Package ‘gen3sis’

July 10, 2020

Type Package

Title General Engine for Eco-Evolutionary Simulations

Version 1.0

Description Contains an engine for spatially-explicit eco-evolutionary mechanistic models with a modular implementation and several support functions. It allows exploring the consequences of ecological and macroevolutionary processes across realistic or theoretical spatio-temporal landscapes on biodiversity patterns as a general term.

License GPL-3

Encoding UTF-8

LazyData true

Depends R (>= 3.5.0)

Imports Rcpp (>= 0.12.16), Matrix, methods, utils, raster, gdistance, sp, stringr, tools, grDevices

Suggests knitr, testthat (>= 2.1.0), rmarkdown

LinkingTo Rcpp, BH

URL `https://github.com/project-Gen3sis/R-package`

BugReports `https://github.com/project-Gen3sis/R-package/issues`

RoxygenNote 7.1.1

VignetteBuilder knitr

SystemRequirements C++11

NeedsCompilation yes

Author ETH Zürich [cph],
Oskar Hagen [aut, cre] (Landscape Ecology, WSL and ETH Zürich, Switzerland),
Benjamin Flueck [aut] (Landscape Ecology, WSL and ETH Zürich, Switzerland),
Fabian Fopp [aut] (Landscape Ecology, WSL and ETH Zürich, Switzerland),
Juliano S. Cabral [aut] (Ecosystem Modeling, Center for Computational and Theoretical Biology, University of Würzburg, Würzburg, Germany),
Florian Hartig [aut] (Theoretical Ecology, University of Regensburg,

Regensburg, Germany),
Mikael Pontarp [aut] (Department of Biology, Lund University, Lund, Sweden),
Charles Novaes de Santana [ctb] (Landscape Ecology, WSL and ETH Zürich, Switzerland),
Thiago F. Rangel [aut] (Department of Ecology, Universidade Federal de Goiás, Goiás, Brazil),
Theo Gaboriau [ctb] (Depatment of Computational Biology, Lausanne University, Switzerland),
Loïc Pellissier [aut, ths] (Landscape Ecology, WSL and ETH Zürich, Switzerland)

Maintainer Oskar Hagen <oskar@hagen.bio>

Repository CRAN

Date/Publication 2020-07-10 16:40:02 UTC

R topics documented:

`apply_ecology` *apply_ecology allows the user to define the ecology of the species that takes place within each cell, which define species survival and species abundance.*

Description

apply_ecology allows the user to define the ecology of the species that takes place within each cell, which define species survival and species abundance.

Usage

```
apply_ecology(abundance, traits, local_environment, config)
```

Arguments

`abundance`	a named vector of abundances with one abundance value per species
`traits`	a named matrix containg the species traits, one row per species
`local_environment`	the environmental values for the given cell
`config`	the config of the simulation

Details

The arguments of the function allows to apply abiotic and biotic ecological rules to species in each cell. Based on those rules, the function updates the abundance of each species in each cell. If the abundance is null, the species is absent or extinct. Ecology can account for local environmental conditions, the abundance of species, or their traits.

Value

an abundance vector with the new abundace values for every species. An abundance value of 0 indicates species death, any other values survival.

`apply_evolution`	*apply_evolution allows defining the function that changes the values of traits of a given species at each time step and in each cell. If no operations are provided, traits are not changing.*

Description

apply_evolution allows defining the function that changes the values of traits of a given species at each time step and in each cell. If no operations are provided, traits are not changing.

Usage

```
apply_evolution(species, cluster_indices, landscape, config)
```

Arguments

`species`	The target species object whose traits will be changed
`cluster_indices`	an index vector indicating the cluster every occupied cell is part of
`landscape`	the current landscape which can co-determine the rate of trait changes
`config`	the current config

Details

This function is called for any single species alongside an index for the geographical clusters within the species.

Value

the mutated species traits matrix

`color_richness`	*define gen3sis richness color scale*

Description

define gen3sis richness color scale

Usage

```
color_richness(n)
```

Arguments

`n`	corresponds to the colorRampPalette parameter

Value

returns a colorRampPalette function with the gen3sis richness colours

---

`create_ancestor_species` *The function create_ancestor_species allows the user to populate the world at the beginning of a simulation.*

Description

The function create_ancestor_species allows the user to populate the world at the beginning of a simulation.

Usage

```
create_ancestor_species(landscape, config)
```

Arguments

`landscape`	the landscape over which to create the species
`config`	the configuration information

Details

Using this function any number of new species can be created. For every species, a number of habitable cells from the landscape are selected and call 'create_species'. In another step, the user must initialize the species[["traits"]] matrix with the desired initial traits values.

Value

a list of species

`create_input_config` *creates either an empty configuration or a pre-filled configuration object from a config file.*

Description

creates either an empty configuration or a pre-filled configuration object from a config file.

Usage

```
create_input_config(config_file = NA)
```

Arguments

`config_file`	the path to a valid configuration file. if NA it creates an empty config.

Value

list of configuration elements, similar generated from reading a config_file.R. The internal elements of this list are: "general", "initialization", "dispersal", "speciation", "mutation" and "ecology".

Examples

```
# create empty config object
config_empty <- create_input_config(config_file = NA)

# create a config object from config_file
# get path to example config
datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")
path_config <- file.path(datapath, "config/config_worldcenter.R")
config_object <- create_input_config(config_file = path_config)

# change seed of config_worldcenter config object
config_object$gen3sis$general$random_seed <- 2020

# run the model for config_object

  sim <- run_simulation(config = config_object,
                        landscape = file.path(datapath, "landscape"),
                        output_directory = tempdir())
```

`create_input_landscape` *create an landscape input from a named list of rasters or raster files*

Description

create an landscape input from a named list of rasters or raster files

Usage

```
create_input_landscape(
  landscapes,
  cost_function,
  directions,
  output_directory,
  timesteps = NULL,
  calculate_full_distance_matrices = FALSE,
  crs = NULL,
  overwrite_output = FALSE,
  verbose = FALSE
)
```

Arguments

`landscapes`	list of named list(s) of raster(s) or raster file(s) name(s). Starting from the present towards the past. The list names are important since these are the environmental names
`cost_function`	function that returns a cost value between a pair of cell (neighbours) that should have the following signature: #' cost_fuinction <- function(src, src_habitable, dest, dest_habitable) #rules for envoriolmental factors to be considered (e.g. elevation) #return(cost value) where src is a vector of environmental conditions for the origin cell, src_habitable (TRUE or FALSE) for habitable condition of the origin cell, dest is a vector of environmental conditions for the destination cell, dest_habitable (TRUE or FALSE) for habitable condition of the destination cell
`directions`	4, 8, 16 oder adjacency matrix (see gdistance package)
`output_directory`	path for storing the gen3sis ready landscape
`timesteps`	vector of names for every time-step to represent the time-step at gen3sis ready landscape. if NULL, tim-steps are sequentially numbered from 0 (present)
`calculate_full_distance_matrices`	TRUE or FALSE. If TRUE calculates the entire distance matrix for every time-step and between all habitable cells (faster CPU time, higher storage required). If FALSE, only local distances are calculated (slower CPU time when simulating but smaller gen3sis landscape size)
`crs`	the coordinate reference system in crs format (see rater::crs)
`overwrite_output`	TRUE or FALSE
`verbose`	print distance calculation progress (default: FALSE)

Value

no return object. This function saves the landscape input files for gen3sis at the output_directory

Examples

```
# load needed library
library(raster)

# get path containing example rasters
datapath <- system.file(file.path("extdata", "InputRasters"), package="gen3sis")

# create raster bricks
temperature_brick <- brick(file.path(datapath, "WorldCenter/temp_rasters.grd"))
aridity_brick <-  brick(file.path(datapath, "WorldCenter/aridity_rasters.grd"))
area_brick <-  brick(file.path(datapath, "WorldCenter/area_rasters.grd"))

# create sub-list of environmental variables for fast example
# (i.e. 10 time-steps)
```

```
landscapes_sub_list <- list(temp=NULL, arid=NULL, area=NULL)
for(i in 1:10){
  landscapes_sub_list$temp <- c(landscapes_sub_list$temp, temperature_brick[[i]])
  landscapes_sub_list$arid <- c(landscapes_sub_list$arid, aridity_brick[[i]])
  landscapes_sub_list$area <- c(landscapes_sub_list$area, area_brick[[i]])
}

# define cost function, crossing water as double as land sites
cost_function_water <- function(source, habitable_src, dest, habitable_dest) {
  if(!all(habitable_src, habitable_dest)) {
    return(2)
  } else {
    return(1)
  }
}

# create input landscape ready for gen3sis from sub-list
# (i.e. 10 time-steps) and only local-distances.
create_input_landscape(
  landscapes = landscapes_sub_list,
  cost_function = cost_function_water,
  output_directory = file.path(tempdir(), "landscape_sub"),
  directions = 8, # surrounding sites for each site
  timesteps = paste0(round(seq(150, 140, length.out = 10),2), "Ma"),
  calculate_full_distance_matrices = FALSE) # full distance matrix



# create list of all environmental variables available
landscapes_list <- list(temp=NULL, arid=NULL, area=NULL)
for(i in 1:nlayers(temperature_brick)){
  landscapes_list$temp <- c(landscapes_list$temp, temperature_brick[[i]])
  landscapes_list$arid <- c(landscapes_list$arid, aridity_brick[[i]])
  landscapes_list$area <- c(landscapes_list$area, area_brick[[i]])
}

# create input landscape ready for gen3sis (~ 3min run-time)
# and full distance matrix
create_input_landscape(
  landscapes = landscapes_list,
  cost_function = cost_function_water,
  output_directory = file.path(tempdir(), "landscape3"),
  directions = 8, # surrounding sites for each site
  timesteps = paste0(round(seq(150, 100, length.out = 301),2), "Ma"),
  calculate_full_distance_matrices = FALSE) # full distance matrix
```

`create_species` *creates a new species.*

Description

creates a new species.

Usage

```
create_species(initial_cells, config)
```

Arguments

`initial_cells`	a list of initial cells (strings) to occupy
`config`	the configuration information

Details

This function is to be used in the create_ancestor_species function at the configuration of a simulation. It will create a species object representing one species in the simulation occupying the given list of initial cells.

Value

returns a newly created species occupying the provided initial cells

Examples

```
## Not run:
# inside a create_ancestor_species function of a config taking a landscape and a config
# create_species creates a new species

# define range of species for the entire world in this case lat long system
range <- c(-180, 180, -90, 90)

## select coordinates within the range stipulated above
# takes landscape coordinates
co <- landscape$coordinates
# select coordinates within the range
selection <- co[, "x"] >= range[1] &
  co[, "x"] <= range[2] &
  co[, "y"] >= range[3] &
  co[, "y"] <= range[4]
# get the initial cells
initial_cells <- rownames(co)[selection]

# call create_species
new_species <- create_species(initial_cells, config)

# extra: set local adaptation to max optimal temp equals local temp
new_species$traits[ , "temp"] <- landscape$environment[,"temp"]

# extra: set a certaintrait (e.g. traitX) to one on all populations of this species
new_species$traits[ , "tratiX"] <- 1
```

```
## End(Not run)
```

`evolution_mode_none` *evolution_mode_none*

Description

evolution_mode_none

Usage

```
evolution_mode_none(species, cluster_indices, landscape, config)
```

Arguments

`species`	the current species
`cluster_indices`	indices to assign cells to geographic clusters
`landscape`	the current landscape
`config`	the general config

`gen3sis` *gen3sis: General Engine for Eco-Evolutionary Simulations*

Description

Contains an engine for spatially-explicit eco-evolutionary mechanistic models with a modular implementation and several support functions. It allows exploring the consequences of ecological and macroevolutionary processes across realistic or theoretical spatio-temporal landscapes on biodiversity patterns as a general term.

Details

Gen3sis is implemented in a mix of R and C++ code, and wrapped into an R-package. All high-level functions that the user may interact with are written in R, and are documented via the standard R / Roxygen help files for R-packages. Runtime-critical functions are implemented in C++ and coupled to R via the Rcpp framework. Additionally, the package provides several convenience functions to generate input data, configuration files and plots, as well as tutorials in the form of vignettes that illustrate how to declare models and run simulations.

References

O. Hagen, B. Flück, F. Fopp, J.S. Cabral, F. Hartig, M. Pontarp, T.F. Rangel, L. Pellissier. (2020). gen3sis: the GENeral Engine for Eco-Evolutionary SImulationS on the origins of biodiversity. (in prep)

See Also

create_input_config create_input_landscape run_simulation plot_summary

Examples

```
## Not run:

# 1. Load gen3sis and all necessary input data is set (landscape and config).

library(gen3sis)

# get path to example input inside package
datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")
path_config <- file.path(datapath, "config/config_worldcenter.R")
path_landscape <- file.path(datapath, "landscape")

# 2. Run simulation

sim <- run_simulation(config = path_config, landscape = path_landscape)

# 3. Visualize the outputs

# plot summary of entire simulation
plot_summary(sim)

# plot richness at a given time-step
# this only works if species is saved for this time-step
landscape_t_150 <- readRDS(file.path(datapath,
"output", "config_worldcenter", "landscapes", "landscape_t_150.rds"))
species_t_150 <- readRDS(file.path(datapath,
"output", "config_worldcenter", "species", "species_t_150.rds"))
plot_richness(species_t_150, landscape_t_150)


## End(Not run)
```

---

get_dispersal_values *get_dispersal_values allows the user to generates dispersal value(s) for a given species. The simulation request the user to return a vector of dispersal values with length num_draws.*

Description

get_dispersal_values allows the user to generates dispersal value(s) for a given species. The simulation request the user to return a vector of dispersal values with length num_draws.

Usage

```
get_dispersal_values(num_draws, species, landscape, config)
```

Arguments

`num_draws`	'num_draws' the number of dispersal values drawn
`species`	'species' the species for which the values are to be produced
`landscape`	'landscape' the landscape of the current time step
`config`	'config' the config of the simulation

Details

Dispersal values are used for two different operations. First, they are used to evaluate pairwise dispersal events between colonized and uninhabited cells. Second, they are used during the geographical clustering of species populations when determining which cells are in range of each other and belong to the same geographic cluster.

num_draws tells the user how many dispersal values are requested by the simulation when this function is called and must be returned. It can be of varying length depending on the operation calling it, i.e. colonisation or geographic clustering. If the dispersal is considered as fixed the function should return a vector of length num_draws with repeated identical values, or varying values in case of more complex dispersal kernels.

Note: if the distances are randomized the cluster formation may be asymetrical. Therefore the ordering of all clustering operations is randomized.

Value

a numerical vector of length num_draws with dispersal values

`get_divergence_factor` *get_divergence_factor allows the user to define the rate at which geographic clusters accumulate differentiation with each other.*

Description

get_divergence_factor allows the user to define the rate at which geographic clusters accumulate differentiation with each other.

Usage

```
get_divergence_factor(species, cluster_indices, landscape, config)
```

Arguments

`species`	the species of the current time step
`cluster_indices`	an index vector indicating the cluster every occupied cell is part of
`landscape`	the landscape of the current time step
`config`	the config of the simulation

Details

this function determines the increase in divergence between separated clusters of a species. This function should return either a value, one there is an homogeneous divergence, or a matrix indicating the divergence that should be accumulated between specific pairwise geographic clusters.

The function can either return a single value or a full cluster by cluster matrix. If only one value is returned it will be used to increment divergence between any given distinct cluster pairs. If a matrix is returned it has to be in the dimension of cluster x cluster, in which case the divergence values will be increased according to the cluster membership of any cell pairs

For every time step, the divergence between geographic clusters can increase by a defined number. The divergence values can be scaled optionally using the species or landscape information. For instance, the divergence between clusters could be higher under warmer temperature, or difference in ecological traits could promote faster divergence between clusters.

Oppositely, for every time step, if cluster are merged their divergence is reduced by 1.

Value

a single value or a matrix of divergences between all clusters occuring in clusters_indices

---

`get_divergence_matrix` *The function get_divergence_matrix returns the full divergence matrix for a given species (cell x cell).*

Description

The function get_divergence_matrix returns the full divergence matrix for a given species (cell x cell).

Usage

```
get_divergence_matrix(species)
```

Arguments

`species`	the species for which the divergence matrix should be produced

Details

The functions allows to extract the full divergence matrix representing the accumulated differentiation between all the cells that are occupied by the species. The input is a species object for any time step.

Value

the full decompressed divergence matrix

Examples

```
# get path containing example rasters
datapath <- system.file(file.path("extdata", "WorldCenter"), package="gen3sis")
# get species at t0
species_t_0 <- readRDS(file.path(datapath, "output/config_worldcenter/species/species_t_0.rds"))
# get divergence matrix from species 1
divergence_sp1_t0 <- get_divergence_matrix(species_t_0[[1]])
# get divergence matrix from species 12
divergence_sp12_t0 <- get_divergence_matrix(species_t_0[[12]])
# note that species 1 has no divergence between it's populations, while 12 has.
```

`get_geo_richness` *calculate the richness of a list of species over a given landscape*

Description

calculate the richness of a list of species over a given landscape

Usage

```
get_geo_richness(species_list, landscape)
```

Arguments

`species_list`	a list of species to include in the richness calculations
`landscape`	the landscape to calculate the richnness over

Value

a vector with the richness for every cell in the input landscape

See Also

`plot_richness`

Examples

```
# get path containing example rasters
datapath <- system.file(file.path("extdata", "WorldCenter"), package="gen3sis")
# get species at t0
species_t_0 <- readRDS(file.path(datapath,
                        "output/config_worldcenter/species/species_t_0.rds"))
# get landscape at t0
landscape_t_0 <- readRDS(file.path(datapath,
                          "output/config_worldcenter/landscapes/landscape_t_0.rds"))
# get geo richness
richness_t_0 <- get_geo_richness(species_t_0, landscape_t_0)

# histogram of richness at t0
```

```
hist(richness_t_0)

## plot richness using raster and gen3sis color_richness (see plot_richness for alternative)
# combine richness and geographical coordinates
geo_richness_t_0 <- cbind(landscape_t_0$coordinates, richness_t_0)
library(raster)
plot(rasterFromXYZ(geo_richness_t_0), col=color_richness(20))
```

plot_landscape *plot the environment variable of a given landscape*

Description

plot the environment variable of a given landscape

Usage

```
plot_landscape(landscape)
```

Arguments

landscape	the landscape to plot the environment from

plot_landscape_overview *plot the outline of a given landscape over time*

Description

plot the outline of a given landscape over time

Usage

```
plot_landscape_overview(landscape, slices = 2, start_end_times = NULL)
```

Arguments

landscape	the input landscape to plot
slices	the amount of slices though time between start and end (default value is 2).
start_end_times	the stating and ending times of the simulation (default is NULL, takes the oldest and most recent available)

`plot_raster_multiple` *plot a set of values onto a given landscape*

Description

plot a set of values onto a given landscape

Usage

```
plot_raster_multiple(values, landscape, no_data = 0)
```

Arguments

`values`	a matrix of values with columns coresponding to sets of values, and rows corresponding to grid cells, this will result in ncol(values) raster plots.
`landscape`	a landscape to plot the values onto
`no_data`	what value should be used for missing values in values

`plot_raster_single` *plot a single set of values onto a given landscape*

Description

plot a single set of values onto a given landscape

Usage

```
plot_raster_single(values, landscape, title, no_data = 0, col)
```

Arguments

`values`	a named list of values, the names must correspond to cells in the landscape
`landscape`	a landscape to plot the values onto
`title`	a title string for resulting plot, the time will be taken and appended from the landscape id
`no_data`	what value should be used for missing values in values
`col`	corresponds to the raster col plot parameter. This can be omitted and colors handled by raster::plot

Examples

```
# get path to output objects
datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")

# plot environmental variables at a given step
landscape_t_150 <- readRDS(
  file.path(datapath, "output", "config_worldcenter", "landscapes", "landscape_t_150.rds"))
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,2))
plot_raster_single(landscape_t_150$environment[,"temp"], landscape_t_150, "Temperature", NA)
# use col to change the color
plot_raster_single(landscape_t_150$environment[,"prec"], landscape_t_150, "Aridity", NA,
                   col=topo.colors(5))
par(oldpar)
# note that these values were scaled by the configuration object
```

## `plot_richness` *plot the richness of the given list of species on a landscape*

Description

plot the richness of the given list of species on a landscape

Usage

```
plot_richness(species_list, landscape)
```

Arguments

`species_list`	a list of species to use in the richness calculation
`landscape`	a landscape to plot the richness onto

Examples

```
## plot from saved outputs
# get path containing example rasters
datapath <- system.file(file.path("extdata", "WorldCenter"), package="gen3sis")
# get species at t0
species_t_0 <- readRDS(file.path(datapath,
                        "output/config_worldcenter/species/species_t_0.rds"))
# get landscape at t0
landscape_t_0 <- readRDS(file.path(datapath,
                          "output/config_worldcenter/landscapes/landscape_t_0.rds"))
# plot richness
plot_richness(species_t_0, landscape_t_0)


## plot from within observer
# call plot_richness from inside the end_of_timestep_observer function
# at the config file:
## Not run:
plot_richness(data$all_species, data$landscape)

## End(Not run)
```

---

plot_species_presence *plot a species' presence on a given landscape*

Description

plot a species' presence on a given landscape

Usage

```
plot_species_presence(species, landscape)
```

Arguments

species	a single species
landscape	a landscape

Examples

```
# get path to output objects
datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")

# load landscape and species at time step zero
landscape_t_0 <- readRDS(
  file.path(datapath, "output/config_worldcenter", "landscapes", "landscape_t_0.rds"))
species_t_0 <- readRDS(
  file.path(datapath, "output/config_worldcenter", "species", "species_t_0.rds"))

# plot species 21 range
plot_species_presence(species_t_0[[21]], landscape_t_0)
# oh, a South American one!

# plot ranges of 3 species (i.e. 1, 21 and 32)
oldpar <- par(no.readonly = TRUE)
par(mfrow=c(1,3))
plot_species_presence(species_t_0[[1]], landscape_t_0)
plot_species_presence(species_t_0[[21]], landscape_t_0)
plot_species_presence(species_t_0[[32]], landscape_t_0)
par(oldpar)
```

plot_summary *plot simulation default summary object*

Description

plot simulation default summary object

Usage

```
plot_summary(output, summary_title = NULL, summary_legend = NULL)
```

Arguments

`output`	tsgen3sis output object resulting from a gen3sis simulation
`summary_title`	summary plot title as character. If NULL, title computed from input name.
`summary_legend`	either a staring with __n for new lines or NULL. If NULL, provides default summary and simulation information.

See Also

run_simulation

Examples

```
# load existing summary example
datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")
output <- readRDS(file.path(datapath, "output/config_worldcenter/sgen3sis.rds"))
# plot output summary
plot_summary(output)


## run simulation and plot summary
# get path or correct input objects
datapath <- system.file(file.path("extdata", "CaseStudy1"), package="gen3sis")
# run simulation and store summary object to output
output <- run_simulation(config = file.path(datapath,"config/config_fast.R"),
                      landscape = file.path(datapath,"landscape"),
                      output_directory = tempdir())
# plot output summary
plot_summary(output)
```

prepare_directories *checks if the necessary directories exist, and otherwise creates them. This function will be called by the simulation, but is made available if the directories should be created manually beforehand, for example to redirect the stdout to a file in the output directory.*

Description

checks if the necessary directories exist, and otherwise creates them. This function will be called by the simulation, but is made available if the directories should be created manually beforehand, for example to redirect the stdout to a file in the output directory.

Usage

```
prepare_directories(
  config_file = NA,
  input_directory = NA,
  output_directory = NA
)
```

Arguments

`config_file`	path to the config file, if NA the default config will be used
`input_directory`	path to input directory, if NA it will be derived from the config file path
`output_directory`	path to output directory, if NA it will be derived from the config file path

Value

returns a named list with the paths for the input and output directories

Examples

```
## Not run:
  # this is an internal function used to attribute directories by deduction
  # called at the start of a simulation run
  datapath <- system.file(file.path("extdata", "WorldCenter"), package = "gen3sis")
  # deducing input directory and setting output directory
  prepare_directories(config_file = file.path(datapath, "config/config_worldcenter.R"))
  # setting output directory
  prepare_directories(config_file = file.path(datapath, "config/config_worldcenter.R"),
                      input_directory = file.path(datapath, "landscape"))

## End(Not run)
```

`run_simulation`	*run a simulation in gen3sis and return a summary object possibly saving outputs and plots to the output folder.*

Description

run a simulation in gen3sis and return a summary object possibly saving outputs and plots to the output folder.

Usage

```
run_simulation(
  config = NA,
  landscape = NA,
  output_directory = NA,
  timestep_restart = NA,
  save_state = NA,
  call_observer = "all",
  enable_gc = FALSE,
  verbose = 1
)
```

Arguments

`config`	configuration file for the simulation or configuration object derived from a config file
`landscape`	directory where the all_geo_hab and distance_matrices reside
`output_directory`	directory for the simulation output
`timestep_restart`	= set the start time time-step. If NA start at the beginning, If "ti" start from the last available time-step, if a number "x" start from timestep x
`save_state`	= save the internal state of the simulation for restarts. If "all" save all time-step, if a vector, saves the desired time-steps if "last", saves only last timestep
`call_observer`	call observer functions if any, NA calls at the start and end times, "all" call all time-steps, You can also provide the number of time-steps equaly spaced between start and end steps, that the observer function is called
`enable_gc`	enable gc in case of memory shortages
`verbose`	integer value (0, 1 ,2 or 3). If 0 no printed statement, 1 time-step progress, 2 enable additional progress outputs regarding current time-step, 3 aditional information from within modules (default is 1)

Details

run a simulation with defined landscape and config objects. Possibly plot and save specified outputs as defined in the end_of_timestep_observer function inside the config object

Value

a summary object containing a minimal summary on simulation and dynamics progress (alive, speciations, extinctions) and some useful simulation data

See Also

plot_summary create_input_config create_input_landscape

Examples

```
# get path or correct input objects
datapath <- system.file(file.path("extdata", "CaseStudy1"), package="gen3sis")

# run simulation and store summary obejct to sim
sim <- run_simulation(config = file.path(datapath,"config/config_fast.R"),
                landscape = file.path(datapath,"landscape"),
                output_directory = tempdir())

# plot summary object
plot_summary(sim)
```

---

save_abundance *This function can be called within the observer function to save the species abundances.*

Description

This function can be called within the observer function to save the species abundances.

Usage

```
save_abundance()
```

See Also

save_species

Examples

```
## Not run:
  ## save abundances from within observer
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_abundance()

## End(Not run)
```

save_divergence

This function can be called within the observer function to save the compressed species divergence.

Description

This function can be called within the observer function to save the compressed species divergence.

Usage

```
save_divergence()
```

See Also

save_species

Examples

```
## Not run:
  ## save divergences from within observer for each species
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_divergence()

## End(Not run)
```

save_landscape

This function can be called within the observer function to save the current landscape, can be called independently by the user and is called by other observer functions relying on the landscape to be present (e.g. save_species)

Description

This function can be called within the observer function to save the current landscape, can be called independently by the user and is called by other observer functions relying on the landscape to be present (e.g. save_species)

Usage

```
save_landscape()
```

See Also

save_species

Examples

```
## Not run:
  ## save landscape from within observer for each species
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_landscape()

## End(Not run)
```

---

save_occupancy *This function can be called within the observer function to save the current occupancy pattern*

Description

This function can be called within the observer function to save the current occupancy pattern

Usage

```
save_occupancy()
```

Examples

```
## Not run:
  ## save occupancies from within observer
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_occupancy()

## End(Not run)
```

---

save_phylogeny *This function can be called within the observer function to save the current phylogeny.*

Description

This function can be called within the observer function to save the current phylogeny.

Usage

```
save_phylogeny()
```

Examples

```
## Not run:
  ## save phylogeny as a nexus tree from within observer for each species
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_phylogeny()

## End(Not run)
```

save_richness *This function can be called within the observer function to save the current richness pattern*

Description

This function can be called within the observer function to save the current richness pattern

Usage

```
save_richness()
```

See Also

save_species

Examples

```
## Not run:
  ## save the current richness pattern from within observer for each species
 # this functions should be called inside the end_of_timestep_observer function at the config file:
  save_richness()

## End(Not run)
```

save_species *This function can be called within the observer function to save the full species list.*

Description

This function can be called within the observer function to save the full species list.

Usage

```
save_species()
```

See Also

save_landscape

Examples

```
## Not run:
#adding the call to the end_of_timestep_observer function at the config file or object
#will automatically save all the species at an rds file at the outputfolder/species folder
# and the respective landscape at outputfolder/landscapes for the times steps the observer
# function is called (i.e. call_observer parameter at the run_simulation function)
save_species()

## End(Not run)
```

---

save_traits *This function can be called within the observer function to save the species traits.*

Description

This function can be called within the observer function to save the species traits.

Usage

```
save_traits()
```

See Also

save_species

Examples

```
## Not run:
  ## save the current traits pattern from within observer for each population of each species
  # this functions should be called inside the end_of_timestep_observer function at the config file:
   save_traits()

## End(Not run)
```

---

skeleton_config *empty skeleton config*

Description

empty skeleton config

Usage

```
skeleton_config()
```

Value

compiled string

verify_config — *This function verifies that all required config fields are provided.*

Description

This function verifies that all required config fields are provided.

Usage

```
verify_config(config)
```

Arguments

`config`	a config object

Value

Returns TRUE for a valid config, FALSE otherwise, in which case a list of missing parameters will be printed out as well

See Also

`create_input_config` `write_config_skeleton`

Examples

```
# get path to input config
datapath <- system.file(file.path("extdata", "WorldCenter"), package="gen3sis")
path_config <- file.path(datapath, "config/config_worldcenter.R")
# create config object
config_object <- create_input_config(path_config)
# check class
class(config_object)
# verify config
verify_config(config_object) # TRUE! this is a valid config

# break config_object, change name random_seed to r4nd0m_s33d
names(config_object$gen3sis$general)[1] <- "r4nd0m_s33d"
verify_config(config_object) # FALSE! this is an invalid config
```

`write_config_skeleton` *writes out a config skeleton, that is, an empty config file to be edited by the user.*

Description

writes out a config skeleton, that is, an empty config file to be edited by the user.

Usage

```
write_config_skeleton(file_path = "./config_skeleton.R", overwrite = FALSE)
```

Arguments

`file_path`	file path to write the file to
`overwrite`	overwrite existing file defaults to FALSE

Value

returns a boolean indicating success or failure

Examples

```
# set config_empty.R file path
config_file_path <- file.path(tempdir(), "config_empty.R")
#writes out a config skeleton
write_config_skeleton(config_file_path)
```

Index