Package 'jackalope'

July 6, 2021

Type Package

Title A Swift, Versatile Phylogenomic and High-Throughput Sequencing Simulator

Version 1.1.3

Description Simply and efficiently
simulates (i) variants from reference genomes and (ii) reads from both Illumina
<https://www.illumina.com/>
and Pacific Biosciences (PacBio) <https://www.pacb.com/> platforms.
It can either read reference genomes from FASTA files or simulate new ones.
Genomic variants can be simulated using summary statistics, phylogenies,
Variant Call Format (VCF) files, and coalescent simulations—the latter of which
can include selection, recombination, and demographic fluctuations.
'jackalope' can simulate single, paired-end, or mate-pair Illumina reads,
as well as PacBio reads.
These simulations include sequencing errors, mapping qualities, multiplexing,
and optical/polymerase chain reaction (PCR) duplicates.
Simulating Illumina sequencing is based on ART
by Huang et al. (2012) <doi:10.1093/bioinformatics/btr708>.
PacBio sequencing simulation is based on
SimLoRD by Stöcker et al. (2016) <doi:10.1093/bioinformatics/btw286>.
All outputs can be written to standard file formats.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

Depends R (>= 2.10)

biocViews

Imports ape, R6, Rcpp (>= 0.12.11), zlibbioc

LinkingTo Rcpp, RcppArmadillo, RcppProgress, Rhtslib, zlibbioc

SystemRequirements GNU make, C++11

RoxygenNote 7.1.1

Suggests coala, knitr, markdown, rmarkdown, scrm, testthat

VignetteBuilder knitr

URL https://github.com/lucasnell/jackalope

BugReports https://github.com/lucasnell/jackalope/issues

NeedsCompilation yes

Author Lucas A. Nell [cph, aut, cre] (<https://orcid.org/0000-0003-3209-0517>)

Maintainer Lucas A. Nell <lucas@lucasnell.com>

Repository CRAN

Date/Publication 2021-07-06 04:40:02 UTC

R topics documented:

`create_genome` *Create a reference genome.*

Description

Random chromosomes are generated to create a new `ref_genome` object. Note that this function will never generate empty chromosomes.

Usage

```
create_genome(
  n_chroms,
  len_mean,
  len_sd = 0,
  pi_tcag = rep(0.25, 4),
  n_threads = 1
)
```

Arguments

n_chroms	Number of chromosomes.
len_mean	Mean for the gamma distribution of chromosome sizes.
len_sd	Standard deviation for the gamma distribution of chromosome sizes. If set to <= 0, all chromosomes will be the same length. Defaults to 0.
pi_tcag	Vector of length 4 containing the nucleotide equilibrium frequencies for "T", "C", "A", and "G", respectively. Defaults to rep(0.25,4).
n_threads	Number of threads to use for parallel processing. This argument is ignored if OpenMP is not enabled. Defaults to 1.

Value

A ref_genome object.

Examples

```
genome <- create_genome(10, 100e3, 100, pi_tcag = c(0.1, 0.2, 0.3, 0.4))
```

---

create_haplotypes *Create haplotypes from a reference genome.*

Description

Uses one of multiple methods to create variant haplotypes from a reference genome. See haps_functions for the methods available.

Usage

```
create_haplotypes(
  reference,
  haps_info,
  sub = NULL,
  ins = NULL,
  del = NULL,
```

```
  epsilon = 0.03,
  n_threads = 1,
  show_progress = FALSE
)
```

Arguments

reference	A ref_genome object from which to generate haplotypes. This argument is required.
haps_info	Output from one of the haps_functions. These functions organize higher-level information for use here. See haps_functions for brief descriptions and links to each method. If this argument is NULL, all arguments other than reference are ignored, and an empty haplotypes object with no haplotypes is returned. This is designed for use when you'd like to add mutations manually. If you create a blank haplotypes object, you can use its add_haps method to add haplotypes manually.
sub	Output from one of the sub_models functions that organizes information for the substitution models. See sub_models for more information on these models and their required parameters. This argument is ignored if you are using a VCF file to create haplotypes. Passing NULL to this argument results in no substitutions. Defaults to NULL.
ins	Output from the indels function that specifies rates of insertions by length. This argument is ignored if you are using a VCF file to create haplotypes. Passing NULL to this argument results in no insertions. Defaults to NULL.
del	Output from the indels function that specifies rates of deletions by length. This argument is ignored if you are using a VCF file to create haplotypes. Passing NULL to this argument results in no deletions. Defaults to NULL.
epsilon	Error control parameter for the "tau-leaping" approximation to the Doob–Gillespie algorithm, as used for the indel portion of the simulations. Smaller values result in a closer approximation. Larger values are less exact but faster. Values must be >= 0 and < 1. For more information on the approximation, see Cao et al. (2006) and Wieder et al. (2011), listed below. If epsilon is 0, then it reverts to the exact Doob–Gillespie algorithm. Defaults to 0.03.
n_threads	Number of threads to use for parallel processing. This argument is ignored if OpenMP is not enabled. Threads are spread across chromosomes, so it doesn't make sense to supply more threads than chromosomes in the reference genome. Defaults to 1.
show_progress	Boolean for whether to show a progress bar during processing. Defaults to FALSE.

Value

A haplotypes object.

References

Cao, Y., D. T. Gillespie, and L. R. Petzold. 2006. Efficient step size selection for the tau-leaping simulation method. *The Journal of Chemical Physics* **124**(4): 044109.

Doob, J. L. 1942. Topics in the theory of markoff chains. *Transactions of the American Mathematical Society* **52**(1): 37–64.

Gillespie, D. T. 1976. A general method for numerically simulating the stochastic time evolution of coupled chemical reactions. *Journal of Computational Physics* **22**(4): 403–434.

Wieder, N., R. H. Fink, and F. von Wegner. 2011. Exact and approximate stochastic simulation of intracellular calcium dynamics. *Journal of Biomedicine and Biotechnology* **2011**: 572492.

Examples

```
r <- create_genome(10, 1000)
v_phylo <- create_haplotypes(r, haps_phylo(ape::rcoal(5)), sub_JC69(0.1))
v_theta <- create_haplotypes(r, haps_theta(0.001, 5), sub_K80(0.1, 0.2))
```

`evo_rates` *Table of evolutionary rates.*

Description

From Table 1 in Sung et al. (2016).

Usage

```
evo_rates
```

Format

A data frame with 15 rows and 8 variables:

`domain` Either `Bacteria` or `Eukarya` for what type of organism the species is.

`species` Species name.

`Ge` Effective genome size using only coding DNA.

`Gc_Gnc` Effective genome size using coding DNA and non-coding DNA that is under purifying selection.

`indels` Rate of insertions and deletions (10^{-10} events per site per generation).

`subs` Base-substitution mutation rate (10^{-10} events per site per generation).

`Ne` Effective population size ($\times 10^6$).

`theta_s` Population mutation rate estimated using θ_s.

`pi_s` Population mutation rate estimated using π_s.

Source

doi: 10.1534/g3.116.030890

References

Sung, W., M. S. Ackerman, M. M. Dillon, T. G. Platt, C. Fuqua, V. S. Cooper, and M. Lynch. 2016. Evolution of the insertion-deletion mutation rate across the tree of life. *G3: Genes | Genomes | Genetics* **6**:2583–2591.

`haplotypes` *An R6 Class Representing Haploid Variants*

Description

Interactive wrapper for a pointer to a C++ object that stores information about variant haplotypes from a single reference genome.

Details

This class should NEVER be created using `haplotypes$new`. Only use `create_haplotypes`. Because this class wraps a pointer to a C++ object, there are no fields to manipulate directly. All manipulations are done through this class's methods.

Connections to `ref_genome` objects

Regarding the `ref_genome` object you use to create a `haplotypes` object, you should note the following:

- **This point is the most important.** Both the `ref_genome` and `haplotypes` objects use the same underlying C++ object to store reference genome information. Thus, if you make any changes to the `ref_genome` object, those changes will also show up in the `haplotypes` object. For example, if you make a `haplotypes` object named `V` based on an existing `ref_genome` object named `R`, then you merge chromosomes in `R`, `V` will now have merged chromosomes. If you've already started adding mutations to `V`, then all the indexes used to store those mutations will be inaccurate. So when you do anything with `V` later, your R session will crash or have errors. **The lesson here is that you shouldn't edit the reference genome after using it to create haplotypes.**
- If a `ref_genome` object is used to create a `haplotypes` object, deleting the `ref_genome` object won't cause issues with the `haplotypes` object. However, the `haplotypes` class doesn't provide methods to edit chromosomes, so only remove the `ref_genome` object when you're done editing the reference genome.

Methods

Public methods:

- `haplotypes$new()`
- `haplotypes$print()`
- `haplotypes$ptr()`
- `haplotypes$n_chroms()`
- `haplotypes$n_haps()`

- haplotypes$sizes()
- haplotypes$chrom_names()
- haplotypes$hap_names()
- haplotypes$chrom()
- haplotypes$gc_prop()
- haplotypes$nt_prop()
- haplotypes$set_names()
- haplotypes$add_haps()
- haplotypes$dup_haps()
- haplotypes$rm_haps()
- haplotypes$add_sub()
- haplotypes$add_ins()
- haplotypes$add_del()

Method `new()`**:** Do NOT use this; only use `create_haplotypes` to make new `haplotypes`.

Usage:

```
haplotypes$new(genomes_ptr, reference_ptr)
```

Arguments:

`genomes_ptr` An `externalptr` object pointing to a C++ object that stores the information about the haplotypes.

`reference_ptr` An `externalptr` object pointing to a C++ object that stores the information about the reference genome.

Method `print()`**:** Print a `haplotypes` object.

Usage:

```
haplotypes$print()
```

Method `ptr()`**:** View pointer to underlying C++ object (this is not useful to end users).

Usage:

```
haplotypes$ptr()
```

Returns: An `externalptr` object.

Method `n_chroms()`**:** View number of chromosomes.

Usage:

```
haplotypes$n_chroms()
```

Returns: Integer number of chromosomes.

Method `n_haps()`**:** View number of haplotypes.

Usage:

```
haplotypes$n_haps()
```

Returns: Integer number of haplotypes.

Method `sizes()`**:** View chromosome sizes for one haplotype.

Usage:

```
haplotypes$sizes(hap_ind)
```

Arguments:

`hap_ind` Index for the focal haplotype.

Returns: Integer vector of chromosome sizes for focal haplotype.

Method `chrom_names()`**:** View chromosome names.

Usage:

```
haplotypes$chrom_names()
```

Returns: Character vector of chromosome names.

Method `hap_names()`**:** View haplotype names.

Usage:

```
haplotypes$hap_names()
```

Returns: Character vector of haplotype names.

Method `chrom()`**:** View one haplotype chromosome.

Usage:

```
haplotypes$chrom(hap_ind, chrom_ind)
```

Arguments:

`hap_ind` Index for the focal haplotype.

`chrom_ind` Index for the focal chromosome.

Returns: A single string representing the chosen haplotype chromosome's DNA sequence.

Method `gc_prop()`**:** View GC proportion for part of one haplotype chromosome.

Usage:

```
haplotypes$gc_prop(hap_ind, chrom_ind, start, end)
```

Arguments:

`hap_ind` Index for the focal haplotype.

`chrom_ind` Index for the focal chromosome.

`start` Point on the chromosome at which to start the calculation (inclusive).

`end` Point on the chromosome at which to end the calculation (inclusive).

Returns: A double in the range [0,1] representing the proportion of DNA sequence that is either `G` or `C`.

Method `nt_prop()`**:** View nucleotide content for part of one haplotype chromosome

Usage:

```
haplotypes$nt_prop(nt, hap_ind, chrom_ind, start, end)
```

Arguments:

`nt` Which nucleotide to calculate the proportion that the DNA sequence is made of. Must be one of `T`, `C`, `A`, `G`, or `N`.

`hap_ind` Index for the focal haplotype.

chrom_ind Index for the focal chromosome.

start Point on the chromosome at which to start the calculation (inclusive).

end Point on the chromosome at which to end the calculation (inclusive).

Returns: A double in the range [0,1] representing the proportion of DNA sequence that is nt.

Method set_names(): Change haplotype names.

Usage:

```
haplotypes$set_names(new_names)
```

Arguments:

new_names Vector of new names to use. This must be the same length as the number of current names.

Returns: This R6 object, invisibly.

Method add_haps(): Add one or more blank, named haplotypes

Usage:

```
haplotypes$add_haps(new_names)
```

Arguments:

new_names Vector of name(s) for the new haplotype(s).

Returns: This R6 object, invisibly.

Method dup_haps(): Duplicate one or more haplotypes by name.

Usage:

```
haplotypes$dup_haps(hap_names, new_names = NULL)
```

Arguments:

hap_names Vector of existing haplotype name(s) that you want to duplicate.

new_names Optional vector specifying the names of the duplicates. If NULL, their names are auto-generated. Defaults to NULL.

Returns: This R6 object, invisibly.

Method rm_haps(): Remove one or more haplotypes by name.

Usage:

```
haplotypes$rm_haps(hap_names)
```

Arguments:

hap_names Vector of existing haplotype name(s) that you want to remove.

Returns: This R6 object, invisibly.

Method add_sub(): Manually add a substitution.

Usage:

```
haplotypes$add_sub(hap_ind, chrom_ind, pos, nt)
```

Arguments:

hap_ind Index for the focal haplotype.

chrom_ind Index for the focal chromosome.

`pos` Position at which to add the mutation.

`nt` Single character representing the nucleotide to change the current one to.

Returns: This `R6` object, invisibly.

Method `add_ins()`**:** Manually add an insertion.

Usage:

```
haplotypes$add_ins(hap_ind, chrom_ind, pos, nts)
```

Arguments:

`hap_ind` Index for the focal haplotype.

`chrom_ind` Index for the focal chromosome.

`pos` Position at which to add the mutation.

`nts` String representing the nucleotide(s) that will be inserted after the designated position.

Returns: This `R6` object, invisibly.

```
\item{`add_del(hap_ind, chrom_ind, pos, n_nts)`}{Manually add a deletion
   for a given haplotype (`hap_ind`), chromosome (`chrom_ind`), and position (`pos`).
    The designated number of nucleotides to delete (`n_nts`) will be deleted
    starting at `pos`, unless `pos` is near the chromosome end and doesn't have
    `n_nts` nucleotides to remove; it simply stops at the chromosome end in
    this case.}
```

Method `add_del()`**:** Manually add a deletion.

Usage:

```
haplotypes$add_del(hap_ind, chrom_ind, pos, n_nts)
```

Arguments:

`hap_ind` Index for the focal haplotype.

`chrom_ind` Index for the focal chromosome.

`pos` Position at which to add the mutation.

`n_nts` Single integer specifying the number of nucleotides to delete. These will be deleted starting at `pos`. If `pos` is near the chromosome end and doesn't have `n_nts` nucleotides to remove, it simply removes nucleotides from `pos` to the chromosome end.

Returns: This `R6` object, invisibly.

See Also

`create_haplotypes`

haps_functions *Organize higher-level information for creating haplotypes.*

Description

The following functions organize information that gets passed to `create_haplotypes` to generate haplotypes from a reference genome. Each function represents a method of generation and starts with `"haps_"`. The first three are phylogenomic methods, and all functions but `haps_vcf` will use molecular evolution information when passed to `create_haplotypes`.

Details

`haps_theta` Uses an estimate for theta, the population-scaled mutation rate, and a desired number of haplotypes.

`haps_phylo` Uses phylogenetic tree(s) from `phylo` object(s) or NEWICK file(s), one tree per chromosome or one for all chromosomes.

`haps_gtrees` Uses gene trees, either in the form of an object from the `scrm` or `coala` package or a file containing output in the style of the `ms` program.

`haps_ssites` Uses matrices of segregating sites, either in the form of `scrm` or `coala` coalescent-simulator object(s), or a `ms`-style output file.

`haps_vcf` Uses a haplotype call format (VCF) file that directly specifies haplotypes.

See Also

`create_haplotypes`

haps_gtrees *Organize information to create haplotypes using gene trees*

Description

This function organizes higher-level information for creating haplotypes from gene trees output from coalescent simulations. Note that all gene trees must be rooted and binary.

Usage

```
haps_gtrees(obj = NULL, fn = NULL)

write_gtrees(gtrees, out_prefix)
```

Arguments

obj	Object containing gene trees. This can be one of the following: (1) A single list with a trees field inside. This field must contain a set of gene trees for each chromosome. (2) A list of lists, each sub-list containing a trees field of length 1. The top-level list must be of the same length as the number of chromosomes. Defaults to NULL.
fn	A single string specifying the name of the file containing the ms-style coalescent output with gene trees. Defaults to NULL.
gtrees	A haps_gtrees_info object output from haps_gtrees.
out_prefix	Prefix for the output file of gene trees. The extension will be .trees.

Details

Using the obj argument is designed after the trees fields in the output from the scrm and coala packages. (These packages are not required to be installed when installing jackalope.) To get gene trees, make sure to add + sumstat_trees() to the coalmodel for coala, or make sure that "-T" is present in args for scrm. If using either of these packages, I encourage you to cite them. For citation information, see output from citation("scrm") or citation("coala").

If using an output file from a command-line program like ms/msms, add the -T option.

Value

A haps_gtrees_info object containing information used in create_haplotypes to create variant haplotypes. This class is just a wrapper around a list of NEWICK tree strings, one for each gene tree, which you can view (but not change) using the object's trees() method.

Functions

- write_gtrees: Write gene trees to ms-style output file.

---

haps_phylo *Organize information to create haplotypes using phylogenetic tree(s)*

Description

This function organizes higher-level information for creating haplotypes from phylogenetic tree(s) output as phylo or multiPhylo objects (both from the ape package) or NEWICK files. Note that all phylogenetic trees must be rooted and binary. If using this function, I encourage you to cite ape. For citation information, see output from citation("ape").

Usage

```
haps_phylo(obj = NULL, fn = NULL)
```

Arguments

obj	Object containing phylogenetic tree(s). This can be (1) a single phylo object that represents all chromosomes in the genome or (2) a list or multiPhylo object containing a phylo object for each reference chromosome. In the latter case, phylogenies will be assigned to chromosomes in the order provided. Defaults to NULL.
fn	One or more string(s), each of which specifies the file name of a NEWICK file containing a phylogeny. If one name is provided, that phylogeny will be used for all chromosomes. If more than one is provided, there must be a phylogeny for each reference genome chromosome, and phylogenies will be assigned to chromosomes in the order provided. Defaults to NULL.

Details

See ?ape::write.tree for writing phylogenies to an output file.

Value

A haps_phylo_info object containing information used in create_haplotypes to create variant haplotypes. This class is just a wrapper around a list containing phylogenetic tree information for each reference chromosome, which you can view (but not change) using the object's phylo() method.

---

haps_ssites *Organize information to create haplotypes using segregating sites matrices*

Description

This function organizes higher-level information for creating haplotypes from matrices of segregating sites output from coalescent simulations.

Usage

```
haps_ssites(obj = NULL, fn = NULL)
```

Arguments

obj	Object containing segregating sites information. This can be one of the following: (1) A single list with a seg_sites field inside. This field must contain a matrix for segregating sites for each chromosome. The matrix itself should contain the haplotype information, coded using 0s and 1s: 0s indicate the ancestral state and 1s indicate mutant. The matrix column names should indicate the positions of the polymorphisms on the chromosome. If positions are in the range (0,1), they're assumed to come from an infinite- sites model and are relative positions. If positions are integers in the range [0, chromosome length - 1] or [1, chromosome length], they're assumed to come from an finite-sites model and are absolute positions. Defaults to NULL.

fn	A single string specifying the name of the file containing the ms-style coalescent output with segregating site info. Defaults to NULL.

Details

For what the seg_sites field should look like in a list, see output from the scrm or coala package. (These packages are not required to be installed when installing jackalope.) If using either of these packages, I encourage you to cite them. For citation information, see output from citation("scrm") or citation("coala").

Value

A haps_ssites_info object containing information used in create_haplotypes to create variant haplotypes. This class is just a wrapper around a list of matrices of segregating site info, which you can view (but not change) using the object's mats() method.

---

haps_theta *Organize information to create haplotypes using theta parameter*

Description

This function organizes higher-level information for creating haplotypes from the population-scaled mutation rate and a desired number of haplotypes.

Usage

```
haps_theta(theta, n_haps)
```

Arguments

theta	Population-scaled mutation rate.
n_haps	Number of desired haplotypes.

Value

A haps_theta_info object containing information used in create_haplotypes to create variant haplotypes. This class is just a wrapper around a list containing the phylogenetic tree and theta parameter, which you can view (but not change) using the object's phylo() and theta() methods, respectively.

haps_vcf *Organize information to create haplotypes using a VCF file*

Description

This function organizes higher-level information for creating haplotypes from Variant Call Format (VCF) files.

Usage

```
haps_vcf(fn, print_names = FALSE)
```

Arguments

fn	A single string specifying the name of the VCF file
print_names	Logical for whether to print all unique chromosome names from the VCF file when VCF chromosome names don't match those from the reference genome. This printing doesn't happen until this object is passed to `create_haplotypes`. This can be useful for troubleshooting. Defaults to `FALSE`.

Value

A `haps_vcf_info` object containing information used in `create_haplotypes` to create variant haplotypes. This class is just a wrapper around a list containing the arguments to this function, which you can view (but not change) using the object's `fn()` and `print_names()` methods.

illumina *Create and write Illumina reads to FASTQ file(s).*

Description

From either a reference genome or set of variant haplotypes, create Illumina reads from error profiles and write them to FASTQ output file(s). I encourage you to cite the reference below in addition to `jackalope` if you use this function.

Usage

```
illumina(obj,
         out_prefix,
         n_reads,
         read_length,
         paired,
         frag_mean = 400,
         frag_sd = 100,
         matepair = FALSE,
```

```
seq_sys = NULL,
profile1 = NULL,
profile2 = NULL,
ins_prob1 = 0.00009,
del_prob1 = 0.00011,
ins_prob2 = 0.00015,
del_prob2 = 0.00023,
frag_len_min = NULL,
frag_len_max = NULL,
haplotype_probs = NULL,
barcodes = NULL,
prob_dup = 0.02,
sep_files = FALSE,
compress = FALSE,
comp_method = "bgzip",
n_threads = 1L,
read_pool_size = 1000L,
show_progress = FALSE,
overwrite = FALSE)
```

Arguments

obj	Sequencing object of class ref_genome or haplotypes.
out_prefix	Prefix for the output file(s), including entire path except for the file extension.
n_reads	Number of reads you want to create.
read_length	Length of reads.
paired	Logical for whether to use paired-end reads. This argument is changed to TRUE if matepair is TRUE.
frag_mean	Mean of the Gamma distribution that generates fragment sizes. Defaults to 400.
frag_sd	Standard deviation of the Gamma distribution that generates fragment sizes. Defaults to 100.
matepair	Logical for whether to simulate mate-pair reads. Defaults to FALSE.
seq_sys	Full or abbreviated name of sequencing system to use. See "Sequencing systems" section for options. See "Sequencing profiles" section for more information on how this argument, profile1, and profile2 are used to specify profiles. Defaults to NULL.
profile1	Custom profile file for read 1. See "Sequencing profiles" section for more information on how this argument, profile2, and seq_sys are used to specify profiles. Defaults to NULL.
profile2	Custom profile file for read 2. See "Sequencing profiles" section for more information on how this argument, profile1, and seq_sys are used to specify profiles. Defaults to NULL.
ins_prob1	Insertion probability for read 1. Defaults to 0.00009.
del_prob1	Deletion probability for read 1. Defaults to 0.00011.
ins_prob2	Insertion probability for read 2. Defaults to 0.00015.

del_prob2 Deletion probability for read 2. Defaults to 0.00023.

frag_len_min Minimum fragment size. A NULL value results in the read length. Defaults to NULL.

frag_len_max Maximum fragment size. A NULL value results in 2^32-1, the maximum allowed value. Defaults to NULL

haplotype_probs
Relative probability of sampling each haplotype. This is ignored if sequencing a reference genome. NULL results in all having the same probability. Defaults to NULL.

barcodes Character vector of barcodes for each haplotype, or a single barcode if sequencing a reference genome. NULL results in no barcodes. Defaults to NULL.

prob_dup A single number indicating the probability of duplicates. Defaults to 0.02.

sep_files Logical indicating whether to make separate files for each haplotype. This argument is coerced to FALSE if the obj argument is not a haplotypes object. Defaults to FALSE.

compress Logical specifying whether or not to compress output file, or an integer specifying the level of compression, from 1 to 9. If TRUE, a compression level of 6 is used. Defaults to FALSE.

comp_method Character specifying which type of compression to use if any is desired. Options include "gzip" and "bgzip". This is ignored if compress is FALSE, and it throws an error if it's set to "gzip" when n_threads > 1 (since I don't have a method to do gzip compression in parallel). Defaults to "bgzip".

n_threads The number of threads to use in processing. If compress is TRUE or > 0 (indicating compressed output), setting n_threads to 2 or more makes this function first create an uncompressed file/files using n_threads threads, then compress that/those file/files also using n_threads threads. There is no speed increase if you try to use multiple threads to create compressed output on the fly, so that option is not included. If you want to be conservative with disk space (by not having an uncompressed file present even temporarily), set n_threads to 1. Threads are NOT spread across chromosomes or haplotypes, so you don't need to think about these when choosing this argument's value. However, all threads write to the same file/files, so there are diminishing returns for providing many threads. This argument is ignored if the package was not compiled with OpenMP. Defaults to 1.

read_pool_size The number of reads to store before writing to disk. Increasing this number should improve speed but take up more memory. Defaults to 1000.

show_progress Logical for whether to show a progress bar. Defaults to FALSE.

overwrite Logical for whether to overwrite existing FASTQ file(s) of the same name, if they exist.

Value

Nothing is returned.

Sequencing profiles

This section outlines how to use the `seq_sys`, `profile1`, and `profile2` arguments. If all arguments are `NULL` (their defaults), a sequencing system is chosen based on the read length. If, however, one or more arguments has been provided, then how they're provided should depend on whether you want single- or paired-end reads.

For single-end reads

- `profile2` should be `NULL`.
- Only `seq_sys` or `profile1` should be provided, not both.

For paired-end reads

- If providing `seq_sys`, don't provide either `profile1` or `profile2`.
- If providing `profile1`, you must also provide `profile2` (they can be the same if you want) and you cannot provide `seq_sys`.

Sequencing systems

Sequencing system options are the following, where, for each system, "name" is the full name, "abbrev" is the abbreviated name, "max_len" indicates the maximum allowed read length, and "paired" indicates whether paired-end sequencing is allowed.

name	abbrev	max_len	paired
Genome Analyzer I	GA1	44	Yes
Genome Analyzer II	GA2	75	Yes
HiSeq 1000	HS10	100	Yes
HiSeq 2000	HS20	100	Yes
HiSeq 2500	HS25	150	Yes
HiSeqX v2.5 PCR free	HSXn	150	Yes
HiSeqX v2.5 TruSeq	HSXt	150	Yes
MiniSeq TruSeq	MinS	50	No
MiSeq v1	MSv1	250	Yes
MiSeq v3	MSv3	250	Yes
NextSeq 500 v2	NS50	75	Yes

ID lines

The ID lines for FASTQ files are formatted as such:
@<genome name>-<chromosome name>-<starting position>-<strand>[/<read#>]

where the part in [] is only for paired-end Illumina reads, and where genome name is always `REF` for reference genomes (as opposed to haplotypes).

References

Huang, W., L. Li, J. R. Myers, and G. T. Marth. 2012. ART: a next-generation sequencing read simulator. *Bioinformatics* **28**:593–594.

Examples

```
rg <- create_genome(10, 100e3, 100)
dir <- tempdir(TRUE)
illumina(rg, paste0(dir, "/illumina_reads"), n_reads = 100,
         read_length = 100, paired = FALSE)
```

`indels` *Insertions and deletions (indels) specification*

Description

Construct necessary information for insertions and deletions (indels) that will be used in `create_haplotypes`.

Usage

```
indels(rate, max_length = 10, a = NULL, rel_rates = NULL)
```

Arguments

`rate`	Single number specifying the overall indel rate among all lengths.
`max_length`	Maximum length of indels. Defaults to `10`.
`a`	Extra parameter necessary for generating rates from a Lavalette distribution. See Details for more info. Defaults to `NULL`.
`rel_rates`	A numeric vector of relative rates for each indel length from 1 to the maximum length. If provided, all arguments other than `rate` are ignored. Defaults to `NULL`.

Details

All indels require the `rate` parameter, which specifies the overall indels rate among all lengths. The `rate` parameter is ultimately combined with a vector of relative rates among the different lengths of indels from 1 to the maximum possible length. There are three different ways to specify/generate relative-rate values.

1. Assume that rates are proportional to `exp(-L)` for indel length `L` from 1 to the maximum length (Albers et al. 2011). This method will be used if the following arguments are provided:
   - `rate`
   - `max_length`
2. Generate relative rates from a Lavalette distribution (Fletcher and Yang 2009), where the rate for length `L` is proportional to `{L * max_length / (max_length -L + 1)}^(-a)`. This method will be used if the following arguments are provided:
   - `rate`
   - `max_length`

- a

3. Directly specify values by providing a numeric vector of relative rates for each insertion/deletion length from 1 to the maximum length. This method will be used if the following arguments are provided:
   - rate
   - rel_rates

Value

An indel_info object, which is an R6 class that wraps the info needed for the create_haplotypes function. It does not allow the user to directly manipulate the info inside, as that should be done using this function. You can use the rates() method to view the indel rates by size.

References

Albers, C. A., G. Lunter, D. G. MacArthur, G. McVean, W. H. Ouwehand, and R. Durbin. 2011. Dindel: accurate indel calls from short-read data. Genome Research 21:961–973.

Fletcher, W., and Z. Yang. 2009. INDELible: a flexible simulator of biological sequence evolution. Molecular Biology and Evolution 26:1879–1888.

Examples

```
# relative rates are proportional to `exp(-L)` for indel
# length `L` from 1 to 5:
indel_rates1 <- indels(0.1, max_length = 5)

# relative rates are proportional to Lavalette distribution
# for length from 1 to 10:
indel_rates2 <- indels(0.2, max_length = 10, a = 1.1)

# relative rates are all the same for lengths from 1 to 100:
indel_rates3 <- indels(0.2, rel_rates = rep(1, 100))
```

---

jackalope *jackalope: An efficient, flexible molecular evolution and sequencing simulator.*

Description

jackalope simply and efficiently simulates (i) haplotypes from reference genomes and (ii) reads from both Illumina and Pacific Biosciences (PacBio) platforms. It can either read reference genomes from FASTA files or simulate new ones. Variant haplotypes can be simulated using summary statistics, phylogenies, Variant Call Format (VCF) files, and coalescent simulations—the latter of which

can include selection, recombination, and demographic fluctuations. jackalope can simulate single, paired-end, or mate-pair Illumina reads, as well as reads from Pacific Biosciences These simulations include sequencing errors, mapping qualities, multiplexing, and optical/PCR duplicates. All outputs can be written to standard file formats.

---

pacbio *Create and write PacBio reads to FASTQ file(s).*

Description

From either a reference genome or set of variant haplotypes, create PacBio reads and write them to FASTQ output file(s). I encourage you to cite the reference below in addition to jackalope if you use this function.

Usage

```
pacbio(obj,
       out_prefix,
       n_reads,
       chi2_params_s = c(0.01214, -5.12, 675, 48303.0732881,
                         1.4691051212330266),
       chi2_params_n = c(0.00189237136, 2.53944970, 5500),
       max_passes = 40,
       sqrt_params = c(0.5, 0.2247),
       norm_params = c(0, 0.2),
       prob_thresh = 0.2,
       ins_prob = 0.11,
       del_prob = 0.04,
       sub_prob = 0.01,
       min_read_length = 50,
       lognorm_read_length = c(0.200110276521, -10075.4363813,
                               17922.611306),
       custom_read_lengths = NULL,
       prob_dup = 0.0,
       haplotype_probs = NULL,
       sep_files = FALSE,
       compress = FALSE,
       comp_method = "bgzip",
       n_threads = 1L,
       read_pool_size = 100L,
       show_progress = FALSE,
       overwrite = FALSE)
```

Arguments

obj	Sequencing object of class ref_genome or haplotypes.
out_prefix	Prefix for the output file(s), including entire path except for the file extension.

n_reads Number of reads you want to create.

chi2_params_s Vector containing the 5 parameters for the curve determining the scale parameter for the chi^2 distribution. Defaults to c(0.01214,-5.12,675,48303.0732881,1.4691051212330266).

chi2_params_n Vector containing the 3 parameters for the function determining the n parameter for the chi^2 distribution. Defaults to c(0.00189237136,2.53944970,5500).

max_passes Maximal number of passes for one molecule. Defaults to 40.

sqrt_params Vector containing the 2 parameters for the square root function for the quality increase. Defaults to c(0.5,0.2247).

norm_params Vector containing the 2 parameters for normal distributed noise added to quality increase square root function Defaults to c(0,0.2).

prob_thresh Upper bound for the modified total error probability. Defaults to 0.2.

ins_prob Probability for insertions for reads with one pass. Defaults to 0.11.

del_prob Probability for deletions for reads with one pass. Defaults to 0.04.

sub_prob Probability for substitutions for reads with one pass. Defaults to 0.01.

min_read_length
Minium read length for lognormal distribution. Defaults to 50.

lognorm_read_length
Vector containing the 3 parameters for lognormal read length distribution. Defaults to c(0.200110276521,-10075.4363813,17922.611306).

custom_read_lengths
Sample read lengths from a vector or column in a matrix; if a matrix, the second column specifies the sampling weights. If NULL, it samples read lengths from the lognormal distribution using parameters in lognorm_read_length. Defaults to NULL.

prob_dup A single number indicating the probability of duplicates. Defaults to 0.0.

haplotype_probs
Relative probability of sampling each haplotype. This is ignored if sequencing a reference genome. NULL results in all having the same probability. Defaults to NULL.

sep_files Logical indicating whether to make separate files for each haplotype. This argument is coerced to FALSE if the obj argument is not a haplotypes object. Defaults to FALSE.

compress Logical specifying whether or not to compress output file, or an integer specifying the level of compression, from 1 to 9. If TRUE, a compression level of 6 is used. Defaults to FALSE.

comp_method Character specifying which type of compression to use if any is desired. Options include "gzip" and "bgzip". This is ignored if compress is FALSE, and it throws an error if it's set to "gzip" when n_threads > 1 (since I don't have a method to do gzip compression in parallel). Defaults to "bgzip".

n_threads The number of threads to use in processing. If compress is TRUE or > 0 (indicating compressed output), setting n_threads to 2 or more makes this function first create an uncompressed file/files using n_threads threads, then compress that/those file/files also using n_threads threads. There is no speed increase

if you try to use multiple threads to create compressed output on the fly, so that option is not included. If you want to be conservative with disk space (by not having an uncompressed file present even temporarily), set `n_threads` to `1`. Threads are NOT spread across chromosomes or haplotypes, so you don't need to think about these when choosing this argument's value. However, all threads write to the same file/files, so there are diminishing returns for providing many threads. This argument is ignored if the package was not compiled with OpenMP. Defaults to `1`.

`read_pool_size` The number of reads to store before writing to disk. Increasing this number should improve speed but take up more memory. Defaults to `100`.

`show_progress` Logical for whether to show a progress bar. Defaults to `FALSE`.

`overwrite` Logical for whether to overwrite existing FASTQ file(s) of the same name, if they exist.

Value

Nothing is returned.

ID lines

The ID lines for FASTQ files are formatted as such:
@<genome name>-<chromosome name>-<starting position>-<strand>

where genome name is always `REF` for reference genomes (as opposed to haplotypes).

References

Stöcker, B. K., J. Köster, and S. Rahmann. 2016. SimLoRD: simulation of long read data. *Bioinformatics* **32**:2704–2706.

Examples

```
rg <- create_genome(10, 100e3, 100)
dir <- tempdir(TRUE)
pacbio(rg, paste0(dir, "/pacbio_reads"), n_reads = 100)
```

`read_fasta` *Read a fasta file.*

Description

Accepts uncompressed and gzipped fasta files.

Usage

```
read_fasta(fasta_files, fai_files = NULL, cut_names = FALSE)
```

Arguments

fasta_files	File name(s) of the fasta file(s).
fai_files	File name(s) of the fasta index file(s). Providing this argument speeds up the reading process significantly. If this argument is provided, it must be the same length as the fasta_files argument. Defaults to NULL, which indicates the fasta file(s) is/are not indexed.
cut_names	Boolean for whether to cut chromosome names at the first space. This argument is ignored if fai_file is not NULL. Defaults to FALSE.

Value

A ref_genome object.

---

ref_genome *R6 Class Representing a Reference Genome*

Description

Interactive wrapper for a pointer to a C++ object that stores reference genome information.

Details

This class should NEVER be created using ref_genome$new. Only use read_fasta or create_genome. Because this class wraps a pointer to a C++ object, there are no fields to manipulate directly. All manipulations are done through this class's methods.

Methods

Public methods:

- ref_genome$new()
- ref_genome$print()
- ref_genome$ptr()
- ref_genome$n_chroms()
- ref_genome$sizes()
- ref_genome$chrom_names()
- ref_genome$chrom()
- ref_genome$gc_prop()
- ref_genome$nt_prop()
- ref_genome$set_names()
- ref_genome$clean_names()
- ref_genome$add_chroms()
- ref_genome$rm_chroms()
- ref_genome$merge_chroms()
- ref_genome$filter_chroms()

- ref_genome$replace_Ns()

Method `new()`**:** Do NOT use this; only use `read_fasta` or `create_genome` to make a new `ref_genome`.

Usage:

```
ref_genome$new(genome_ptr)
```

Arguments:

`genome_ptr` An `externalptr` object pointing to a C++ object that stores the information about the reference genome.

Method `print()`**:** Print a `ref_genome` object.

Usage:

```
ref_genome$print()
```

Method `ptr()`**:** View pointer to underlying C++ object (this is not useful to end users).

Usage:

```
ref_genome$ptr()
```

Returns: An `externalptr` object.

Method `n_chroms()`**:** View number of chromosomes.

Usage:

```
ref_genome$n_chroms()
```

Returns: Integer number of chromosomes.

Method `sizes()`**:** View chromosome sizes.

Usage:

```
ref_genome$sizes()
```

Returns: Integer vector of chromosome sizes.

Method `chrom_names()`**:** View chromosome names.

Usage:

```
ref_genome$chrom_names()
```

Returns: Character vector of chromosome names.

Method `chrom()`**:** View one reference chromosome.

Usage:

```
ref_genome$chrom(chrom_ind)
```

Arguments:

`chrom_ind` Index for the focal chromosome.

Returns: A single string representing the chosen chromosome's DNA sequence.

Method `gc_prop()`**:** View GC proportion for part of one reference chromosome.

Usage:

```
ref_genome$gc_prop(chrom_ind, start, end)
```

Arguments:

`chrom_ind` Index for the focal chromosome.

`start` Point on the chromosome at which to start the calculation (inclusive).

`end` Point on the chromosome at which to end the calculation (inclusive).

Returns: A double in the range [0,1] representing the proportion of DNA sequence that is either `G` or `C`.

Method `nt_prop()`**:** View nucleotide content for part of one reference chromosome

Usage:

```
ref_genome$nt_prop(nt, chrom_ind, start, end)
```

Arguments:

`nt` Which nucleotide to calculate the proportion that the DNA sequence is made of. Must be one of `T`, `C`, `A`, `G`, or `N`.

`chrom_ind` Index for the focal chromosome.

`start` Point on the chromosome at which to start the calculation (inclusive).

`end` Point on the chromosome at which to end the calculation (inclusive).

Returns: A double in the range [0,1] representing the proportion of DNA sequence that is `nt`.

Method `set_names()`**:** Change chromosome names.

Usage:

```
ref_genome$set_names(new_names)
```

Arguments:

`new_names` Vector of new names to use. This must be the same length as the number of current names.

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 10)
ref$set_names(c("a", "b", "c", "d"))
```

Method `clean_names()`**:** Clean chromosome names, converting " :;=%,\\|/\"\'" to "`_`".

Usage:

```
ref_genome$clean_names()
```

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 10)
ref$set_names(c("a:", "b|", "c;", "d'"))
ref$clean_names()
```

Method `add_chroms()`**:** Add one or more chromosomes.

Usage:

```
ref_genome$add_chroms(new_chroms, new_names = NULL)
```

Arguments:

`new_chroms` Character vector of DNA strings representing new chromosomes.

`new_names` Optional character vector of names for the new chromosomes. It should be the same length as `new_chroms`. If `NULL`, new names will be automatically generated. Defaults to `NULL`.

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 10)
ref$add_chroms("TCAGTCAG")
```

Method `rm_chroms()`**:** Remove one or more chromosomes by name

Usage:

```
ref_genome$rm_chroms(chrom_names)
```

Arguments:

`chrom_names` Vector of the name(s) of the chromosome(s) to remove.

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 10)
ref$set_names(c("a", "b", "c", "d"))
ref$rm_chroms("b")
```

Method `merge_chroms()`**:** Merge chromosomes into one.

Usage:

```
ref_genome$merge_chroms(chrom_names)
```

Arguments:

`chrom_names` Vector of the names of the chromosomes to merge into one. Duplicates are not allowed, and chromosomes are merged in the order they're provided. If this is `NULL`, then all chromosomes are merged after first shuffling their order.

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 10)
ref$merge_chroms(ref$chrom_names()[1:2])
ref$merge_chroms(NULL)
```

Method `filter_chroms()`**:** Filter chromosomes by size or for a proportion of total bases.

Usage:

```
ref_genome$filter_chroms(threshold, method)
```

Arguments:

`threshold` Number used as a threshold. If `method == "size"`, then this is the minimum length of a chromosome that will remain after filtering. If `method == "prop"`, chromosomes are first size-sorted, then the largest `N` chromosomes are retained that allow at least threshold * sum(<all chromosome sizes>) base pairs remaining after filtering.

`method` String indicating which filter method to use: chromosome size (`method = "size"`) or proportion of total bases (`method = "prop"`).

Returns: This `R6` object, invisibly.

Examples:

```
ref <- create_genome(4, 100, 50)
ref$filter_chroms(90, "size")
ref$filter_chroms(0.4, "prop")
```

Method `replace_Ns()`**:** Replace `Ns` in the reference genome.

Usage:

```
ref_genome$replace_Ns(pi_tcag, n_threads = 1, show_progress = FALSE)
```

Arguments:

`pi_tcag` Numeric vector (length 4) indicating the sampling weights for `T`, `C`, `A`, and `G`, respectively, for generating new nucleotides with which to replace the `Ns`.

`n_threads` Optional integer specifying the threads to use. Ignored if the package wasn't compiled with OpenMP. Defaults to `1`.

`show_progress` Optional logical indicating whether to show a progress bar. Defaults to `FALSE`.

Returns: This `R6` object, invisibly.

See Also

`read_fasta` `create_genome`

Examples

```
## ------------------------------------------------
## Method `ref_genome$set_names`
## ------------------------------------------------

ref <- create_genome(4, 10)
ref$set_names(c("a", "b", "c", "d"))


## ------------------------------------------------
## Method `ref_genome$clean_names`
## ------------------------------------------------

ref <- create_genome(4, 10)
ref$set_names(c("a:", "b|", "c;", "d'"))
ref$clean_names()
```

```
## ------------------------------------------------
## Method `ref_genome$add_chroms`
## ------------------------------------------------

ref <- create_genome(4, 10)
ref$add_chroms("TCAGTCAG")

## ------------------------------------------------
## Method `ref_genome$rm_chroms`
## ------------------------------------------------

ref <- create_genome(4, 10)
ref$set_names(c("a", "b", "c", "d"))
ref$rm_chroms("b")


## ------------------------------------------------
## Method `ref_genome$merge_chroms`
## ------------------------------------------------

ref <- create_genome(4, 10)
ref$merge_chroms(ref$chrom_names()[1:2])
ref$merge_chroms(NULL)


## ------------------------------------------------
## Method `ref_genome$filter_chroms`
## ------------------------------------------------

ref <- create_genome(4, 100, 50)
ref$filter_chroms(90, "size")
ref$filter_chroms(0.4, "prop")
```

---

sub_models *Construct necessary information for substitution models.*

Description

For a more detailed explanation, see `vignette("sub-models")`.

Usage

```
sub_JC69(lambda, mu = 1, gamma_shape = NULL, gamma_k = 5, invariant = 0)

sub_K80(alpha, beta, mu = 1, gamma_shape = NULL, gamma_k = 5, invariant = 0)

sub_F81(pi_tcag, mu = 1, gamma_shape = NULL, gamma_k = 5, invariant = 0)

sub_HKY85(
```

```
  pi_tcag,
  alpha,
  beta,
  mu = 1,
  gamma_shape = NULL,
  gamma_k = 5,
  invariant = 0
)

sub_F84(
  pi_tcag,
  beta,
  kappa,
  mu = 1,
  gamma_shape = NULL,
  gamma_k = 5,
  invariant = 0
)

sub_TN93(
  pi_tcag,
  alpha_1,
  alpha_2,
  beta,
  mu = 1,
  gamma_shape = NULL,
  gamma_k = 5,
  invariant = 0
)

sub_GTR(
  pi_tcag,
  abcdef,
  mu = 1,
  gamma_shape = NULL,
  gamma_k = 5,
  invariant = 0
)

sub_UNREST(Q, mu = 1, gamma_shape = NULL, gamma_k = 5, invariant = 0)
```

Arguments

lambda	Substitution rate for all possible substitutions.
mu	Total rate of substitutions. Defaults to 1, which makes branch lengths in units of substitutions per site. Passing NULL results in no scaling.
gamma_shape	Numeric shape parameter for discrete Gamma distribution used for among-site variability. Values must be greater than zero. If this parameter is NULL, among-

site variability is not included. Defaults to NULL.

gamma_k — The number of categories to split the discrete Gamma distribution into. Values must be an integer in the range [2,255]. This argument is ignored if gamma_shape is NA. Defaults to 5.

invariant — Proportion of sites that are invariant. Values must be in the range [0,1). Defaults to 0.

alpha — Substitution rate for transitions.

beta — Substitution rate for transversions.

pi_tcag — Vector of length 4 indicating the equilibrium distributions of T, C, A, and G respectively. Values must be >= 0, and they are forced to sum to 1.

kappa — The transition/transversion rate ratio.

alpha_1 — Substitution rate for T <-> C transition.

alpha_2 — Substitution rate for A <-> G transition.

abcdef — A vector of length 6 that contains the off-diagonal elements for the substitution rate matrix. See vignette("sub-models") for how the values are ordered in the matrix.

Q — Matrix of substitution rates for "T", "C", "A", and "G", respectively. Item Q[i,j] is the rate of substitution from nucleotide i to nucleotide j. Do not include indel rates here! Values on the diagonal are calculated inside the function so are ignored.

Value

A sub_info object, which is an R6 class that wraps the info needed for the create_haplotypes function. It does not allow the user to directly manipulate the info inside, as that should be done using the sub_models functions. You can use the following methods from the class to view information:

Q() View a list of substitution rate matrices, one for each Gamma category.

pi_tcag() View the equilibrium nucleotide frequencies.

gammas() View the discrete Gamma-class values.

invariant() View the proportion of invariant sites.

model() View the substitution model.

U() View list of the U matrices (one matrix per Gamma category) used for calculating transition-probability matrices. This is empty for UNREST models.

Ui() View list of the U^-1 matrices (one matrix per Gamma category) used for calculating transition-probability matrices. This is empty for UNREST models.

L() View list of the lambda vectors (one vector per Gamma category) used for calculating transition-probability matrices. This is empty for UNREST models.

Functions

- sub_JC69: JC69 model.
- sub_K80: K80 model.
- sub_F81: F81 model.
- sub_HKY85: HKY85 model.
- sub_F84: F84 model.
- sub_TN93: TN93 model.
- sub_GTR: GTR model.
- sub_UNREST: UNREST model.

See Also

create_haplotypes

Examples

```
# Same substitution rate for all types:
obj_JC69 <- sub_JC69(lambda = 0.1)

# Transitions 2x more likely than transversions:
obj_K80 <- sub_K80(alpha = 0.2, beta = 0.1)

# Incorporating equilibrium frequencies:
obj_HKY85 <- sub_HKY85(pi_tcag = c(0.1, 0.2, 0.3, 0.4),
                       alpha = 0.2, beta = 0.1)

# 10-category Gamma distribution for among-site variability:
obj_K80 <- sub_K80(alpha = 0.2, beta = 0.1,
                   gamma_shape = 1, gamma_k = 10)

# Invariant sites:
obj_K80 <- sub_K80(alpha = 0.2, beta = 0.1,
                   invariant = 0.25)
```

---

write_fasta *Write a* ref_genome *or* haplotypes *object to a FASTA file.*

Description

This file produces 1 FASTA file for a ref_genome object and one file for each haplotype in a haplotypes object.

Usage

```
write_fasta(
  obj,
  out_prefix,
  compress = FALSE,
  comp_method = "bgzip",
  text_width = 80,
  show_progress = FALSE,
  n_threads = 1,
  overwrite = FALSE
)
```

Arguments

obj	A ref_genome or haplotypes object.
out_prefix	Prefix for the output file.
compress	Logical specifying whether or not to compress output file, or an integer specifying the level of compression, from 1 to 9. If TRUE, a compression level of 6 is used. Defaults to FALSE.
comp_method	Character specifying which type of compression to use if any is desired. Options include "gzip" and "bgzip". This is ignored if compress is FALSE. Defaults to "bgzip".
text_width	The number of characters per line in the output fasta file. Defaults to 80.
show_progress	Logical for whether to show a progress bar. Defaults to FALSE.
n_threads	Number of threads to use if writing from a haplotypes object. Threads are split among haplotypes, so it's not useful to provide more threads than haplotypes. This argument is ignored if obj is a ref_genome object, or if OpenMP is not enabled. Defaults to 1.
overwrite	Logical for whether to overwrite existing file(s) of the same name, if they exist. Defaults to FALSE.

Value

NULL

---

write_vcf *Write haplotype info from a* haplotypes *object to a VCF file.*

Description

Compression in this function always uses "bgzip" for compatibility with "tabix".

Usage

```
write_vcf(
  haps,
  out_prefix,
  compress = FALSE,
  sample_matrix = NULL,
  show_progress = FALSE,
  overwrite = FALSE
)
```

Arguments

haps	A haplotypes object.
out_prefix	Prefix for the output file.
compress	Logical specifying whether or not to compress output file, or an integer specifying the level of compression, from 1 to 9. If TRUE, a compression level of 6 is used. Defaults to FALSE.
sample_matrix	Matrix to specify how haplotypes are grouped into samples if samples are not haploid. There should be one row for each sample, and each row should contain indices or names for the haplotypes present in that sample. Indices/names for haplotypes cannot be repeated. Instead of repeating indices here, you should use the dup_haps method of the haplotypes class to duplicate the necessary haplotype(s). The number of columns indicates the ploidy level: 2 columns for diploid, 3 for triploid, 4 for tetraploid, and so on; there is no limit to the ploidy level. If this argument is NULL, it's assumed that each haplotype is its own separate sample. Defaults to NULL.
show_progress	Logical for whether to show a progress bar. Defaults to FALSE.
overwrite	Logical for whether to overwrite existing file(s) of the same name, if they exist. Defaults to FALSE.

Value

NULL

Index