Package ‘slendr’

August 19, 2022

Title A Simulation Framework for Spatiotemporal Population Genetics

Version 0.3.0

Description A framework for simulating spatially explicit genomic data which leverages real cartographic information for programmatic and visual encoding of spatiotemporal population dynamics on real geographic landscapes. Population genetic models are then automatically executed by the 'SLiM' software by Haller et al. (2019) <doi:10.1093/molbev/msy228> behind the scenes, using a custom built-in simulation 'SLiM' script. Additionally, fully abstract spatial models not tied to a specific geographic location are supported, and users can also simulate data from standard, non-spatial, random-mating models. These can be simulated either with the 'SLiM' built-in back-end script, or using an efficient coalescent population genetics simulator 'msprime' by Baumdicker et al. (2022) <doi:10.1093/genetics/iyab229> with a custom-built 'Python' script bundled with the R package. Simulated genomic data is saved in a tree-sequence format and can be loaded, manipulated, and summarised using tree-sequence functionality via an R interface to the 'Python' module 'tskit' by Kelleher et al. (2019) <doi:10.1038/s41588-019-0483-y>. Complete model configuration, simulation and analysis pipelines can be therefore constructed without a need to leave the R environment, eliminating friction between disparate tools for population genetic simulations and data analysis.

Depends R (>= 3.6.0)

Encoding UTF-8

RoxygenNote 7.2.0

SystemRequirements 'SLiM' is a forward simulation software for population genetics and evolutionary biology. See <https://messerlab.org/slim/> for installation instructions and further information. The 'Python' coalescent framework 'msprime' and the 'tskit' module can by installed by following the instructions at <https://tskit.dev/>.

Imports sf, stars, ggplot2, dplyr, purrr, readr, magrittr, reticulate, tidyr, rnaturalearth, gganimate, png, ijtiff, shinyWidgets, shiny, ape

Suggests testthat (>= 3.0.0), knitr, rmarkdown, admixr, units, rgdal, magick, cowplot, forcats, rsvg

VignetteBuilder knitr

URL https://github.com/bodkan/slendr

BugReports https://github.com/bodkan/slendr/issues

Config/testthat/edition 3

NeedsCompilation no

Author Martin Petr [aut, cre] (<https://orcid.org/0000-0003-4879-8421>)

Maintainer Martin Petr <contact@bodkan.net>

Repository CRAN

Date/Publication 2022-08-19 09:20:23 UTC

R topics documented:

animate_model 3
area 4
as.phylo.slendr_phylo 5
check_dependencies 5
check_env 6
clear_env 6
compile_model 7
distance 9
expand_range 10
explore_model 12
gene_flow 13
join 15
move 16
msprime 18
overlap 19
plot_map 20
plot_model 21
population 22
print.slendr_pop 24
print.slendr_ts 25
read_model 26
region 26
reproject 27
resize 28
schedule_sampling 30
setup_env 32
set_dispersal 32
set_range 34
shrink_range 37
slim 39

subtract 41
ts_afs 42
ts_ancestors 43
ts_coalesced 44
ts_descendants 45
ts_divergence 46
ts_diversity 47
ts_draw 48
ts_edges 49
ts_eigenstrat 50
ts_f2 50
ts_fst 53
ts_genotypes 54
ts_load 54
ts_metadata 56
ts_mutate 57
ts_nodes 58
ts_phylo 59
ts_recapitate 60
ts_samples 62
ts_save 62
ts_segregating 63
ts_simplify 64
ts_table 65
ts_tajima 67
ts_tree 68
ts_vcf 69
world 69

Index **71**

`animate_model` *Animate the simulated population dynamics*

Description

Animate the simulated population dynamics

Usage

```
animate_model(model, file, steps, gif = NULL, width = 800, height = 560)
```

Arguments

`model`	Compiled `slendr_model` model object
`file`	Path to the table of saved individual locations
`steps`	How many frames should the animation have?
`gif`	Path to an output GIF file (animation object returned by default)
`width, height`	Dimensions of the animation in pixels

Value

If `gif = NULL`, return gganimate animation object. Otherwise a GIF file is saved and no value is returned.

`area` *Calculate the area covered by the given slendr object*

Description

Calculate the area covered by the given slendr object

Usage

```
area(x)
```

Arguments

`x`	Object of the class `slendr`

Value

Area covered by the input object. If a `slendr_pop` was given, a table with an population range area in each time point will be returned. If a `slendr_region` or `slendr_world` object was specified, the total area covered by this object's spatial boundary will be returned.

Examples

```
region_a <- region("A", center = c(20, 50), radius = 20)
region_b <- region("B", polygon = list(c(50, 40), c(70, 40), c(70, 60), c(50, 60)))
plot_map(region_a, region_b)

# note that area won't be *exactly* equal to pi*r^2:
#   https://stackoverflow.com/a/65280376
area(region_a)

area(region_b)
```

`as.phylo.slendr_phylo` *Convert an annotated* `slendr_phylo` *object to a* `phylo` *object*

Description

This function servers as a workaround around a ggtree error: `Error in UseMethod("as.phylo") : no applicable method for 'as.phylo' applied to an object of class "c('phylo', 'slendr_phylo')"`

Usage

```
## S3 method for class 'slendr_phylo'
as.phylo(x)
```

Arguments

`x`	Tree object of the class `slendr_phylo`

Value

Standard phylogenetic tree object implemented by the R package ape

---

`check_dependencies` *Check that all dependencies are available for slendr examples*

Description

Check that all dependencies are available for slendr examples

Usage

```
check_dependencies(python = FALSE, slim = FALSE)
```

Arguments

`python`	Is the slendr Python environment required?
`slim`	Is SLiM required?

Value

No return value. Called only to result in an error message if a particular software dependency is missing for an example to run.

`check_env` *Check that the active Python environment is setup for slendr*

Description

This function inspects the Python environment which has been activated by the reticulate package and prints the versions of all slendr Python dependencies to the console.

Usage

```
check_env(quiet = FALSE)
```

Arguments

`quiet`	Should a log message be printed? If `FALSE`, only a logical value is returned (invisibly).

Value

Either `TRUE` (slendr Python environment is present) or `FALSE` (slendr Python environment is not present).

Examples

```
check_env()
```

`clear_env` *Remove the automatically created slendr Python environment*

Description

Remove the automatically created slendr Python environment

Usage

```
clear_env(force = FALSE)
```

Arguments

`force`	Ask before deleting the environment?

Value

No return value, called for side effects

`compile_model` *Compile the spatial demographic model*

Description

First, compiles the vectorized population spatial maps into a series of binary raster PNG files, which is the format that SLiM understands and uses it to define population boundaries. Then extracts the demographic model defined by the user (i.e. population divergences and gene flow events) into a series of tables which are later used by the built-in SLiM script to program the timing of simulation events.

Usage

```
compile_model(
  populations,
  generation_time,
  path = NULL,
  resolution = NULL,
  competition = NULL,
  mating = NULL,
  dispersal = NULL,
  gene_flow = list(),
  overwrite = FALSE,
  force = FALSE,
  simulation_length = NULL,
  direction = NULL,
  slim_script = NULL,
  description = "",
  sim_length = NULL
)
```

Arguments

`populations`	Object(s) of the `slendr_pop` class (multiple objects need to be specified in a `list`)
`generation_time`	Generation time (in model time units)
`path`	Output directory for the model configuration files which will be loaded by the backend SLiM script. If `NULL`, model configuration files will be saved to a temporary directory.
`resolution`	How many distance units per pixel?
`competition, mating`	Maximum spatial competition and mating choice distance
`dispersal`	Standard deviation of the normal distribution of the parent-offspring distance
`gene_flow`	Gene flow events generated by the `gene_flow` function (either a list of data.frame objects in the format defined by the `gene_flow` function, or a single data.frame)

`overwrite`	Completely delete the specified directory, in case it already exists, and create a new one?
`force`	Force a deletion of the model directory if it is already present? Useful for non-interactive uses. In an interactive mode, the user is asked to confirm the deletion manually.
`simulation_length`	Total length of the simulation (required for forward time models, optional for models specified in backward time units which by default run to "the present time")
`direction`	Intended direction of time. Under normal circumstances this parameter is inferred from the model and does not need to be set manually.
`slim_script`	Path to a SLiM script to be used for executing the model (by default, a bundled backend script will be used). If `NULL`, the SLiM script bundled with slendr will be used.
`description`	Optional short description of the model
`sim_length`	Deprecated. Use `simulation_length` instead.

Value

Compiled `slendr_model` model object which encapsulates all information about the specified model (which populations are involved, when and how much gene flow should occur, what is the spatial resolution of a map, and what spatial dispersal and mating parameters should be used in a SLiM simulation, if applicable)

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)
```

```
# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events ------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -----------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`distance` *Calculate the distance between a pair of spatial boundaries*

Description

Calculate the distance between a pair of spatial boundaries

Usage

```
distance(x, y, measure, time = NULL)
```

Arguments

`x, y`	Objects of the class `slendr`
`measure`	How to measure distance? This can be either `'border'` (distance between the borders of `x` and `y`) or `'center'` (distance between their centroids).
`time`	Time closest to the spatial maps of `x` and `y` if they represent `slendr_pop` population boundaries (ignored for general `slendr_region` objects)

Value

If the coordinate reference system was specified, a distance in projected units (i.e. meters) is returned. Otherwise the function returns a normal Euclidean distance.

Examples

```
# create two regions on a blank abstract landscape
region_a <- region("A", center = c(20, 50), radius = 20)
region_b <- region("B", center = c(80, 50), radius = 20)
plot_map(region_a, region_b)

# compute the distance between the centers of both population ranges
distance(region_a, region_b, measure = "center")

# compute the distance between the borders of both population ranges
distance(region_a, region_b, measure = "border")
```

`expand_range` *Expand the population range*

Description

Expands the spatial population range by a specified distance in a given time-window

Usage

```
expand_range(
  pop,
  by,
  end,
  start,
  overlap = 0.8,
  snapshots = NULL,
  polygon = NULL,
  lock = FALSE,
  verbose = TRUE
)
```

Arguments

`pop`	Object of the class `slendr_pop`
`by`	How many units of distance to expand by?
`start, end`	When does the expansion start/end?
`overlap`	Minimum overlap between subsequent spatial boundaries
`snapshots`	The number of intermediate snapshots (overrides the `overlap` parameter)
`polygon`	Geographic region to restrict the expansion to
`lock`	Maintain the same density of individuals. If `FALSE` (the default), the number of individuals in the population will not change. If `TRUE`, the number of individuals simulated will be changed (increased or decreased) appropriately, to match the new population range area.
`verbose`	Report on the progress of generating intermediate spatial boundaries?

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
```

```
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events --------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`explore_model` *Open an interactive browser of the spatial model*

Description

Open an interactive browser of the spatial model

Usage

```
explore_model(model)
```

Arguments

`model`	Compiled `slendr_model` model object

Value

No return value, called in order to start an interactive browser-based interface to explore the dynamics of a slendr model

---

`gene_flow` *Define a gene-flow event between two populations*

Description

Define a gene-flow event between two populations

Usage

```
gene_flow(from, to, rate, start, end, overlap = TRUE)
```

Arguments

`from, to`	Objects of the class `slendr_pop`
`rate`	Scalar value in the range (0, 1] specifying the proportion of migration over given time period
`start, end`	Start and end of the gene-flow event
`overlap`	Require spatial overlap between admixing populations? (default `TRUE`)

Value

Object of the class data.frame containing parameters of the specified gene-flow event.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
```

```
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events --------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`join` *Merge two spatial* `slendr` *objects into one*

Description

Merge two spatial `slendr` objects into one

Usage

```
join(x, y, name = NULL)
```

Arguments

`x`	Object of the class `slendr`
`y`	Object of the class `slendr`
`name`	Optional name of the resulting geographic region. If missing, name will be constructed from the function arguments.

Value

Object of the class `slendr_region` which encodes a standard spatial object of the class `sf` with several additional attributes (most importantly a corresponding `slendr_map` object, if applicable).

Examples

```
# create a blank abstract world 1000x1000 distance units in size
blank_map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# it is possible to construct custom landscapes (islands, corridors, etc.)
island1 <- region("island1", polygon = list(c(10, 30), c(50, 30), c(40, 50), c(0, 40)))
island2 <- region("island2", polygon = list(c(60, 60), c(80, 40), c(100, 60), c(80, 80)))
island3 <- region("island3", center = c(20, 80), radius = 10)
archipelago <- island1 %>% join(island2) %>% join(island3)

custom_map <- world(xrange = c(1, 100), c(1, 100), landscape = archipelago)

# real Earth landscapes can be defined using freely-available Natural Earth
# project data and with the possibility to specify an appropriate Coordinate
# Reference System, such as this example of a map of Europe

real_map <- world(xrange = c(-15, 40), yrange = c(30, 60), crs = "EPSG:3035")
```

`move` *Move the population to a new location in a given amount of time*

Description

This function defines a displacement of a population along a given trajectory in a given time frame

Usage

```
move(
  pop,
  trajectory,
  end,
  start,
  overlap = 0.8,
  snapshots = NULL,
  verbose = TRUE
)
```

Arguments

`pop`	Object of the class `slendr_pop`
`trajectory`	List of two-dimensional vectors (longitude, latitude) specifying the migration trajectory
`start`, `end`	Start/end points of the population migration
`overlap`	Minimum overlap between subsequent spatial boundaries
`snapshots`	The number of intermediate snapshots (overrides the `overlap` parameter)
`verbose`	Show the progress of searching through the number of sufficient snapshots?

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)
```

```
# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events -------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation ------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
```

```
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`msprime` *Run a slendr model in msprime*

Description

This function will execute a built-in msprime script and run a compiled slendr demographic model.

Usage

```
msprime(
  model,
  sequence_length,
  recombination_rate,
  samples = NULL,
  output = NULL,
  random_seed = NULL,
  load = TRUE,
  verbose = FALSE,
  debug = FALSE,
  sampling = NULL
)
```

Arguments

`model`	Model object created by the `compile` function
`sequence_length`	Total length of the simulated sequence (in base-pairs)
`recombination_rate`	Recombination rate of the simulated sequence (in recombinations per basepair per generation)
`samples`	A data frame of times at which a given number of individuals should be remembered in the tree-sequence (see `schedule_sampling` for a function that can generate the sampling schedule in the correct format). If missing, only individuals present at the end of the simulation will be recorded in the tree-sequence output file.
`output`	Path to the output tree-sequence file. If `NULL` (the default), tree sequence will be saved to a temporary file.
`random_seed`	Random seed (if missing, SLiM's own seed will be used)
`load`	Should the final tree sequence be immediately loaded and returned? Default is `TRUE`. The alternative (`FALSE`) is useful when a tree-sequence file is written to a custom location to be loaded at a later point.

`verbose`	Write the output log to the console (default `FALSE`)?
`debug`	Write msprime's debug log to the console (default `FALSE`)?
`sampling`	Deprecated in favor of `samples`.

Value

A tree-sequence object loaded via Python-R reticulate interface function `ts_load` (internally represented by the Python object `tskit.trees.TreeSequence`)

Examples

```
# load an example model
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# afr and eur objects would normally be created before slendr model compilation,
# but here we take them out of the model object already compiled for this
# example (in a standard slendr simulation pipeline, this wouldn't be necessary)
afr <- model$populations[["AFR"]]
eur <- model$populations[["EUR"]]
chimp <- model$populations[["CH"]]

# schedule the sampling of a couple of ancient and present-day individuals
# given model at 20 ky, 10 ky, 5ky ago and at present-day (time 0)
modern_samples <- schedule_sampling(model, times = 0, list(afr, 10), list(eur, 100), list(chimp, 1))
ancient_samples <- schedule_sampling(model, times = c(40000, 30000, 20000, 10000), list(eur, 1))

# sampling schedules are just data frames and can be merged easily
samples <- rbind(modern_samples, ancient_samples)

# run a simulation using the msprime back end from a compiled slendr model object
ts <- msprime(model, sequence_length = 1e5, recombination_rate = 0, samples = samples)

# automatic loading of a simulated output can be prevented by `load = FALSE`, which can be
# useful when a custom path to a tree-sequence output is given for later downstream analyses
output_file <- tempfile(fileext = ".trees")
msprime(model, sequence_length = 1e5, recombination_rate = 0, samples = samples,
        output = output_file, load = FALSE, random_seed = 42)
# ... at a later stage:
ts <- ts_load(output_file, model)

summary(ts)
```

`overlap` *Generate the overlap of two* `slendr` *objects*

Description

Generate the overlap of two `slendr` objects

Usage

```
overlap(x, y, name = NULL)
```

Arguments

`x`	Object of the class `slendr`
`y`	Object of the class `slendr`
`name`	Optional name of the resulting geographic region. If missing, name will be constructed from the function arguments.

Value

Object of the class `slendr_region` which encodes a standard spatial object of the class `sf` with several additional attributes (most importantly a corresponding `slendr_map` object, if applicable).

`plot_map` *Plot* `slendr` *geographic features on a map*

Description

Plots objects of the three `slendr` spatial classes (`slendr_map`, `slendr_region`, and `slendr_pop`).

Usage

```
plot_map(
  ...,
  time = NULL,
  gene_flow = FALSE,
  graticules = "original",
  intersect = TRUE,
  show_map = TRUE,
  title = NULL,
  interpolated_maps = NULL
)
```

Arguments

`...`	Objects of classes `slendr_map`, `slendr_region`, or `slendr_pop`
`time`	Plot a concrete time point
`gene_flow`	Indicate geneflow events with an arrow
`graticules`	Plot graticules in the original Coordinate Reference System (such as longitude-latitude), or in the internal CRS (such as meters)?
`intersect`	Intersect the population boundaries against landscape and other geographic boundaries (default TRUE)?
`show_map`	Show the underlying world map

`title`	Title of the plot
`interpolated_maps`	Interpolated spatial boundaries for all populations in all time points (this is only used for plotting using the `explore` shiny app)

Value

A ggplot2 object with the visualized slendr map

`plot_model` *Plot demographic history encoded in a slendr model*

Description

Plot demographic history encoded in a slendr model

Usage

```
plot_model(model, sizes = TRUE, proportions = FALSE, log = FALSE)
```

Arguments

`model`	Compiled `slendr_model` model object
`sizes`	Should population size changes be visualized?
`proportions`	Should gene flow proportions be visualized (`FALSE` by default to prevent cluttering and overplotting)
`log`	Should the y-axis be plotted on a log scale? Useful for models over very long time-scales.

Value

A ggplot2 object with the visualized slendr model

Examples

```
# load an example model with an already simulated tree sequence
path <- system.file("extdata/models/introgression", package = "slendr")
model <- read_model(path)

plot_model(model, sizes = FALSE, log = TRUE)
```

`population` *Define a population*

Description

Defines the parameters of a population (non-spatial and spatial).

Usage

```
population(
  name,
  time,
  N,
  parent = "ancestor",
  map = FALSE,
  center = NULL,
  radius = NULL,
  polygon = NULL,
  remove = NULL,
  intersect = TRUE,
  competition = NA,
  mating = NA,
  dispersal = NA,
  dispersal_fun = NULL,
  aquatic = FALSE
)
```

Arguments

`name`	Name of the population
`time`	Time of the population's first appearance
`N`	Number of individuals at the time of first appearance
`parent`	Parent population object or "ancestor" (indicating that the population does not have an ancestor, and that it is the first population in its "lineage")
`map`	Object of the type `slendr_map` which defines the world context (created using the `world` function). If the value `FALSE` is provided, a non-spatial model will be run.
`center`	Two-dimensional vector specifying the center of the circular range
`radius`	Radius of the circular range
`polygon`	List of vector pairs, defining corners of the polygon range or a geographic region of the class `slendr_region` from which the polygon coordinates will be extracted (see the `region() function`)
`remove`	Time at which the population should be removed
`intersect`	Intersect the population's boundaries with landscape features?

`competition, mating`	Maximum spatial competition and mating choice distance
`dispersal`	Standard deviation of the normal distribution of the distance that offspring disperses from its parent
`dispersal_fun`	Distribution function governing the dispersal of offspring. One of "normal", "uniform", "cauchy", "exponential", or "brownian" (in which vertical and horizontal displacements are drawn from a normal distribution independently).
`aquatic`	Is the species aquatic (`FALSE` by default, i.e. terrestrial species)?

Details

There are four ways to specify a spatial boundary: i) circular range specified using a center coordinate and a radius, ii) polygon specified as a list of two-dimensional vector coordinates, iii) polygon as in ii), but defined (and named) using the `region` function, iv) with just a world map specified (circular or polygon range parameters set to the default `NULL` value), the population will be allowed to occupy the entire landscape.

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
```

```
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events ------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -----------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`print.slendr_pop` *Print a short summary of a* `slendr` *object*

Description

All spatial objects in the slendr package are internally represented as Simple Features (`sf`) objects. This fact is hidden in most circumstances this, as the goal of the slendr package is to provide functionality at a much higher level (population boundaries, geographic regions, instead of individual polygons and other "low-level" geometric objects), without the users having to worry about low-level details involved in handling spatial geometries. However, the full `sf` object representation can be always printed by calling `x[]`.

Usage

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

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

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

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

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

Arguments

`x`	Object of a class `slendr` (either `slendr_pop`, `slendr_map`, `slendr_region`, or `slendr_table`)
`...`	Additional arguments passed to `print`

Value

No return value, used only for printing

`print.slendr_ts` *Print tskit's summary table of the Python tree-sequence object*

Description

Print tskit's summary table of the Python tree-sequence object

Usage

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

Arguments

`x`	Tree object of the class `slendr_phylo`
`...`	Additional arguments normally passed to `print` (not used in this case)

Value

No return value, simply prints the tskit summary table to the terminal

---

`read_model` *Read a previously serialized model configuration*

Description

Reads all configuration tables and other model data from a location where it was previously compiled to by the `compile` function.

Usage

```
read_model(path)
```

Arguments

`path`	Directory with all required configuration files

Value

Compiled `slendr_model` model object which encapsulates all information about the specified model (which populations are involved, when and how much gene flow should occur, what is the spatial resolution of a map, and what spatial dispersal and mating parameters should be used in a SLiM simulation, if applicable)

Examples

```
# load an example model with an already simulated tree sequence
path <- system.file("extdata/models/introgression", package = "slendr")
model <- read_model(path)

plot_model(model, sizes = FALSE, log = TRUE)
```

`region` *Define a geographic region*

Description

Creates a geographic region (a polygon) on a given map and gives it a name. This can be used to define objects which can be reused in multiple places in a slendr script (such as `region` arguments of `population`) without having to repeatedly define polygon coordinates.

Usage

```
region(name = NULL, map = NULL, center = NULL, radius = NULL, polygon = NULL)
```

Arguments

`name`	Name of the geographic region
`map`	Object of the type `sf` which defines the map
`center`	Two-dimensional vector specifying the center of the circular range
`radius`	Radius of the circular range
`polygon`	List of vector pairs, defining corners of the polygon range or a geographic region of the class `slendr_region` from which the polygon coordinates will be extracted (see the `region() function`)

Value

Object of the class `slendr_region` which encodes a standard spatial object of the class `sf` with several additional attributes (most importantly a corresponding `slendr_map` object, if applicable).

Examples

```
# create a blank abstract world 1000x1000 distance units in size
blank_map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# it is possible to construct custom landscapes (islands, corridors, etc.)
island1 <- region("island1", polygon = list(c(10, 30), c(50, 30), c(40, 50), c(0, 40)))
island2 <- region("island2", polygon = list(c(60, 60), c(80, 40), c(100, 60), c(80, 80)))
island3 <- region("island3", center = c(20, 80), radius = 10)
archipelago <- island1 %>% join(island2) %>% join(island3)

custom_map <- world(xrange = c(1, 100), c(1, 100), landscape = archipelago)

# real Earth landscapes can be defined using freely-available Natural Earth
# project data and with the possibility to specify an appropriate Coordinate
# Reference System, such as this example of a map of Europe

real_map <- world(xrange = c(-15, 40), yrange = c(30, 60), crs = "EPSG:3035")
```

`reproject` *Reproject coordinates between coordinate systems*

Description

Converts between coordinates on a compiled raster map (i.e. pixel units) and different Geographic Coordinate Systems (CRS).

Usage

```
reproject(
  from,
  to,
  x = NULL,
```

```
  y = NULL,
  coords = NULL,
  model = NULL,
  add = FALSE,
  input_prefix = "",
  output_prefix = "new"
)
```

Arguments

`from, to`	Either a CRS code accepted by GDAL, a valid integer EPSG value, an object of class `crs`, the value "raster" (converting from/to pixel coordinates), or "world" (converting from/to whatever CRS is set for the underlying map)
`x, y`	Coordinates in two dimensions (if missing, coordinates are expected to be in the `data.frame` specified in the `coords` parameter as columns "x" and "y")
`coords`	data.frame-like object with coordinates in columns "x" and "y"
`model`	Object of the class `slendr_model`
`add`	Add column coordinates to the input data.frame `coords` (coordinates otherwise returned as a separate object)?
`input_prefix, output_prefix`	Input and output prefixes of data frame columns with spatial coordinates

Value

Data.frame with converted two-dimensional coordinates given as input

Examples

```
lon_lat_df <- data.frame(x = c(30, 0, 15), y = c(60, 40, 10))

reproject(
  from = "epsg:4326",
  to = "epsg:3035",
  coords = lon_lat_df,
  add = TRUE # add converted [lon,lat] coordinates as a new column
)
```

`resize` *Change the population size*

Description

Resizes the population starting from the current value of `N` individuals to the specified value

Usage

```
resize(pop, N, how, time, end = NULL)
```

Arguments

`pop`	Object of the class `slendr_pop`
`N`	Population size after the change
`how`	How to change the population size (options are `"step"` or `"exponential"`)
`time`	Time of the population size change
`end`	End of the population size change period (used for exponential change events)

Details

In the case of exponential size change, if the final `N` is larger than the current size, the population will be exponentially growing over the specified time period until it reaches `N` individuals. If `N` is smaller, the population will shrink exponentially.

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
```

```
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events --------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`schedule_sampling` *Define sampling events for a given set of populations*

Description

Schedule sampling events at specified times and, optionally, a given set of locations on a landscape

Usage

```
schedule_sampling(model, times, ..., locations = NULL, strict = FALSE)
```

Arguments

`model`	Object of the class `slendr_model`
`times`	Integer vector of times (in model time units) at which to schedule remembering of individuals in the tree-sequence
`...`	Lists of two elements (`slendr_pop` population object-<number of individuals to sample), representing from which populations should how many individuals be remembered at times given by `times`
`locations`	List of vector pairs, defining two-dimensional coordinates of locations at which the closest number of individuals from given populations should be sampled. If `NULL` (the default), individuals will be sampled randomly throughout their spatial boundary.
`strict`	Should any occurence of a population not being present at a given time result in an error? Default is `FALSE`, meaning that invalid sampling times for any populations will be quietly ignored.

Details

If both times and locations are given, the the sampling will be scheduled on each specified location in each given time-point. Note that for the time-being, in the interest of simplicity, no sanity checks are performed on the locations given except the restriction that the sampling points must fall within the bounding box around the simulated world map. Other than that, slendr will simply instruct its SLiM backend script to sample individuals as close to the sampling points given as possible, regardless of whether those points lie within a population spatial boundary at that particular moment of time.

Value

Data frame with three columns: time of sampling, population to sample from, how many individuals to sample

Examples

```
# load an example model with an already simulated tree sequence
path <- system.file("extdata/models/introgression", package = "slendr")
model <- read_model(path)

# afr and eur objects would normally be created before slendr model compilation,
# but here we take them out of the model object already compiled for this
# example (in a standard slendr simulation pipeline, this wouldn't be necessary)
afr <- model$populations[["AFR"]]
eur <- model$populations[["EUR"]]

# schedule the recording of 10 African and 100 European individuals from a
# given model at 20 ky, 10 ky, 5ky ago and at present-day (time 0)
schedule <- schedule_sampling(
  model, times = c(20000, 10000, 5000, 0),
  list(afr, 10), list(eur, 100)
)
```

```
# the result of `schedule_sampling` is a simple data frame (note that the locations
# of sampling locations have `NA` values because the model is non-spatial)
schedule
```

`setup_env` *Setup a dedicated Python virtual environment for slendr*

Description

This function will automatically download a Python miniconda distribution dedicated to an R-Python interface. It will also create a slendr-specific Python environment with all the required Python dependencies.

Usage

```
setup_env(quiet = FALSE, agree = FALSE, pip = NULL)
```

Arguments

`quiet`	Should informative messages be printed to the console? Default is `FALSE`.
`agree`	Automatically agree to all questions?
`pip`	Should pip be used instead of conda for installing slendr's Python dependencies? Note that this will still use the conda distribution to install Python itself, but will change the repository from which slendr will install its Python dependencies. Unless explicitly set to `TRUE`, Python dependencies will be installed from conda repositories by default, expect for the case of osx-arm64 Mac architecture, for which conda dependencies are broken.

Value

No return value, called for side effects

`set_dispersal` *Change dispersal parameters*

Description

Changes either the competition interactive distance, mating choice distance, or the dispersal of offspring from its parent

Usage

```
set_dispersal(
  pop,
  time,
  competition = NA,
  mating = NA,
  dispersal = NA,
  dispersal_fun = NULL
)
```

Arguments

`pop`	Object of the class `slendr_pop`
`time`	Time of the population size change
`competition, mating`	Maximum spatial competition and mating choice distance
`dispersal`	Standard deviation of the normal distribution of the distance that offspring disperses from its parent
`dispersal_fun`	Distribution function governing the dispersal of offspring. One of "normal", "uniform", "cauchy", "exponential", or "brownian" (in which vertical and horizontal displacements are drawn from a normal distribution independently).

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = ”blank”)

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population(”pop1”, N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is ”inherited” from the parent)
pop2 <- population(”pop2”, N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
```

```
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events -----------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation ----------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`set_range` *Update the population range*

Description

This function allows a more manual control of spatial map changes in addition to the `expand` and `move` functions

Usage

```
set_range(
  pop,
  time,
  center = NULL,
  radius = NULL,
  polygon = NULL,
  lock = FALSE
)
```

Arguments

`pop`	Object of the class `slendr_pop`
`time`	Time of the change
`center`	Two-dimensional vector specifying the center of the circular range
`radius`	Radius of the circular range
`polygon`	List of vector pairs, defining corners of the polygon range (see also the `region` argument) or a geographic region of the class `slendr_region` from which the polygon coordinates will be extracted
`lock`	Maintain the same density of individuals. If `FALSE` (the default), the number of individuals in the population will not change. If `TRUE`, the number of individuals simulated will be changed (increased or decreased) appropriately, to match the new population range area.

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)
```

```
# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events --------------------------------------------------------

# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
```

```
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`shrink_range` *Shrink the population range*

Description

Shrinks the spatial population range by a specified distance in a given time-window

Usage

```
shrink_range(
  pop,
  by,
  end,
  start,
  overlap = 0.8,
  snapshots = NULL,
  lock = FALSE,
  verbose = TRUE
)
```

Arguments

`pop`	Object of the class `slendr_pop`
`by`	How many units of distance to shrink by?
`start, end`	When does the boundary shrinking start/end?
`overlap`	Minimum overlap between subsequent spatial boundaries
`snapshots`	The number of intermediate snapshots (overrides the `overlap` parameter)
`lock`	Maintain the same density of individuals. If `FALSE` (the default), the number of individuals in the population will not change. If `TRUE`, the number of individuals simulated will be changed (increased or decreased) appropriately, to match the new population range area.
`verbose`	Report on the progress of generating intermediate spatial boundaries?

Value

Object of the class `slendr_pop`, which contains population parameters such as name, time of appearance in the simulation, parent population (if any), and its spatial parameters such as map and spatial boundary.

Examples

```
# spatial definitions -----------------------------------------------------

# create a blank abstract world 1000x1000 distance units in size
map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# create a circular population with the center of a population boundary at
# [200, 800] and a radius of 100 distance units, 1000 individuals at time 1
# occupying a map just specified
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100)

# printing a population object to a console shows a brief summary
pop1

# create another population occupying a polygon range, splitting from pop1
# at a given time point (note that specifying a map is not necessary because
# it is "inherited" from the parent)
pop2 <- population("pop2", N = 100, time = 50, parent = pop1,
                        polygon = list(c(100, 100), c(320, 30), c(500, 200),
                                  c(500, 400), c(300, 450), c(100, 400)))

pop3 <- population("pop3", N = 200, time = 80, parent = pop2,
                   center = c(800, 800), radius = 200)

# move "pop1" to another location along a specified trajectory and saved the
# resulting object to the same variable (the number of intermediate spatial
# snapshots can be also determined automatically by leaving out the
# `snapshots = ` argument)
pop1_moved <- move(pop1, start = 100, end = 200, snapshots = 6,
                   trajectory = list(c(600, 820), c(800, 400), c(800, 150)))
pop1_moved

# many slendr functions are pipe-friendly, making it possible to construct
# pipelines which construct entire history of a population
pop1 <- population("pop1", N = 1000, time = 1,
                   map = map, center = c(200, 800), radius = 100) %>%
  move(start = 100, end = 200, snapshots = 6,
       trajectory = list(c(400, 800), c(600, 700), c(800, 400), c(800, 150))) %>%
  set_range(time = 300, polygon = list(
    c(400, 0), c(1000, 0), c(1000, 600), c(900, 400), c(800, 250),
    c(600, 100), c(500, 50))
  )

# population ranges can expand by a given distance in all directions
pop2 <- expand_range(pop2, by = 200, start = 50, end = 150, snapshots = 3)

# we can check the positions of all populations interactively by plotting their
# ranges together on a single map
plot_map(pop1, pop2, pop3)

# gene flow events --------------------------------------------------------
```

```
# individual gene flow events can be saved to a list
gf <- list(
  gene_flow(from = pop1, to = pop3, start = 150, end = 200, rate = 0.15),
  gene_flow(from = pop1, to = pop2, start = 300, end = 330, rate = 0.25)
)

# compilation -------------------------------------------------------------

# compile model components in a serialized form to dist, returning a single
# slendr model object (in practice, the resolution should be smaller)
model <- compile_model(
  populations = list(pop1, pop2, pop3), generation_time = 1,
  resolution = 100, simulation_length = 500,
  competition = 5, mating = 5, dispersal = 1
)
```

`slim` *Run a slendr model in SLiM*

Description

This function will execute a SLiM script generated by the `compile` function during the compilation of a slendr demographic model.

Usage

```
slim(
  model,
  sequence_length,
  recombination_rate,
  samples = NULL,
  output = NULL,
  burnin = 0,
  max_attempts = 1,
  spatial = !is.null(model$world),
  coalescent_only = TRUE,
  method = c("batch", "gui"),
  random_seed = NULL,
  verbose = FALSE,
  load = TRUE,
  locations = NULL,
  slim_path = NULL,
  sampling = NULL
)
```

Arguments

`model`	Model object created by the `compile` function
`sequence_length`	Total length of the simulated sequence (in base-pairs)
`recombination_rate`	Recombination rate of the simulated sequence (in recombinations per basepair per generation)
`samples`	A data frame of times at which a given number of individuals should be remembered in the tree-sequence (see `schedule_sampling` for a function that can generate the sampling schedule in the correct format). If missing, only individuals present at the end of the simulation will be recorded in the tree-sequence output file.
`output`	Path to the output tree-sequence file. If `NULL` (the default), tree sequence will be saved to a temporary file.
`burnin`	Length of the burnin (in model's time units, i.e. years)
`max_attempts`	How many attempts should be made to place an offspring near one of its parents? Serves to prevent infinite loops on the SLiM backend. Default value is 1.
`spatial`	Should the model be executed in spatial mode? By default, if a world map was specified during model definition, simulation will proceed in a spatial mode.
`coalescent_only`	Should `initializeTreeSeq(retainCoalescentOnly = <...>)` be set to `TRUE` (the default) or `FALSE`? See "retainCoalescentOnly" in the SLiM manual for more detail.
`method`	How to run the script? ("gui" - open in SLiMgui, "batch" - run on the command-line)
`random_seed`	Random seed (if missing, SLiM's own seed will be used)
`verbose`	Write the SLiM output log to the console (default `FALSE`)?
`load`	Should the final tree sequence be immediately loaded and returned? Default is `TRUE`. The alternative (`FALSE`) is useful when a tree-sequence file is written to a custom location to be loaded at a later point.
`locations`	If `NULL`, locations are not saved. Otherwise, the path to the file where locations of each individual throughout the simulation will be saved (most likely for use with `animate_model`).
`slim_path`	Optional way to specify path to an appropriate SLiM binary (this is useful if the `slim` binary is not on the `$PATH`).
`sampling`	Deprecated in favor of `samples`.

Value

A tree-sequence object loaded via Python-R reticulate interface function `ts_load` (internally represented by the Python object `tskit.trees.TreeSequence`)

Examples

```
# load an example model
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# afr and eur objects would normally be created before slendr model compilation,
# but here we take them out of the model object already compiled for this
# example (in a standard slendr simulation pipeline, this wouldn't be necessary)
afr <- model$populations[["AFR"]]
eur <- model$populations[["EUR"]]
chimp <- model$populations[["CH"]]

# schedule the sampling of a couple of ancient and present-day individuals
# given model at 20 ky, 10 ky, 5ky ago and at present-day (time 0)
modern_samples <- schedule_sampling(model, times = 0, list(afr, 5), list(eur, 5), list(chimp, 1))
ancient_samples <- schedule_sampling(model, times = c(30000, 20000, 10000), list(eur, 1))

# sampling schedules are just data frames and can be merged easily
samples <- rbind(modern_samples, ancient_samples)

# run a simulation using the SLiM back end from a compiled slendr model object and return
# a tree-sequence output
ts <- slim(model, sequence_length = 1e5, recombination_rate = 0, samples = samples)

# automatic loading of a simulated output can be prevented by `load = FALSE`, which can be
# useful when a custom path to a tree-sequence output is given for later downstream analyses
output_file <- tempfile(fileext = ".trees")
slim(model, sequence_length = 1e5, recombination_rate = 0, samples = samples,
     output = output_file, load = FALSE)
# ... at a later stage:
ts <- ts_load(output_file, model)

ts
```

`subtract` *Generate the difference between two* `slendr` *objects*

Description

Generate the difference between two `slendr` objects

Usage

```
subtract(x, y, name = NULL)
```

Arguments

`x`	Object of the class `slendr`
`y`	Object of the class `slendr`

`name`	Optional name of the resulting geographic region. If missing, name will be constructed from the function arguments.

Value

Object of the class `slendr_region` which encodes a standard spatial object of the class `sf` with several additional attributes (most importantly a corresponding `slendr_map` object, if applicable).

`ts_afs` *Compute the allele frequency spectrum (AFS)*

Description

This function computes the AFS with respect to the given set of individuals

Usage

```
ts_afs(
  ts,
  sample_sets = NULL,
  mode = c("site", "branch", "node"),
  windows = NULL,
  span_normalise = FALSE,
  polarised = FALSE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set). If NULL, allele frequency spectrum for all individuals in the tree sequence will be computed.
`mode`	The mode for the calculation ("sites" or "branch")
`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) are added automatically)
`span_normalise`	Argument passed to tskit's `allele_frequency_spectrum` method
`polarised`	When FALSE (the default) the allele frequency spectrum will be folded (i.e. the counts will not depend on knowing which allele is ancestral)

Details

For more information on the format of the result and dimensions, in particular the interpretation of the first and the last element of the AFS, please see the tskit manual at `https://tskit.dev/tskit/docs/stable/python-api.html`

Value

Allele frequency spectrum values for the given sample set

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

samples <- ts_samples(ts) %>% .[.$pop %in% c("AFR", "EUR"), ]

# compute AFS for the given set of individuals
ts_afs(ts, sample_sets = list(samples$name))
```

`ts_ancestors` *Extract (spatio-)temporal ancestral history for given nodes/individuals*

Description

Extract (spatio-)temporal ancestral history for given nodes/individuals

Usage

```
ts_ancestors(ts, x, verbose = FALSE, complete = TRUE)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`x`	Either an individual name or an integer node ID
`verbose`	Report on the progress of ancestry path generation?
`complete`	Does every individual in the tree sequence need to have complete metadata recorded? If `TRUE`, only individuals/nodes with complete metadata will be included in the reconstruction of ancestral relationships. For instance, nodes added during the coalescent recapitation phase will not be included because they don't have spatial information associated with them.

Value

A table of ancestral nodes of a given tree-sequence node all the way up to the root of the tree sequence

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# find the complete ancestry information for a given individual
ts_ancestors(ts, "EUR_1", verbose = TRUE)
```

`ts_coalesced` *Check that all trees in the tree sequence are fully coalesced*

Description

Check that all trees in the tree sequence are fully coalesced

Usage

```
ts_coalesced(ts, return_failed = FALSE)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`return_failed`	Report back which trees failed the coalescence check?

Value

TRUE or FALSE value if `return_failed = FALSE`, otherwise a vector of (tskit Python 0-based) indices of trees which failed the coalescence test

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

ts_coalesced(ts) # is the tree sequence fully coalesced? (TRUE or FALSE)

# returns a vector of tree sequence segments which are not coalesced
not_coalesced <- ts_coalesced(ts, return_failed = TRUE)
```

`ts_descendants` *Extract all descendants of a given tree-sequence node*

Description

Extract all descendants of a given tree-sequence node

Usage

```
ts_descendants(ts, x, verbose = FALSE, complete = TRUE)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`x`	An integer node ID of the ancestral node
`verbose`	Report on the progress of ancestry path generation?
`complete`	Does every individual in the tree sequence need to have complete metadata recorded? If `TRUE`, only individuals/nodes with complete metadata will be included in the reconstruction of ancestral relationships. For instance, nodes added during the coalescent recapitation phase will not be included because they don't have spatial information associated with them.

Value

A table of descendant nodes of a given tree-sequence node all the way down to the leaves of the tree sequence

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# find the complete descendancy information for a given individual
ts_descendants(ts, x = 62, verbose = TRUE)
```

`ts_divergence` *Calculate pairwise divergence between sets of individuals*

Description

Calculate pairwise divergence between sets of individuals

Usage

```
ts_divergence(
  ts,
  sample_sets,
  mode = c("site", "branch", "node"),
  windows = NULL,
  span_normalise = TRUE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set)
`mode`	The mode for the calculation ("sites" or "branch")
`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) do not have to be specified as they are added automatically.
`span_normalise`	Divide the result by the span of the window? Default TRUE, see the tskit documentation for more detail.

Value

For each pairwise calculation, either a single divergence value or a vector of divergence values (one for each window)

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# collect sampled individuals from all populations in a list
sample_sets <- ts_samples(ts) %>%
  split(., .$pop) %>%
```

```
  lapply(function(pop) pop$name)

# compute the divergence between individuals from each sample set (list of
# individual names generated in the previous step)
ts_divergence(ts, sample_sets) %>% .[order(.$divergence), ]
```

`ts_diversity` *Calculate diversity in given sets of individuals*

Description

Calculate diversity in given sets of individuals

Usage

```
ts_diversity(
  ts,
  sample_sets,
  mode = c("site", "branch", "node"),
  windows = NULL,
  span_normalise = TRUE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set). If a simple vector is provided, it will be interpreted as `as.list(sample_sets)`, meaning that a given statistic will be calculated for each individual separately.
`mode`	The mode for the calculation ("sites" or "branch")
`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) are added automatically)
`span_normalise`	Divide the result by the span of the window? Default TRUE, see the tskit documentation for more detail.

Value

For each set of individuals either a single diversity value or a vector of diversity values (one for each window)

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))
```

```
# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# collect sampled individuals from all populations in a list
sample_sets <- ts_samples(ts) %>%
  split(., .$pop) %>%
  lapply(function(pop) pop$name)

# compute diversity in each population based on sample sets extracted
# in the previous step
ts_diversity(ts, sample_sets) %>% .[order(.$diversity), ]
```

`ts_draw` *Plot a graphical representation of a single tree*

Description

This function first obtains an SVG representation of the tree by calling the `draw_svg` method of tskit and renders it as a bitmap image in R. All of the many optional keyword arguments of the `draw_svg` method can be provided and will be automatically passed to the method behind the scenes.

Usage

```
ts_draw(
  x,
  width = 1500,
  height = 500,
  labels = FALSE,
  sampled_only = TRUE,
  ...
)
```

Arguments

`x`	A single tree extracted by `ts_tree`
`width, height`	Pixel dimensions of the rendered bitmap
`labels`	Label each node with the individual name?
`sampled_only`	Should only individuals explicitly sampled through simplification be labeled? This is relevant in situations in which sampled individuals can themselves be among the ancestral nodes.
`...`	Keyword arguments to the tskit `draw_svg` function.

Value

No return value, called for side effects

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract the first tree in the tree sequence and draw it
tree <- ts_tree(ts, i = 1)

# ts_draw accepts various optional arguments of tskit.Tree.draw_svg
ts_draw(tree, time_scale = "rank")
```

`ts_edges` *Extract spatio-temporal edge annotation table from a given tree or tree sequence*

Description

Extract spatio-temporal edge annotation table from a given tree or tree sequence

Usage

```
ts_edges(x)
```

Arguments

`x`	Tree object generated by `ts_phylo` or a slendr tree sequence object produced by `ts_load`, `ts_recapitate`, `ts_simplify`, or `ts_mutate`

Value

Data frame of the `sf` type containing the times of nodes and start-end coordinates of edges across space

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract an annotated table with (spatio-)temporal edge information
ts_edges(ts)
```

`ts_eigenstrat` *Convert genotypes to the EIGENSTRAT file format*

Description

EIGENSTRAT data produced by this function can be used by the admixr R package (`https://bodkan.net/admixr/`).

Usage

```
ts_eigenstrat(ts, prefix, chrom = "chr1", outgroup = NULL)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`prefix`	EIGENSTRAT trio prefix
`chrom`	The name of the chromosome in the EIGENSTRAT snp file (default "chr1")
`outgroup`	Should a formal, artificial outgroup be added? If `NULL` (default), no outgroup is added. A non-NULL character name will serve as the name of the outgroup in an ind file.

Details

In case an outgroup was not formally specified in a slendr model which generated the tree sequence data, it is possible to artificially create an outgroup sample with the name specified by the `outgroup` argument, which will carry all ancestral alleles (i.e. value "2" in a geno file for each position in a snp file).

Value

Object of the class EIGENSTRAT created by the admixr package

`ts_f2` *Calculate the f2, f3, f4, and f4-ratio statistics*

Description

Calculate the f2, f3, f4, and f4-ratio statistics

Usage

```
ts_f2(
  ts,
  A,
  B,
  mode = c("site", "branch", "node"),
  span_normalise = TRUE,
  windows = NULL
)

ts_f3(
  ts,
  A,
  B,
  C,
  mode = c("site", "branch", "node"),
  span_normalise = TRUE,
  windows = NULL
)

ts_f4(
  ts,
  W,
  X,
  Y,
  Z,
  mode = c("site", "branch", "node"),
  span_normalise = TRUE,
  windows = NULL
)

ts_f4ratio(
  ts,
  X,
  A,
  B,
  C,
  O,
  mode = c("site", "branch"),
  span_normalise = TRUE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`mode`	The mode for the calculation ("sites" or "branch")
`span_normalise`	Divide the result by the span of the window? Default TRUE, see the tskit documentation for more detail.

`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) do not have to be specified as they are added automatically.
`W, X, Y, Z, A, B, C, O`	Character vectors of individual names (following the nomenclature of Patterson et al. 2021)

Value

Data frame with statistics calculated for the given sets of individuals

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# calculate f2 for two individuals in a previously loaded tree sequence
ts_f2(ts, A = "AFR_1", B = "EUR_1")

# calculate f2 for two sets of individuals
ts_f2(ts, A = c("AFR_1", "AFR_2"), B = c("EUR_1", "EUR_3"))

# calculate f3 for two individuals in a previously loaded tree sequence
ts_f3(ts, A = "EUR_1", B = "AFR_1", C = "NEA_1")

# calculate f3 for two sets of individuals
ts_f3(ts, A = c("AFR_1", "AFR_2", "EUR_1", "EUR_2"),
          B = c("NEA_1", "NEA_2"),
          C = "CH_1")

# calculate f4 for single individuals
ts_f4(ts, W = "EUR_1", X = "AFR_1", Y = "NEA_1", Z = "CH_1")

# calculate f4 for sets of individuals
ts_f4(ts, W = c("EUR_1", "EUR_2"),
          X = c("AFR_1", "AFR_2"),
          Y = "NEA_1",
          Z = "CH_1")

# calculate f4-ratio for a given set of target individuals X
ts_f4ratio(ts, X = c("EUR_1", "EUR_2", "EUR_4", "EUR_5"),
               A = "NEA_1", B = "NEA_2", C = "AFR_1", O = "CH_1")
```

ts_fst Calculate pairwise statistics between sets of individuals

Description

For a discussion on the difference between "site", "branch", and "node" options of the mode argument, please see the tskit documentation at https://tskit.dev/tskit/docs/stable/stats.html#sec-stats-mode.

Usage

```
ts_fst(
  ts,
  sample_sets,
  mode = c("site", "branch", "node"),
  windows = NULL,
  span_normalise = TRUE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set)
`mode`	The mode for the calculation ("sites" or "branch")
`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) do not have to be specified as they are added automatically.
`span_normalise`	Divide the result by the span of the window? Default TRUE, see the tskit documentation for more detail.

Value

For each pairwise calculation, either a single Fst value or a vector of Fst values (one for each window)

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# compute F_st between two sets of individuals in a given tree sequence ts
```

```
ts_fst(ts, sample_sets = list(afr = c("AFR_1", "AFR_2", "AFR_3"),
                              eur = c("EUR_1", "EUR_2")))
```

`ts_genotypes` *Extract genotype table from the tree sequence*

Description

Extract genotype table from the tree sequence

Usage

```
ts_genotypes(ts)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`

Value

Data frame object of the class `tibble` containing genotypes of simulated individuals in columns

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE, mutate = TRUE,
              mutation_rate = 1e-8, random_seed = 42)

# extract the genotype matrix (this could take  a long time consume lots
# of memory!)
gts <- ts_genotypes(ts)
```

`ts_load` *Load a tree sequence file produced by a given model*

Description

This function loads a tree sequence file simulated from a given slendr model. Optionally, the tree sequence can be recapitated and simplified.

Usage

```
ts_load(
  file,
  model = NULL,
  recapitate = FALSE,
  simplify = FALSE,
  mutate = FALSE,
  recombination_rate = NULL,
  mutation_rate = NULL,
  Ne = NULL,
  random_seed = NULL,
  simplify_to = NULL,
  keep_input_roots = FALSE,
  demography = NULL
)
```

Arguments

`file`	A path to the tree-sequence file (either originating from a slendr model or a standard non-slendr tree sequence)
`model`	Optional `slendr_model` object which produced the tree-sequence `file`. Used for adding various annotation data and metadata to the standard tskit tree-sequence object.
`recapitate`	Should the tree sequence be recapitated?
`simplify`	Should the tree sequence be simplified down to a set of sampled individuals (those explicitly recorded)?
`mutate`	Should the tree sequence be mutated?
`recombination_rate, Ne`	Arguments passed to `ts_recapitate`
`mutation_rate`	Mutation rate passed to `ts_mutate`
`random_seed`	Random seed passed to pyslim's `recapitate` method
`simplify_to`	A character vector of individual names. If NULL, all remembered individuals will be retained. Only used when `simplify` = `TRUE`.
`keep_input_roots`	Should the history ancestral to the MRCA of all samples be retained in the tree sequence? Default is `FALSE`.
`demography`	Ancestral demography to be passed internally to `msprime.sim_ancestry()` (see msprime's documentation for mode detail)

Details

The loading, recapitation and simplification is performed using the Python module pyslim which serves as a link between tree sequences generated by SLiM and the tskit module for manipulation of tree sequence data. All of these steps have been modelled after the official pyslim tutorial and documentation available at: `https://tskit.dev/pyslim/docs/latest/tutorial.html`.

The recapitation and simplification steps can also be performed individually using the functions `ts_recapitate` and `ts_simplify`.

Value

Tree-sequence object of the class `slendr_ts`, which serves as an interface point for the Python module tskit using slendr functions with the `ts_` prefix.

See Also

`ts_nodes` for extracting useful information about individuals, nodes, coalescent times and geospatial locations of nodes on a map

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load tree sequence generated by a given model
ts <- ts_load(slendr_ts, model)

# even tree sequences generated by non-slendr models can be
msprime_ts <- system.file("extdata/models/msprime.trees", package = "slendr")
ts <- ts_load(msprime_ts)

# load tree sequence and immediately simplify it only to sampled individuals
# (note that the example tree sequence is already simplified so this operation
# does not do anything in this case)
ts <- ts_load(slendr_ts, model = model, simplify = TRUE)

# load tree sequence and simplify it to a subset of sampled individuals
ts_small <- ts_simplify(ts, simplify_to = c("CH_1", "NEA_1", "NEA_2",
                                            "AFR_1", "AFR_2", "EUR_1", "EUR_2"))

# load tree sequence, recapitate it and simplify it
ts <- ts_load(slendr_ts, model, recapitate = TRUE, simplify = TRUE,
              recombination_rate = 1e-8, Ne = 10000, random_seed = 42)

# load tree sequence, recapitate it, simplify it and overlay neutral mutations
ts <- ts_load(slendr_ts, model, recapitate = TRUE, simplify = TRUE, random_seed = 42,
              recombination_rate = 1e-8, Ne = 10000, mutation_rate = 1e-8)

ts
```

`ts_metadata` *Extract list with tree sequence metadata saved by SLiM*

Description

Extract list with tree sequence metadata saved by SLiM

Usage

```
ts_metadata(ts)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`

Value

List of metadata fields extracted from the tree-sequence object

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model)

# extract the list of metadata information from the tree sequence
ts_metadata(ts)
```

`ts_mutate` *Add mutations to the given tree sequence*

Description

Add mutations to the given tree sequence

Usage

```
ts_mutate(
  ts,
  mutation_rate,
  random_seed = NULL,
  keep_existing = TRUE,
  mut_type = NULL
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`mutation_rate`	Mutation rate used by msprime to simulate mutations
`random_seed`	Random seed passed to msprime's `mutate` method
`keep_existing`	Keep existing mutations?

`mut_type`	Assign SLiM mutation type to neutral mutations? If `NULL` (default), no special mutation type will be used. If an integer number is given, mutations of the SLiM mutation type with that integer identifier will be created.

Value

Tree-sequence object of the class `slendr_ts`, which serves as an interface point for the Python module tskit using slendr functions with the `ts_` prefix.

See Also

`ts_nodes` for extracting useful information about individuals, nodes, coalescent times and geospatial locations of nodes on a map

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

ts <- ts_load(slendr_ts, model)
ts_mutate <- ts_mutate(ts, mutation_rate = 1e-8, random_seed = 42)

ts_mutate
```

`ts_nodes` *Extract combined annotated table of individuals and nodes*

Description

This function combines information from the table of individuals and table of nodes into a single data frame which can be used in downstream analyses.

Usage

```
ts_nodes(x, sf = TRUE)
```

Arguments

`x`	Tree sequence object of the class `slendr_ts` or a `phylo` object extracted by `ts_phylo`
`sf`	Should spatial data be returned in an sf format? If `FALSE`, spatial geometries will be returned simply as x and y columns, instead of the standard POINT data type.

Details

The source of data (tables of individuals and nodes recorded in the tree sequence generated by SLiM) are combined into a single data frame. If the model which generated the data was spatial, coordinates of nodes (which are pixel-based by default because SLiM spatial simulations occur on a raster), the coordinates are automatically converted to an explicit spatial object of the `sf` class unless `spatial = FALSE`. See `https://r-spatial.github.io/sf/` for an extensive introduction to the sf package and the ways in which spatial data can be processed, analysed, and visualised.

Value

Data frame with processed information from the tree sequence object. If the model which generated this data was spatial, result will be returned as a spatial object of the class `sf`.

See Also

`ts_table` for accessing raw tree sequence tables without added metadata annotation. See also `ts_ancestors` to learn how to extract information about relationship beteween nodes in the tree sequence, and how to analysed data about distances between nodes in the spatial context.

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract an annotated table with (spatio-)temporal node information
ts_nodes(ts)
```

`ts_phylo` *Convert a tree in the tree sequence to an object of the class* `phylo`

Description

Convert a tree in the tree sequence to an object of the class `phylo`

Usage

```
ts_phylo(
  ts,
  i,
  mode = c("index", "position"),
  labels = c("tskit", "pop"),
  quiet = FALSE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`i`	Position of the tree in the tree sequence. If `mode = "index"`, an i-th tree will be returned (in one-based indexing), if `mode = "position"`, a tree covering an i-th base of the simulated genome will be returned.
`mode`	How should the `i` argument be interpreted? Either "index" as an i-th tree in the sequence of genealogies, or "position" along the simulated genome.
`labels`	What should be stored as node labels in the final `phylo` object? Options are either a population name or a tskit integer node ID (which is a different thing from a `phylo` class node integer index).
`quiet`	Should ape's internal phylo validity test be printed out?

Value

Standard phylogenetic tree object implemented by the R package ape

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract the 1st tree from a given tree sequence, return ape object
tree <- ts_phylo(ts, i = 1, mode = "index", quiet = TRUE)
tree

# extract the tree at a 42th basepair in the given tree sequence
tree <- ts_phylo(ts, i = 42, mode = "position", quiet = TRUE)

# because the tree is a standard ape phylo object, we can plot it easily
plot(tree, use.edge.length = FALSE)
ape::nodelabels()
```

`ts_recapitate` *Recapitate the tree sequence*

Description

Recapitate the tree sequence

Usage

```
ts_recapitate(
  ts,
  recombination_rate,
  Ne = NULL,
  demography = NULL,
  random_seed = NULL
)
```

Arguments

`ts`	Tree sequence object loaded by `ts_load`
`recombination_rate`	A constant value of the recombination rate
`Ne`	Effective population size during the recapitation process
`demography`	Ancestral demography to be passed internally to `msprime.sim_ancestry()` (see msprime's documentation for mode detail)
`random_seed`	Random seed passed to pyslim's `recapitate` method

Value

Tree-sequence object of the class `slendr_ts`, which serves as an interface point for the Python module tskit using slendr functions with the `ts_` prefix.

See Also

`ts_nodes` for extracting useful information about individuals, nodes, coalescent times and geospatial locations of nodes on a map

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

ts <-
  ts_load(slendr_ts, model) %>%
  ts_recapitate(recombination_rate = 1e-8, Ne = 10000, random_seed = 42)

ts
```

`ts_samples` *Extract names and times of individuals of interest in the current tree sequence (either all sampled individuals or those that the user simplified to)*

Description

Extract names and times of individuals of interest in the current tree sequence (either all sampled individuals or those that the user simplified to)

Usage

```
ts_samples(ts)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`

Value

Table of individuals scheduled for sampling across space and time

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract the table of individuals scheduled for simulation and sampling
ts_samples(ts)
```

`ts_save` *Save a tree sequence to a file*

Description

Save a tree sequence to a file

Usage

```
ts_save(ts, file)
```

Arguments

`ts`	Tree sequence object loaded by `ts_load`
`file`	File to which the tree sequence should be saved

Value

No return value, called for side effects

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree sequence
ts <- ts_load(slendr_ts, model)

# save the tree-sequence object to a different location
another_file <- paste(tempfile(), ".trees")
ts_save(ts, another_file)
```

`ts_segregating` *Calculate the density of segregating sites for the given sets of individuals*

Description

Calculate the density of segregating sites for the given sets of individuals

Usage

```
ts_segregating(
  ts,
  sample_sets,
  mode = c("site", "branch", "node"),
  windows = NULL,
  span_normalise = FALSE
)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set). If a simple vector is provided, it will be interpreted as `as.list(sample_sets)`, meaning that a given statistic will be calculated for each individual separately.
`mode`	The mode for the calculation ("sites" or "branch")

`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) are added automatically)
`span_normalise`	Divide the result by the span of the window? Default TRUE, see the tskit documentation for more detail.

Value

For each set of individuals either a single diversity value or a vector of diversity values (one for each window)

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# collect sampled individuals from all populations in a list
sample_sets <- ts_samples(ts) %>%
  split(., .$pop) %>%
  lapply(function(pop) pop$name)

ts_segregating(ts, sample_sets)
```

`ts_simplify` *Simplify the tree sequence down to a given set of individuals*

Description

This function is a convenience wrapper around the `simplify` method implemented in tskit, designed to work on tree sequence data simulated by SLiM using the **slendr** R package.

Usage

```
ts_simplify(ts, simplify_to = NULL, keep_input_roots = FALSE)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`simplify_to`	A character vector of individual names. If NULL, all explicitly remembered individuals (i.e. those specified via the `schedule_sampling` function will be left in the tree sequence after the simplification.
`keep_input_roots`	Should the history ancestral to the MRCA of all samples be retained in the tree sequence? Default is `FALSE`.

Details

The simplification process is used to remove redundant information from the tree sequence and retains only information necessary to describe the genealogical history of a set of samples.

For more information on how simplification works in pyslim and tskit, see the official documentation at `https://tskit.dev/tskit/docs/stable/python-api.html#tskit.TreeSequence.simplify` and `https://tskit.dev/pyslim/docs/latest/tutorial.html#simplification`.

A very clear description of the difference between remembering and retaining and how to use these techniques to implement historical individuals (i.e. ancient DNA samples) is in the pyslim documentation at `https://tskit.dev/pyslim/docs/latest/tutorial.html#historical-individuals`.

Value

Tree-sequence object of the class `slendr_ts`, which serves as an interface point for the Python module tskit using slendr functions with the `ts_` prefix.

See Also

`ts_nodes` for extracting useful information about individuals, nodes, coalescent times and geospatial locations of nodes on a map

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

ts <- ts_load(slendr_ts, model)
ts

# simplify tree sequence to sampled individuals
ts_simplified <- ts_simplify(ts)

# simplify to a subset of sampled individuals
ts_small <- ts_simplify(
  ts,
  simplify_to = c("CH_1", "NEA_1", "NEA_2", "AFR_1",
                  "AFR_2", "EUR_1", "EUR_2")
)

ts_small
```

`ts_table` *Get the table of individuals/nodes/edges/mutations from the tree sequence*

Description

This function extracts data from a given tree sequence table. All times are converted to model-specific time units from tskit's "generations backwards" time direction.

Usage

```
ts_table(ts, table = c("individuals", "edges", "nodes", "mutations"))
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`table`	Which tree sequence table to return

Details

For further processing and analyses, the output of the function `ts_nodes` might be more useful, as it merges the information in node and individual tables into one table and further annotates it with useful information from the model configuration data.

Value

Data frame with the information from the give tree-sequence table (can be either a table of individuals, edges, nodes, or mutations).

See Also

`ts_nodes` and `ts_edges` for accessing an annotated, more user-friendly and analysis-friendly tree-sequence table data

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE, mutate = TRUE,
              mutation_rate = 1e-8, random_seed = 42)

# get the 'raw' tskit table of individuals
ts_table(ts, "individuals")

# get the 'raw' tskit table of edges
ts_table(ts, "edges")

# get the 'raw' tskit table of nodes
ts_table(ts, "nodes")

# get the 'raw' tskit table of mutations
ts_table(ts, "mutations")
```

`ts_tajima` *Calculate Tajima's D for given sets of individuals*

Description

For a discussion on the difference between "site" and "branch" options of the `mode` argument, please see the tskit documentation at https://tskit.dev/tskit/docs/stable/stats.html#sec-stats-mode

Usage

```
ts_tajima(ts, sample_sets, mode = c("site", "branch", "node"), windows = NULL)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`sample_sets`	A list (optionally a named list) of character vectors with individual names (one vector per set). If a simple vector is provided, it will be interpreted as `as.list(sample_sets)`, meaning that a given statistic will be calculated for each individual separately.
`mode`	The mode for the calculation ("sites" or "branch")
`windows`	Coordinates of breakpoints between windows. The first coordinate (0) and the last coordinate (equal to `ts$sequence_length`) are added automatically)

Value

For each set of individuals either a single Tajima's D value or a vector of Tajima's D values (one for each window)

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, mutate = TRUE, mutation_rate = 1e-8, random_seed = 42)

# calculate Tajima's D for given sets of individuals in a tree sequence ts
ts_tajima(ts, list(eur = c("EUR_1", "EUR_2", "EUR_3", "EUR_4", "EUR_5"),
                   nea = c("NEA_1", "NEA_2")))
```

`ts_tree` *Get a tree from a given tree sequence*

Description

For more information about optional keyword arguments see tskit documentation: `https://tskit.dev/tskit/docs/stable/python-api.html#the-treesequence-class`

Usage

```
ts_tree(ts, i, mode = c("index", "position"), ...)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`i`	Position of the tree in the tree sequence. If `mode = "index"`, an i-th tree will be returned (in one-based indexing), if `mode = "position"`, a tree covering an i-th base of the simulated genome will be returned.
`mode`	How should the `i` argument be interpreted? Either "index" as an i-th tree in the sequence of genealogies, or "position" along the simulated genome.
`...`	Additional keyword arguments accepted by `tskit.TreeSequence.at` and `tskit.TreeSequence.at_in` methods

Value

Python-reticulate-based object of the class tskit.trees.Tree

Examples

```
# load an example model with an already simulated tree sequence
slendr_ts <- system.file("extdata/models/introgression.trees", package = "slendr")
model <- read_model(path = system.file("extdata/models/introgression", package = "slendr"))

# load the tree-sequence object from disk
ts <- ts_load(slendr_ts, model, simplify = TRUE)

# extract the first tree in the tree sequence
tree <- ts_tree(ts, i = 1)

# extract the tree at a position 100000bp in the tree sequence
tree <- ts_tree(ts, i = 100000, mode = "position")
```

`ts_vcf` *Save genotypes from the tree sequence as a VCF file*

Description

Save genotypes from the tree sequence as a VCF file

Usage

```
ts_vcf(ts, path, chrom = NULL, individuals = NULL)
```

Arguments

`ts`	Tree sequence object of the class `slendr_ts`
`path`	Path to a VCF file
`chrom`	Chromosome name to be written in the CHROM column of the VCF
`individuals`	A character vector of individuals in the tree sequence. If missing, all individuals present in the tree sequence will be saved.

Value

No return value, called for side effects

`world` *Define a world map for all spatial operations*

Description

Defines either an abstract geographic landscape (blank or containing user-defined landscape) or using a real Earth cartographic data from the Natural Earth project (`https://www.naturalearthdata.com`).

Usage

```
world(
  xrange,
  yrange,
  landscape = "naturalearth",
  crs = NULL,
  scale = c("small", "medium", "large")
)
```

Arguments

`xrange`	Two-dimensional vector specifying minimum and maximum horizontal range ("longitude" if using real Earth cartographic data)
`yrange`	Two-dimensional vector specifying minimum and maximum vertical range ("latitude" if using real Earth cartographic data)
`landscape`	Either "blank" (for blank abstract geography), "naturalearth" (for real Earth geography) or an object of the class `sf` defining abstract geographic features of the world
`crs`	EPSG code of a coordinate reference system to use for spatial operations. No CRS is assumed by default (`NULL`), implying an abstract landscape not tied to any real-world geographic region (when `landscape = "blank"` or when `landscape` is a custom-defined geographic landscape), or implying WGS-84 (EPSG 4326) coordinate system when a real Earth landscape was defined (`landscape = "naturalearth"`).
`scale`	If Natural Earth geographic data is used (i.e. `landscape = "naturalearth"`), this parameter determines the resolution of the data used. The value "small" corresponds to 1:110m data and is provided with the package, values "medium" and "large" correspond to 1:50m and 1:10m respectively and will be downloaded from the internet. Default value is "small".

Value

Object of the class `slendr_map`, which encodes a standard spatial object of the class `sf` with additional slendr-specific attributes such as requested x-range and y-range.

Examples

```
# create a blank abstract world 1000x1000 distance units in size
blank_map <- world(xrange = c(0, 1000), yrange = c(0, 1000), landscape = "blank")

# it is possible to construct custom landscapes (islands, corridors, etc.)
island1 <- region("island1", polygon = list(c(10, 30), c(50, 30), c(40, 50), c(0, 40)))
island2 <- region("island2", polygon = list(c(60, 60), c(80, 40), c(100, 60), c(80, 80)))
island3 <- region("island3", center = c(20, 80), radius = 10)
archipelago <- island1 %>% join(island2) %>% join(island3)

custom_map <- world(xrange = c(1, 100), c(1, 100), landscape = archipelago)

# real Earth landscapes can be defined using freely-available Natural Earth
# project data and with the possibility to specify an appropriate Coordinate
# Reference System, such as this example of a map of Europe

real_map <- world(xrange = c(-15, 40), yrange = c(30, 60), crs = "EPSG:3035")
```

Index

animate_model, 3
area, 4
as.phylo.slendr_phylo, 5

check_dependencies, 5
check_env, 6
clear_env, 6
compile_model, 7

distance, 9

expand_range, 10
explore_model, 12

gene_flow, 13

join, 15

move, 16
msprime, 18

overlap, 19

plot_map, 20
plot_model, 21
population, 22
print.slendr_map *(print.slendr_pop)*, 24
print.slendr_model *(print.slendr_pop)*, 24
print.slendr_nodes *(print.slendr_pop)*, 24
print.slendr_pop, 24
print.slendr_region *(print.slendr_pop)*, 24
print.slendr_ts, 25

read_model, 26
region, 26
reproject, 27
resize, 28

schedule_sampling, 30, *64*
set_dispersal, 32
set_range, 34
setup_env, 32
shrink_range, 37
slim, 39
subtract, 41

ts_afs, 42
ts_ancestors, 43, *59*
ts_coalesced, 44
ts_descendants, 45
ts_divergence, 46
ts_diversity, 47
ts_draw, 48
ts_edges, 49, *66*
ts_eigenstrat, 50
ts_f2, 50
ts_f3 *(ts_f2)*, 50
ts_f4 *(ts_f2)*, 50
ts_f4ratio *(ts_f2)*, 50
ts_fst, 53
ts_genotypes, 54
ts_load, 54
ts_metadata, 56
ts_mutate, 57
ts_nodes, *56*, *58*, 58, *61*, *65*, *66*
ts_phylo, 59
ts_recapitate, *55*, 60
ts_samples, 62
ts_save, 62
ts_segregating, 63
ts_simplify, *55*, 64
ts_table, *59*, 65
ts_tajima, 67
ts_tree, *48*, 68
ts_vcf, 69

world, 69