The rates are assumed to be symmetric (e.g., the rate for T to G is equal to the one for G to T).

See Also

For using rates that variate between the loci in a model: par_variation, par_zero_inflation

For adding recombination: feat_recombination.

For creating a model: coal_model

Other features: feat_growth(), feat_ignore_singletons(), feat_migration(), feat_outgroup(), feat_pop_merge(), feat_recombination(), feat_selection(), feat_size_change(), feat_unphased()

Examples

```
# A model with a constant mutation rate of 5:
model <- coal_model(5, 1) + feat_mutation(5) + sumstat_seg_sites()
simulate(model)

# A model with a mutation of 5.0 for the first 10 loci, and 7.5 for the
# second 10 loci
model <- coal_model(4) +
  locus_averaged(10, 100) +
  locus_averaged(10, 100) +
  feat_mutation(5.0, locus_group = 1) +
  feat_mutation(7.5, locus_group = 2) +
  sumstat_seg_sites()
simulate(model)

# A model with 7 mutations per locus:
model <- coal_model(5, 1) +
  feat_mutation(7, fixed = TRUE) +
  sumstat_seg_sites()
simulate(model)

# A model using the HKY model:
model <- coal_model(c(10, 1), 2) +
 feat_mutation(7.5, model = "HKY", tstv_ratio = 2,
               base_frequencies = c(.25, .25, .25, .25)) +
  feat_outgroup(2) +
  feat_pop_merge(1.0, 2, 1) +
  sumstat_seg_sites()
## Not run: simulate(model)

# A model using the GTR model:
model <- coal_model(c(10, 1), 1, 25) +
  feat_mutation(7.5, model = "GTR",
```

```
                gtr_rates = c(1, 1, 1, 1, 1, 1) / 6) +
  feat_outgroup(2) +
  feat_pop_merge(1.0, 2, 1) +
  sumstat_dna()
## Not run: simulate(model)$dna
```

`feat_outgroup` *Feature: Outgroup*

Description

This feature declares an existing population as outgroup. Outgroups are used to determine the ancestral allele in finite sites simulations and are required there. All individuals of the outgroup are ignored when calculating summary statistics. If the outgroup consists of multiple individuals, all positions where the individuals have different alleles are ignored.

Usage

```
feat_outgroup(population, locus_group = "all")
```

Arguments

`population`	The population that is marked as outgroup. If finite-sites mutation models are used, the last population must be specified as outgroup.
`locus_group`	The loci for which this features is used. Can either be `"all"` (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in `locus_` commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has `locus_single(10) + locus_averaged(10, 11) + locus_single(12)` and this argument is `c(2, 3)`, than the feature is used for all but the first locus (that is locus 2 - 12).

Value

The feature, which can be added to a model created with `coal_model` using `+`.

See Also

For creating a model: `coal_model`

Other features: `feat_growth()`, `feat_ignore_singletons()`, `feat_migration()`, `feat_mutation()`, `feat_pop_merge()`, `feat_recombination()`, `feat_selection()`, `feat_size_change()`, `feat_unphased()`

Examples

```
# A simple finite sites model
model <- coal_model(c(4, 6, 1), 2, 10) +
   feat_outgroup(3) +
   feat_pop_merge(0.5, 2, 1) +
   feat_pop_merge(2, 3, 1) +
   feat_mutation(5, model = "GTR", gtr_rates = 1:6)
```

---

feat_pop_merge *Feature: Population Merge*

Description

Backwards in time, this feature merges one population into another. Forwards in time, this corresponds to a speciation event.

Usage

```
feat_pop_merge(time, pop_source, pop_target, locus_group = "all")
```

Arguments

time	The time at which the merge occurs.
pop_source	The population from which all lines are moved. This is the newly created population in the speciation event.
pop_target	The population to which the lines are moved. This is the population in which the speciation event occurs.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Details

In addition to the merge, the growth rate of and all migration rates from the source population will be set to 0 at the time of the merge to mimic a speciation event forwards in time. Technically, pop_source is still present in the model, but not used unless migration to the population is manually enabled.

Value

The feature, which can be added to a model created with coal_model using +.

See Also

For creating a model: coal_model

Other features: feat_growth(), feat_ignore_singletons(), feat_migration(), feat_mutation(), feat_outgroup(), feat_recombination(), feat_selection(), feat_size_change(), feat_unphased()

Examples

```
# Two population which merge after 0.5 time units:
model <- coal_model(c(25,25), 1) +
  feat_pop_merge(0.5, 2, 1) +
  feat_mutation(5) +
  sumstat_tajimas_d()
simulate(model)

# An standard isolation-with-migration model:
model_iwm <- model +
  feat_migration(.75, symmetric = TRUE)
simulate(model_iwm)
```

---

feat_recombination *Feature: Recombination*

Description

This feature adds intra-locus recombination to a model. The rate is per locus for unlinked loci and per trio for linked locus trios. By default, the same recombination rate is used for all loci, but it is possible to change this with par_variation and par_zero_inflation. Coala assumes that recombination events can occur between all neighboring bases.

Usage

```
feat_recombination(rate, locus_group = "all")
```

Arguments

rate	The recombination rate. Can be a numeric or a parameter. The rate is equal to $4 * N0 * r$, where r is the probability that a recombination event within the locus occurs in one generation.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Value

The feature, which can be added to a model using +.

The feature, which can be added to a model created with coal_model using +.

See Also

For creating a model: coal_model

For adding recombination: feat_mutation.

Other features: feat_growth(), feat_ignore_singletons(), feat_migration(), feat_mutation(), feat_outgroup(), feat_pop_merge(), feat_selection(), feat_size_change(), feat_unphased()

Examples

```
# Simulate a 1.5kb sequence for 10 individuals with recombination:
model <- coal_model(10, 2, 1500) +
  feat_recombination(1.5) +
  feat_mutation(5) +
  sumstat_sfs()
simulate(model)
```

---

feat_selection *Feature: Selection*

Description

This feature adds selection to a model. Only one site per locus can be under selection. Using this feature requires that msms is installed, see activate_msms.

Usage

```
feat_selection(
  strength_AA = 0,
  strength_Aa = 0,
  strength_aa = 0,
  strength_A = NULL,
  population = "all",
  time,
  start = TRUE,
  start_frequency = 5e-04,
  Ne = 10000,
  position = 0.5,
  force_keep = TRUE,
  locus_group = "all"
)
```

Arguments

strength_AA	The selection strength for the selected homozygote. The parameter is valid for the chosen population and the time further past-wards from either time 0 on if start = TRUE, or from time onwards. The same applies for strength_Aa, strength_aa and strength_A.
strength_Aa	The selection strength for the heterozygote.
strength_aa	The selection strength for the recessive homozygote.
strength_A	This sets the strength for the selected allele in a haploid model or a diploid model with additive selection. strength_AA, strength_Aa, strength_aa are ignored when this is argument is given.
population	The population in which the allele is selected. Can either be all for all population, or the number of a population.
time	The time at which the selection starts if start == TRUE (looking forwards in time), or the time at which the selection strength changes if start == FALSE. The new strength applies for the time period further into the past in this case.
start	Whether selection should start at this time point. At this time, the selected allele is introduced in the population with an initial starting frequency. This must be set to TRUE for exactly one selection feature in the model. The values of start_frequency, Ne, position and force_keep are used for the model. You can add additional selection feature to the model to set the selection strength for more demes or change it at different time points, but these need to have start = FALSE.
start_frequency	The start frequency at which the selected allele is introduced at time. If the model has multiple population, this can either be a numeric vector that contains the initial frequency for each population or a single number. In the latter case, the value is used for all population specified with populations, and 0 is used for all other populations.
Ne	The effective population size that is used for the forward simulations.
position	The position of the selected site, relative to the simulated sequence. Values between 0 and 1 are within the simulated area, while smaller values are to the left of it and larger ones to the right.
force_keep	Whether to restart simulations in which the selected goes to extinction or not.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Value

The feature, which can be added to a model created with coal_model using +.

See Also

For using rates that variate between the loci in a model: par_variation, par_zero_inflation

For summary statistics that are sensitive for selection: sumstat_tajimas_d, sumstat_ihh, sumstat_omega, sumstat_mcmf

For creating a model: coal_model

Other features: feat_growth(), feat_ignore_singletons(), feat_migration(), feat_mutation(), feat_outgroup(), feat_pop_merge(), feat_recombination(), feat_size_change(), feat_unphased()

Examples

```
# Positive additive selection in population 2:
model <- coal_model(c(10, 13), 1, 10000) +
  feat_pop_merge(.5, 2, 1) +
  feat_selection(strength_A = 1000,
                 population = 2,
                 time = par_named("tau")) +
  feat_mutation(100) +
  feat_recombination(10) +
  sumstat_tajimas_d(population = 2)
## Not run: simulate(model, pars = c(tau = 0.03))
```

---

feat_size_change *Feature: Instantaneous Size Change*

Description

This feature changes the effective population size of one population. The change is performed at a given time point and applies to the time interval further on into the past from this point. The population size is set to a fraction of N0.

Usage

```
feat_size_change(new_size, population = 1, time = "0", locus_group = "all")
```

Arguments

new_size	A parameter giving the new size of the population, as a fraction of N0.
population	The number of the population whichs size changes. Can also be set to "all". Then the size changes applies to all populations.
time	The time at which the population's size is changed.
locus_group	The loci for which this features is used. Can either be "all" (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in locus_ commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has locus_single(10) + locus_averaged(10, 11) + locus_single(12) and this argument is c(2, 3), than the feature is used for all but the first locus (that is locus 2 - 12).

Value

The feature, which can be added to a model using +.

The feature, which can be added to a model created with coal_model using +.

See Also

For continuous size changes over time: feat_growth.

For creating a model: coal_model

Other features: feat_growth(), feat_ignore_singletons(), feat_migration(), feat_mutation(), feat_outgroup(), feat_pop_merge(), feat_recombination(), feat_selection(), feat_unphased()

Examples

```
# A model with one smaller population:
model <- coal_model(c(20, 5), 3) +
  feat_size_change(.1, population = 2) +
  feat_mutation(1.0) +
  feat_migration(0.5, 2, 1) +
  sumstat_sfs()
simulate(model)

# A model of one population that experienced a bottleneck:
model <- coal_model(10, 1) +
  feat_size_change(0.1, time = 0.3) +
  feat_size_change(1.0, time = 0.5) +
  feat_mutation(20) +
  sumstat_sfs()
simulate(model)
```

---

feat_unphased *Feature: Unphased Sequences*

Description

This simulates unphased data by randomly mixing the sites within one individual. Each position is randomly taken from a phased chromosome..

Usage

```
feat_unphased(samples_per_ind, locus_group = "all")
```

Arguments

samples_per_ind
: The number of pseudo-chromosomes that are created from the phased chromosomes for each individual.

`locus_group` The loci for which this features is used. Can either be `"all"` (default), in which case the feature is used for simulating all loci, or a numeric vector. In the latter case, the feature is only used for the loci added in `locus_` commands with the corresponding index starting from 1 in order in which the commands where added to the model. For example, if a model has `locus_single(10) + locus_averaged(10, 11) + locus_single(12)` and this argument is `c(2, 3)`, than the feature is used for all but the first locus (that is locus 2 - 12).

Details

For each individual, `ploidy` chromosomes are simulated, and `samples_per_ind` pseudo-chromosomes are created of these.

Value

The feature, which can be added to a model using `+`.

The feature, which can be added to a model created with `coal_model` using `+`.

See Also

For creating a model: `coal_model`

Other features: `feat_growth()`, `feat_ignore_singletons()`, `feat_migration()`, `feat_mutation()`, `feat_outgroup()`, `feat_pop_merge()`, `feat_recombination()`, `feat_selection()`, `feat_size_change()`

Examples

```
# Simulate unphased data in a diploid population
model <- coal_model(10, 1, ploidy = 2) +
  feat_mutation(10) +
  feat_unphased(2) +
  sumstat_seg_sites()
simulate(model)

# The same as before, but return only one chromosome for
# each individual:
model <- coal_model(10, 1, ploidy = 2) +
  feat_mutation(10) +
  feat_unphased(1) +
  sumstat_seg_sites()
simulate(model)
```

`list_simulators` *Returns the available simulators*

Description

This functions returns the usable simulators

Usage

```
list_simulators()
```

Examples

```
list_simulators()
```

`locus` *Loci*

Description

This functions adds one or more loci to a model. A locus is a continuous stretch of DNA of a given length. All loci are simulated independently of each other, and are genetically unlinked. A model can contain a large number of different loci created with `locus_single`. This will, however, slow down the simulation. For performance reasons, it is better to add the same number of loci with averaged length using `locus_averaged` if this simplification is justifiable. Both can also be combined in a single model. In the results, the summary statistics for the loci are returned in order in which they are added to the model.

Usage

```
locus_single(length)

locus_averaged(number, length)
```

Arguments

`length`	The length of the locus in base pairs.
`number`	The number of loci to add.

Functions

- `locus_single()`: Adds a single locus.
- `locus_averaged()`: Adds multiple loci with equal length.

See Also

For adding three loci which are linked to each other: `locus_trio`

Examples

```
# A model with one locus of length 1005 bp:
coal_model(10) + locus_single(1005)
# This is equivalent to:
coal_model(10, 1, 1005)

# A model can contain multiple loci:
coal_model(5) + locus_single(100) + locus_single(200) + locus_single(300)
# Or more efficient with averaged length:
coal_model(5) + locus_averaged(3, 200)
# Or equivalently:
coal_model(5, 3, 200)

# Single and averaged loci can also be combined arbitrarily:
coal_model(15) + locus_averaged(10, 150) + locus_single(250)
coal_model(15, 10, 150) + locus_single(250) + locus_averaged(10, 350)
```

---

locus_trio *Locus Trios*

Description

This functions adds a group of three loci to the model that are genetically linked to each other. They are still unlinked to all other loci or locus trios in the model. Simulating linked loci that are far apart from each other can be very slow. Please mind that mutation and recombination rates for locus trios are rates per trio and not per locus, i.e. they account for mutations that occur on the tree loci and the sequences in-between them together.

Usage

```
locus_trio(
  locus_length = c(left = 1000, middle = 1000, right = 1000),
  distance = c(left_middle = 500, middle_right = 500),
  number = 1
)
```

Arguments

`locus_length`	An integer vector of length 3, giving the length of each of the three loci (left, middle and right).
`distance`	A vector of two, giving the distance between left and middle, and middle an right locus, in base pairs.
`number`	The number of loci to add.

See Also

For adding unlinked loci: `locus`

Examples

```
# A model with one locus trio
coal_model(25) +
  locus_trio(locus_length=c(1250, 1017, 980), distance=c(257, 814))

# Ten identical locus trios:
coal_model(25) +
  locus_trio(locus_length=c(1250, 1017, 980), distance=c(257, 814), number = 10)

# Two different ones:
coal_model(25) +
  locus_trio(locus_length=c(1000, 500, 900), distance=c(200, 400)) +
  locus_trio(locus_length=c(700, 500, 800), distance=c(350, 150))
```

---

parameter *Model Parameters*

Description

These functions add parameters to a model. Parameters can either be used in features, or added directly to a model using the plus operator. The value of parameters can be specified in the simulation command (for `par_named` and `par_range`), sampled from a prior distribution (`par_prior`) or can be derived from other parameters (`par_expr`).

Usage

```
par_expr(expr)

par_const(constant)

par_named(name)

par_range(name, lower, upper)

par_prior(name, prior)
```

Arguments

`expr`	An R expression. This allows to define a parameter using an R expression. It can contain other named parameters (e.g. `2 * a` will create an parameter that is twice the value of an existing parameter `a`). Make sure that the expression always evaluates to a valid parameter value (a single numeric in almost all cases).
`constant`	An R expression. The constant value of the parameter. Different to `expr`, the expression is evaluated immediately and can not depend on other named parameters.
`name`	Character. The name of the parameter. Must be unique in a model.
`lower`	A numeric. The lower boundary of the parameter's range.

upper	A numeric. The upper boundary of the parameter's range.
prior	An expression. Evaluation this expression should give a sample from the prior distribution you want for the parameter. For example using `rnorm(1)` gives a standard normal prior.

Functions

- `par_expr()`: Creates a parameter with value determined by evaluating an expression.
- `par_const()`: Creates an parameter that is equal to a fixed value. Different to `par_expr`, the value is evaluated on parameter creation.
- `par_named()`: Creates an parameter whose value is specified via the `pars` argument in `simulate.coalmodel`.
- `par_range()`: Creates an parameter that can take a range of possible values. Similar to `par_named`, the value of the parameter used in a simulation is set via the `pars` argument. This is primarily intended for creating model parameters for **jaatha**.
- `par_prior()`: Creates a named parameter with a prior distribution. Before each simulation, the expression for the prior is evaluated. The resulting value can be used in `par_expr` under the chosen name.

Author(s)

Paul Staab

See Also

For parameters that variate between the loci in a model: `par_variation`, `par_zero_inflation`

Examples

```
# A parameter (here for the mutation rate) that is always
# equal to '5':
model_base <- coal_model(20, 1) +
  sumstat_nucleotide_div()

model <- model_base +
  feat_mutation(par_const(5))
simulate(model)

# With using a prior:
model <- model_base +
  feat_mutation(par_prior("theta", rnorm(1, 5, .1)))
simulate(model)

# Using a named parater:
model <- model_base +
  feat_mutation(par_named("theta"))
simulate(model, pars = c(theta = 5))

# or similarly a ranged parameter:
model <- model_base +
  feat_mutation(par_range("theta", 1, 10))
```

```
simulate(model, pars = c(theta = 5))

# Expressions can be used to derive parameters from
# other parameters:
model <- model_base +
  par_named("theta_half") +
  feat_mutation(par_expr(theta_half * 2))
simulate(model, pars = c(theta_half = 2.5))

model <- model_base +
  par_named("theta_log") +
  feat_mutation(par_expr(exp(theta_log)))
simulate(model, pars = c(theta_log = log(5)))
```

`par_variation` *Variable Parameters*

Description

This function can be used to let the values of a parameter vary between the different loci. When used, the values for the enclosed parameter will follow a gamma distribution with mean of the parameters original value, and the variance specified as argument `variance`. This requires that the original value is positive. When using this, the simulators are called separately for each locus, which can dramatically increase the time needed to simulate models with many loci.

Usage

```
par_variation(par, variance)
```

Arguments

`par`	A parameter whichs value will be made variable between the loci.
`variance`	The variance of the gamma distribution, which the values used for simulation will follow.

See Also

For parameters that are identical for all loci: `parameter`

Examples

```
model <- coal_model(5, 10) +
  feat_mutation(par_variation(par_const(5), 10)) +
  sumstat_nucleotide_div()
simulate(model)
```

par_zero_inflation *Zero Inflation for Parameters*

Description

This adds a zero inflation to the distribution of a parameter for the different loci. When using this, each locus will be simulated with a parameter value of 0 with probability `prob`, or with parameter's original value in the remaining cases. are called separately for each locus, which can dramatically increase the time needed to simulate models with many loci.

Usage

```
par_zero_inflation(par, prob, random = TRUE)
```

Arguments

`par`	A parameter which will be set to 0 for part of the loci.
`prob`	The probability that the parameters value will be set to `0` for each locus if `random` is `TRUE`. Otherwise, it's the fixed fraction of loci which will have a parameter value of `0`.
`random`	Whether the number of loci which are simulated with a value of `0` should be random (`TRUE`) or a fixed fraction (`FALSE`). See `prob` for details.

See Also

For parameters that are identical for all loci: `parameter`

Examples

```
# Simulate half of the loci with recomination and the other half without it:
model <- coal_model(4, 4) +
  feat_recombination(par_zero_inflation(par_named("rho"), .5, random = FALSE)) +
  sumstat_trees()
simulate(model, pars = c(rho = 1))

# Use a zero inflated gamma distribution:
model <- coal_model(4, 4) +
  feat_recombination(par_zero_inflation(par_variation(1, 10), .3)) +
  sumstat_trees()
simulate(model)
```

scale_model *Function that downscales a coalescent model*

Description

This function reduces the number of loci in all averaged loci by a certain factor. Non-averaged loci as created with `locus_single` are not modified in any way. This function is primarily designed for jaatha, and might be unsuitable for other purposes.

Usage

```
scale_model(model, scaling_factor)
```

Arguments

`model`	The model to downscale
`scaling_factor`	The factor by which the number of loci are reduced. A value of 2 changes to numbers to half their value (rounded), a value of 3 to a third an so on.

Examples

```
model <- coal_model(10, loci_number = 10) + locus_single(100)
model
# Group 1: 10 loci; group 2: 1 locus

model <- scale_model(model, 3)
model
# Group 1: 3 loci; group 2: 1 locus
```

simulate.coalmodel *Simulate Data According to a Demographic Model*

Description

This function simulates a model created with `coal_model`. The model can be extended with features, `parameters` and loci. Read the 'coala-introduction' vignette for detailed instructions on creating and simulating such models.

Usage

```
## S3 method for class 'coalmodel'
simulate(object, nsim = 1, seed, ..., pars = numeric(0), cores = 1)
```

Arguments

object	The coalescent model to be simulated
nsim	currently unused
seed	A random seed that is set before simulation.
...	currently unused
pars	Values for parameters specified with par_named or par_range. Should be a named numeric vector.
cores	The number of cores that the independent repetitions from nsim will be distributed on. Must be 1 on Windows, and should also be 1 when using R in a graphical environment (e.g. Rstudio).

Value

A list of summary statistics.

Examples

```
model <- coal_model(10, 3) +
  feat_mutation(5) +
  sumstat_sfs() +
  sumstat_tajimas_d()
simulate(model, nsim = 2)

model <- coal_model(c(5,10), 20) +
  feat_pop_merge(par_range('tau', 0.01, 5), 2, 1) +
  feat_mutation(par_range('theta', 1, 10)) +
  sumstat_jsfs()
simulate(model, pars=c(tau = 1, theta = 5))
```

---

simulator_ms *Simulator: ms*

Description

This function adds the simulator 'ms' to the list of available simulators. In order to use 'ms', you need to install the CRAN package **phyclust**. By default, 'scrm' will be preferred over 'ms'. Raise the priority of 'ms' to change this behavior.

Usage

```
activate_ms(priority = 300)
```

Arguments

priority	The priority for this simulator. If multiple simulators can simulate a model, the one with the highest priority will be used.

References

Richard R. Hudson. Generating samples under a Wright-Fisher neutral model of genetic variation. Bioinformatics (2002) 18 (2): 337-338 doi:10.1093/bioinformatics/18.2.337

See Also

Other simulators: simulator_msms, simulator_scrm, simulator_seqgen

Examples

```
# To prefer ms to scrm:
## Not run: activate_ms(priority = 500)
```

`simulator_msms` *Simulator: msms*

Description

This adds the simulator 'msms' to the list of available simulators. To add msms, you need to download the jar file and have Java installed on your system.

Usage

```
activate_msms(jar = NULL, java = NULL, priority = 200, download = FALSE)
```

Arguments

jar	The path of the msms jar file.
java	The path of the java executable on your system.
priority	The priority for this simulator. If multiple simulators can simulate a model, the one with the highest priority will be used.
download	If set to TRUE, coala will try to download the msms jar file. In that case, the jar argument is not required.

References

Gregory Ewing and Joachim Hermisson. MSMS: a coalescent simulation program including recombination, demographic structure and selection at a single locus. Bioinformatics (2010) 26 (16): 2064-2065 doi:10.1093/bioinformatics/btq322

See Also

Other simulators: simulator_ms, simulator_scrm, simulator_seqgen

Examples

```
# Download and activate msms (requires Java)
## Not run: activate_msms(download = TRUE)
```

simulator_scrm *Simulator: scrm*

Description

This function adds the simulator 'scrm' to the list of available simulators. It is provided via the CRAN package **scrm** and should be always installed alongside with **coala**. It should be activated automatically, and this function is only needed to change it `priority`.

Usage

```
activate_scrm(priority = 400)
```

Arguments

`priority`	The priority for this simulator. If multiple simulators can simulate a model, the one with the highest priority will be used.

References

Paul R. Staab, Sha Zhu, Dirk Metzler and Gerton Lunter (2015). "scrm: efficiently simulating long sequences using the approximated coalescent with recombination." Bioinformatics, 31(10), pp. 1680-1682. http://dx.doi.org/10.1093/bioinformatics/btu861

See Also

Other simulators: `simulator_ms`, `simulator_msms`, `simulator_seqgen`

Examples

```
# Change scrm's priority
model <- coal_model(10, 1) + feat_mutation(5)
model # scrm is used by default
activate_scrm(250)
model # Now ms is used instead (if installed)
activate_scrm(550)
model # Now scrm is used again
```

simulator_seqgen *Simulator: seq-gen*

Description

This allows you to use seq-gen to simulate finite sites mutation models. When using seq-gen, coala will simulate ancestral tress using the other simulators, and call seq-gen to simulate finite sites mutations using the trees. Seq-gen has a low priority, but will always be used when finite sites mutation models are used.

Usage

```
activate_seqgen(binary = NULL, priority = 100)
```

Arguments

binary	The path of the seqgen binary that will be used for simulations. If none is provided, coala will look for a binary called 'seqgen' or 'seq-gen' using the PATH variable.
priority	The priority for this simulator. If multiple simulators can simulate a model, the one with the highest priority will be used.

Installation

You need to download the program from http://tree.bio.ed.ac.uk/software/seqgen/ and compile the binary prior to invoking this function. On Debian-based systems, you can alternatively install the package 'seg-gen'.

References

Andrew Rambaut and Nicholas C. Grassly. Seq-Gen: an application for the Monte Carlo simulation of DNA sequence evolution along phylogenetic trees. Comput Appl Biosci (1997) 13 (3): 235-238 doi:10.1093/bioinformatics/13.3.235

See Also

Other simulators: simulator_ms, simulator_msms, simulator_scrm

Examples

```
## Not run: activate_seqgen("./bin/seqgen")
```

---

sumstat_dna *Summary Statistic: DNA*

Description

This summary statistic returns the actual DNA sequences from finite sites simulations. It can not be calculated together with other summary statistics or when assuming an infinite sites mutation model. No outgroup is needed for it, and the outgroup sequences will also be returned if present.

Usage

```
sumstat_dna(name = "dna", transformation = identity)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

A list of sequences for each locus. Each entries is a character matrix decoding the sequences. Each row is an individual, and each column is a genetic position.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_nucleotide_div(), sumstat_omega(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
model <- coal_model(5, 1, 10) +
 feat_mutation(5, model = "GTR", gtr_rates = rep(1, 6)) +
 sumstat_dna()
## Not run: simulate(model)$dna
```

---

sumstat_file *Summary Statistic: Files*

Description

This "summary statistic" returns files with the raw results of the simulation. Multiple files are returned in case coala needs multiple calls to simulators to simulate the model. These files do not contain any post processing of the results done by coala, e.g. feat_unphased and feat_ignore_singletons.

Usage

```
sumstat_file(folder)
```

Arguments

folder	The path to a folder. The files will be created there.

Value

A character vector containing the files in order in which they where created.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_nucleotide_div(), sumstat_omega(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
folder <- tempfile("coala-test")
model <- coal_model(10, 1) +
  feat_mutation(5) +
  sumstat_file(folder)
simulate(model)$file

unlink(folder, recursive = TRUE)  # Clean up
```

---

sumstat_four_gamete *Summary Statistic: Four-Gamete-Condition*

Description

This summary statistic calculates a number of values (see 'Value') related to the Four-Gamete-Condition (see 'Details'). It is sensitive for recombination and particularly useful when estimating recombination rates with **jaatha** or Approximate Bayesian Computation.

Usage

```
sumstat_four_gamete(
  name = "four_gamete",
  population = 1,
  transformation = identity,
  na.rm = FALSE
)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
population	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations. Default is population 1.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.
na.rm	should missing data be ignored? Default is FALSE.

Details

The Four-Gamete-Condition for two SNPs is violated if all four combinations of derived and ancestral alleles at the SNPs are observed in a gamete/a haplotype. Under an Infinite-Sites mutation model, a violation indicates that there must have been at least one recombination event between the SNPs.

Value

The statistic generates a matrix where each row represents one locus, and the columns give the statistic for different classes of pairs of SNPs:

mid_near The value for all pairs of SNPs that are close together, that is within 10 percent of the locus" length. If locus trios are used, only pairs of SNPs were both SNPs are on the middle locus are considered.

mid_far Same as `mid_near`, but for pairs of SNPs that are more that 10 percent of the locus" length apart.

outer Only when using locus trios. The statistic for pairs where both SNPs are on the same outer locus.

between Only when using locus trios. The statistic for pairs where one SNPs is on the middle locus, and the other is on an outer one.

mid The value for all pairs on the middle locus or all pairs when not using trios.

perc_polym The percentage of positions that are polymorpic.

Unphased Data

For unphased data, the four gamete condition is only counted as violated if it is violated for all possible phasing of the data.

See Also

To create a demographic model: `coal_model`

To calculate this statistic from data: `calc_sumstats_from_data`

Other summary statistics: `sumstat_dna()`, `sumstat_file()`, `sumstat_ihh()`, `sumstat_jsfs()`, `sumstat_mcmf()`, `sumstat_nucleotide_div()`, `sumstat_omega()`, `sumstat_seg_sites()`, `sumstat_sfs()`, `sumstat_tajimas_d()`, `sumstat_trees()`

Examples

```
model <- coal_model(5, 2) +
 feat_mutation(50) +
 feat_recombination(10) +
 sumstat_four_gamete()
stats <- simulate(model)
print(stats$four_gamete)
```

sumstat_ihh *Summary Statistic: Integrated Extended Haplotype Homozygosity*

Description

This summary statistic calculates a number of values based on extended haplotype homozygosity (EHH), including iHH, iES and optionally iHS. Coala relies on scan_hh from package **rehh** to calculate this statistic. Please refer to their documentation for detailed information on the implementation. Please cite the corresponding publication (see below) if you use the statistic for a publication.

Usage

```
sumstat_ihh(
  name = "ihh",
  population = 1,
  max_snps = 1000,
  calc_ihs = FALSE,
  transformation = identity
)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
population	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations. Default is population 1.
max_snps	The maximal number of SNPs per locus that are used for the calculation. If a locus has more SNPs, only a random subset of them will be used to increase performance. Set to Inf to use all SNPs.
calc_ihs	If set to TRUE, additionally standardized iHS is calculated.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

If calc_ihs = FALSE, a data.frame with values for iHH and iES is returned. Otherwise, a list of two data frames are returned, one for IHH and IES values and the other one for IHS values.

In all data.frames rows are SNPs and the columns present the following values for each SNP:

- CHR: The SNP's locus
- POSITION: The SNP's absolute position on its locus
- FREQ_A: The frequency of the ancestral allele
- FREQ_D: The frequency of the derived allele

- IHH_A: integrated EHH for the ancestral allele
- IHH_D: integrated EHH for the derived allele
- IES: integrated EHHS
- INES: integrated normalized EHHS
- IHS: iHS, normalized over all loci.

References

- Mathieu Gautier and Renaud Vitalis, rehh: an R package to detect footprints of selection in genome-wide SNP data from haplotype structure. Bioinformatics (2012) 28 (8): 1176-1177 first published online March 7, 2012 doi:10.1093/bioinformatics/bts115
- Voight et al., A map of recent positive selection in the human genome. PLoS Biol, 4(3):e72, Mar 2006.

Author(s)

Paul Staab

See Also

To create a demographic model: `coal_model`

To calculate this statistic from data: `calc_sumstats_from_data`

Other summary statistics: `sumstat_dna()`, `sumstat_file()`, `sumstat_four_gamete()`, `sumstat_jsfs()`, `sumstat_mcmf()`, `sumstat_nucleotide_div()`, `sumstat_omega()`, `sumstat_seg_sites()`, `sumstat_sfs()`, `sumstat_tajimas_d()`, `sumstat_trees()`

Examples

```
model <- coal_model(20, 1, 1000) +
  feat_mutation(1000) +
  sumstat_ihh()

  stat <- simulate(model)
  print(stat$ihh)
```

---

sumstat_jsfs *Summary Statistic: Joint Site Frequency Spectrum*

Description

The summary statistic calculates the joint site frequency spectrum (JSFS) for multiple populations.

Usage

```
sumstat_jsfs(
  name = "jsfs",
  populations = c(1, 2),
  per_locus = FALSE,
  transformation = identity
)
```

Arguments

`name`	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
`populations`	An integer vector containing the populations for which the JSFS is generated.
`per_locus`	If `TRUE`, the JSFS is returned for each locus instead of globally. In this case, the result is a list, where each entry is the JSFS for the corresponding locus.
`transformation`	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

The JSFS, given as an array. The dimensions correspond to the populations as given in the `populations` argument.

See Also

To create a demographic model: `coal_model`

To calculate this statistic from data: `calc_sumstats_from_data`

Other summary statistics: `sumstat_dna()`, `sumstat_file()`, `sumstat_four_gamete()`, `sumstat_ihh()`, `sumstat_mcmf()`, `sumstat_nucleotide_div()`, `sumstat_omega()`, `sumstat_seg_sites()`, `sumstat_sfs()`, `sumstat_tajimas_d()`, `sumstat_trees()`

Examples

```
model <- coal_model(c(2, 3, 4), 2) +
  feat_mutation(5) +
  feat_migration(1, symmetric = TRUE) +
  sumstat_jsfs("jsfs_12", populations = c(1, 2)) +
  sumstat_jsfs("jsfs_123", populations = c(1, 2, 3))
stats <- simulate(model)
print(stats$jsfs_12)
print(stats$jsfs_123)
```

sumstat_mcmf *Summary Statistic: Most Common Mutation's Frequency*

Description

This summary statistic calculates the observed frequency of the mutational pattern that is observed most often in the data.

Usage

```
sumstat_mcmf(
  name = "mcmf",
  population = 1,
  transformation = identity,
  expand_mcmf = FALSE,
  type_expand = 1
)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
population	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.
expand_mcmf	Whether to use or not the expanded MCMF. See Details
type_expand	Specifies the type of expanded MCMF to be used. See Details

Details

The expand_mcmf = FALSE calculates the mcmf per locus and returns a vector. The expand_mcmf = TRUE and type_expand = 1 returns the same results as the first column of a Matrix. The expand_mcmf = TRUE and type_expand = 2 adds the frequency of derived alleles in the most frequently observed mutational pattern as a second column. The expand_mcmf = TRUE and type_expand = 3 adds the percentage of positions that are polymorpic. When expanded_mcmf = TRUE results are returned as a matrix.

Value

A numeric vector or matrix containing MCMF for each locus.

mcmf The observed frequency of the mutational pattern that is observed most often in the data.

bal The frequency of derived alleles in the most frequently observed mutational pattern.

perc_poly The percentage of positions that are polymorpic.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_nucleotide_div(), sumstat_omega(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
# Calculate MCMF for a panmictic population
model <- coal_model(10, 2) +
  feat_mutation(50) +
  sumstat_mcmf()
simulate(model)
```

`sumstat_nucleotide_div`

Summary Statistic: Nucleotide Diversity

Description

The summary statistic calculates the nucleotide diversity (π) per locus, which is the mean number of pairwise difference for two individuals. It is a commonly used estimator for the scaled mutation rate θ.

Usage

```
sumstat_nucleotide_div(name = "pi", population = 1, transformation = identity)
```

Arguments

`name`	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
`population`	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations. Default is population 1.
`transformation`	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Details

The nucleotide diversity was introduced by

Nei and Li (1979). "Mathematical Model for Studying Genetic Variation in Terms of Restriction Endonucleases". PNAS 76 (10): 5269-73. doi:10.1073/pnas.76.10.5269.

Value

On simulation, this returns a vector with the value of pi for each locus.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_omega(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
model <- coal_model(5, 2) +
  feat_mutation(5) +
  sumstat_nucleotide_div()
stats <- simulate(model)
print(stats$pi)
```

---

sumstat_omega *Summary Statistic: Omega*

Description

Calculates the Omega Statistic introduced by Kim & Nielsen (2004) from the simulated data. The statistic is sensitive for hard selective sweeps. To calculate the statistic, coala relies on the command line program OmegaPlus, which needs to be downloaded and compiled manually in order to use the statistic.

Usage

```
sumstat_omega(
  name = "omega",
  min_win = 100,
  max_win = 1000,
  grid = 1000,
  binary = "automatic",
  transformation = identity
)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
min_win	The minimum distance from the grid point that a SNP must have to be included in the calculation of omega.

max_win	The maximum distance from the grid point that a SNP must have to be included in the calculation of omega.
grid	The number of points for which omega is calculated on each locus. Should be significantly lower than the locus length.
binary	The path of the binary for OmegaPlus. If set to "automatic", coala will try to find a binary called "OmegaPlus" using the PATH environment variable.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

A data frame listing of locus, genetic position and the calculated omega value.

References

Linkage disequilibrium as a signature of selective sweeps. Y. Kim and R. Nielsen (2004). Genetics, 167, 1513-1524.

OmegaPlus: a scalable tool for rapid detection of selective sweeps in whole-genome datasets. N. Alachiotis, A. Stamatakis and P. Pavlidis (2012). Bioinformatics Vol. 28 no. 17 2012, pages 2274-2275 doi:10.1093/bioinformatics/bts419

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_nucleotide_div(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
## Not run:
model <- coal_model(20, 1, 50000) +
  feat_recombination(50) +
  feat_mutation(1000) +
  feat_selection(strength_A = 1000, time = 0.03) +
  sumstat_omega()
stats <- simulate(model)
plot(stats$omega$omega, type = "l")
## End(Not run)
```

`sumstat_seg_sites` *Summary Statistic: Segregating Sites*

Description

This summary statistics generates a matrix of segregating sites. This is useful for calculating summary statistics that **coala** does not support..

Usage

```
sumstat_seg_sites(name = "seg_sites", transformation = identity)
```

Arguments

`name`	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
`transformation`	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

A list of `segsites` objects. These can be treated as a matrix for most applications. Rows are individuals, columns are SNPs.

See Also

For a description of the segregating sites class: `create_segsites`

To create a demographic model: `coal_model`

To calculate this statistic from data: `calc_sumstats_from_data`

Other summary statistics: `sumstat_dna()`, `sumstat_file()`, `sumstat_four_gamete()`, `sumstat_ihh()`, `sumstat_jsfs()`, `sumstat_mcmf()`, `sumstat_nucleotide_div()`, `sumstat_omega()`, `sumstat_sfs()`, `sumstat_tajimas_d()`, `sumstat_trees()`

Examples

```
model <- coal_model(5, 1) +
  feat_mutation(5) +
  sumstat_seg_sites("segsites")
stats <- simulate(model)
print(stats$segsites)
```

sumstat_sfs *Summary Statistic: Site Frequency Spectrum*

Description

The Site Frequency Spectrum (SFS) counts how many SNPs are in a sample according to their number of derived alleles.

Usage

```
sumstat_sfs(name = "sfs", population = "all", transformation = identity)
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
population	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations. Default is population 1.
transformation	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_nucleotide_div(), sumstat_omega(), sumstat_seg_sites(), sumstat_tajimas_d(), sumstat_trees()

Examples

```
model <- coal_model(20, 500) +
  feat_mutation(2) +
  sumstat_sfs()
stats <- simulate(model)
barplot(stats$sfs)
```

`sumstat_tajimas_d` *Summary Statistic: Tajima's D*

Description

This statistic calculates Tajima's D from the simulation results when added to a model. Tajima's D primarily measures an deviation of singletons from the neutral expectation of an equilibrium model. Negative values indicate an excess of singletons, while positive values code a depletion of them.

Usage

```
sumstat_tajimas_d(
  name = "tajimas_d",
  population = "all",
  transformation = identity
)
```

Arguments

`name`	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.
`population`	The population for which the statistic is calculated. Can also be "all" to calculate it from all populations. Default is population 1.
`transformation`	An optional function for transforming the results of the statistic. If specified, the results of the transformation are returned instead of the original values.

Value

On simulation, this returns a vector with the value of Tajima's D for each locus.

References

Tajima, F. (1989). "Statistical method for testing the neutral mutation hypothesis by DNA polymorphism.". Genetics 123 (3): 585-95.

See Also

To create a demographic model: `coal_model`

To calculate this statistic from data: `calc_sumstats_from_data`

Other summary statistics: `sumstat_dna()`, `sumstat_file()`, `sumstat_four_gamete()`, `sumstat_ihh()`, `sumstat_jsfs()`, `sumstat_mcmf()`, `sumstat_nucleotide_div()`, `sumstat_omega()`, `sumstat_seg_sites()`, `sumstat_sfs()`, `sumstat_trees()`

Examples

```
# A neutral model that should yield values close to zero:
model <- coal_model(5, 2) +
  feat_mutation(20) +
  feat_recombination(10) +
  sumstat_tajimas_d("taji_d")
stats <- simulate(model)
print(stats$taji_d)
```

---

sumstat_trees *Summary Statistic: Ancestral Trees*

Description

This statistic returns ancestral tress in NEWICK format.

Usage

```
sumstat_trees(name = "trees")
```

Arguments

name	The name of the summary statistic. When simulating a model, the value of the statistics are written to an entry of the returned list with this name. Summary statistic names must be unique in a model.

See Also

To create a demographic model: coal_model

To calculate this statistic from data: calc_sumstats_from_data

Other summary statistics: sumstat_dna(), sumstat_file(), sumstat_four_gamete(), sumstat_ihh(), sumstat_jsfs(), sumstat_mcmf(), sumstat_nucleotide_div(), sumstat_omega(), sumstat_seg_sites(), sumstat_sfs(), sumstat_tajimas_d()

Examples

```
# Without recombination:
model <- coal_model(4, 2) + sumstat_trees()
stats <- simulate(model)
print(stats$trees)

# With recombination:
model <- model + feat_recombination(5)
stats <- simulate(model)
print(stats$trees)
```

Index