Package ‘landsepi’

July 26, 2021

Type Package
Encoding UTF-8
Title Landscape Epidemiology and Evolution
Version 1.1.0
Date 2021-07-26
Maintainer Jean-François Rey <jean-francois.rey@inrae.fr>

Description A stochastic, spatially-explicit, demo-genetic model simulating the spread and evolution of a plant pathogen in a heterogeneous landscape to assess resistance deployment strategies. It is based on a spatial geometry for describing the landscape and allocation of different cultivars, a dispersal kernel for the dissemination of the pathogen, and a SEIR ('Susceptible-Exposed-Infectious-Removed') structure with a discrete time step. It provides a useful tool to assess the performance of a wide range of deployment options with respect to their epidemiological, evolutionary and economic outcomes.


URL https://csiro-inra.pages.biosp.inrae.fr/landsepi/,
https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi

BugReports https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi/-/issues
License GPL (>= 2) | file LICENSE
LazyData true
BuildVignettes true
NeedsCompilation yes
Biarch true

SystemRequirements C++11, gsl

Depends methods, utils, grDevices (>= 3.0.0), graphics (>= 3.0.0), R (>= 3.3.0), sp (>= 1.0-17),
Imports Rcpp (>= 0.9.0), stats (>= 3.0.2), Matrix, mvtnorm, fields, splancs, sf, DBI, RSQLite, foreach, parallel, doParallel
R topics documented:

**Collate** 'Math-Functions.R' 'RcppExports.R' 'graphics.R' 'AgriLand.R'
   'Class-LandsepiParams.R' 'GPKGTools.R' 'tools.R'
   'Methods-LandsepiParams.R' 'demo_landsepi.R' 'landsepi.R'
   'output.R' 'runShiny.R' 'simul_landsepi.R'

**LinkingTo** Rcpp, testthat

**RoxygenNote** 7.1.1

**Suggests** testthat, shiny, shinyjs, DT, knitr, rmarkdown

**VignetteBuilder** knitr

**Author** Loup Rimbaud [aut],
   Julien Papaix [aut],
   Jean-François Rey [aut, cre] (<https://orcid.org/0000-0003-3281-6701>),
   Jean-Loup Gaussen [ctb]

**Repository** CRAN

**Date/Publication** 2021-07-26 16:40:02 UTC

R topics documented:

landsepi-package .............................................................. 4
AgriLand ................................................................. 9
allocateCroptypeCultivars .................................................. 11
allocateCultivarGenes ..................................................... 13
allocateLandscapeCroptypes ............................................... 14
checkCroptypes ............................................................ 16
checkCultivars .............................................................. 17
checkCultivarsGenes ....................................................... 17
checkDispersalHost ........................................................ 18
checkDispersalPathogen ................................................... 18
checkGenes ................................................................. 19
checkInoculum .............................................................. 19
checkLandscape ............................................................ 20
checkOutputs ............................................................... 20
checkPathogen .............................................................. 21
checkSimulParams ........................................................ 21
checkTime ................................................................. 22
createLandscapeGPKG ..................................................... 22
createSimulParams ........................................................ 23
CroptypeBDD2Params ....................................................... 24
CultivarBDD2Params ........................................................ 24
CultivarGeneBDD2Params ................................................... 25
demo_landsepi .............................................................. 25
dispP ................................................................. 26
epid_output ................................................................. 27
evol_output ................................................................. 30
GeneBDD2Params ............................................................ 32
getGPKGArea ............................................................... 32
getGPKGCroptypes .......................................................... 33
## R topics documented:

- `getGPKGCroptypesRaw` .................................................. 33
- `getGPKGCultivars` ....................................................... 34
- `getGPKGCultivarsGenes` .................................................. 34
- `getGPKGGeneIDForCultivar` .............................................. 35
- `getGPKGGenes` .......................................................... 35
- `getGPKGRotation` ......................................................... 36
- `GPKGAddInputData` ....................................................... 36
- `GPKGAddTables` .......................................................... 37
- `initialize, LandsepiParams-method` .................................. 37
- `invlogit` ............................................................... 39
- `is.in.01` ............................................................... 39
- `is.positive` ............................................................ 40
- `is.strict.positive` ...................................................... 40
- `is.wholenumber` ........................................................ 41
- `landscapeTEST` .......................................................... 42
- `LandsepiParams` ......................................................... 42
- `loadCroptypes` .......................................................... 43
- `loadCultivar` ........................................................... 44
- `loadDispersalHost` ....................................................... 45
- `loadDispersalPathogen` .................................................. 46
- `loadGene` ............................................................... 47
- `loadLandscape` .......................................................... 48
- `loadOutputs` ............................................................. 48
- `loadPathogen` ........................................................... 50
- `loadSimulParams` ........................................................ 50
- `logit` ................................................................. 51
- `model_landsepi` .......................................................... 52
- `multiN` ................................................................. 56
- `params2CroptypeBDD` .................................................... 57
- `params2CultivarBDD` ..................................................... 58
- `params2GeneBDD` ........................................................ 58
- `params2GeneListBDD` .................................................... 59
- `periodic_cov` ............................................................ 59
- `plotland` .............................................................. 60
- `plot_allocation` ........................................................ 61
- `plot_freqPatho` .......................................................... 62
- `print` ................................................................. 63
- `resetCultivarsGenes` ..................................................... 63
- `runShinyApp` ............................................................ 64
- `runSimul` .............................................................. 64
- `saveDeploymentStrategy` ................................................ 67
- `setCroptypes` ........................................................... 68
- `setCultivars` ........................................................... 69
- `setDispersalHost` ........................................................ 71
- `setDispersalPathogen` ................................................... 72
- `setGenes` .............................................................. 73
- `setInoculum` ............................................................ 74
- `setLandscape` ........................................................... 75
Description

A stochastic, spatially-explicit, demo-genetic model simulating the spread and evolution of a plant pathogen in a heterogeneous landscape to assess resistance deployment strategies.

Details

- **Package:** landsepi
- **Type:** Package
- **Version:** 1.1.0
- **Date:** 2021-07-26
- **License:** GPL (>=2)

The landsepi package implements a spatially explicit stochastic model able to assess the epidemiological, evolutionary and economic outcomes of strategies to deploy plant resistance to pathogens. It is based on a spatial geometry for describing the landscape and allocation of different cultivars, a dispersal kernel for the dissemination of the pathogen, and a SEIR (‘susceptible-exposed-infectious-removed’, renamed HLIR for ‘healthy-latent-infectious-removed’ to avoid confusions with ‘susceptible host’) structure with a discrete time step. It simulates the spread and evolution of a pathogen in a heterogeneous cropping landscape, across cropping seasons split by host harvests which impose potential bottlenecks to the pathogen.

The landscape is represented by a set of polygons where the pathogen can disperse (the basic spatial unit is an individual polygon; an agricultural field may be composed of a single or several polygons). landsepi includes built-in simulated landscapes (and associated dispersal matrices for rust pathogens, see below), but is it possible to use your own landscape (in shapefile format) and dispersal matrix.

A wide array of resistance deployment strategies can be simulated in landsepi: fields of the landscape are cultivated with different croptypes that can rotate through time; each croptype is composed of either a pure cultivar or a mixture; and each cultivar may carry one or several resistance
genes. Thus, all combinations of rotations, mosaics, mixtures and pyramiding strategies are possible. Resistance genes affect several possible pathogen aggressiveness components: infection rate, durations of the latent period and the infectious period, and propagule production rate. Resistance may be complete (i.e. complete inhibition of the targeted aggressiveness component) or partial (i.e. the targeted aggressiveness component is only softened), and expressed from the beginning of the season, or later (to simulate Adult Plant Resistance (APR), also called Mature Plant Resistance). Cultivar allocation can be realised via an algorithm (allocateCrotypeCultivars()) but it is possible to use your own cultivar allocation if it is included in the shapefile containing the landscape.

To each resistance gene in the host (whether it may be a major gene or a QTL for quantitative resistance) is associated a pathogenicity gene in the pathogen. Through mutation of pathogenicity genes, the pathogen can restore its aggressiveness on resistance hosts and thus adapt to resistance (leading to sudden breakdown or gradual erosion of resistance genes). Pathogenicity genes may also be reassorted via sexual reproduction or gene recombination. Increased in aggressiveness on a resistant host (i.e. adaptation to the corresponding resistance genes) can be penalised by a fitness cost on susceptible hosts, i.e. pathogen genotypes adapted to a resistance gene have a reduced aggressiveness on hosts that do not carry this gene. The relation between pathogen aggressiveness on susceptible and resistant hosts is defined by a trade-off relationship whose shape depends on the strength of the trade-off. Strong trade-off means that the gain in fitness on resistant hosts is smaller than the cost on susceptible hosts.

This model provides a useful tool to assess the performance of a wide range of deployment options via epidemiological, evolutionary and economic outputs. It also helps investigate the effect of landscape organisation, the considered pathosystem and the epidemio-evolutionary context on the performance of a given strategy.

The package includes five examples of landscape structures and a default parameterisation to represent plant pathogens as typified by rusts of cereal crops (genus *Puccinia*, e.g. stripe rust, stem rust and leaf rust of wheat and barley). The main function of the package is runSimul(). It can be parameterised to simulate various resistance deployment strategies using either the provided landscapes and parameters for cereal rusts, or landscapes and parameters set by the user. See demo_landsepi() for a demonstration, and our tutorials (browseVignettes("landsepi")) for details on how to use landsepi.

Assumptions (in bold those that can be relaxed with appropriate parameterization):

1. The spatial unit is a polygon, i.e. a piece of land delimited by boundaries and possibly cultivated with a crop. Such crop may be host or non-host, and the polygon is considered a homogeneous mixture of host individuals (i.e. there is no intra-polygon structuration). A field may be composed of a single or several polygons.

2. Host individuals are in one of these four categories: H (healthy), E (latent, i.e. infected but not infectious nor symptomatic), I (infectious and symptomatic), or R (removed, i.e. epidemiologically inactive).

3. A host ‘individual’ is an infection unit and may correspond to a given amount of plant tissue (where a local infection may develop, e.g. fungal lesion) or a whole plant (e.g. systemic viral infection). In the first case, plant growth increases the amount of available plant tissue (hence the number of individuals) during the cropping season. Plant growth is deterministic (logistic growth) and only healthy hosts (state H) contribute to plant growth (castrating pathogen).

4. The decreasing availability of healthy host tissues (as epidemics spread) makes pathogen infection less likely (i.e. density-dependence due to plant architecture).
5. **Host are cultivated, thus there is no host reproduction, dispersal and natural death.**

6. Environmental and climate conditions are constant, and host individuals of a given genotype are equally susceptible to disease from the first to the last day of every cropping season.

7. Crop yield depends on the average amount of producing host individuals during the cropping season and does not depend on the time of epidemic peak. **Only healthy individuals (state H) contribute to crop yield.**

8. Components of a mixture are independent each other (i.e. there is neither plant-plant interaction nor competition for space, and harvests are segregated).

9. The pathogen is haploid.

10. Initially, the pathogen is not adapted to any source of resistance, and is only present on susceptible hosts (at state I).

11. **Pathogen dispersal is isotropic (i.e. equally probable in every direction).**

12. **Pathogen reproduction is clonal.**

13. Pathogenicity genes mutate independently from each other.

14. **Pathogen adaptation to a given resistance gene consists in restoring the same aggressiveness component as the one targeted by the resistance gene.**

15. If a fitness cost penalises pathogen adaptation to a given resistance gene, this cost is paid on hosts that do not carry this gene, and consists in a reduction in the same aggressiveness component as the one targeted by the resistance gene.

16. When there is a delay for activation of a given resistance gene (APR), the time to activation is the same for all hosts carrying this gene and located in the same field.

17. Variances of the durations of the latent and the infectious periods of the pathogen are not affected by plant resistance.

18. If there is sexual reproduction (or gene recombination), it occurs only between parental infections located in the same polygon and the same host genotype. The host population is panmictic (i.e. all pairs of parents have the same probability to occur). The propagule production rate of a couple is the sum of the propagule production rates of the parents. The genotype of each daughter propagule is issued from random loci segregation between parental loci.

**Epidemiological outputs** The epidemiological outcome of a deployment strategy is evaluated using:

1. the area under the disease progress curve (AUDPC) to measure disease severity (i.e. the average number of diseased plant tissue -status I and R- per time step and square meter),

2. the relative area under the disease progress curve (AUDPCr) to measure the average proportion of diseased tissue (status I and R) relative to the total number of existing host individuals (H+L+I+R).

3. the Green Leaf Area (GLA) to measure the average amount of healthy plant tissue (status H) per time step and square meter,

4. the relative Green Leaf Area (GLAr) to measure the average proportion of healthy tissue (status H) relative to the total number of existing host individuals (H+L+I+R).

A set of graphics and a video showing epidemic dynamics can also be generated.

**Evolutionary outputs** The evolutionary outcome is assessed by measuring:

1. the dynamics of pathogen genotype frequencies,
2. the evolution of pathogen aggressiveness,
3. the durability of resistance genes. Durability can be estimated using the time until the pathogen reaches the three steps to adapt to plant resistance: (1) first appearance of adapted mutants, (2) initial migration to resistant hosts and infection, and (3) broader establishment in the resistant host population (i.e. the point at which extinction becomes unlikely).

**Economic outputs** The economic outcome of a simulation can be evaluated using:
1. the crop yield: yearly crop production (e.g. grains, fruits, wine) in weight (or volume) units per hectare (depends on the number of productive hosts and associated theoretical yield),
2. the crop products: yearly products generated from sales, in monetary units per hectare (depends on crop yield and market value),
3. the crop operational costs: yearly costs associated with crop planting, in monetary units per hectare (depends on initial host density and planting cost),
4. the margin, i.e. products - operational costs, in monetary units per hectare.

**Future versions:**
Future versions of the package will include in particular:

- Sets of pathogen parameters to simulate other pathosystems (e.g. canola blackleg, grapevine downy mildew, potato virus Y on pepper).
- More flexible initial conditions (e.g. size, location and composition of pathogen inoculum at the beginning of the simulation).

**Dependencies:**
The package for compiling needs:

- g++
- libgsl2
- libgsl-dev

and the following R packages:

- Rcpp
- sp
- stats
- Matrix
- mvtnorm
- maptools
- fields
- splines
- sf
- DBI
- RSQLite
• foreach
• parallel
• doParallel

In addition, to generate videos the package will need ffmpeg.

Author(s)

Loup Rimbaud <loup.rimbaud@inrae.fr>
Julien Papaix <julien.papaix@inrae.fr>
Jean-Francois Rey <jean-francois.rey@inrae.fr>
Jean-Loup Gaussen <jean-loup-thomas.gaussen@inrae.fr>
Maintainer: Jean-Francois Rey <jean-francois.rey@inrae.fr>

References

When referencing the simulation model, please cite the following article::

When referencing the R package, please cite the following package::

See Also

Useful links:
• https://csiro-inra.pages.biopsp.inrae.fr/landsepi/
• https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi
• Report bugs at https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi/-/issues

Examples

## Not run:
library("landsepi")

## Run demonstrations (in 20-year simulations) for different deployment strategies:
demo_landsepi(strat = "MO") ## for a mosaic of cultivars
demo_landsepi(strat = "MI") ## for a mixture of cultivars
demo_landsepi(strat = "RO") ## for a rotation of cultivars
demo_landsepi(strat = "PY") ## for a pyramid of resistance genes

## End(Not run)
**AgriLand**

---

**Landscape allocation**

---

**Description**

Generates a landscape composed of fields where croptypes are allocated with controlled proportions and spatio-temporal aggregation.

**Usage**

\[
\text{AgriLand(}
\quad \text{landscape,}
\text{Nyears,}
\quad \text{rotation_period} = 0,
\quad \text{rotation_sequence} = \text{list(c}(0, 1, 2)),
\quad \text{rotation_realloc} = \text{FALSE},
\quad \text{prop} = \text{list(c}(1/3, 1/3, 1/3)),
\quad \text{aggreg} = \text{list}(1),
\quad \text{algo} = \text{"periodic"},
\quad \text{croptype_names} = \text{c()},
\quad \text{graphic} = \text{FALSE},
\quad \text{outputDir} = \text{"./"}
\text{)}
\]

**Arguments**

- **landscape**
  a spatialpolygon object containing field coordinates.

- **Nyears**
  an integer giving the number of simulated cropping seasons.

- **rotation_period**
  number of years before rotation of the landscape. There is no rotation if rotation_period=0 or rotation_period=Nyears.

- **rotation_sequence**
  a list, each element of the list contains indices of croptypes that are cultivated during a period given by "rotation_period". There is no change in cultivated croptypes if the list contains only one element (e.g. only one vector c(0,1,2), indicating cultivation of croptypes 0, 1 and 2).

- **rotation_realloc**
  a logical indicating if a new random allocation of croptypes is performed when the landscape is rotated (FALSE=static allocation, TRUE=dynamic allocation). Note that if rotation_realloc=FALSE, all elements of the list "rotation_sequence" must have the same length, and only the first element of the lists "prop" and "aggreg" will be used.

- **prop**
  a list of the same size as "rotation_sequence", each element of the list contains a vector of the proportions (in surface) associated with the croptypes in "rotation_sequence". A single vector can be given instead of a list if all elements of "rotation_sequence" are associated with the same proportions.
aggreg  a list of the same size as "rotation_sequence", each element of the list is a single double indicating the degree of aggregation of the landscape. This double must greater or equal 0; the greater its value, the higher the degree of spatial aggregation (roughly, aggreg between 0 and 0.1 for fragmented landscapes, between 0.1 and 0.5 for balanced landscapes, between 0.5 and 3 for aggregated landscapes, and above 3 for highly aggregated landscapes). A single double can be given instead of a list if all elements of "rotation_sequence" are associated with the same level of aggregation.

algo  the algorithm used for the computation of the variance-covariance matrix of the multivariate normal distribution: "exp" for exponential function, "periodic" for periodic function, "random" for random draw (see details of function multiN). If algo="random", the parameter aggreg is not used. Algorithm "exp" is preferable for big landscapes.

croptype_names  a vector of croptype names (for legend in graphic).

graphic  a logical indicating if a graphic of the landscape must be generated (TRUE) or not (FALSE).

outputDir  a directory to save graphic

Details
An algorithm based on latent Gaussian fields is used to allocate two different croptypes across the simulated landscapes (e.g. a susceptible and a resistant cultivar, denoted as SC and RC, respectively). This algorithm allows the control of the proportions of each croptype in terms of surface coverage, and their level of spatial aggregation. A random vector of values is drawn from a multivariate normal distribution with expectation 0 and a variance-covariance matrix which depends on the pairwise distances between the centroids of the fields. Next, the croptypes are allocated to different fields depending on whether each value drawn from the multivariate normal distribution is above or below a threshold. The proportion of each cultivar in the landscape is controlled by the value of this threshold. To allocate more than two croptypes, AgriLand uses sequentially this algorithm. For instance, the allocation of three croptypes (e.g. SC, RC1 and RC2) is performed as follows:

1. the allocation algorithm is run once to segregate the fields where the susceptible cultivar is grown, and
2. the two resistant cultivars (RC1 and RC2) are assigned to the remaining candidate fields by re-running the allocation algorithm.

Value
a gpkg (shapefile) containing the landscape structure (i.e. coordinates of field boundaries), the area and composition (i.e. croptypes) in time (i.e. each year) for each field. A png graphic can be generated if graphic=TRUE.

References
allocateCroptypeCultivars

Allocate cultivars to one croptype

Description

Updates a given croptype by allocating cultivars composing it.

See Also

multiN, periodic_cov, allocateLandscapeCroptypes

Examples

```r
## Not run:
data(landscapeTEST)
landscape <- get("landscapeTEST1")
set.seed(12345)
## Generate a mosaic of three croptypes in balanced proportions
## and high level of spatial aggregation
AgriLand(landscape, 
  Nyears = 10, 
  rotation_sequence = c(0, 1, 2), prop = rep(1/3, 3), 
  aggreg = rep(10, 3), algo = "periodic", 
  graphic = TRUE, outputDir = getwd()
)

## Generate a dynamic mosaic of two croptypes in unbalanced proportions
## and low level of spatial aggregation,
## the second croptype being replaced every 5 years without changing field allocation
AgriLand(landscape, 
  Nyears = 20, rotation_period = 5, rotation_sequence = list(c(0, 1), c(0, 2)),
  prop = c(1/3, 2/3), aggreg = c(0.07, 0.07), algo = "periodic", graphic = TRUE, 
  outputDir = getwd()
)

## Generate a dynamic mosaic of four croptypes in balanced proportions
## and medium level of spatial aggregation,
## with field allocation changing every year
AgriLand(landscape, 
  Nyears = 5, rotation_period = 1, rotation_realloc = TRUE, 
  rotation_sequence = c(0, 1, 2, 3),
  prop = rep(1/4, 4), aggreg = 0.25, algo = "exp", graphic = TRUE, outputDir = getwd()
)
## End(Not run)
```
allocateCroptypeCultivars

Usage

allocateCroptypeCultivars(
  croptypes,
  croptypeName,
  cultivarsInCroptype,
  prop = NULL
)

Arguments

croptypes      a dataframe containing all croptypes, initialised via loadCroptypes
croptypeName   the name of the croptype to be allocated
cultivarsInCroptype name of cultivars composing the croptype
prop            vector of proportions of each cultivar in the croptype. Default to balanced proportions.

Value

a croptype data.frame updated for the concerned croptype.

See Also

setCroptypes, setCultivars

Examples

## Not run:
simul_params <- createSimulParams()
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop", "Mixture"))
croptypes
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture", c("Resistant1", "Resistant2"))
croptypes

## End(Not run)
allocateCultivarGenes  Allocate genes to a cultivar

Description

Updates a LandsepiParams object with, for a given cultivar, the list of genes it carries

Usage

allocateCultivarGenes(params, cultivarName, listGenesNames = c"

Arguments

params a LandsepiParams object.
cultivarName the name of the cultivar to be allocated.
listGenesNames the names of the genes the cultivar carries

Value

a LandsepiParams object

See Also

setGenes, setCultivars

Examples

## Not run:
simul_params <- createSimulParams()
gene1 <- loadGene(name = "MG 1", type = "majorGene")
gene2 <- loadGene(name = "MG 2", type = "majorGene")
genes <- data.frame(rbind(gene1, gene2), stringsAsFactors = FALSE)
simul_params <- setGenes(simul_params, genes)
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
simul_params <- allocateCultivarGenes(simul_params, "Resistant", c("MG 1", "MG 2"))
simul_params@CultivarsGenes

## End(Not run)
allocateLandscapeCroptypes

Allocate croptypes to the landscape

Description

Updates the landscape of a LandsepiParams object with croptype allocation in every field of the landscape and every year of simulation. Allocation is based on an algorithm which controls croptype proportions (in surface) and spatio-temporal aggregation.

Usage

allocateLandscapeCroptypes(
  params,
  rotation_period,
  rotation_sequence,
  rotation_realloc = FALSE,
  prop,
  aggreg,
  algo = "periodic",
  graphic = TRUE
)

Arguments

params a LandsepiParams Object.
rotation_period number of years before rotation of the landscape. There is no rotation if rotation_period=0 or rotation_period=Nyears.
rotation_sequence a list, each element of the list contains indices of croptypes that are cultivated during a period given by "rotation_period". There is no change in cultivated croptypes if the list contains only one element (e.g. only one vector c(0,1,2), indicating cultivation of croptypes 0, 1 and 2).
rotation_realloc a logical indicating if a new random allocation of croptypes is performed when the landscape is rotated (FALSE=static allocation, TRUE=dynamic allocation). Note that if rotation_realloc=FALSE, all elements of the list "rotation_sequence" must have the same length, and only the first element of the lists "prop" and "aggreg" will be used.
prop a list of the same size as "rotation_sequence", each element of the list contains a vector of the proportions (in surface) associated with the croptypes in "rotation_sequence". A single vector can be given instead of a list if all elements of "rotation_sequence" are associated with the same proportions.
allocateLandscapeCroptypes

**aggreg**
a list of the same size as "rotation_sequence", each element of the list is a single
double indicating the degree of aggregation of the landscape. This double must
greater or equal 0; the greater its value, the higher the degree of spatial aggrega-
tion (roughly, aggreg between 0 and 0.1 for fragmented landscapes, between 0.1
and 0.5 for balanced landscapes, between 0.5 and 3 for aggregated landscapes,
and above 3 for highly aggregated landscapes). A single double can be given
instead of a list if all elements of "rotation_sequence" are associated with the
same level of aggregation.

**algo**
the algorithm used for the computation of the variance-covariance matrix of the
multivariate normal distribution: "exp" for exponential function, "periodic" for
periodic function, "random" for random draw (see details of function multiN). If
algo="random", the parameter aggreg is not used. Algorithm "exp" is preferable
for big landscapes.

**graphic**
a logical indicating if graphics must be generated (TRUE) or not (FALSE).

**Details**
An algorithm based on latent Gaussian fields is used to allocate two different croptypes across the
simulated landscapes (e.g. a susceptible and a resistant cultivar, denoted as SC and RC, respec-
tively). This algorithm allows the control of the proportions of each croptype in terms of surface
coverage, and their level of spatial aggregation. A random vector of values is drawn from a mul-
tivariate normal distribution with expectation 0 and a variance-covariance matrix which depends
on the pairwise distances between the centroids of the fields. Next, the croptypes are allocated to
different fields depending on whether each value drawn from the multivariate normal distribution
is above or below a threshold. The proportion of each cultivar in the landscape is controlled by
the value of this threshold. To allocate more than two croptypes, AgriLand uses sequentially this
algorithm. For instance, the allocation of three croptypes (e.g. SC, RC1 and RC2) is performed as follows:

1. the allocation algorithm is run once to segregate the fields where the susceptible cultivar is
grown, and

2. the two resistant cultivars (RC1 and RC2) are assigned to the remaining candidate fields by
re-running the allocation algorithm.

**Value**
a LandsepiParams object with Landscape updated with the layer "croptypeID". It contains croptype
allocation in every field of the landscape for all years of simulation.

**Examples**
```r
## Not run:
## Initialisation
simul_params <- createSimulParams(outputDir = getwd())
## Time parameters
simul_params <- setTime(simul_params, Nyears = 10, nTspY = 120)
## Landscape
simul_params <- setLandscape(simul_params, loadLandscape(1))
## Cultivars
```
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)

## Allocate cultivars to croptypes

croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop", "Resistant crop 1", "Resistant crop 2"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop 1", "Resistant1")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop 2", "Resistant2")
simul_params <- setCroptypes(simul_params, croptypes)

## Allocate croptypes to landscape

rotation_sequence <- croptypes$croptypeID ## No rotation -> 1 rotation_sequence element
rotation_period <- 0 ## same croptypes every years
prop <- c(1/3, 1/3, 1/3) ## croptypes proportions
aggreg <- 10 ## aggregated landscape

simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = rotation_period, rotation_sequence = rotation_sequence, rotation_realloc = FALSE, prop = prop, aggreg = aggreg)
simul_params$Landscape

## End(Not run)

---

**checkCroptypes**  

**Check croptypes**

**Description**

checks croptypes validity

**Usage**

checkCroptypes(params)

**Arguments**

- params: a LandsepiParams object.

**Value**

a boolean, TRUE if OK, FALSE otherwise
checkCultivars  

Description
check cultivars validity

Usage
checkCultivars(params)

Arguments
params a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise

checkCultivarsGenes  

Description
Checks CultivarsGene data.frame validity

Usage
checkCultivarsGenes(params)

Arguments
params a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise
checkDispersalHost  
*Check host dispersal*

**Description**

Checks host dispersal matrix validity.

**Usage**

```
checkDispersalHost(params)
```

**Arguments**

- `params` a LandsepiParams Object.

**Value**

a boolean TRUE if OK, FALSE otherwise

---

checkDispersalPathogen  
*Check pathogen dispersal*

**Description**

Checks pathogen dispersal validity

**Usage**

```
checkDispersalPathogen(params)
```

**Arguments**

- `params` a LandsepiParams Object.

**Value**

a boolean TRUE if OK, FALSE otherwise
checkGenes

Description
checks Genes data.frame validity

Usage
checkGenes(params)

Arguments
params a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise

checkInoculum

Description
Checks inoculum validity.

Usage
checkInoculum(params)

Arguments
params a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise
**checkLandscape**  
*Check the landscape*

**Description**
Checks landscape validity

**Usage**
checkLandscape(params)

**Arguments**
params a LandsepiParams Object.

**Value**
TRUE if Ok, FALSE otherwise

---

**checkOutputs**  
*Check outputs*

**Description**
Checks outputs validity.

**Usage**
checkOutputs(params)

**Arguments**
params a LandsepiParams object.

**Value**
a boolean, TRUE if OK, FALSE otherwise
checkPathogen  

Description
Checks pathogen validity

Usage
checkPathogen(params)

Arguments
params a LandsepiParams Object.

Value
a boolean, TRUE if OK, FALSE otherwise

checkSimulParams  

Description
Checks validity of a LandsepiParams object.

Usage
checkSimulParams(params)

Arguments
params a LandsepiParams Object.

Value
TRUE if OK for simulation, FALSE otherwise
checkTime  

**Description**
Checks time parameters validity

**Usage**
checkTime(params)

**Arguments**
- **params** a LandsepiParams Object.

**Value**
a boolean TRUE if times are setted.

createLandscapeGPKG  

**Description**
Creates a GPKG file from a sf object

**Usage**
createLandscapeGPKG(landscape, outputfile)

**Arguments**
- **landscape** a sf object
- **outputfile** GPKG output file name

**Details**
Generates a GPKG file with two layouts. One layout called "croptypeID" containing the croptypes ID for every year. One another layout called "area" containing the area of every polygon.

**Value**
the outputfile name (full path)
createSimulParams

Create a LandsepiParams object.

Description

Creates a default object of class LandsepiParams.

Usage

createSimulParams(outputDir = "./"")

Arguments

outputDir ouput directory for simulation (default: current directory)

Details

Create a default object of class LandsepiParams used to store all simulation parameters. It also creates a subdirectory in outputDir using the date; this directory will contain all simulation outputs.

Value

a LandsepiParams object initialised with the following context:

- random seed
- all pathogen parameters fixed at 0
- no between-field dispersal (neither pathogen nor host)
- no pathogen introduction
- no resistance gene
- no output to generate.

Examples

## Not run:
createSimulParams()

## End(Not run)
**CroptypeBDD2Params**

**Description**

Converts BDD table to Croptype LandspeParams object

**Usage**

CroptypeBDD2Params(inputGPKG)

**Arguments**

- `inputGPKG` a GPKG filename

**Value**

a data.frame LandspeParams@Croptypes compatible

---

**CultivarBDD2Params**

**Description**

Converts BDD table Cultivar to a Cultivar LandsepiParams object

**Usage**

CultivarBDD2Params(inputGPKG)

**Arguments**

- `inputGPKG` a GPKG filename

**Value**

a data.frame LandspeParams@Cultivars compatible
**CultivarGeneBDD2Params**

*Description*

Converts BDD table to LandsepiParams object

*Usage*

```r
CultivarGeneBDD2Params(inputGPKG)
```

*Arguments*

- `inputGPKG`: a GPKG filename

*Value*

a data.frame LandsepiParams@CultivarsGenes compatible

---

**demo_landsepi**

*Package demonstration*

*Description*

run a simulation demonstration with landsepi

*Usage*

```r
demo_landsepi(
  seed = 12345,
  strat = "MO",
  Nyears = 20,
  nTSpY = 120,
  videoMP4 = TRUE
)
```

*Arguments*

- `seed`: an integer used as seed for Random Number Generator (default 12345).
- `strat`: a string specifying the deployment strategy: "MO" for mosaic of resistant cultivars, "MI" for intra-field mixtures, "RO" for cultivar rotations, and "PY" for resistance gene pyramiding in a cultivar.
- `Nyears`: number of cropping seasons (years) to simulate.
- `nTSpY`: number of time-steps (days) per cropping season.
- `videoMP4`: a logical indicating if a video must be generated (TRUE, default) or not (FALSE).
Details

In these examples, 2 completely efficient resistance sources (typical of major resistance genes) are deployed in the landscape according to one of the following strategies:

- **Mosaic:** 3 pure crops (S + R1 + R2) with very high spatial aggregation.
- **Mixture:** 1 pure susceptible crop + 1 mixture of two resistant cultivars, with high aggregation.
- **Rotation:** 1 susceptible pure crop + 2 resistant crops in alternation every 2 years, with moderate aggregation.
- **Pyramiding:** 1 susceptible crop + 1 pyramided cultivar in a fragmented landscape (low aggregation).

Value

A set of text files, graphics and a video showing epidemic dynamics.

See Also

runSimul, runShinyApp

Examples

```r
## Not run:
## Run demonstrations (in 20-year simulations) for different deployment strategies:
demo_landsepi(strat = "MO") ## for a mosaic of cultivars
demo_landsepi(strat = "MI") ## for a mixture of cultivars
demo_landsepi(strat = "RO") ## for a rotation of cultivars
demo_landsepi(strat = "PY") ## for a pyramid of resistance genes
## End(Not run)
```

dispP

*Dispersal matrices for rust fungi of cereal crops.*

Description

Five vectorised dispersal matrices of pathogens as typified by rust fungi of cereal crops (genus *Puccinia*), and associated with landscapes 1 to 5 (composed of 155, 154, 152, 153 and 156 fields, respectively).

Usage

dispP_1
dispP_2
dispP_3
dispP_4
dispP_5
**Format**

The format is: num [1:24025] 8.81e-01 9.53e-04 7.08e-10 1.59e-10 3.29e-06 ...

**Details**

The pathogen dispersal matrix gives the probability for a pathogen in a field $i$ (row) to migrate to field $i'$ (column) through dispersal. It is computed based on a dispersal kernel and the euclidian distance between each point in fields $i$ and $i'$, using the CaliFloPP algorithm (Bouvier et al. 2009). The dispersal kernel is an isotropic power-law function of equation: 

$$f(x) = \frac{(b-2)*(b-1)}{(2*pi*a^2)} * (1 + x/a)^{-(b-2)}$$

with $a=40$ a scale parameter and $b=7$ related to the weight of the dispersal tail. The expected mean dispersal distance is given by $2*a/(b-3)=20$ m.

**References**


**Examples**

dispP_1
summary(dispP_1)
## maybe str(dispP_1); plot(dispP_1) ...

---

**epid_output**

*Generation of epidemiological and economic model outputs*

**Description**

Generates epidemiological and economic outputs from model simulations.

**Usage**

epid_output(
  types = "all",
  time_param,
  Npatho,
  area,
  rotation,
  croptypes,
  cultivars_param,
  eco_param,
  GLAnoDis = cultivars_param$max_density[1],
  ylim_param = list(audpc = c(0, 0.76), audpc_rel = c(0, 1), gla = c(0, 1.48), gla_rel = c(0, 1), eco_cost = c(0, NA), eco_yield = c(0, NA), eco_product = c(0, NA),
    eco_margin = c(NA, NA)),
  writeTXT = TRUE,
  graphic = TRUE,
  path = getwd())
)
Arguments

types  
a character string (or a vector of character strings if several outputs are to be computed) specifying the type of outputs to generate (see details):
  • "audpc": Area Under Disease Progress Curve
  • "audpc_rel": Relative Area Under Disease Progress Curve
  • "gla": Green Leaf Area
  • "gla_rel": Relative Green Leaf Area
  • "eco_yield": Total crop yield
  • "eco_cost": Operational crop costs
  • "eco_product": Crop products
  • "eco_margin": Margin (products - operational costs)
  • "HLIR_dynamics", "H_dynamics", "L_dynamics", "IR_dynamics", "HLI_dynamics", etc.: Epidemic dynamics related to the specified sanitary status (H, L, I or R and all their combinations). Graphics only, works only if graphic=TRUE.
  • "all": compute all these outputs (default).

time_param  
list of simulation parameters:
  • Nyears = number cropping seasons,
  • nTSpY = number of time-steps per cropping season.

Npatho  
number of pathogen genotypes.

area  
a vector containing polygon areas (must be in square meters).

rotation  
a dataframe containing for each field (rows) and year (columns, named "year_1", "year_2", etc.), the index of the cultivated croptype. Importantly, the matrix must contain 1 more column than the real number of simulated years.

croptypes  
a dataframe with three columns named 'croptypeID' for croptype index, 'cultivarID' for cultivar index and 'proportion' for the proportion of the cultivar within the croptype.

cultivars_param  
list of parameters associated with each host genotype (i.e. cultivars):
  • name = vector of cultivar names,
  • initial_density = vector of host densities (per square meter) at the beginning of the cropping season as if cultivated in pure crop,
  • max_density = vector of maximum host densities (per square meter) at the end of the cropping season as if cultivated in pure crop,
  • cultivars_genes_list = a list containing, for each host genotype, the indices of carried resistance genes.

eco_param  
a list of economic parameters for each host genotype as if cultivated in pure crop:
  • yield_perHa = a dataframe of 4 columns for the theoretical yield associated with hosts in sanitary status H, L, I and R, as if cultivated in pure crops, and one row per host genotype (yields are expressed in weight or volume units / ha / cropping season),
  • planting_cost_perHa = a vector of planting costs (in monetary units / ha / cropping season),
• market_value = a vector of market values of the production (in monetary units / weight or volume unit).

GLAnoDis the value of absolute GLA in absence of disease (required to compute economic outputs).

ylim_param a list of graphical parameters for each required output: bounds for y-axes for audpc, gla, gla_rel, eco_cost, eco_yield, eco_product, eco_margin.

writeTXT a logical indicating if the output is written in a text file (TRUE) or not (FALSE).

graphic a logical indicating if a tiff graphic of the output is generated (only if more than one year is simulated).

path path of text file (if writeTXT = TRUE) and tiff graphic (if graphic = TRUE) to be generated.

Details

Outputs are computed every year for every cultivar as well as for the whole landscape.

Epidemiological outputs. The epidemiological impact of pathogen spread can be evaluated by different measures:

1. Area Under Disease Progress Curve (AUDPC): average number of diseased host individuals (status I + R) per time step and square meter.

2. Relative Area Under Disease Progress Curve (AUDPCr): average proportion of diseased host individuals (status I + R) relative to the total number of existing hosts (H+L+I+R).

3. Green Leaf Area (GLA): average number of healthy host individuals (status H) per time step and per square meter.

4. Relative Green Leaf Area (GLAr): average proportion of healthy host individuals (status H) relative to the total number of existing hosts (H+L+I+R).

Economic outputs. The economic outcome of a simulation can be evaluated using:

1. Crop yield: yearly crop yield (e.g. grains, fruits, wine) in weight (or volume) units per hectare (depends on the number of productive hosts and associated theoretical yield).

2. Crop products: yearly product generated from sales, in monetary units per hectare (depends on crop yield and market value).

3. Operational crop costs: yearly costs associated with crop planting in monetary units per hectare (depends on initial host density and planting cost).

4. Crop margin, i.e. products - operational costs, in monetary units per hectare.

Value

A list containing, for each required type of output, a matrix summarising the output for each year and cultivar (as well as the whole landscape). Each matrix can be written in a txt file (if writeTXT=TRUE), and illustrated in a graphic (if graphic=TRUE).

References

evol_output

See Also
evol_output

Examples

```r
## Not run:
demo_landsepi()

## End(Not run)
```

evol_output  

*Generation of evolutionary model outputs*

Description

Generates evolutionary outputs from model simulations.

Usage

```r
evol_output(
  types = "all",
  time_param,
  Npoly,
  cultivars_param,
  genes_param,
  thres_breakdown = 50000,
  writeTXT = TRUE,
  graphic = TRUE,
  path = getwd()
)
```

Arguments

types  
a character string (or a vector of character strings if several outputs are to be computed) specifying the type of outputs to generate (see details):

- "evol_patho": Dynamics of pathogen genotype frequencies
- "evol_aggr": Evolution of pathogen aggressiveness
- "durability": Durability of resistance genes
- "all": compute all these outputs (default)

time_param  
list of simulation parameters:

- Nyears = number cropping seasons,
- nTSpY = number of time-steps per cropping season.

Npoly  
number of fields in the landscape.

cultivars_param  
list of parameters associated with each host genotype (i.e. cultivars) when cultivated in pure crops:
evol_output

- name = vector of cultivar names,
- cultivars_genes_list = a list containing, for each host genotype, the indices of carried resistance genes.

genes_param = list of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene:

- name = vector of names of resistance genes,
- Nlevels_aggressiveness = vector containing the number of adaptation levels related to each resistance gene (i.e. 1 + number of required mutations for a pathogenicity gene to fully adapt to the corresponding resistance gene),

thres_breakdown

- an integer (or vector of integers) giving the threshold (i.e. number of infections) above which a pathogen genotype is unlikely to go extinct and resistance is considered broken down, used to characterise the time to invasion of resistant hosts (several values are computed if several thresholds are given in a vector).

writeTXT = a logical indicating if the output is written in a text file (TRUE) or not (FALSE).

graphic = a logical indicating if graphics must be generated (TRUE) or not (FALSE).

path = a character string indicating the path of the repository where simulation output files are located and where .txt files and graphics will be generated.

Details

For each pathogen genotype, several computations are performed:

- appearance: time to first appearance (as propagule);
- R_infection: time to first true infection of a resistant host;
- R_invasion: time when the number of infections of resistant hosts reaches a threshold above which the genotype is unlikely to go extinct.

The value Nyears + 1 time step is used if the genotype never appeared/infected/invaded.

Value

A list containing, for each required type of output, a matrix summarising the output. Each matrix can be written in a txt file (if writeTXT=TRUE), and illustrated in a graphic (if graphic=TRUE).

References


See Also

epid_output
getGPKGArea

Examples

## Not run:
demo_landsepi()

## End(Not run)

GeneBDD2Params

Description

Converts BDD table Gene to LandsepiParams object

Usage

GeneBDD2Params(inputGPKG)

Arguments

inputGPKG a LandsepiParams

Value

a data.frame LandsepiParams@Genes compatible

getGPKGArea

Description

Gets "area" layer as a vector of a GPKG file

Usage

getGPKGArea(gpkgfile)

Arguments

gpkgfile a GPKG file

Value

a vector of the area of each polygons
getGPKGCroptypes

Description
Gets Croptypes and Cultivars proportions from a GPKG file

Usage
getGPKGCroptypes(inputGPKG)

Arguments
inputGPKG 
a GPKG filename

Value
a data.frame with croptypeID, CultivarNames, and proportions

getGPKGCroptypesRaw

Description
Gets Croptypes and Cultivars proportions from a GPKG file without refactoring

Usage
getGPKGCroptypesRaw(inputGPKG)

Arguments
inputGPKG 
a GPKG filename

Value
a data.frame with croptypeID, CultivarID, and proportions
getGPKGCultivars

Description

Gets Cultivars from a GPKG file

Usage

getGPKGCultivars(inputGPKG)

Arguments

inputGPKG  a GPKG filename

Value

a data.frame

getGPKGCultivarsGenes

Description

Gets Cultivars Genes from a GPKG file

Usage

getGPKGCultivarsGenes(inputGPKG)

Arguments

inputGPKG  a GPKG filename

Value

a data.frame
getGPKGGeneIDForCultivar

Description
Gets GeneID of a cultivar

Usage
getGPKGGeneIDForCultivar(inputGPKG, cultivarID)

Arguments
inputGPKG a GPKG filename
cultivarID a cultivarID

Value
a data.frame of GeneID

getGPKGGenes

Description
Gets Genes from a GPKG file

Usage
getGPKGGenes(inputGPKG)

Arguments
inputGPKG a GPKG filename

Value
a data.frame
### getGPKGRotation

**Description**

Gets CropTypes ID rotation years as a matrix

**Usage**

```r
getGPKGRotation(gpkgfile)
```

**Arguments**

- `gpkgfile` a GPKG file

**Value**

a matrix as rows for polygons and cols for years

### GPKGAddInputData

**Description**

GPKGAddInputData

**Usage**

```r
GPKGAddInputData(
    gpkgfile,
    table = "",
    data = data.frame(),
    deleteExistingData = FALSE
)
```

**Arguments**

- `gpkgfile` a gpkg filename
- `table` table name
- `data` a data to write in BDD, should be the return of a function `param2XXXXXBBDD`
- `deleteExistingData` if TRUE overwrite data if already present in gpkg file, default FALSE

**Details**

Adds data 'data' values in the table 'table' using 'data' colnames
**Description**

Adds non spatial data table definitions (sqlite) into a GPKG file. It adds Cultivar, CultivarList, Gene, GeneList tables.

**Usage**

GPKGAddTables(gpkgfile)

**Arguments**

gpkgfile a GPKF file

**Value**

the GPKG file name

---

**initialize,LandsepiParams-method**

*LandsepiParams*

**Description**

Creates and initialises a LandespiParams object with default parameters.

**Usage**

```r
## S4 method for signature 'LandsepiParams'
initialize(
  .Object,
  Landscape = st_sf(st_sfc()),
  CropTypes = data.frame(),
  Cultivars = data.frame(matrix(ncol = length(.cultivarsColNames), nrow = 0, dimnames = list(NULL, .cultivarsColNames))),
  CultivarsGenes = data.frame(),
  Genes = data.frame(matrix(ncol = length(.geneColNames), nrow = 0, dimnames = list(NULL, .geneColNames))),
  Pathogen = list(name = "no pathogen", survival_prob = 0, repro_sex_prob = 0, infection_rate = 0, propagule_prod_rate = 0, latent_period_exp = 0, latent_period_var = 0, infectious_period_exp = 0, infectious_period_var = 0, sigmoid_kappa = 0, sigmoid_sigma = 0, sigmoid_plateau = 1),
  PI0 = 0,
  DispHost = vector(),
)```
initialize.LandseiParams-method

```r
DispPatho = vector(),
OutputDir = normalizePath(character(getwd())),
OutputGPKG = "landsepi_landscape.gpkg",
Outputs = list(epid_outputs = "", evol_outputs = "", thres_breakdown = NA, GLAnoDis = NA, audpc100S = NA),
TimeParam = list(),
Seed = NULL,
...
```

Arguments

- **Object**: a LandseiParam object.
- **Landscape**: a landscape as sf object.
- **Croptypes**: a dataframe with three columns named 'croptypeID' for croptype index, 'cultivarID' for cultivar index and 'proportion' for the proportion of the cultivar within the croptype.
- **Cultivars**: a dataframe of parameters associated with each host genotype (i.e. cultivars, lines) when cultivated in pure crops.
- **CultivarsGenes**: a list containing, for each host genotype, the indices of carried resistance genes.
- **Genes**: a data.frame of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene.
- **Pathogen**: a list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance.
- **PI0**: initial probability for the first host (whose index is 0) to be infectious (i.e. state I) at the beginning of the simulation. Must be between 0 and 1.
- **DispHost**: a vectorized matrix giving the probability of host dispersal from any field of the landscape to any other field.
- **DispPatho**: a vectorized matrix giving the probability of pathogen dispersal from any field of the landscape to any other field.
- **OutputDir**: the directory for simulation outputs.
- **OutputGPKG**: the name of the output GPKG file containing parameters of the deployment strategy.
- **Outputs**: a list of outputs parameters.
- **TimeParam**: a list of time parameters.
- **Seed**: an integer used as seed value (for random number generator).
- **...**: more options
**invlogit**

*Inverse logit function*

**Description**

Given a numeric object, return the invlogit of the values. Missing values (NAs) are allowed.

**Usage**

```r
invlogit(x)
```

**Arguments**

- `x` a numeric object

**Details**

The invlogit is defined by \( \frac{\exp(x)}{1+\exp(x)} \). Values in `x` of -\( \infty \) or \( \infty \) return invlogits of 0 or 1 respectively. Any NAs in the input will also be NAs in the output.

**Value**

An object of the same type as `x` containing the invlogits of the input values.

**Examples**

```r
invlogit(10)
```

---

**is.in.01**

*is.in.01*

**Description**

Tests if a number or vector is in the interval [0,1]

**Usage**

```r
is.in.01(x, exclude0 = FALSE)
```

**Arguments**

- `x` a number or vector or matrix
- `exclude0` TRUE is 0 is excluded, FALSE otherwise (default)

**Value**

a logical of the same size as `x`
**is.positive**

Tests if a number or vector is positive (including 0)

**Usage**

`is.positive(x)`

**Arguments**

- **x**: a number or vector or matrix

**Value**

a logical of the same size as x

**Examples**

- `is.positive(-5)`
- `is.positive(10)`
- `is.positive(2.5)`
- `is.positive(matrix(1:9, nrow=3))`

---

**is.strict.positive**

Tests if a number or vector is strictly positive (i.e. excluding 0)

**Usage**

`is.strict.positive(x)`

**Description**

Tests if a number or vector is strictly positive (i.e. excluding 0)

**Usage**

`is.strict.positive(x)`

**Arguments**

- **x**: a number or vector or matrix

**Value**

a logical of the same size as x

**Examples**

- `is.strict.positive(-5)`
- `is.strict.positive(10)`
- `is.strict.positive(2.5)`
- `is.strict.positive(matrix(1:9, nrow=3))`
Arguments

\( x \)  
a number or vector or matrix

Value

a logical of the same size as \( x \)

Examples

\begin{verbatim}
is.strict.positive(-5)  
is.strict.positive(10)  
is.strict.positive(2.5)  
is.strict.positive(matrix(1:9, nrow=3))
\end{verbatim}

Description

Tests if a number or vector is a whole number

Usage

\begin{verbatim}
is.wholenumber(x, tol = .Machine$double.eps^0.5)
\end{verbatim}

Arguments

\( x \)  
a number or vector or matrix  
\( tol \)  
double tolerance

Value

a logical of the same format as \( x \)

Examples

\begin{verbatim}
is.wholenumber(-5)  
is.wholenumber(10)  
is.wholenumber(2.5)  
is.wholenumber(matrix(1:9, nrow=3))
\end{verbatim}
Description

Five simulated landscapes, composed of 155, 154, 152, 153 and 156 fields, respectively.

Usage

landscapeTEST1
landscapeTEST2
landscapeTEST3
landscapeTEST4
landscapeTEST5

Format

Landscapes have been generated using a T-tessellation algorithm. The format is a formal class 'SpatialPolygons' [package "sp"].

Details

The landscape structure is simulated using a T-tessellation algorithm (Kiêu et al. 2013) in order to control specific features such as number, area and shape of the fields.

References


Examples

library(sp)
library(landsepi)
landscapeTEST1
plot(landscapeTEST1)

LandsepiParams

Class LandsepiParams

Description

Landsepi simulation parameters

Details

An object of class LandsepiParams that can be created by calling createSimulParams
Slots

Landscape a landscape as sf object. See loadLandscape, loadLandscape
Croptypes a dataframe with three columns named 'croptypeID' for croptype index, 'cultivarID' for cultivar index and 'proportion' for the proportion of the cultivar within the croptype. See loadCroptypes, setCroptypes and See allocateCroptypeCultivars
Cultivars a dataframe of parameters associated with each host genotype (i.e. cultivars, lines) when cultivated in pure crops. See loadCultivar and setCultivars
CultivarsGenes a list containing, for each host genotype, the indices of carried resistance genes. See allocateCultivarGenes
Genes a dataframe of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene. See loadGene and setGenes
Pathogen a list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance. See loadPathogen and setPathogen
PI0 initial probability for the first host (whose index is 0) to be infectious (i.e. state I) at the beginning of the simulation. Must be between 0 and 1. See setInoculum
DispHost a vectorized matrix giving the probability of host dispersal from any field of the landscape to any other field. See loadDispersalHost and setDispersalHost
DispPatho a vectorized matrix giving the probability of pathogen dispersal from any field of the landscape to any other field. See loadDispersalPathogen and setDispersalPathogen
OutputDir the directory for simulation outputs
OutputGPKG the name of the output GPKG file containing parameters of the deployment strategy
Outputs a list of outputs parameters. See setOutputs
TimeParam a list of time parameters. See setTime
Seed an integer used as seed value (for random number generator). See setTime

loadCroptypes Load Croptypes

Description

Creates a data.frame containing croptype parameters and filled with 0

Usage

loadCroptypes(params, croptypeIDs = NULL, names = NULL)

Arguments

params a LandsepiParams Object.
croptypeIDs a vector of indices of croptypes (must match with croptypes in the landscape)
names a vector containing the names of all croptypes
loadCultivar

Description

create a data.frame containing cultivar parameters depending of his type

Usage

loadCultivar(name, type = "growingHost")

Arguments

name a character string (without space) specifying the cultivar name.

type the cultivar type: "growingHost", "nongrowingHost" or "nonhost" (default = "nonhost").
Details

- "growingHost" is adapted to situations where the infection unit is a piece of leaf (e.g. where a fungal lesion can develop); the number of available infection units increasing during the season due to plant growth.
- "nongrowingHost" corresponds to situations where the infection unit is the whole plant (e.g. for viral systemic infection); thus the number of infection units is constant.
- "nonCrop" is not planted, does not cost anything and does not yield anything (e.g. forest, fallow).

Value

a dataframe of parameters associated with each host genotype (i.e. cultivars, lines) when cultivated in pure crops.

See Also

setCultivars

Examples

c1 <- loadCultivar("winterWheat", type = "growingHost")
c1
c2 <- loadCultivar("forest", type = "nonhost")
c2

loadDispersalHost Load a host dispersal matrix

Description

It loads a vectorised diagonal matrix to simulate no host dispersal.

Usage

loadDispersalHost(params, type = "no")

Arguments

params a LandsepiParams Object.
type a character string specifying the type of dispersal ("no" for no dispersal)

Details

as the size of the matrix depends on the number of fields in the landscape, the landscape must be defined before calling loadDispersalHost.
loadDispersalPathogen

Load a pathogen dispersal matrix

Description

It loads one of the five built-in vectorised dispersal matrices of rust fungi associated with the five built-in landscapes. Landscape and DispersalPathogen ID must be the same.

Usage

loadDispersalPathogen(id = 1)

Arguments

id

a matrix ID between 1 to 5 (must match the ID of the landscape loaded with loadLandscape).

Details

landsepi includes built-in dispersal matrices to represent rust dispersal in the five built-in landscapes. These have been computed from a power-law dispersal kernel: 

\[ g(d) = \frac{(b - 2) \times (b - 1)/(2 \times pi \times a^2)) \times (1 + d/a)^{(-b)}}{a=40 \text{ the scale parameter and } b=7 \text{ a parameter related to the width of the dispersal kernel. The expected mean dispersal distance is given by } 2\times a/(b-3)=20 \text{ m.}} \]

Value

a vectorised dispersal matrix.

See Also

dispP, setDispersalPathogen
Examples

```r
d <- loadDispersalPathogen(1)
d
```

---

**loadGene**

*Load a gene*

**Description**

Creates a data.frame containing parameters of a gene depending of his type

**Usage**

```r
loadGene(name, type = "majorGene")
```

**Arguments**

- `name`: name of the gene
- `type`: type of the gene: "majorGene", "APR", "QTL" or "immunity" (default = "majorGene")

**Details**

- "majorGene" means a completely efficient gene that can be broken down via a single pathogen mutation
- "APR" means a major gene that is active only after a delay of 30 days after planting
- "QTL" means a partial resistance (50% efficiency) that requires several pathogen mutations to be completely eroded
- "immunity" means a completely efficient resistance that the pathogen has no way to adapt (i.e. the cultivar is nonhost).

For different scenarios, the data.frame can be manually updated later.

**Value**

a data.frame with gene parameters

**See Also**

`setGenes`

**Examples**

```r
gene1 <- loadGene(name = "MG 1", type = "majorGene")
gene1
gene2 <- loadGene(name = "Lr34", type = "APR")
gene2
```
loadLandscape  

Load a landscape

Description

Loads one of the five built-in landscapes simulated using a T-tessellation algorithm and composed of 155, 154, 152, 153 and 156 fields, respectively. Each landscape is identified by a numeric from 1 to 5.

Usage

loadLandscape(id = 1)

Arguments

id  
a landscape ID between 1 to 5 (default = 1)

Value

a landscape in sp format

See Also

landscapeTEST, setLandscape

Examples

land <- loadLandscape(1)
length(land)

loadOutputs  

Load outputs

Description

Creates an output list

Usage

loadOutputs(epid_outputs = "all", evol_outputs = "all")
Arguments

epid_outputs a character string (or a vector of character strings if several outputs are to be computed) specifying the type of epidemiological and economic outputs to generate (see details):

- "audpc" : Area Under Disease Progress Curve (average number of diseased host individuals per time step and square meter)
- "audpc_rel" : Relative Area Under Disease Progress Curve (average proportion of diseased host individuals relative to the total number of existing hosts)
- "gla" : Green Leaf Area (average number of healthy host individuals per time step and square meter)
- "gla_rel" : Relative Green Leaf Area (average proportion of healthy host individuals relative to the total number of existing hosts)
- "eco_yield" : total crop yield (in weight or volume units per ha)
- "eco_cost" : operational crop costs (in monetary units per ha)
- "eco_product" : total crop products (in monetary units per ha)
- "eco_margin" : Margin (products - operational costs, in monetary units per ha)
- "HLIR_dynamics", "H_dynamics", "L_dynamics", "IR_dynamics", "HLI_dynamics", etc.: Epidemic dynamics related to the specified sanitary status (H, L, I or R and all their combinations). Graphics only, works only if graphic=TRUE.
- "all" : compute all these outputs (default)
- "" : none of these outputs will be generated.

evol_outputs a character string (or a vector of character strings if several outputs are to be computed) specifying the type of evolutionary outputs to generate:

- "evol_patho": Dynamics of pathogen genotype frequencies
- "evol_aggr": Evolution of pathogen aggressiveness
- "durability": Durability of resistance genes
- "all": compute all these outputs (default)
- "" : none of these outputs will be generated.

Value

a list of outputs and parameters for output generation

See Also

setOutputs

Examples

outputList <- loadOutputs(epid_outputs = "audpc", evol_outputs = "durability")
outputList
loadPathogen

Load pathogen parameters

Description

Loads default pathogen parameters for a specific disease

Usage

loadPathogen(disease = "rust")

Arguments

disease a disease name (default: "rust")

Details

Available diseases:

• "rust"

Value

a list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance

See Also

setPathogen

Examples

basic_patho_params <- loadPathogen()
basic_patho_params

loadSimulParams

Load simulation parameters

Description

Loads a GPKG file from the output of a landsepi simulation.

Usage

loadSimulParams(inputGPKG = "")
logit

**Arguments**

inputGPKG  
name of the GPKG file.

**Details**

See `saveDeploymentStrategy`.

**Value**

a LandsepiParams object.

---

### logit  
*Logit function*

**Description**

Given a numeric object, return the logit of the values. Missing values (NAs) are allowed.

**Usage**

`logit(x)`

**Arguments**

x  
a numeric object containing values between 0 and 1

**Details**

The logit is defined by log(x/(1-x)). Values in x of 0 or 1 return logits of -Inf or Inf respectively. Any NAs in the input will also be NAs in the output.

**Value**

An object of the same type as x containing the logits of the input values.

**Examples**

`logit(0.5)`
model_landsepi  

Model for Landscape Epidemiology & Evolution

Description

Stochastic, spatially-explicit, demo-genetic model simulating the spread and evolution of a plant pathogen in a heterogeneous landscape.

Usage

model_landsepi(
  time_param,
  area_vector,
  rotation_matrix,
  croptypes_cultivars_prop,
  dispersal,
  inits,
  seed,
  cultivars_param,
  basic_patho_param,
  genes_patho
)

Arguments

time_param list of simulation parameters:
  • Nyears = number cropping seasons,
  • nTSpY = number of time-steps per cropping season.

area_vector a vector containing areas of polygons (i.e. fields), in surface units.

rotation_matrix a matrix containing for each field (rows) and year (columns, named "year_1", "year_2", etc.), the index of the cultivated croptype. Importantly, the matrix must contain 1 more column than the real number of simulated years.

croptypes_cultivars_prop a matrix with three columns named 'croptypeID' for croptype index, 'cultivarID' for cultivar index and 'proportion' for the proportion of the cultivar within the croptype.

dispersal list of dispersal parameters:
  • disp_patho = vectorised dispersal matrix of the pathogen,
  • disp_host = vectorised dispersal matrix of the host.

inits list of initial conditions:
  • pI0 = initial probability for the first host (whose index is 0) to be infectious (i.e. state I) at t=0.

seed seed (for random number generation).
cultivars_param
list of parameters associated with each host genotype (i.e. cultivars) when cultivated in pure crops:

- initial_density = vector of host densities (per surface unit) at the beginning of the cropping season,
- max_density = vector of maximum host densities (per surface unit) at the end of the cropping season,
- growth_rate = vector of host growth rates,
- reproduction_rate = vector of host reproduction rates,
- death_rate = vector of host death rates,
- sigmoid_kappa_host = kappa parameter for the sigmoid invasion function (for host dispersal),
- sigmoid_sigma_host = sigma parameter for the sigmoid invasion function (for host dispersal),
- sigmoid_plateau_host = plateau parameter for the sigmoid invasion function (for host dispersal),
- cultivars_genes_list = a list containing, for each host genotype, the indices of carried resistance genes.

basic_patho_param
list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance:

- infection_rate = maximal expected infection rate of a propagule on a healthy host,
- propagule_prod_rate = maximal expected reproduction_rate of an infectious host per timestep,
- latent_period_exp = minimal expected duration of the latent period,
- latent_period_var = variance of the latent period duration,
- infectious_period_exp = maximal expected duration of the infectious period,
- infectious_period_var = variance of the infectious period duration,
- survival_prob = probability for a propagule to survive the off-season,
- repro_sex_prob = probability for an infectious host to reproduce via sex rather than via cloning,
- sigmoid_kappa = kappa parameter of the sigmoid contamination function,
- sigmoid_sigma = sigma parameter of the sigmoid contamination function,
- sigmoid_plateau = plateau parameter of the sigmoid contamination function.

genes_param
list of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene:

- target_trait = vector of aggressiveness components (IR, LAT, IP, or PR) targeted by resistance genes,
- efficiency = vector of resistance gene efficiencies (percentage of reduction of the targeted aggressiveness component: IR, 1/LAT, IP and PR),
- time_to_activ_exp = vector of expected delays to resistance activation (for APRs),
• time_to_activ_var = vector of variances of the delay to resistance activation (for APRs),
• mutation_prob = vector of mutation probabilities for pathogenicity genes (each of them corresponding to a resistance gene),
• Nlevels_aggressiveness = vector of number of adaptation levels related to each resistance gene (i.e. 1 + number of required mutations for a pathogenicity gene to fully adapt to the corresponding resistance gene),
• fitness_cost = vector of fitness penalties paid by pathogen genotypes fully adapted to the considered resistance genes on hosts that do not carry this gene,
• tradeoff_strength = vector of strengths of the trade-off relationships between the level of aggressiveness on hosts that do and do not carry the resistance genes.

Details
See \_landsepi for details on the model and assumptions. Briefly, the model is stochastic, spatially explicit (the basic spatial unit is an individual field), based on a SEIR ("susceptible-exposed-infectious-removed", renamed HLIR for 'healthy-latent-infectious-removed' to avoid confusions with 'susceptible host') structure with a discrete time step. It simulates the spread and evolution of a pathogen in a heterogeneous cropping landscape, across cropping seasons split by host harvests which impose potential bottlenecks to the pathogen. A wide array of resistance deployment strategies can be simulated.

Value
A set of binary files is generated for every year of simulation and every compartment:

• H: healthy hosts,
• Hjuv: juvenile healthy hosts,
• L: latently infected hosts,
• I: infectious hosts,
• R: removed hosts,
• P: propagules.

Each file indicates for every time-step the number of individuals in each field, and when appropriate for each host and pathogen genotypes).

References
Examples

```r
### Not run:
#### Spatially-implicit simulation with 2 patches (S + R) during 3 years ####

**Simulation parameters**

```r
time_param <- list(Nyears=3, nTSpY=120)
Npoly=2
Npatho=2
area <- c(100000, 100000)
cultivars <- as.list(rbind(loadCultivar(name="Susceptible", type="growingHost"), loadCultivar(name="Resistant", type="growingHost")))
names(cultivars)[names(cultivars)=="cultivarName"] <- "name"
cultivars <- c(cultivars, list(sigmoid_kappa_host=0.002, sigmoid_sigma_host=1.001, sigmoid_plateau_host=1, cultivars_genes_list=list(numeric(0),0)))
rotation <- data.frame(year_1=c(0,1), year_2=c(0,1), year_3=c(0,1), year_4=c(0,1))
croptypes_cultivars_prop <- data.frame(croptypeID=c(0,1), cultivarID=c(0,1), proportion=c(1,1))
genes <- as.list(loadGene(name="MG", type="majorGene"))
```

**run simulation**

```r
model_landsepi(seed=1,
time_param = time_param,
basic_patho_param = loadPathogen(disease = "rust"),
inits = list(pI0=0.01), area_vector = area,
dispersal = list(disp_patho=c(0.99,0.01,0.01,0.99), disp_host=c(1,0,0,1)),
rotation_matrix = as.matrix(rotation),
croptypes_cultivars_prop = as.matrix(croptypes_cultivars_prop),
cultivars_param = cultivars, genes_param = genes)
```

**Compute outputs**

```r
eco_param <- list(yield_perHa = cbind(H = as.numeric(cultivars$yield_H), L = as.numeric(cultivars$yield_L), I = as.numeric(cultivars$yield_I), R = as.numeric(cultivars$yield_R)), planting_cost_perHa = as.numeric(cultivars$planting_cost), market_value = as.numeric(cultivars$market_value))
evol_res <- evol_output(, time_param, Npoly, cultivars, genes)
epid_output(, time_param, Npatho, area, rotation, croptypes_cultivars_prop, cultivars, eco_param)
```

### 1-year simulation of a rust epidemic in pure susceptible crop in a single 1 ha patch ###

```r
time_param <- list(Nyears=1, nTSpY=120)
Npoly=1
Npatho=1
area <- c(100000)
cultivars <- as.list(rbind(loadCultivar(name="Susceptible", type="growingHost")))
names(cultivars)[names(cultivars)=="cultivarName"] <- "name"
cultivars <- c(cultivars, list(sigmoid_kappa_host=0.002, sigmoid_sigma_host=1.001, sigmoid_plateau_host=1, cultivars_genes_list=list(numeric(0))))
rotation <- data.frame(year_1=c(0), year_2=c(0))
```
multiN

Description

Algorithm based on latent Gaussian fields to allocate two different types of crops across a landscape.

Usage

multiN(d, area, prop, range = 0, algo = "random")

Arguments

d       a symmetric matrix of the pairwise distances between the centroids of the fields of the landscape.
area    vector containing field areas.
prop    proportion of landscape surface covered by the second type of crop.
range   range of spatial autocorrelation between fields (must be greater or equal 0). The greater the value of range, the higher the degree of spatial aggregation (roughly, range between 0 and 0.1 for fragmented landscapes, between 0.1 and 0.5 for balanced landscapes, between 0.5 and 3 for aggregated landscapes, and above 3 for highly aggregated landscapes).
algo    the algorithm used for the computation of the variance-covariance matrix of the multivariate normal distribution: "exp" for exponential function, "periodic" for periodic function, "random" for random draw (see details). If algo="random", the parameter range is ignored.
Details

This algorithm allows the control of the proportions of each type of crop in terms of surface coverage, and their level of spatial aggregation. A random vector of values is drawn from a multivariate normal distribution with expectation 0 and a variance-covariance matrix which depends on the pairwise distances between the centroids of the fields. Two different functions allow the computation of the variance-covariance matrix to allocate crops with more or less spatial aggregation (depending on the value of the range parameter). The exponential function codes for an exponential decay of the spatial autocorrelation as distance between fields increases. The periodic function codes for a periodic fluctuation of the spatial autocorrelation as distance between fields increases. Alternatively, a normal distribution can be used for a random allocation of the types of crops. Next, the two types of crops are allocated to different fields depending on whether the value drawn from the multivariate normal distribution is above or below a threshold. The proportion of each type of crop in the landscape is controlled by the value of this threshold (parameter prop).

Value

A dataframe containing the index of each field (column 1) and the index (0 or 1) of the type of crop grown on these fields (column 2).

Examples

```r
## Not run:
d <- matrix(rpois(100, 100), nrow = 10)
d <- d + t(d) ## ensures that d is symmetric
area <- data.frame(id = 1:10, area = 10)
multiN(d, area, prop = 0.5, range = 0.5, algo = "periodic")
## End(Not run)
```

Description

Converts a LandsepiParams object to a value compatible with BDD croptype Table

Usage

```r
params2CroptypeBDD(params)
```

Arguments

- `params`: a LandsepiParams object.

Value

- a data.frame BDD compatible
**params2CultivarBDD**

**Description**

Converts a LandsepiParams object to a value compatible with BDD cultivar Table

**Usage**

```r
params2CultivarBDD(params)
```

**Arguments**

- `params` a LandsepiParam object.

**Value**

a data.frame BDD compatible

---

**params2GeneBDD**

**Description**

Converts a LandsepiParams object to a value compatible with BDD Gene Table

**Usage**

```r
params2GeneBDD(params)
```

**Arguments**

- `params` a LandsepiParam object.

**Value**

a data.frame BDD compatible
Description

Converts a LandsepiParams object to a value compatible with BDD cultivarsGenes Table

Usage

params2GeneListBDD(params)

Arguments

params a LandsepiParam object.

Value

a data.frame BDD compatible

________________________________________________________

describe_bdd

Description

A function that describes the BDD cultivarsGenes Table

Usage

describe_bdd()

Value

A data.frame with the description of the BDD cultivarsGenes Table.

________________________________________________________

periodic_cov

Periodic covariance function

Description

Periodic function used to compute the variance-covariance matrix of the fields of the landscape.

Usage

periodic_cov(d, range, phi = 1)

Arguments

d a numeric object containing pairwise distances between the centroids of the fields
range range (half-period of oscillations)
phi amplitude of the oscillations

Details

The periodic covariance is defined by \( \exp(-2 * \sin(\pi/(2 \text{range}))^2 / \phi^2) \). It is used to generate highly fragmented or highly aggregated landscapes.

Value

An object of the same type as d.
Examples

periodic_cov(10, range = 5)

Description

Plots a landscape with colors or hatched lines to represent different types of fields

Usage

plotland(
  landscape,
  COL = rep(0, length(landscape)),
  DENS = rep(0, length(landscape)),
  ANGLE = rep(30, length(landscape)),
  COL.LEG = unique(COL),
  DENS.LEG = unique(DENS),
  ANGLE.LEG = unique(ANGLE),
  TITLE = "",
  SUBTITLE = "",
  LEGEND1 = rep("", length(COL.LEG)),
  LEGEND2 = rep("", length(COL.LEG)),
  TITLE.LEG2 = ""
)

Arguments

landscape a spatialpolygon object containing field coordinates
COL vector containing the color of each field
DENS vector containing the density of hatched lines for each field
ANGLE vector containing the angle of hatched lines for each field
COL.LEG vector containing the colors in the first legend
DENS.LEG vector containing the density of hatched lines in the second legend
ANGLE.LEG vector containing the angle of hatched lines in the second legend
TITLE title of the graphic
SUBTITLE subtitle of the graphic
LEGEND1 labels in the first legend (colors)
LEGEND2 labels in the second legend (hatched lines)
TITLE.LEG2 title for the second legend
Examples

```r
## Not run:
## Draw a landscape with various colours
landscapeTEST1
plotland(landscapeTEST1,
   COL = 1:length(landscapeTEST1),
   DENS = rep(0, length(landscapeTEST1)), ANGLE = rep(30, length(landscapeTEST1))
)

## End(Not run)
```

plot_allocation  Plotting allocation of crop types in a landscape

Description

Plots crop type allocation in the landscape at a given year of the simulation

Usage

```r
plot_allocation(
   landscape, 
   year, 
   crop_type_names = c(),
   title = "", 
   subtitle = "", 
   filename = "landscape.png"
)
```

Arguments

- `landscape`: a SpatialPolygonsDataFrame
- `year`: year to be plotted
- `crop_type_names`: crop type names (for legend)
- `title`: title of the graphic
- `subtitle`: subtitle of the graphic
- `filename`: name of the .png file to be generated

Value

a png file.

See Also

plotland
Examples

```r
## Not run:
landscape <- landscapeTEST1
croptypes <- data.frame(sample.int(3, length(landscape), replace = TRUE))
allocation <- SpatialPolygonsDataFrame(landscape, croptypes, match.ID = TRUE)
plot_allocation(allocation, 1,
   title = "Simulated landscape", subtitle = "Year 1",
   filename = paste(getwd(), "/landscape.png", sep = ""))

## End(Not run)
```

---

**plot_freqPatho**

*Plotting pathotype frequencies*

**Description**

Plots in a .tiff file the dynamics of pathotype frequencies with respect to pathogen adaptation to a specific resistance gene.

**Usage**

```r
plot_freqPatho(
   name_gene,
   Nlevels_aggressiveness,
   I_aggrProp,
   nTS,
   Nyears,
   nTSpY
)
```

**Arguments**

- `name_gene` : a string specifying the name of the gene under investigation
- `Nlevels_aggressiveness` : number of pathotypes with respect to the gene under investigation
- `I_aggrProp` : a matrix giving the frequency of every pathotype (rows) for every time-step (columns)
- `nTS` : number of simulated time-steps
- `Nyears` : number of simulated cropping seasons
- `nTSpY` : number of time-steps per cropping season
Examples

```r
## Not run:
freqMatrix <- matrix(0, nrow = 2, ncol = 100)
freqMatrix[2, 26:100] <- (26:100) / 100
freqMatrix[1, ] <- 1 - freqMatrix[2, ]
plot_freqPatho(
  index_gene = 1,
  Nlevels_aggressiveness = 2,
  freqMatrix,
  nTS = 100,
  Nyears = 10,
  nTSpY = 10
)

## End(Not run)
```

---

**Description**

Prints a LandespiParams object.

**Usage**

```r
## S4 method for signature 'LandespiParams'
print(x, ...)
```

**Arguments**

- `x` a LandespiParams object
- `...` print options

---

**resetCultivarsGenes**

**Reset cultivars genes**

**Description**

Resets the lists of genes carried by all cultivars

**Usage**

```r
resetCultivarsGenes(params)
```

**Arguments**

- `params` a LandespiParams object.
runSimul

Value

- a LandsepiParams object

---

runShinyApp

Description

Launches landsepi shiny application into browser

Usage

runShinyApp()

Details

R packages needed to run the shiny app: install.packages(c("shiny", "DT", "shinyjs", "gridExtra", "png", "grid", "future", "promises", "tools"))

---

runSimul

Description

Run a simulation

Runs a simulation with landsepi, a stochastic, spatially-explicit, demo-genetic model simulating the spread and evolution of a pathogen in a heterogeneous landscape and generating a wide range of epidemiological, evolutionary and economic outputs.

Usage

runSimul(
  params,
  graphic = TRUE,
  writeTXT = TRUE,
  videoMP4 = FALSE,
  keepRawResults = FALSE
)

---
Arguments

params: a LandsepiParams Object containing all simulation parameters. Must be initialised with `createSimulParams` and updated using `set*()` methods (see vignettes for details).

graphic: a logical indicating if graphics must be generated (TRUE, default) or not (FALSE).

writeTXT: a logical indicating if outputs must be written in text files (TRUE, default) or not (FALSE).

videoMP4: a logical indicating if a video must be generated (TRUE) or not (FALSE, default). Works only if graphic=TRUE and audpc_rel is computed.

keepRawResults: a logical indicating if binary files must be kept after the end of the simulation (default=FALSE). Careful, many files may be generated if keepRawResults=TRUE.

Details

See `?landsepi` for details on the model, assumptions and outputs, and our vignettes for tutorials (browseVignettes("landsepi")). The function runs the model simulation using a LandsepiParams object. Briefly, the model is stochastic, spatially explicit (the basic spatial unit is an individual field), based on a SEIR ('susceptible-exposed-infectious-removed', renamed HLIR for 'healthy-latent-infectious-removed' to avoid confusions with 'susceptible host') structure with a discrete time step. It simulates the spread and evolution of a pathogen in a heterogeneous cropping landscape, across cropping seasons split by host harvests which impose potential bottlenecks to the pathogen. A wide array of resistance deployment strategies can be simulated and evaluated using several possible outputs to assess the epidemiological, evolutionary and economic performance of deployment strategies.

Value

A list containing all required outputs. A set of text files, graphics and a video showing epidemic dynamics can be generated. If keepRawResults=TRUE, a set of binary files is generated for every year of simulation and every compartment:

- H: healthy hosts,
- Hjuv: juvenile healthy hosts,
- L: latently infected hosts,
- I: infectious hosts,
- R: removed hosts,
- P: propagules.

Each file indicates for every time-step the number of individuals in each field, and when appropriate for each host and pathogen genotypes.

References

Examples

```r
## Not run:
## Here is an example of simulation of a mosaic of three cultivars (S + R1 + R2). See our tutorials for more examples.
## Initialisation
simul_params <- createSimulParams(outputDir = getwd())
## Seed & Time parameters
simul_params <- setSeed(simul_params, seed = 1)
simul_params <- setTime(simul_params, Nyears = 10, nTSpY = 120)
## Pathogen & inoculum parameters
simul_params <- setPathogen(simul_params, loadPathogen("rust"))
simul_params <- setInoculum(simul_params, 5e-4)
## Landscape & dispersal
simul_params <- setLandscape(simul_params, loadLandscape(1))
simul_params <- setDispersalPathogen(simul_params, loadDispersalPathogen(1))
## Genes
gene1 <- loadGene(name = "MG 1", type = "majorGene")
gene2 <- loadGene(name = "MG 2", type = "majorGene")
genes <- data.frame(rbind(gene1, gene2), stringsAsFactors = FALSE)
simul_params <- setGenes(simul_params, genes)
## Cultivars
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
## Allocate genes to cultivars
simul_params <- allocateCultivarGenes(simul_params, "Resistant1", c("MG 1"))
simul_params <- allocateCultivarGenes(simul_params, "Resistant2", c("MG 2"))
## Allocate cultivars to croptypes
croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop",
          "Susceptible crop 1",
          "Susceptible crop 2"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop 1", "Resistant1")
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop 2", "Resistant2")
simul_params <- setCroptypes(simul_params, croptypes)
## Allocate croptypes to landscape
rotation_sequence <- croptypes$croptypeID  # No rotation -> 1 rotation_sequence element
rotation_period <- 0  # # same croptypes every years
prop <- c(1/3, 1/3, 1/3)  # # croptypes proportions
agreg <- 10  # aggregated landscape
simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = rotation_period, rotation_sequence = rotation_sequence,
rotation_realloc = FALSE, prop = prop, agreg = agreg)
## list of outputs to be generated
simul_params <- setOutputs(simul_params, loadOutputs())
## Check simulation parameters
checkSimulParams(simul_params)
## Save deployment strategy into GPKG file
simul_params <- saveDeploymentStrategy(simul_params)
## Run simulation
```
saveDeploymentStrategy

Save landscape and deployment strategy

Description
Generates a GPKG file containing the landscape and all parameters of the deployment strategy

Usage
saveDeploymentStrategy(
  params,
  outputGPKG = "landsepi_landscape.gpkg",
  overwrite = FALSE
)

Arguments
params: a LandsepiParams Object.
outputGPKG: name of the GPKG output (default: "landsepi_landscape.gpkg") to be generated.
overwrite: a boolean specifying if existing files can be overwritten (TRUE) or not (FALSE, default).

Details
The function generates a GPKG file in the simulation path. The GPKG file contains all input parameters needed to restore the landscape (sf object) and deployment strategy (croptypes, cultivars and genes).

Value
an updated LandsepiParams object.

Examples
## Not run:
## Initialisation
simul_params <- createSimulParams(outputDir = getwd())
## Time parameters
simul_params <- setTime(simul_params, Nyears = 10, nTSpY = 120)
## Landscape
simul_params <- setLandscape(simul_params, loadLandscape(1))
## Genes
gene1 <- loadGene(name = "MG 1", type = "majorGene")
setCroptypes

Description

Updates a LandsepiParams object with croptypes and their composition with regard to cultivar proportions

Usage

setCroptypes(params, dfCroptypes)

Arguments

params a LandsepiParams Object.
dfCroptypes a data.frame containing cultivar proportions in each croptype (see details). It can be generated manually, or initialised with loadCroptypes and later updated with allocateCroptypeCultivars.
setCultivars

Details

The data.frame for cultivar allocations into croptypes must take this format (example):

<table>
<thead>
<tr>
<th>croptypeID</th>
<th>croptypeName</th>
<th>cultivarName1</th>
<th>cultivarName2</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>&quot;crop1&quot;</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>&quot;crop2&quot;</td>
<td>0.5</td>
<td>0.5</td>
<td></td>
</tr>
</tbody>
</table>

croptypeIDs have to match values from landscape "croptypeID" layer with feature year_X. Cultivars names have to match cultivar names in the cultivars data.frame.

Value

a LandsepiParams object

See Also

loadCroptypes

Examples

```r
## Not run:
simul_params <- createSimulParams()
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop", "Mixture"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture", c("Resistant1", "Resistant2"))
simul_params <- setCroptypes(simul_params, croptypes)
simul_params@Croptypes

## End(Not run)
```

Description

Updates a LandsepiParams object with cultivars parameters

Usage

```r
setCultivars(params, dfCultivars)
```
Arguments

params a landsepiParams object.

dfCultivars a data.frame defining the cultivars (see details). It can be generated manually or, alternatively, via `loadCultivar`.

Details

dfCultivars is a dataframe of parameters associated with each host genotype (i.e. cultivars, lines) when cultivated in pure crops. Columns of the dataframe are:

- cultivarName: cultivar names (cannot accept space),
- initial_density: host densities (per square meter) at the beginning of the cropping season as if cultivated in pure crop,
- max_density: maximum host densities (per square meter) at the end of the cropping season as if cultivated in pure crop,
- growth rate: host growth rates,
- reproduction rate: host reproduction rates,
- death rate: host death rates,
- yield_H: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status H as if cultivated in pure crop,
- yield_L: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status L as if cultivated in pure crop,
- yield_I: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status I as if cultivated in pure crop,
- yield_R: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status R as if cultivated in pure crop,
- planting_cost = planting costs (in monetary units / ha / cropping season) as if cultivated in pure crop,
- market_value = market values of the production (in monetary units / weight or volume unit).

The data.frame must be defined as follow (example):

<table>
<thead>
<tr>
<th>cultivarName</th>
<th>initial_density</th>
<th>max_density</th>
<th>growth_rate</th>
<th>reproduction_rate</th>
<th>death_rate</th>
<th>yield_H</th>
<th>yield_L</th>
<th>yield_I</th>
<th>yield_R</th>
<th>planting_cost</th>
<th>market_value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Susceptible</td>
<td>0.1</td>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>225</td>
<td>200</td>
</tr>
<tr>
<td>Resistant1</td>
<td>0.1</td>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>225</td>
<td>200</td>
</tr>
<tr>
<td>Resistant2</td>
<td>0.1</td>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
<td>0.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>225</td>
<td>200</td>
</tr>
</tbody>
</table>

Value

a LandsepiParams object

See Also

`loadCultivar`
setDispersalHost

Examples

```r
## Not run:
simul_params <- createSimulParams()
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
simul_params@Cultivars
## End(Not run)
```

setDispersalHost  Set host dispersal

Description

Updates a LandsepiParams object with a host dispersal matrix.

Usage

```r
setDispersalHost(params, mat)
```

Arguments

- `params`: a LandsepiParams Object.
- `mat`: a square matrix giving the probability of host dispersal from any field of the landscape to any other field. It can be generated manually, or, alternatively, via `loadDispersalHost`. The size of the matrix must match the number of fields in the landscape.

Details

The dispersal matrix gives the probability for a host individual in a field i (row) to migrate to field j (column) through dispersal. If the host is a cultivated plant: seeds are harvested and do not disperse. Thus the dispersal matrix is the identity matrix.

Value

a LandsepiParam object.

See Also

- `loadDispersalHost`
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setLandscape(simul_params, loadLandscape(1))
d <- loadDispersalHost(simul_params)
simul_params <- setDispersalHost(simul_params, d)
simul_params@DispHost

## End(Not run)
```

Description

Updates a LandsepiParams object with a pathogen dispersal matrix.

Usage

```r
setDispersalPathogen(params, mat)
```

Arguments

- `params`: a LandsepiParams Object.
- `mat`: a square matrix giving the probability of pathogen dispersal from any field of the landscape to any other field. It can be generated manually, or, alternatively, via `loadDispersalPathogen`. The size of the matrix must match the number of fields in the landscape, and lines of the matrix must sum to 1.

Details

See tutorial (vignettes) on how to use your own landscape and compute your own pathogen dispersal kernel. The dispersal matrix a square matrix whose size is the number of fields in the landscape and whose elements are, for each line i and each column i’ the probability that propagules migrate from field i to field i’.

Value

a LandsepiParam object.

See Also

- `loadDispersalPathogen`
**Examples**

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setLandscape(simul_params, loadLandscape(1))
d <- loadDispersalPathogen(1)
simul_params <- setDispersalPathogen(simul_params, d)
simul_params@DispPatho

## End(Not run)
```

---

**Description**

Updates a LandsepiParams object with parameters associated with resistance genes and pathogen adaptation.

**Usage**

```r
setGenes(params, dfGenes)
```

**Arguments**

- `params`: a LandsepiParams object
- `dfGenes`: a data.frame containing gene parameters. It can be defined manually, or, alternatively, with `loadGene`.

**Details**

dfGenes is a data.frame of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene. Columns of the dataframe are:

- `geneName`: names of resistance genes,
- `target_trait`: aggressiveness components (IR, LAT, IP, or PR) targeted by resistance genes,
- `efficiency`: resistance gene efficiencies, i.e. the percentage of reduction of the targeted aggressiveness component (IR, 1/LAT, IP and PR),
- `time_to_activ_exp`: expected delays to resistance activation (for APRs),
- `time_to_activ_var`: variances of the delay to resistance activation (for APRs),
- `mutation_prob`: mutation probabilities for pathogenicity genes (each of them corresponding to a resistance gene),
- `Nlevels_aggressiveness`: number of adaptation levels related to each resistance gene (i.e. 1 + number of required mutations for a pathogenicity gene to fully adapt to the corresponding resistance gene),
- `fitness_cost`: fitness penalties paid by pathogen genotypes fully adapted to the considered resistance genes on host that do not carry these genes.
setInoculum

• tradeoff_strength: strengths of the trade-off relationships between the level of aggressiveness on hosts that do and do not carry the resistance genes.

The data.frame must be defined as follow (example):

<table>
<thead>
<tr>
<th>geneName</th>
<th>efficiency</th>
<th>time_to_activ_exp</th>
<th>time_to_activ_var</th>
<th>mutation_prob</th>
<th>Nlevels_aggressiveness</th>
<th>fitness_cost</th>
<th>tradeoff_strength</th>
</tr>
</thead>
<tbody>
<tr>
<td>MG1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1e-07</td>
<td>2</td>
<td>0.5</td>
<td>IR</td>
</tr>
<tr>
<td>QTL1</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>0.0001</td>
<td>10</td>
<td>0.74</td>
<td></td>
</tr>
</tbody>
</table>

Value

a LandsepiParams object.

See Also

loadGene

Examples

```r
## Not run:
simul_params <- createSimulParams()
gene1 <- loadGene(name = "MG 1", type = "majorGene")
gene2 <- loadGene(name = "MG 2", type = "majorGene")
genes <- data.frame(rbind(gene1, gene2), stringsAsFactors = FALSE)
simul_params <- setGenes(simul_params, genes)
simul_params@Genes

## End(Not run)
```

Description

Updates a LandsepiParams with the initial probability for the first host (whose index is 0) to be infectious (i.e. state I) at the beginning of the simulation.

Usage

```r
setInoculum(params, val = 5e-04)
```

Arguments

- **params**: a LandsepiParams object.
- **val**: a numeric value (default = 5e-4). Must be between 0 and 1.

Value

a LandsepiParams object
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setInoculum(simul_params, 1E-3)
simul_params@PI0

## End(Not run)
```

---

**setLandscape**

*Set the landscape*

**Description**

Updates a LandsepiParams object with a sp or sf object as landscape.

**Usage**

```r
setLandscape(params, land)
```

**Arguments**

- `params` a LandsepiParams Object.
- `land` a landscape as sp or sf object

**Details**

The landscape should be a sp or sf object. Built-in landscape are available using `loadLandscape`. See our tutorial (vignettes) for details on how to use your own landscape. If the landscape contains only polygons, croptypes can be allocated later using `allocateLandscapeCroptypes`. Otherwise the landscape has to contain a data.frame specifying for every year, the index of the croptype cultivated in each polygon. Each feature has a field identified by "year_XX" (XX <- seq(1:Nyears+1)) and containing the croptype ID.

<table>
<thead>
<tr>
<th>Features/fields</th>
<th>year_1</th>
<th>year_2</th>
<th>... year_Nyears+1</th>
</tr>
</thead>
<tbody>
<tr>
<td>polygons1</td>
<td>13</td>
<td>10</td>
<td>13</td>
</tr>
<tr>
<td>polygonsX</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

**Value**

a LandsepiParams object.

**See Also**

`loadLandscape`
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setLandscape(simul_params, loadLandscape(1))
simul_params@Landscape

## End(Not run)
```

---

**setOutputs**

*Set outputs*

**Description**

Updates a LandsepiParams object with a list of output parameters.

**Usage**

```r
setOutputs(params, output_list)
```

**Arguments**

- **params**: a LandsepiParams object.
- **output_list**: a list of outputs to be generated and parameters for output generation. It can be generated manually or, alternatively, via `loadOutputs`. This list is composed of:
  - `epid_outputs` = epidemiological outputs to compute (see details)
  - `evol_outputs` = evolutionary outputs to compute (see details)
  - `thres_breakdown` = an integer (or vector of integers) giving the threshold (i.e. number of infections) above which a pathogen genotype is unlikely to go extinct, used to characterise the time to invasion of resistant hosts (several values are computed if several thresholds are given in a vector).
  - `GLAnoDis` = the absolute Green Leaf Area in absence of disease (used to compute economic outputs).
  - `audpc100S` = the audpc in a fully susceptible landscape (used as reference value for graphics).

**Details**

- "epid_outputs" is a character string (or a vector of character strings if several outputs are to be computed) specifying the type of epidemiological and economic outputs to generate:
  - "audpc" : Area Under Disease Progress Curve (average number of diseased host individuals per time step and square meter)
  - "audpc_rel" : Relative Area Under Disease Progress Curve (average proportion of diseased host individuals relative to the total number of existing hosts)
  - "gla" : Green Leaf Area (average number of healthy host individuals per square meter)
setOutputs

- "gla_rel" : Relative Green Leaf Area (average proportion of healthy host individuals relative to the total number of existing hosts)
- "eco_yield" : total crop yield (in weight or volume units per ha)
- "eco_cost" : operational crop costs (in monetary units per ha)
- "eco_product" : total crop products (in monetary units per ha)
- "eco_margin" : Margin (products - costs, in monetary units per ha)
- "HLIR_dynamics", "H_dynamics", "L_dynamics", "IR_dynamics", "HLI_dynamics", etc.: Epidemic dynamics related to the specified sanitary status (H, L, I or R and all their combinations). Graphics only, works only if graphic=TRUE.
- "all" : compute all these outputs (default)
- "" : none of these outputs will be generated.

"evol_outputs" is a character string (or a vector of character strings if several outputs are to be computed) specifying the type of evolutionary outputs to generate:

- "evol_patho": Dynamics of pathogen genotype frequencies
- "evol_aggr": Evolution of pathogen aggressiveness
- "durability": Durability of resistance genes
- "all": compute all these outputs (default)
- "" : none of these outputs will be generated.

Value

a LandsepiParams object.

See Also

loadOutputs

Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setOutputs(simul_params, loadOutputs())
simul_params@Outputs

## End(Not run)
```
setPathogen

Set the pathogen

Description

Updates a LandsepiParams object with pathogen parameters

Usage

setPathogen(params, patho_params)

Arguments

params a LandsepiParams Object.
patho_params a list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance:
  • infection_rate = maximal expected infection rate of a propagule on a healthy host,
  • propagule_prod_rate = maximal expected effective propagule production rate of an infectious host per time step,
  • latent_period_exp = minimal expected duration of the latent period,
  • latent_period_var = variance of the latent period duration,
  • infectious_period_exp = maximal expected duration of the infectious period,
  • infectious_period_var = variance of the infectious period duration,
  • survival_prob = probability for a propagule to survive the off-season,
  • repro_sex_prob = probability for an infectious host to reproduce via sex rather than via cloning,
  • sigmoid_kappa = kappa parameter of the sigmoid contamination function,
  • sigmoid_sigma = sigma parameter of the sigmoid contamination function,
  • sigmoid_plateau = plateau parameter of the sigmoid contamination function.

It can be generated manually, or, alternatively, via loadPathogen.

Details

a set of parameters representative of rust fungi can be loaded via loadPathogen.

Value

a LandsepiParams object

See Also

loadPathogen
### setSeed

**Set the seed**

#### Description

Updates a LandsepiParams object with a seed value for random number generator

#### Usage

```r
setSeed(params, seed)
```

#### Arguments

- `params`: a LandsepiParams Object.
- `seed`: an integer used as seed value (for random number generator).

#### Value

a LandsepiParams object.

#### Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setPathogen(simul_params, loadPathogen())
simul_params@Pathogen

## End(Not run)
```
**setSeedValue**

**Description**
Set RNG seed to seed value if not NULL, otherwise set it to timestamps value

**Usage**
```
setSeedValue(seed = NULL)
```

**Arguments**
- **seed**
  - an integer as seed value or NULL

**Details**
Sets seed for "Mersenne-Twister" algorithm using Inversion generation

**Value**
the new seed value for RNG

**Examples**
```
setSeedValue(seed = 10)
```

---

**setTime**

**Set time parameters**

**Description**
Updates a LandsepiParams object with time parameters : Nyears and nTSpY

**Usage**
```
setTime(params, Nyears, nTSpY)
```

**Arguments**
- **params**
  - a LandsepiParams Object.
- **Nyears**
  - an integer giving the number of cropping seasons (e.g. years) to simulate.
- **nTSpY**
  - an integer giving the number of time steps per cropping season (e.g. days).

**Value**
a LandsepiParams object.
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears=10, nTSpY=120)
simul_params@TimeParam

## End(Not run)
```

Description

Shows a LandespiParams object.

Usage

```r
## S4 method for signature 'LandsepiParams'
show(object)
```

Arguments

- `object`: a LandsepiParams object

simul_landsepi Simulation with input parameters as data.frames.

Description

Stochastic, spatially-explicit, demo-genetic model simulating the spread and evolution of a pathogen in a heterogeneous landscape and generating a wide range of epidemiological, evolutionary and economic outputs.

Usage

```r
simul_landsepi(
  seed = 12345,
  time_param = list(Nyears = 20, nTSpY = 120),
  croptype_names,
  croptypes_cultivars_prop,
  cultivars,
  cultivars_genes_list,
  genes,
  landscape = NULL,
  area,
  rotation,
```
basic_patho_param, disp_patho, disp_host, pi0 = 5e-04, epid_outputs = "all", evol_outputs = "all", thres_breakdown = 50000, GLAnoDis = 1.48315, audpc100S = 0.76, writeTXT = TRUE, graphic = TRUE, videoMP4 = FALSE, keepRawResults = FALSE
)

Arguments

seed an integer used as seed value (for random number generator).
time_param a list of simulation parameters:
  - Nyears = number cropping seasons,
  - nTSpY = number of time-steps per cropping season.
croptype_names a vector of croptypes names.
croptypes_cultivars_prop a dataframe with three columns named 'croptypeID' for croptype index, 'cultivarID' for cultivar index and 'proportion' for the proportion of the cultivar within the croptype.
cultivars a dataframe of parameters associated with each host genotype (i.e. cultivars) when cultivated in pure crops. Columns of the dataframe are:
  - cultivarName: cultivar names,
  - initial_density: host densities (per square meter) at the beginning of the cropping season as if cultivated in pure crop,
  - max_density: maximum host densities (per square meter) at the end of the cropping season as if cultivated in pure crop,
  - growth_rate: host growth rates,
  - reproduction rate: host reproduction rates,
  - death_rate: host death rates,
  - yield_H: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status H as if cultivated in pure crop,
  - yield_L: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status L as if cultivated in pure crop,
  - yield_I: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status I as if cultivated in pure crop,
  - yield_R: theoretical yield (in weight or volume units / ha / cropping season) associated with hosts in sanitary status R as if cultivated in pure crop,
  - planting_cost = planting costs (in monetary units / ha / cropping season) as if cultivated in pure crop,
• market_value = market values of the production (in monetary units / weight or volume unit).

cultivars_genes_list
a list containing, for each host genotype, the indices of carried resistance genes.

genes
a data.frame of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene. Columns of the data frame are:

• geneName: names of resistance genes,
• target_trait: aggressiveness components (IR, LAT, IP, or PR) targeted by resistance genes,
• efficiency: resistance gene efficiencies (percentage of reduction of targeted aggressiveness components: IR, 1/LAT, IP and PR),
• time_to_activ_exp: expected delays to resistance activation (for APRs),
• time_to_activ_var: variances of the delay to resistance activation (for APRs),
• mutation_prob: mutation probabilities for pathogenicity genes (each of them corresponding to a resistance gene),
• Nlevels_aggressiveness: number of adaptation levels related to each resistance gene (i.e. 1 + number of required mutations for a pathogenicity gene to fully adapt to the corresponding resistance gene),
• fitness_cost: fitness penalties paid by pathogen genotypes fully adapted to the considered resistance genes on host that do not carry the resistance genes,
• tradeoff_strength: strengths of the trade-off relationships between the level of aggressiveness on hosts that do and do not carry the resistance genes.

landscape
a sp object containing the landscape (required only if videoMP4=TRUE).

area
a vector containing polygon areas (must be in square meters).

rotation
a data frame containing for each field (rows) and year (columns, named "year_1", "year_2", etc.), the index of the cultivated croptype. Importantly, the matrix must contain 1 more column than the real number of simulated years.

basic_patho_param
a list of pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance:

• infection_rate = maximal expected infection rate of a propagule on a healthy host,
• propagule_prod_rate = maximal expected effective propagule production rate of an infectious host per time step,
• latent_period_exp = minimal expected duration of the latent period,
• latent_period_var = variance of the latent period duration,
• infectious_period_exp = maximal expected duration of the infectious period,
• infectious_period_var = variance of the infectious period duration,
• survival_prob = probability for a propagule to survive the off-season,
• repro_sex_prob = probability for an infectious host to reproduce via sex rather than via cloning,
sigmoid_kappa = kappa parameter of the sigmoid contamination function,
sigmoid_sigma = sigma parameter of the sigmoid contamination function,
sigmoid_plateau = plateau parameter of the sigmoid contamination function.

disp_patho a vectorized matrix giving the probability of pathogen dispersal from any field of the landscape to any other field.
disp_host a vectorized matrix giving the probability of host dispersal from any field of the landscape to any other field
pI0 initial probability for the first host (whose index is 0) to be infectious (i.e. state I) at the beginning of the simulation. Must be between 0 and 1.
epid_outputs a character string (or a vector of character strings if several outputs are to be computed) specifying the type of epidemiological and economic outputs to generate (see details):
  • "audpc": Area Under Disease Progress Curve (average number of diseased host individuals per time step and square meter)
  • "audpc_rel": Relative Area Under Disease Progress Curve (average proportion of diseased host individuals relative to the total number of existing hosts)
  • "gla": Green Leaf Area (average number of healthy host individuals per time step and square meter)
  • "gla_rel": Relative Green Leaf Area (average proportion of healthy host individuals relative to the total number of existing hosts)
  • "eco_yield": total crop yield (in weight or volume units per ha)
  • "eco_cost": operational crop costs (in monetary units per ha)
  • "eco_product": total crop products (in monetary units per ha)
  • "eco_margin": Margin (products - operational costs, in monetary units per ha)
  • "HLIR_dynamics", "H_dynamics", "L_dynamics", "IR_dynamics", "HLI_dynamics", etc.: Epidemic dynamics related to the specified sanitary status (H, L, I or R and all their combinations). Graphics only, works only if graphic=TRUE.
  • "all": compute all these outputs (default)
  • "": none of these outputs will be generated.

evol_outputs a character string (or a vector of character strings if several outputs are to be computed) specifying the type of evolutionary outputs to generate:
  • "evol_patho": Dynamics of pathogen genotype frequencies
  • "evol_aggr": Evolution of pathogen aggressiveness
  • "durability": Durability of resistance genes
  • "all": compute all these outputs (default)
  • "": none of these outputs will be generated.

thres_breakdown an integer (or vector of integers) giving the threshold (i.e. number of infections) above which a pathogen genotype is unlikely to go extinct, used to characterise the time to invasion of resistant hosts (several values are computed if several thresholds are given in a vector).
the absolute Green Leaf Area in absence of disease (used to compute economic outputs).

the audpc in a fully susceptible landscape (used as reference value for graphics).

a logical indicating if outputs must be written in text files (TRUE, default) or not (FALSE).

a logical indicating if graphics must be generated (TRUE, default) or not (FALSE).

a logical indicating if a video must be generated (TRUE) or not (FALSE, default). Works only if graphic=TRUE and epid_outputs="audpc_rel" (or epid_outputs="all").

a logical indicating if binary files must be kept after the end of the simulation (default=FALSE). Careful, many files may be generated if keepRawResults=TRUE.

See ?landsepi for details on the model and assumptions. Briefly, the model is stochastic, spatially explicit (the basic spatial unit is an individual field), based on a SEIR ("susceptible-exposed-infectious-removed", renamed HLIR for "healthy-latent-infectious-removed" to avoid confusions with "susceptible host") structure with a discrete time step. It simulates the spread and evolution of a pathogen in a heterogeneous cropping landscape, across cropping seasons split by host harvests which impose potential bottlenecks to the pathogen. A wide array of resistance deployment strategies can be simulated and evaluated using several possible outputs to assess the epidemiological, evolutionary and economic performance of deployment strategies (See ?epid_output and ?evol_output for details).

A list containing all outputs that have been required via "epid_outputs" and "evol_outputs". A set of text files, graphics and a video showing epidemic dynamics can be generated. If keepRawResults=TRUE, a set of binary files is generated for every year of simulation and every compartment:

- H: healthy hosts,
- Hjuv: juvenile healthy hosts,
- L: latently infected hosts,
- I: infectious hosts,
- R: removed hosts,
- P: propagules.

Each file indicates for every time-step the number of individuals in each field, and when appropriate for each host and pathogen genotypes.

See Also

model_landsepi, epid_output, evol_output, video, runSimul

Examples

```r
## Not run:
#### Spatially-implicit simulation with 2 patches (S + R) during 3 years ####

## Simulation parameters
time_param <- list(Nyears = 3, nTSpY = 120)
area <- c(100000, 100000)
rotation <- data.frame(year_1 = c(0, 1), year_2 = c(0, 1), year_3 = c(0, 1), year_4 = c(0, 1))
croptype_names <- c("Susceptible crop", "Resistant crop")
croptypes_cultivars_prop <- data.frame(
  croptypeID = c(0, 1),
  cultivarID = c(0, 1),
  proportion = c(1, 1)
)
cultivars <- rbind(
  loadCultivar(name = "Susceptible", type = "growingHost"),
  loadCultivar(name = "Resistant", type = "growingHost")
)
genesis <- loadGene(name = "MG", type = "majorGene")
cultivars_genes_list <- list(numeric(0), 0)
## Run simulation
simul_landsepi(
  seed = 12345, time_param, croptype_names, croptypes_cultivars_prop, cultivars,
  cultivars_genes_list, genes, landscape = NULL, area, rotation,
  basic_patho_param = loadPathogen(disease = "rust"),
  disp_patho = c(0.99, 0.01, 0.01, 0.99), disp_host = c(1, 0, 0, 1), pI0 = 5e-4
)

#### Spatially-explicit simulation with built-in landscape during 10 years ####

# Generate a mosaic of four croptypes in balanced proportions
# and medium level of spatial aggregation

## Simulation and Landscape parameters
Nyears <- 10
landscape <- loadLandscape(1)
Npoly <- length(landscape)
library(sf)
area <- st_area(st_as_sf(landscape))
rotation <- AgriLand(landscape, Nyears,
  rotation_period = 1, rotation_realloc = FALSE,
  rotation_sequence = c(0, 1, 2, 3),
  prop = rep(1 / 4, 4), aggreg = 0.5, graphic = TRUE, outputDir = getwd())
rotation <- data.frame(rotation)[, 1:(Nyears + 1)]
croptype_names <- c("Susceptible crop", "Resistant crop")
```
croptypes_cultivars_prop <- data.frame(croptypeID = c(0, 1, 2, 3), cultivarID = c(0, 1, 2, 3), proportion = c(1, 1, 1, 1))
cultivars <- data.frame(rbind(
  loadCultivar(name = "Susceptible", type = "growingHost"),
  loadCultivar(name = "Resistant1", type = "growingHost"),
  loadCultivar(name = "Resistant2", type = "growingHost"),
  loadCultivar(name = "Resistant3", type = "growingHost")
), stringsAsFactors = FALSE)
genesis <- data.frame(rbind(
  loadGene(name = "MG 1", type = "majorGene"),
  loadGene(name = "MG 2", type = "majorGene"),
  loadGene(name = "MG 3", type = "majorGene")
), stringsAsFactors = FALSE)
cultivars_genesis_list <- list(numeric(0), 0, 1, 2)

## Run simulation
simul_landsepi(
  seed = 12345, time_param = list(Nyears = Nyears, nTSpY = 120),
  croptype_names, croptypes_cultivars_prop, cultivars,
  cultivars_genesis_list, genes, landscape, area, rotation,
  basic_patho_param = loadPathogen(disease = "rust"),
  disp_patho = loadDispersalPathogen(1),
  disp_host = as.numeric(diag(Npoly)),
  pI0 = 5e-4
)

## End(Not run)

---

**Summary**

Prints the summary of a LandsepiParams object.

**Usage**

```r
## S4 method for signature 'LandsepiParams'
summary(object)
```

**Arguments**

- `object` a LandsepiParams object.
### switch_patho_to_aggr

**Switch from index of genotype to indices of aggressiveness on different components**

**Description**

Finds the level of aggressiveness on different components (targeted by different resistance genes) from the index of a given pathogen genotype.

**Usage**

```
switch_patho_to_aggr(index_patho, Ngenes, Nlevels_aggressiveness)
```

**Arguments**

- `index_patho`: index of pathogen genotype
- `Ngenes`: number of resistance genes
- `Nlevels_aggressiveness`: vector of the number of adaptation levels related to each resistance gene

**Value**

A vector containing the indices of aggressiveness on the different components targeted by the resistance genes.

**Examples**

```
switch_patho_to_aggr(5, 3, c(2, 2, 3))
```

---

### video

**Generation of a video**

**Description**

Generates a video showing the epidemic dynamics on a map representing the cropping landscape. (requires ffmpeg library).

**Usage**

```
video(
  audpc,
  time_param,
  Npatho,
  landscape,
  area,
)```
rotation,  
croptypes,  
croptype_names = c(),  
cultivars_param,  
keyDates = NULL,  
nMapPY = 5,  
path = getwd()  
)

Arguments

audpc  
A dataframe containing audpc outputs (generated through epid_output). 1 year per line and 1 column per cultivar, with an additional column for the average audpc in the landscape.

time_param  
list of simulation parameters:
• Nyears = number cropping seasons,
• nTSpY = number of time-steps per cropping season.

Npatho  
number of pathogen genotypes.

landscape  
a sp object containing the landscape.

area  
a vector containing polygon areas (must be in square meters).

rotation  
a dataframe containing for each field (rows) and year (columns, named "year_1", "year_2", etc.), the index of the cultivated croptype. Importantly, the matrix must contain 1 more column than the real number of simulated years.

croptypes  
a dataframe with three columns named ‘croptypeID’ for croptype index, ‘cultivarID’ for cultivar index and ‘proportion’ for the proportion of the cultivar within the croptype.

croptype_names  
a vector of croptype names (for legend).

cultivars_param  
a list of parameters associated with each host genotype (i.e. cultivars) when cultivated in pure crops:
• name = vector of cultivar names,
• max_density = vector of maximum host densities (per square meter) at the end of the cropping season as if cultivated in pure crops,
• cultivars_genes_list = a list containing, for each host genotype, the indices of carried resistance genes.

keyDates  
a vector of times (in time steps) where to draw vertical lines in the AUDPC graphic. Usually used to delimit durabilities of the resistance genes. No line is drawn if keyDates=NULL (default).

nMapPY  
an integer specifying the number of epidemic maps per year to generate.

path  
path where binary files are located and where the video will be generated.

Details

The left panel shows the year-after-year dynamics of AUDPC, for each cultivar as well as the global average. The right panel illustrates the landscape, where fields are hatched depending on the cultivated croptype, and coloured depending on the prevalence of the disease. Note that up to 9 different croptypes can be represented properly in the right panel.
Value

A video file of format mp4.

Examples

```r
## Not run:
demo_landsepi()

## End(Not run)
```
Index

- SEIR
  landsepi-package, 4
- datasets
  dispP, 26
  landscapeTEST, 42
- demo-genetic
  landsepi-package, 4
- deployment
  landsepi-package, 4
- durability
  landsepi-package, 4
- model
  landsepi-package, 4
- resistance
  landsepi-package, 4
- spatial
  landsepi-package, 4
- stochastic
  landsepi-package, 4
 Phú PACKAGE (landsepi-package), 4

AgriLand, 9
allocateCroptypeCultivars, 11, 43, 44, 68
allocateCultivarGenes, 13, 43
allocateLandscapeCroptypes, 11, 14, 75
checkCroptypes, 16
checkCultivars, 17
checkCultivarGenes, 17
checkDispersalHost, 18
checkDispersalPathogen, 18
checkGenes, 19
checkInoculum, 19
checkLandscape, 20
checkOutputs, 20
checkPathogen, 21
checkSimulParams, 21
checkTime, 22
createLandscapeGPKG, 22
createSimulParams, 23, 42, 65

CroptypeBDD2Params, 24
CultivarBDD2Params, 24
CultivarGeneBDD2Params, 25
demo_landsepi, 25
dispP, 26, 46
dispP_1(dispP), 26
dispP_2(dispP), 26
dispP_3(dispP), 26
dispP_4(dispP), 26
dispP_5(dispP), 26
epid_output, 27, 31, 86
evol_output, 30, 30, 86
GeneBDD2Params, 32
getGPKGArea, 32
getGPKGCroptypes, 33
getGPKGCroptypesRaw, 33
getGPKGCultivars, 34
getGPKGCultivarGenes, 34
getGPKGGeneIDForCultivar, 35
getGPKGGenes, 35
getGPKGRotation, 36
GPKGAddInputData, 36
GPKGAddTables, 37
initialize, LandsepiParams-method, 37
invlogit, 39
is.in.01, 39
is.positive, 40
is.strict.positive, 40
is.wholenumber, 41
landscapeTEST, 42, 48
landscapeTEST1 (landscapeTEST), 42
landscapeTEST2 (landscapeTEST), 42
landscapeTEST3 (landscapeTEST), 42
landscapeTEST4 (landscapeTEST), 42
landscapeTEST5 (landscapeTEST), 42
landsepi (landsepi-package), 4
landsepi-package, 4
LandsepiParams, 42
LandsepiParams-class (LandsepiParams), 42
loadCroptypes, 12, 43, 43, 43, 43, 68, 69
loadCultivar, 43, 44, 70
loadDispersalHost, 43, 45, 71
loadDispersalPathogen, 43, 46, 72
loadGene, 43, 47, 73, 74
loadLandscape, 43, 46, 48, 75
loadOutputs, 48, 76, 77
loadPathogen, 43, 50, 78
loadSimulParams, 50
logit, 51

model_landsepi, 52, 86
multiN, 11, 56

params2CroptypeBDD, 57
params2CultivarBDD, 58
params2GeneBDD, 58
params2GeneListBDD, 59
periodic_cov, 11, 59
plot_allocation, 61
plot_freqPatho, 62
plotland, 60, 61
print, 63
print, LandsepiParams-method (print), 63

resetCultivarsGenes, 63
runShinyApp, 26, 64
runSimul, 26, 64, 86

saveDeploymentStrategy, 51, 67
setCroptypes, 12, 43, 44, 68
setCultivars, 12, 13, 43, 43, 45, 69
setDispersalHost, 43, 46, 71
setDispersalPathogen, 43, 46, 72
setGenes, 13, 43, 47, 73
setInoculum, 43, 74
setLandscape, 48
setLandscape (setLansdcape), 75
setLansdcape, 75
setOutputs, 43, 49, 76
setPathogen, 43, 50, 78
setSeed, 79
setSeedValue, 80
setTime, 43, 80
show, 81

show, LandsepiParams-method (show), 81
simul_landsepi, 81
summary, 87
summary, LandsepiParams-method (summary), 87
switch_patho_to_aggr, 88
video, 86, 88