

# Package ‘GenomeAdmixR’

June 20, 2025

**Type** Package

**Title** Simulate Admixture of Genomes

**Version** 2.1.12

**Description** Individual-based simulations forward in time,  
simulating how patterns in ancestry along the genome change after  
admixture. Full description can be found in Janzen (2021)  
<[doi:10.1111/2041-210X.13612](https://doi.org/10.1111/2041-210X.13612)>.

**License** GPL (>= 2)

**URL** <https://github.com/thijsjanzen/GenomeAdmixR>

**BugReports** <https://github.com/thijsjanzen/GenomeAdmixR/issues>

**Imports** ggplot2, ggridges, hierfstat, Rcpp, RcppParallel, rlang,  
stringr, tibble, vcfR

**Suggests** dplyr, junctions, knitr, magrittr, rmarkdown, testit,  
testthat, pbapply

**LinkingTo** Rcpp, RcppArmadillo, RcppParallel

**VignetteBuilder** knitr

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**Depends** R (>= 2.10)

**NeedsCompilation** yes

**Author** Thijs Janzen [aut, cre],  
Fernando Diaz G. [ctb],  
Richèl J.C. Bilderbeek [ctb]

**Maintainer** Thijs Janzen <[thijsjanzen@gmail.com](mailto:thijsjanzen@gmail.com)>

**Repository** CRAN

**Date/Publication** 2025-06-20 10:40:02 UTC

## Contents

GenomeAdmixR-package . . . . .	3
ancestry_module . . . . .	5
calculate_allele_frequencies . . . . .	6
calculate_average_ld . . . . .	7
calculate_dist_junctions . . . . .	7
calculate_fst . . . . .	8
calculate_heterozygosity . . . . .	9
calculate_ld . . . . .	9
calculate_marker_frequency . . . . .	10
combine_input_data . . . . .	11
create_artificial_genomeadmixr_data . . . . .	12
create_iso_female . . . . .	12
dgrp2.3R.5k.data . . . . .	13
iso_female_ancestry . . . . .	14
iso_female_sequence . . . . .	15
load_population . . . . .	16
migration_settings . . . . .	17
plink_to_genomeadmixr_data . . . . .	18
plot.individual . . . . .	18
plot_chromosome . . . . .	19
plot_difference_frequencies . . . . .	20
plot_dist_junctions . . . . .	21
plot_frequencies . . . . .	21
plot_joyplot_frequencies . . . . .	22
plot_over_time . . . . .	23
plot_start_end . . . . .	24
print.genomeadmixr_data . . . . .	25
print.individual . . . . .	26
print.population . . . . .	26
read_input_data . . . . .	27
save_population . . . . .	28
sequence_module . . . . .	28
simulate_admixture . . . . .	29
simulate_ancestry . . . . .	31
simulate_ancestry_migration . . . . .	32
simulate_sequence . . . . .	35
simulate_sequence_migration . . . . .	36
simulation_data_to_genomeadmixr_data . . . . .	39
vcfR_to_genomeadmixr_data . . . . .	39
write_as_plink . . . . .	40

## Index

41

## Description

Individual-based simulations forward in time, simulating how patterns in ancestry along the genome change after admixture. The simulation assumes Wright-Fisher dynamics, e.g. random mating and non-overlapping generations. In the simulation, instead of specific alleles, local ancestry is tracked, thus assuming that local molecular data can always be uniquely traced back to one of the founding individuals (populations). The package provides functionality to perform such simulations, but also to perform post-hoc statistical analyses and to visualize the obtained results.

Version 2.1.10 - Fixed memory bug, improved documentation

Version 2.1.9 - updated `tbb::task_scheduler_init` to `tbb::global_control`

Version 2.1.7 - Improve documentation

Version 2.1.6 - check classes with `inherits`

Version 2.1.5 - Removed debugging output

Version 2.1.4 - Only output when `verbose = TRUE`

Version 2.1.3 - Changed DOI link in description

Version 2.1.2 - Improved testing

Version 2.1.1 - Removed GNU make dependency

Version 2.1 - Removed error in `calculate_allele_frequency`

Version 2.0.1 - Moved migration outside the modules

Version 2.0 - Added `ancestry_module` and `sequence_module` to distinguish between implementations of the model

Version 1.2 - Added example sequencing data

Version 1.2 - Added the option to load sequence data for admixing

Version 1.1 - Fixed a minor bug with `plot_joyplot_frequencies`

Version 1.1 - Improved tests

Version 1.1 - Improved recombination code (again)

Version 1.0 - Release associated with bioRxiv submission, to be found here: <https://doi.org/10.1101/2020.10.19.343491>

Version 0.66 - Improved recombination code, about twice as fast

Version 0.65 - Added testing and added logo

Version 0.64 - Reduced cyclomatic complexity

Version 0.63 - Updated random number generation

Version 0.62 - Updated to Roxygen

Version 0.61 - Added `plot_over_time`

Version 0.60 - Added admixture with migration

Version 0.59 - Updated frequency code under the hood

Version 0.58 - Renamed to GenomeAdmixR

Version 0.58 - Collapsed and improved many functions

Version 0.57 - Added function to generate admixed individuals

Version 0.56 - Added starting frequencies to `'simulate_admixture'`

Version 0.55 - Extended `'calculate_marker_frequency'` to handle a vector of locations

Version 0.55 - Increased accuracy of choosing a random position for recombination, this should prevent the rare bug fixed in version 0.54

Version 0.54 - Fixed a MAJOR bug regarding recombination: in rare cases, a crossover position

could be picked on an existing junction, due to the limited number of digits in `uniform()`

Version 0.54 - Improved `plot_difference_frequencies` to handle modified input

Version 0.53 - Added `multiplicative_selection`

Version 0.52 - Added `plot_difference_frequencies`

Version 0.51 - Added `tajima's d` calculation

Version 0.50 - Added `simulated_admixture` until

Version 0.49 - Added `'simulate'` to `cpp`

Version 0.48 - Added a general `'simulate'` function

Version 0.47 - Changed the effect of migration

Version 0.46 - Added `joyplot` & `increase_ancestor`

Version 0.45 - Removed `create_two_populations`

Version 0.44 - Added tracking regions

Version 0.43 - Fixed bugs in `select_population`

Version 0.42 - Added initial and final frequency tables

Version 0.41 - Added multiple marker support

Version 0.40 - Collapsed selection functions

Version 0.39 - Added support for non-additive selection

Version 0.38 - Added track frequencies

Version 0.37 - Removed selection on regions

Version 0.36 - Added `progress_bar` option

Version 0.35 - Added `calculate_marker_frequency`

Version 0.34 - Added `selection_markers`

Version 0.33 - Fixed bugs in selection

Version 0.32 - Moved `Fish.h` code to `Fish.cpp`

Version 0.31 - Changed random number generator to R based

Version 0.30 - Added `Recombination = 1` code

Version 0.29 - Changed internal junction representation: removed `.left`

Version 0.28 - Reverted to Agner Fog Random number generation

Version 0.27 - Speed up return types

Version 0.26 - Added class verification code

Version 0.25 - Squashed plotting bug

Version 0.24 - Removed `Output.cpp`

Version 0.23 - Removed `number_of_founders` from `calc_allele_spectrum`

Version 0.22 - Added save and load functions

Version 0.21 - Changed random-seed management

Version 0.20 - Removed superfluous code

Version 0.19 - Removed `number_of_founders` from `Fst` and `LD` code

Version 0.18 - Start of tracking changes

## Author(s)

Thijs Janzen Maintainer: (thijsjanzen@gmail.com)

## References

Janzen T, Diaz F. Individual-based simulations of genome evolution with ancestry: The GenomeAdmixR R package. *Methods Ecol Evol.* 2021; 12: 1346–1357. <https://doi.org/10.1111/2041-210X.13612>

---

ancestry_module	<i>Creates a module to start simulations tracking local ancestry</i>
-----------------	--

---

## Description

Module to perform simulations based on local ancestry

## Usage

```
ancestry_module(  
  input_population = NA,  
  number_of_founders = 2,  
  initial_frequencies = NA,  
  morgan = 1,  
  markers = NA,  
  track_junctions = FALSE  
)
```

## Arguments

input_population	Potential earlier simulated population used as starting point for the simulation. If not provided by the user, the simulation starts from scratch.
number_of_founders	Number of unique ancestors / ancestries to be tracked in the simulation
initial_frequencies	A vector describing the initial frequency of each ancestor / ancestry. By default, equal frequencies are assumed. If a vector not summing to 1 is provided, the vector is normalized.
morgan	Length of the genomic stretch simulated, expressed in Morgan (e.g. the number of crossovers during meiosis)
markers	A vector of locations of markers, with the location in Morgan. Ancestry at these marker positions is tracked for every generation.
track_junctions	Tracks the average number of junctions over time if TRUE

## Value

list with type = "Ancestry". Can be used in simulate\_admixture.

---

 calculate\_allele\_frequencies

*Calculate allele frequencies*


---

## Description

Calculate for a number of regularly spaced markers the relative frequency of each ancestor in the population.

## Usage

```
calculate_allele_frequencies(
  source_pop,
  locations = seq(0, 1, length.out = 100),
  progress_bar = FALSE
)
```

## Arguments

source_pop	Population for which to estimate allele frequencies
locations	A vector indicating the locations (in Morgan) where to calculate the allele frequencies.
progress_bar	Displays a progress_bar if TRUE. Default value is TRUE

## Details

Markers are equidistantly spaced, with a distance of step\_size in between them.

## Value

A tibble containing the allele frequencies

## Examples

```
number_founders = 20
wildpop = simulate_admixture(
  module = ancestry_module(number_of_founders = 20, morgan = 1),
  pop_size = 1000,
  total_runtime = 10,
  num_threads = 1)

freq_output <- calculate_allele_frequencies(wildpop,
                                             progress_bar = TRUE)

require(ggplot2)
ggplot(freq_output, aes(x=location, y = frequency,
                       col = as.factor(ancestor))) +
  geom_line()
```

---

calculate\_average\_ld    *Calculates the ld between two alleles*

---

**Description**

calculate the average ld between two loci

**Usage**

calculate\_average\_ld(alleles\_pos\_1, alleles\_pos\_2)

**Arguments**

alleles\_pos\_1    alleles at locus 1

alleles\_pos\_2    alleles at locus 2

**Value**

a list with two entries: LD and r\_squared

---

calculate\_dist\_junctions  
                          *collect the full distribution of junctions in the population*

---

**Description**

calculates the distribution of junctions across the population

**Usage**

calculate\_dist\_junctions(pop)

**Arguments**

pop                    object of the class 'population'

**Value**

vector with two entries per individual, each indicating the number of junctions in the respective chromosomes





```
FST <- calculate_fst(pop1 = two_populations$population_1,
  pop2 = two_populations$population_2,
  sampled_individuals = 10,
  number_of_markers = 100,
  random_markers = TRUE)
```

---

calculate\_heterozygosity  
*Calculate heterozygosity*

---

### Description

Calculate the average population level heterozygosity

### Usage

```
calculate_heterozygosity(source_pop, locations, progress_bar = FALSE)
```

### Arguments

source_pop	Population for which to estimate allele frequencies, or a list of individuals for which to calculate average heterozygosity
locations	A vector indicating the locations (in Morgan) of markers for which to calculate the heterozygosity
progress_bar	Displays a progress_bar if TRUE. Default value is TRUE

### Value

A tibble containing the heterozygosities

---

calculate_ld	<i>Calculate linkage disequilibrium statistics This function calculates two matrices, once containing all pairwise linkage disequilibrium (ld) values, and one matrix containing all pairwise r statistics</i>
--------------	--

---

### Description

Calculate linkage disequilibrium statistics This function calculates two matrices, once containing all pairwise linkage disequilibrium (ld) values, and one matrix containing all pairwise r statistics

### Usage

```
calculate_ld(pop, sampled_individuals = 10, markers = NA, verbose = FALSE)
```

**Arguments**

pop	focal population
sampled_individuals	Number of individuals randomly sampled to calculate the LD matrices
markers	vector of markers. If a single number is used, that number of markers is randomly placed along the genome.
verbose	display verbose output, default is FALSE.

**Value**

An object containing two items:

ld_matrix	Pairwise ld statistics for all markers
rsq_matrix	Pairwise rsq statistics for all markers

**Examples**

```
wildpop = simulate_admixture(
  module = ancestry_module(number_of_founders = 10, morgan = 1),
  pop_size = 1000,
  total_runtime = 10)

ld_results <- calculate_ld(pop = wildpop,
  markers = 10)

plot(ld_results$ld_matrix~ld_results$dist_matrix,
  pch = 16,
  xlab="Distance between markers",
  ylab = "Linkage Disequilibrium")
```

---

calculate\_marker\_frequency

*Calculate allele frequencies at a specific marker location*

---

**Description**

Calculate the relative frequency of each ancestor in the population at a specific marker location

**Usage**

```
calculate_marker_frequency(pop, location)
```

**Arguments**

pop	Population for which to estimate allele frequencies at the given marker
location	A vector or scalar of location(s) along the chromosome for which allele frequencies are to be calculated. Locations are in Morgan.

**Value**

A tibble containing the frequency of each present ancestor at the provided location. Ancestors with frequency = 0 are dropped out of the table. The tibble contains three columns: location, ancestor and frequency.

**Examples**

```
wildpop = simulate_admixture(
  module = ancestry_module(number_of_founders = 20, morgan = 1),
  pop_size = 1000,
  total_runtime = 10)

avg_frequencies <- calculate_marker_frequency(pop = wildpop,
                                             location = 0.5)

frequencies <-
  calculate_marker_frequency(pop = wildpop,
                           location = seq(0.4, 0.5, by = 0.01))
require(ggplot2)
ggplot(frequencies, aes(x = location, y = frequency, col = ancestor)) +
  geom_step()
```

---

combine_input_data	<i>combine sequence data that was previously read from file into a population</i>
--------------------	---

---

**Description**

Create data in a format that can be used by GenomeAdmixR, entries are sampled randomly from each input data set, with replacement. Probability of sampling from each input data set is driven by the input frequencies, and total number of individuals sampled is driven by pop\_size.

**Usage**

```
combine_input_data(input_data_list, frequencies = NA, pop_size)
```

**Arguments**

input_data_list	list where each entry is the result of create_input_data
frequencies	frequency of each entry in the list in the starting population
pop_size	intended population size

**Value**

the input data entries are combined to one single population that can be used to seed simulate\_admixture\_data. Output is identical to create\_input\_data

---

```
create_artificial_genomeadmixmap_data
```

*function to generate artificial genomeadmixmap\_data*

---

### Description

function to generate artificial genomeadmixmap\_data

### Usage

```
create_artificial_genomeadmixmap_data(
  number_of_individuals,
  marker_locations,
  used_nucleotides = 1:4,
  nucleotide_frequencies = NA
)
```

### Arguments

`number_of_individuals`  
number of individuals

`marker_locations`  
location of markers, either in bp or Morgan

`used_nucleotides`  
subset or full set of [1/2/3/4] (reflecting a/c/t/g)

`nucleotide_frequencies`  
frequencies of the used nucleotides, if not provided, equal frequencies are assumed.

### Value

genomeadmixmap\_data object ready for simulate\_admixture\_data

---

```
create_iso_female
```

*function to simulate creation of an isofemale line*

---

### Description

create\_isofemale simulates the creation of an isofemale line through extreme inbreeding.

**Usage**

```
create_iso_female(  
  module = ancestry_module(),  
  n = 1,  
  inbreeding_pop_size = 100,  
  run_time = 2000,  
  num_threads = 1,  
  verbose = FALSE  
)
```

**Arguments**

- module            Source population from which isofemales are generated
- n                Number of isofemales to be generated
- inbreeding\_pop\_size      Population size of the population used to generate homozygous individuals
- run\_time        Maximum runtime used for inbreeding
- num\_threads     number of threads. Default is 1. Set to -1 to use all available threads
- verbose        Displays verbose output if TRUE. Default value is FALSE

**Details**

To create an isofemale, two individuals are randomly picked from the source population. Using these two individuals, a new population is seeded, of size `inbreeding_pop_size`. Then, this population is allowed to inbreed until either `run_time` is reached, or until all individuals are homozygous and genetically identical, whatever happens first.

**Value**

A list of length `n`, where each entry is a fully homozygous isofemale.

---

dgrp2.3R.5k.data	<i>A subset of sequencing data from the Drosophila Genetics Reference Panel</i>
------------------	---

---

**Description**

This data set contains sequences from the 3R chromosome. Included are 4603 SNPs with at least 0.05 minor allele frequency, sequenced across 410 isofemale lines. Sequences were downloaded from <<http://dgrp2.gnets.ncsu.edu/data.html>>.

**Usage**

```
data("dgrp2.3R.5k.data")
```

**Format**

genomeadmixr\_data object

**References**

Mackay, T., Richards, S., Stone, E. et al. The *Drosophila melanogaster* Genetic Reference Panel. *Nature* 482, 173–178 (2012). <<https://doi.org/10.1038/nature10811>>

**Examples**

```
data("dgrp2.3R.5k.data")
simulate_admixture(
  module = sequence_module(molecular_data = dgrp2.3R.5k.data),
  pop_size = 100,
  total_runtime = 10)
```

---

iso\_female\_ancestry      *Create isofemale*

---

**Description**

Creates isofemale individuals, given a population

**Usage**

```
iso_female_ancestry(
  source_pop = NA,
  n = 1,
  inbreeding_pop_size = 100,
  run_time = 2000,
  morgan = 1,
  num_threads = 1,
  verbose = FALSE
)
```

**Arguments**

source_pop	Source population from which isofemales are generated
n	Number of isofemales to be generated
inbreeding_pop_size	Population size of the population used to generate homozygous individuals
run_time	Maximum runtime used for inbreeding
morgan	Size of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
num_threads	number of threads. Default is 1. Set to -1 to use all available threads
verbose	Displays verbose output if TRUE. Default value is FALSE

**Details**

To create an isofemale, two individuals are randomly picked from the source population. Using these two individuals, a new population is seeded, of size `inbreeding_pop_size`. Then, this population is allowed to inbreed until either `run_time` is reached, or until all individuals are homozygous and genetically identical, whatever happens first.

**Value**

A list of length `n`, where each entry is a fully homozygous isofemale.

---

<code>iso_female_sequence</code>	<i>Create isofemale</i>
----------------------------------	-------------------------

---

**Description**

Creates isofemale individuals, given a population

**Usage**

```
iso_female_sequence(
  input_data = NA,
  n = 1,
  inbreeding_pop_size = 100,
  run_time = 2000,
  morgan = 1,
  recombination_rate = NA,
  num_threads = 1,
  verbose = FALSE
)
```

**Arguments**

<code>input_data</code>	Source population from which isofemales are generated
<code>n</code>	Number of isofemales to be generated
<code>inbreeding_pop_size</code>	Population size of the population used to generate homozygous individuals
<code>run_time</code>	Maximum runtime used for inbreeding
<code>morgan</code>	Size of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
<code>recombination_rate</code>	rate in cM / Mbp, used to map recombination to the markers. If the <code>recombination_rate</code> is not set, the value for Morgan is used, assuming that the markers included span an entire chromosome.
<code>num_threads</code>	number of threads. Default is 1. Set to -1 to use all available threads
<code>verbose</code>	Displays verbose output if TRUE. Default value is FALSE

**Details**

To create an isofemale, two individuals are randomly picked from the source population. Using these two individuals, a new population is seeded, of size `inbreeding_pop_size`. Then, this population is allowed to inbreed until either `run_time` is reached, or until all individuals are homozygous and genetically identical, whatever happens first.

**Value**

A list of length `n`, where each entry is a fully homozygous isofemale.

---

load_population	<i>Load a population from file</i>
-----------------	------------------------------------

---

**Description**

Loads a population that has previously been written to file.

**Usage**

```
load_population(file_name)
```

**Arguments**

file_name	Name of the file to save the population
-----------	---

**Details**

This function is a wrapper for `readRDS`.

**Value**

A population object

**See Also**

[save\\_population](#)



---

migration_settings	<i>Function to manage settings associated with migration</i>
--------------------	--

---

## Description

creates a list with settings associated with migration.

## Usage

```
migration_settings(
  migration_rate = NA,
  stop_at_critical_fst = FALSE,
  critical_fst = NA,
  population_size = c(100, 100),
  initial_frequencies = list(c(1, 0), c(0, 1)),
  generations_between_update = 10,
  sampled_individuals = 10,
  number_of_markers = 100,
  random_markers = TRUE
)
```

## Arguments

migration_rate	Rate of migration between the two populations. Migration is implemented such that with probability $m$ (migration rate) one of the two parents of a new offspring is from the other population, with probability $1-m$ both parents are of the focal population.
stop_at_critical_fst	option to stop at a critical FST value , default is FALSE
critical_fst	the critical fst value to stop, if stop_simulation_at_critical_fst is TRUE
population_size	vector of population sizes, one size for each population
initial_frequencies	A list describing the initial frequency of each ancestor in each population. Each entry in the list contains a vector with the frequencies for all ancestor. The length of the vector indicates the number of unique ancestors. If a vector not summing to 1 is provided, the vector is normalized.
generations_between_update	The number of generations after which the simulation has to check again whether the critical Fst value is exceeded
sampled_individuals	Number of individuals to be sampled at random from the population to estimate Fst
number_of_markers	Number of markers to be used to estimate Fst
random_markers	Are the markers to estimate Fst randomly distributed, or regularly distributed? Default is TRUE.

**Value**

list with migration associated settings. To be used to pass on migration settings to simulate\_admixture.

---

```
plink_to_genomeadmixr_data
      function to convert plink style (ped/map) data to genome_admixr_data
```

---

**Description**

function to convert plink style (ped/map) data to genome\_admixr\_data

**Usage**

```
plink_to_genomeadmixr_data(
  ped_data,
  map_data,
  chosen_chromosome,
  verbose = FALSE
)
```

**Arguments**

ped_data	result of read.table(ped_file, header = F)
map_data	result of read.table(map_file, header = F)
chosen_chromosome	chromosome of choice
verbose	verbose output

**Value**

genomeadmixr\_data object ready for simulate\_admixture\_data

---

```
plot.individual      plot the genome of an individual
```

---

**Description**

visualise ancestry blocks on both chromosomes

**Usage**

```
## S3 method for class 'individual'
plot(x, cols = NA, ...)
```

**Arguments**

x	object of type individual
cols	colors for the different ancestors
...	other arguments

**Value**

No return value

---

plot_chromosome	<i>plots a chromosome</i>
-----------------	---------------------------

---

**Description**

This function plots a chromosome in the range [xmin, xmax]. Colors indicate different ancestry.

**Usage**

```
plot_chromosome(chrom, xmin = 0, xmax = 1)
```

**Arguments**

chrom	object of type chromosome, typically a table with two columns. The first column indicates the start of an ancestry block (location in Morgan), the second column indicates the ancestry type.
xmin	minimum value of the range, default = 0.
xmax	maximum value of the range, default = 1.

**Value**

No return value

**Examples**

```
wildpop = simulate_admixture(
  module = ancestry_module(number_of_founders = 10, morgan = 1),
  pop_size = 1000,
  total_runtime = 10)

isofemale <- create_iso_female(
  module = ancestry_module(input_population = wildpop,
                           morgan = 1),
  n = 1,
  inbreeding_pop_size = 100,
  run_time = 10)

plot_chromosome(chrom = isofemale[[1]]$chromosome1)
```

```
# and a detail of the chromosome:
plot_chromosome(chrom = isofemale[[1]]$chromosome1,
                xmin = 0.4,
                xmax = 0.6)
```

---

plot\_difference\_frequencies

*Plot the change in frequency between the start and end of a simulation*

---

## Description

This function plots the change in frequency of one or multiple ancestors after performing a simulation.

## Usage

```
plot_difference_frequencies(
  results,
  picked_ancestor = "ALL",
  picked_population = 1
)
```

## Arguments

results	An object which is the result of <code>simulate_admixture</code> being a list with four properties: population, frequencies, initial_frequencies and final frequencies
picked_ancestor	Default is "ALL", where different colors indicate different ancestors. Alternatively, for clarity, the user can specify a specific ancestral allele, and only that allele is plotted
picked_population	If multiple populations were simulated (in the case of <code>simulate_admixture_migration</code> ), which population should be plotted? Default is <code>population_1</code> .

## Value

a ggplot2 object

## Examples

```
s <- 0.1
select_matrix <- matrix(nrow = 1, ncol = 5)
select_matrix[1, ] <- c(0.25, 1.0, 1 + 0.5 * s, 1 + s, 0)

markers <- seq(from = 0.2, to = 0.3, length.out = 100)

selected_pop <- simulate_admixture(
  module = ancestry_module(number_of_founders = 10,
```

```

                                morgan = 1,
                                markers = markers),
                                pop_size = 1000,
                                total_runtime = 11,
                                select_matrix = select_matrix)
require(ggplot2)
plot_difference_frequencies(results = selected_pop,
                           picked_ancestor = "ALL")

```

---

plot\_dist\_junctions     *plot the distribution of junctions*

---

### Description

plots the distribution of junctions in the population using base R

### Usage

```
plot_dist_junctions(pop)
```

### Arguments

pop                      of the class 'population'

### Value

No return value

---

plot\_frequencies         *Plot the frequencies of all ancestors along the genome.*

---

### Description

This function plots the frequency of all ancestors after performing a simulation.

### Usage

```

plot_frequencies(
  result,
  locations = seq(0, 1, length.out = 100),
  progress_bar = FALSE
)

```

**Arguments**

result	An object which is the result of select_population or create_population_selection, being a list with four properties: population, frequencies, initial_frequencies and final frequencies
locations	A vector indicating the locations (in Morgan) where to calculate the allele frequencies.
progress_bar	Displays a progress_bar if TRUE. Default value is FALSE

**Value**

a ggplot2 object

**Examples**

```
pop <- simulate_admixture(
  module = ancestry_module(number_of_founders = 4),
  pop_size = 1000,
  total_runtime = 11)
require(ggplot2)
plot_frequencies(result = pop)
```

---

plot\_joyplot\_frequencies

*make a joy plot of the distribution of allele frequencies within a region*

---

**Description**

This function plots the distribution of allele frequencies within a region over time, making use of a 'joyplot'

**Usage**

```
plot_joyplot_frequencies(
  frequencies,
  time_points,
  picked_ancestor = "ALL",
  picked_population = 1
)
```

**Arguments**

frequencies	A tibble containing four columns: time, location, ancestor, frequency. Typically one of the items returned by create_population_selection or select_population when the user specifies track_frequency.
time_points	A sequence of time points for which the user wants to create the joyplot

picked\_ancestor

Default is "ALL", where different colors indicate different ancestors. Alternatively, for clarity, the user can specify a specific ancestral allele, and only that allele is plotted

picked\_population

If multiple populations were simulated (in the case of simulate\_admixture\_migration), which population should be plotted? Default is population\_1.

## Value

a ggplot object

## Examples

```
s <- 0.01
select_matrix <- matrix(nrow = 1, ncol = 5)
select_matrix[1, ] <- c(0.25, 1.0, 1 + 0.5 * s, 1 + s, 0)

markers <- seq(from = 0.2, to = 0.3, length.out = 100)

selected_pop <- simulate_admixture(
  module = ancestry_module(number_of_founders = 10,
                           morgan = 1,
                           markers = markers),
  pop_size = 1000,
  total_runtime = 11,
  select_matrix = select_matrix)

require(ggplot2)
plot_joyplot_frequencies(frequencies = selected_pop$frequencies,
  time_points = 0:11,
  picked_ancestor = "ALL")

# joyplot frequencies returns a ggplot object, so we can
# add extra elements:
plot_joyplot_frequencies(frequencies = selected_pop$frequencies,
  time_points = 0:11,
  picked_ancestor = "ALL") +
  ggplot2::xlab("Location") +
  ggplot2::ylab("Generations")
```

---

plot\_over\_time

*Plot the frequencies of all ancestors over time*

---

## Description

This function plots the frequency of all ancestors over time at a specific location on the chromosome, after performing a simulation.

**Usage**

```
plot_over_time(frequencies, focal_location)
```

**Arguments**

**frequencies** A tibble containing four columns: time, location, ancestor, frequency. A fifth column population can be included if the tibble is the result of `simulate_admixture_migration`.

**focal\_location** Location (in Morgan) where to plot the allele frequencies.

**Value**

a ggplot2 object

**Examples**

```
pop <- simulate_admixture(
  module = ancestry_module(number_of_founders = 10,
                           markers = 0.5),
  pop_size = 1000,
  total_runtime = 11)
require(ggplot2)
plot_over_time(frequencies = pop$frequencies,
               focal_location = 0.5)
```

---

plot_start_end	<i>Plot both the starting frequencies and the final frequencies in one plot</i>
----------------	---

---

**Description**

This function plots the distribution of both the starting and the final frequencies in one plot

**Usage**

```
plot_start_end(results, picked_ancestor = "ALL", picked_population = 1)
```

**Arguments**

**results** An object which is the result of `simulate_admixture`, being a list with four properties: population, frequencies, initial\_frequencies and final frequencies

**picked\_ancestor** Default is "ALL", where different colors indicate different ancestors. Alternatively, for clarity, the user can specify a specific ancestral allele, and only that allele is plotted

**picked\_population** If multiple populations were simulated (in the case of `simulate_admixture_migration`), which population should be plotted? Default is population\_1.



**Value**

a ggplot object

**Examples**

```
markers <- seq(from = 0.2, to = 0.3, length.out = 100)

pop <- simulate_admixture(
  module = ancestry_module(number_of_founders = 3,
                           morgan = 1,
                           markers = markers),
  pop_size = 1000,
  total_runtime = 11)
require(ggplot2)
plot_start_end(pop,
  picked_ancestor = "ALL")
plot_start_end(pop,
  picked_ancestor = 1)
```

---

```
print.genomeadmixr_data
```

*print an individual to the console*

---

**Description**

prints an object of class genomeadmixr\_data to the console

**Usage**

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

**Arguments**

x	individual
...	other arguments

**Value**

No return value

print.individual	<i>print an individual to the console</i>
------------------	---

---

**Description**

prints an object of class individual to the console

**Usage**

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

**Arguments**

x	individual
...	other arguments

**Value**

No return value

---

print.population	<i>print a population object</i>
------------------	----------------------------------

---

**Description**

prints the contents of a population nicely

**Usage**

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

**Arguments**

x	input population
...	other arguments

**Value**

No return value

---

read_input_data	<i>read sequence data from file to be used in simulation</i>
-----------------	--

---

## Description

Create data in a format that can be used by GenomeAdmixR

## Usage

```
read_input_data(  
  file_names,  
  type,  
  chosen_chromosome,  
  number_of_snps = NA,  
  random_snps = TRUE,  
  verbose = FALSE  
)
```

## Arguments

file_names	names of input files
type	type of data, options are 'ped' and 'vcf'
chosen_chromosome	GenomeAdmixR simulates only a single chromosome.
number_of_snps	number of snps to be loaded from file, default is to load all snps
random_snps	if a subset of all snps has to be taken, should these be sampled sequentially (e.g. the first 100 snps) or randomly (100 randomly sampled snps) (examples are for 'number_of_snps' = 100).
verbose	give verbose output

## Value

list with two properties: `genomes` a matrix with the sequence translated to numerics, such that [actg] corresponds to [1234], and missing data is represented with "-". Rows in the matrix correspond to chromosomes, and columns represent bases. Two consecutive rows represent an individual, such that rows 1-2 are individual, rows 3-4 are one individual etc. `markers` corresponds to the locations of the markers (in bp) on the chosen chromosome.

---

save_population	<i>Save a population to file</i>
-----------------	----------------------------------

---

**Description**

Saves a population to file for later use

**Usage**

```
save_population(population, file_name, compression = TRUE)
```

**Arguments**

population	Object of class population
file_name	Name of the file to save the population
compression	By default, the population is compressed to reduce file size. See for more information saveRDS

**Details**

This function functions as a wrapper for the base function saveRDS.

**Value**

No return value

---

sequence_module	<i>create sequence module</i>
-----------------	-------------------------------

---

**Description**

creates a sequence module, which contains all relevant information in order to perform a simulation based on sequence data.

**Usage**

```
sequence_module(  
  molecular_data = NA,  
  initial_frequencies = NA,  
  morgan = 1,  
  recombination_rate = NA,  
  markers = NA,  
  mutation_rate = 0,  
  substitution_matrix = matrix(1/4, 4, 4)  
)
```

**Arguments**

- molecular\_data** Genomic data used as input, should be of type `genomeadmixture_data`. Either a single dataset is provided, or a list of multiple `genomeadmixture_data` objects.
- initial\_frequencies** A vector describing the initial contribution of each provided input data set to the starting hybrid swarm. By default, equal frequencies are assumed. If a vector not summing to 1 is provided, the vector is normalized.
- morgan** Length of the molecular sequence in Morgan (e.g. the number of crossovers during meiosis), alternatively, the recombination rate can be used, see below.
- recombination\_rate** rate in cM / Mbp, used to map recombination to the markers. If the `recombination_rate` is not set, the value for Morgan is used, assuming that the markers included span an entire chromosome.
- markers** A vector of locations of markers, these markers are tracked for every generation.
- mutation\_rate** the per base probability of mutation. Default is 0.
- substitution\_matrix** a 4x4 matrix representing the probability of mutating to another base (where  $[1/2/3/4] = [a/c/t/g]$ ), conditional on the event of a mutation happening. Default is the JC69 matrix, with equal probabilities for all transitions / transversions.

**Value**

sequence module object, used as starting point for `simulate_admixture`.

---

<code>simulate_admixture</code>	<i>Individual based simulation of the breakdown of contiguous ancestry blocks.</i>
---------------------------------	--

---

**Description**

Individual based simulation of the breakdown of contiguous ancestry blocks, with or without selection. Simulations can be started from scratch, or from a predefined input population.

**Usage**

```
simulate_admixture(
  module = ancestry_module(),
  pop_size = 100,
  total_runtime = 100,
  migration = migration_settings(),
  select_matrix = NA,
  multiplicative_selection = TRUE,
  verbose = FALSE,
  num_threads = 1
)
```

**Arguments**

<code>module</code>	Chosen module to simulate, either created with <code>module_ancestry</code> or <code>module_sequence</code> .
<code>pop_size</code>	The number of individuals in the population. If the number is larger than the number of individuals in the input population (if provided), additional individuals are sampled randomly from the input population to reach the intended size.
<code>total_runtime</code>	Number of generations
<code>migration</code>	settings associated with migration, should be created with <a href="#">migration_settings</a>
<code>select_matrix</code>	Selection matrix indicating the markers which are under selection. If not provided by the user, the simulation proceeds neutrally. If provided, each row in the matrix should contain five entries: <ul style="list-style-type: none"> <li>• location of the marker under selection (in Morgan)</li> <li>• fitness of wildtype (aa)</li> <li>• fitness of heterozygote (aA)</li> <li>• fitness of homozygote mutant (AA)</li> <li>• Ancestral type that represents the mutant allele A</li> </ul>
<code>multiplicative_selection</code>	Default: TRUE. If TRUE, fitness is calculated for multiple markers by multiplying fitness values for each marker. If FALSE, fitness is calculated by adding fitness values for each marker.
<code>verbose</code>	Verbose output if TRUE. Default value is FALSE
<code>num_threads</code>	number of threads. Default is 1. Set to -1 to use all available threads

**Value**

A list with: population a population object, and three tibbles with allele frequencies (only contain values of a vector was provided to the argument markers: `frequencies`, `initial_frequencies` and `final_frequencies`). Each tibble contains four columns, time, location, ancestor and frequency, which indicates the number of generations, the location along the chromosome of the marker, the ancestral allele at that location in that generation, and finally, the frequency of that allele.

**Examples**

```
# local ancestry simulation
two_populations <- simulate_admixture(
  module = ancestry_module(number_of_founders = 3,
                           morgan = 0.8),
  migration = migration_settings(
    migration_rate = 0.01,
    population_size = c(100, 100)),
  total_runtime = 10)

# sequence simulation
data(dgrp2.3R.5k.data)

sequence_population <-
  simulate_admixture(
    module = sequence_module(molecular_data = dgrp2.3R.5k.data,
```

```

        recombination_rate = 0.2,
        mutation_rate = 1e-5),
pop_size = 1000,
total_runtime = 10)

```

---

simulate_ancestry	<i>Individual based simulation of the breakdown of contiguous ancestry blocks.</i>
-------------------	--

---

## Description

Individual based simulation of the breakdown of contiguous ancestry blocks, with or without selection. Simulations can be started from scratch, or from a predefined input population.

## Usage

```

simulate_ancestry(
  input_population = NA,
  pop_size = NA,
  number_of_founders = 2,
  initial_frequencies = NA,
  total_runtime = 100,
  morgan = 1,
  num_threads = 1,
  select_matrix = NA,
  markers = NA,
  verbose = FALSE,
  track_junctions = FALSE,
  multiplicative_selection = TRUE
)

```

## Arguments

input_population	Potential earlier simulated population used as starting point for the simulation. If not provided by the user, the simulation starts from scratch.
pop_size	The number of individuals in the population. If the number is larger than the number of individuals in the input population (if provided), additional individuals are sampled randomly from the input population to reach the intended size.
number_of_founders	Number of unique ancestors
initial_frequencies	A vector describing the initial frequency of each ancestor. By default, equal frequencies are assumed. If a vector not summing to 1 is provided, the vector is normalized.
total_runtime	Number of generations

morgan	Length of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
num_threads	number of threads. Default is 1. Set to -1 to use all available threads
select_matrix	Selection matrix indicating the markers which are under selection. If not provided by the user, the simulation proceeds neutrally. If provided, each row in the matrix should contain five entries: <ul style="list-style-type: none"> <li>• location of the marker under selection (in Morgan)</li> <li>• fitness of wildtype (aa)</li> <li>• fitness of heterozygote (aA)</li> <li>• fitness of homozygote mutant (AA)</li> <li>• Ancestral type that represents the mutant allele A</li> </ul>
markers	A vector of locations of markers (relative locations in [0, 1]). If a vector is provided, ancestry at these marker positions is tracked for every generation.
verbose	Verbose output if TRUE. Default value is FALSE
track_junctions	Track the average number of junctions over time if TRUE
multiplicative_selection	Default: TRUE. If TRUE, fitness is calculated for multiple markers by multiplying fitness values for each marker. If FALSE, fitness is calculated by adding fitness values for each marker.

### Value

A list with: population a population object, and three tibbles with allele frequencies (only contain values of a vector was provided to the argument markers: frequencies, initial\_frequencies and final\_frequencies. Each tibble contains four columns, time, location, ancestor and frequency, which indicates the number of generations, the location along the chromosome of the marker, the ancestral allele at that location in that generation, and finally, the frequency of that allele.

---

simulate\_ancestry\_migration

*Individual based simulation of the breakdown of contiguous ancestry blocks in two populations linked by migration*

---

### Description

Individual based simulation of the breakdown of contiguous ancestry blocks, with or without selection. Simulations can be started from scratch, or from a predefined input population. Two populations are simulated, connected by migration



**Usage**

```
simulate_ancestry_migration(
  input_population_1 = NA,
  input_population_2 = NA,
  pop_size = c(100, 100),
  initial_frequencies = list(c(1, 0), c(0, 1)),
  total_runtime = 100,
  morgan = 1,
  num_threads = 1,
  select_matrix = NA,
  markers = NA,
  verbose = FALSE,
  track_junctions = FALSE,
  multiplicative_selection = TRUE,
  migration_rate = 0,
  stop_at_critical_fst = FALSE,
  critical_fst = 0.1,
  generations_between_update = 100,
  sampled_individuals = 10,
  number_of_markers = 100,
  random_markers = TRUE
)
```

**Arguments**

<code>input_population_1</code>	Potential earlier simulated population used as starting point for the simulation. If not provided by the user, the simulation starts from scratch.
<code>input_population_2</code>	Potential earlier simulated population used as starting point for the simulation. If not provided by the user, the simulation starts from scratch.
<code>pop_size</code>	Vector containing the number of individuals in both populations.
<code>initial_frequencies</code>	A list describing the initial frequency of each ancestor in each population. Each entry in the list contains a vector with the frequencies for all ancestor. The length of the vector indicates the number of unique ancestors. If a vector not summing to 1 is provided, the vector is normalized.
<code>total_runtime</code>	Number of generations
<code>morgan</code>	Length of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
<code>num_threads</code>	number of threads. Default is 1. Set to -1 to use all available threads
<code>select_matrix</code>	Selection matrix indicating the markers which are under selection. If not provided by the user, the simulation proceeds neutrally. If provided, each row in the matrix should contain five entries: <ul style="list-style-type: none"> <li>• location of the marker under selection (in Morgan)</li> <li>• fitness of wildtype (aa)</li> </ul>

	<ul style="list-style-type: none"> <li>• fitness of heterozygote (aA)</li> <li>• fitness of homozygote mutant (AA)</li> <li>• Ancestral type that represents the mutant allele A</li> </ul>
markers	A vector of locations of markers (relative locations in [0, 1]). If a vector is provided, ancestry at these marker positions is tracked for every generation.
verbose	Verbose output if TRUE. Default value is FALSE
track_junctions	Track the average number of junctions over time if TRUE
multiplicative_selection	Default: TRUE. If TRUE, fitness is calculated for multiple markers by multiplying fitness values for each marker. If FALSE, fitness is calculated by adding fitness values for each marker.
migration_rate	Rate of migration between the two populations. Migration is implemented such that with probability m (migration rate) one of the two parents of a new offspring is from the other population, with probability 1-m both parents are of the focal population.
stop_at_critical_fst	option to stop at a critical FST value , default is FALSE
critical_fst	the critical fst value to stop, if stop_simulation_at_critical_fst is TRUE
generations_between_update	The number of generations after which the simulation has to check again whether the critical Fst value is exceeded
sampled_individuals	Number of individuals to be sampled at random from the population to estimate Fst
number_of_markers	Number of markers to be used to estimate Fst
random_markers	Are the markers to estimate Fst randomly distributed, or regularly distributed? Default is TRUE.

## Value

A list with: population\_1, population\_2 two population objects, and three tibbles with allele frequencies (only contain values of a vector was provided to the argument markers: frequencies, initial\_frequencies and final\_frequencies. Each tibble contains five columns, time, location, ancestor, frequency and population, which indicates the number of generations, the location along the chromosome of the marker, the ancestral allele at that location in that generation, the frequency of that allele and the population in which it was recorded (1 or 2). If a critical fst value was used to terminate the simulation, and object FST with the final FST estimate is returned as well.

---

simulate_sequence	<i>Individual based simulation of the breakdown of contiguous ancestry blocks.</i>
-------------------	--

---

## Description

Individual based simulation of the breakdown of contiguous ancestry blocks, with or without selection. Simulations can be started from scratch, or from a predefined input population.

## Usage

```
simulate_sequence(
  input_data = NA,
  pop_size = NA,
  initial_frequencies = NA,
  total_runtime = 100,
  morgan = 1,
  recombination_rate = NA,
  num_threads = 1,
  select_matrix = NA,
  markers = NA,
  verbose = FALSE,
  multiplicative_selection = TRUE,
  mutation_rate = 0,
  substitution_matrix = matrix(1/4, 4, 4)
)
```

## Arguments

input_data	Genomic data used as input, should be of type <code>genomeadmixr_data</code> . Either a single dataset is provided, or a list of multiple <code>genomeadmixr_data</code> objects.
pop_size	Vector containing the number of individuals in both populations.
initial_frequencies	A vector describing the initial contribution of each provided input data set to the starting hybrid swarm. By default, equal frequencies are assumed. If a vector not summing to 1 is provided, the vector is normalized.
total_runtime	Number of generations
morgan	Length of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
recombination_rate	rate in cM / Mbp, used to map recombination to the markers. If the <code>recombination_rate</code> is not set, the value for Morgan is used, assuming that the markers included span an entire chromosome.
num_threads	number of threads. Default is 1. Set to -1 to use all available threads

select_matrix	Selection matrix indicating the markers which are under selection. If not provided by the user, the simulation proceeds neutrally. If provided, each row in the matrix should contain five entries: <ul style="list-style-type: none"> <li>• location of the marker under selection (in Morgan)</li> <li>• fitness of wildtype (aa)</li> <li>• fitness of heterozygote (aA)</li> <li>• fitness of homozygote mutant (AA)</li> <li>• Ancestral type that represents the mutant allele A</li> </ul>
markers	A vector of locations of markers (relative locations in [0, 1]). If a vector is provided, ancestry at these marker positions is tracked for every generation.
verbose	Verbose output if TRUE. Default value is FALSE
multiplicative_selection	Default: TRUE. If TRUE, fitness is calculated for multiple markers by multiplying fitness values for each marker. If FALSE, fitness is calculated by adding fitness values for each marker.
mutation_rate	the per base probability of mutation. Default is 0.
substitution_matrix	a 4x4 matrix representing the probability of mutating to another base (where [1/2/3/4] = [a/c/t/g]), conditional on the event of a mutation happening. Default is the JC69 matrix, with equal probabilities for all transitions / transversions.

### Value

A list with: population a population object, and three tibbles with allele frequencies (only contain values of a vector was provided to the argument markers: frequencies, initial\_frequencies and final\_frequencies. Each tibble contains four columns, time, location, ancestor and frequency, which indicates the number of generations, the location along the chromosome of the marker, the ancestral allele at that location in that generation, and finally, the frequency of that allele.

---

simulate\_sequence\_migration

*Individual based simulation of the breakdown of contiguous ancestry blocks in two populations linked by migration*

---

### Description

Individual based simulation of the breakdown of contiguous ancestry blocks, with or without selection. Simulations can be started from scratch, or from a predefined input population. Two populations are simulated, connected by migration

**Usage**

```
simulate_sequence_migration(
  input_data_population_1 = NA,
  input_data_population_2 = NA,
  pop_size = c(100, 100),
  total_runtime = 100,
  morgan = 1,
  recombination_rate = NA,
  num_threads = 1,
  select_matrix = NA,
  markers = NA,
  verbose = FALSE,
  multiplicative_selection = TRUE,
  migration_rate = 0,
  stop_at_critical_fst = FALSE,
  critical_fst = NA,
  generations_between_update = 100,
  sampled_individuals = 10,
  number_of_markers = 100,
  random_markers = TRUE,
  mutation_rate = 0,
  substitution_matrix = matrix(1/4, 4, 4)
)
```

**Arguments**

input_data_population_1	Genomic data used as input, should be created by the function <code>create_input_data</code> or by the function <code>combine_input_data</code>
input_data_population_2	Genomic data used as input, should be created by the function <code>create_input_data</code> or by the function <code>combine_input_data</code>
pop_size	Vector containing the number of individuals in both populations.
total_runtime	Number of generations
morgan	Length of the chromosome in Morgan (e.g. the number of crossovers during meiosis)
recombination_rate	rate in cM / Mbp, used to map recombination to the markers. If the <code>recombination_rate</code> is not set, the value for <code>morgan</code> is used, assuming that the markers included span an entire chromosome.
num_threads	number of threads. Default is 1. Set to -1 to use all available threads
select_matrix	Selection matrix indicating the markers which are under selection. If not provided by the user, the simulation proceeds neutrally. If provided, each row in the matrix should contain five entries: <ul style="list-style-type: none"> <li>• location of the marker under selection (in Morgan)</li> <li>• fitness of wildtype (aa)</li> </ul>

	<ul style="list-style-type: none"> <li>• fitness of heterozygote (aA)</li> <li>• fitness of homozygote mutant (AA)</li> <li>• Ancestral type that represents the mutant allele A</li> </ul>
markers	A vector of locations of markers (relative locations in [0, 1]). If a vector is provided, ancestry at these marker positions is tracked for every generation.
verbose	Verbose output if TRUE. Default value is FALSE
multiplicative_selection	Default: TRUE. If TRUE, fitness is calculated for multiple markers by multiplying fitness values for each marker. If FALSE, fitness is calculated by adding fitness values for each marker.
migration_rate	Rate of migration between the two populations. Migration is implemented such that with probability m (migration rate) one of the two parents of a new offspring is from the other population, with probability 1-m both parents are of the focal population.
stop_at_critical_fst	option to stop at a critical FST value , default is FALSE
critical_fst	the critical fst value to stop, if stop_simulation_at_critical_fst is TRUE
generations_between_update	The number of generations after which the simulation has to check again whether the critical Fst value is exceeded
sampled_individuals	Number of individuals to be sampled at random from the population to estimate Fst
number_of_markers	Number of markers to be used to estimate Fst
random_markers	Are the markers to estimate Fst randomly distributed, or regularly distributed? Default is TRUE.
mutation_rate	the per base probability of mutation. Default is 0.
substitution_matrix	a 4x4 matrix representing the probability of mutating to another base (where [1/2/3/4] = [a/c/t/g]), conditional on the event of a mutation happening. Default is the JC69 matrix, with equal probabilities for all transitions / transversions.

## Value

A list with: population\_1, population\_2 two population objects, and three tibbles with allele frequencies (only contain values of a vector was provided to the argument markers: frequencies, initial\_frequencies and final\_frequencies. Each tibble contains five columns, time, location, ancestor, frequency and population, which indicates the number of generations, the location along the chromosome of the marker, the ancestral allele at that location in that generation, the frequency of that allele and the population in which it was recorded (1 or 2). If a critical fst value was used to terminate the simulation, and object FST with the final FST estimate is returned as well.

---

```
simulation_data_to_genomeadmixr_data  
  function to convert ped/map data to genome_admixr_data
```

---

**Description**

function to convert ped/map data to genome\_admixr\_data

**Usage**

```
simulation_data_to_genomeadmixr_data(  
  simulation_data,  
  markers = NA,  
  verbose = FALSE  
)
```

**Arguments**

simulation_data	result of simulate_admixture
markers	vector of locations of markers (in Morgan). If no vector is provided, the function searches for marker locations in the simulation_data.
verbose	provide verbose output (default is FALSE)

**Value**

genomeadmixr\_data object ready for simulate\_admixture\_data

---

```
vcfR_to_genomeadmixr_data  
  function to convert a vcfR object to genome_admixr_data
```

---

**Description**

function to convert a vcfR object to genome\_admixr\_data

**Usage**

```
vcfR_to_genomeadmixr_data(  
  vcfR_object,  
  chosen_chromosome,  
  number_of_snps = NA,  
  random_snps = TRUE,  
  verbose = FALSE  
)
```

**Arguments**

vcfr_object	result of vcfr::read.vcfr
chosen_chromosome	chromosome of choice
number_of_snps	number of snps to be loaded from the vcf file, default is to load all snps
random_snps	if a subset of all snps has to be taken, should these be sampled sequentially (e.g. the first 100 snps) or randomly (100 randomly sampled snps) (examples are for 'number_of_snps' = 100).
verbose	if true, print progress bar

**Value**

genomeadmixr\_data object ready for simulate\_admixture\_data

---

write_as_plink	<i>function to write simulation output as PLINK style data</i>
----------------	--

---

**Description**

function to write simulation output as PLINK style data

**Usage**

```
write_as_plink(
  input_pop,
  marker_locations,
  file_name_prefix,
  chromosome = 1,
  recombination_rate = 1
)
```

**Arguments**

input_pop	input population, either of class "population" or of class "genomeadmixr_data"
marker_locations	location of markers, in bp
file_name_prefix	prefix of the ped/map files.
chromosome	chromosome indication for map file
recombination_rate	recombination rate in cM / kb

**Value**

No return value



# Index

- \* **datasets**
  - dgrp2.3R.5k.data, [13](#)
- ancestry\_module, [5](#)
- calculate\_allele\_frequencies, [6](#)
- calculate\_average\_ld, [7](#)
- calculate\_dist\_junctions, [7](#)
- calculate\_fst, [8](#)
- calculate\_heterozygosity, [9](#)
- calculate\_ld, [9](#)
- calculate\_marker\_frequency, [10](#)
- combine\_input\_data, [11](#)
- create\_artificial\_genomeadmixr\_data,  
[12](#)
- create\_iso\_female, [12](#)
- dgrp2.3R.5k.data, [13](#)
- GenomeAdmixR (GenomeAdmixR-package), [3](#)
- GenomeAdmixR-package, [3](#)
- iso\_female\_ancestry, [14](#)
- iso\_female\_sequence, [15](#)
- load\_population, [16](#)
- migration\_settings, [17](#), [30](#)
- plink\_to\_genomeadmixr\_data, [18](#)
- plot.individual, [18](#)
- plot\_chromosome, [19](#)
- plot\_difference\_frequencies, [20](#)
- plot\_dist\_junctions, [21](#)
- plot\_frequencies, [21](#)
- plot\_joyplot\_frequencies, [22](#)
- plot\_over\_time, [23](#)
- plot\_start\_end, [24](#)
- print.genomeadmixr\_data, [25](#)
- print.individual, [26](#)
- print.population, [26](#)
- read\_input\_data, [27](#)
- save\_population, [16](#), [28](#)
- sequence\_module, [28](#)
- simulate\_admixture, [29](#)
- simulate\_ancestry, [31](#)
- simulate\_ancestry\_migration, [32](#)
- simulate\_sequence, [35](#)
- simulate\_sequence\_migration, [36](#)
- simulation\_data\_to\_genomeadmixr\_data,  
[39](#)
- vcfR\_to\_genomeadmixr\_data, [39](#)
- write\_as\_plink, [40](#)