# Package ‘landsepi’

March 12, 2024

**Type** Package  
**Encoding** UTF-8  
**Title** Landscape Epidemiology and Evolution  
**Version** 1.4.0  
**Date** 2024-03-12  
**Maintainer** Jean-François Rey &lt;jean-francois.rey@inrae.fr&gt;  
**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. 

**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++, gsl  
**Depends** R (>= 4.2.0), sp (>= 1.0-17)  
**Imports** methods, utils, stats (>= 3.0.2), grDevices (>= 3.0.0), graphics (>= 3.0.0), parallel, Rcpp (>= 0.9.0), Matrix (>= 1.3-0), mvtnorm, fields, splancs, sf, DBI, RSQLite, foreach, doParallel, deSolve
Collate 'Math-Functions.R' 'RcppExports.R' 'graphics.R' 'AgriLand.R'
   'Class-LandsepiParams.R' 'Cultivars_List.R' 'GPKGTools.R'
   'tools.R' 'Methods-LandsepiParams.R' 'demo_landsepi.R'
   'landsepi.R' 'output.R' 'runShiny.R' 'simul_landsepi.R'

LinkingTo Rcpp, testthat

RoxygenNote 7.3.1

Suggests testthat (>= 3.0.0), shiny, shinyjs, DT, knitr, rmarkdown

VignetteBuilder knitr

Author Loup Rimbaud [aut] (<https://orcid.org/0000-0002-8098-9984>),
   Julien Papaix [aut],
   Jean-François Rey [aut, cre] (<https://orcid.org/0000-0003-3281-6701>),
   Marta Zaffaroni [ctb] (<https://orcid.org/0000-0002-2951-8626>),
   Jean-Loup Gaussen [ctb],
   Manon Couty [ctb]

Repository CRAN

Date/Publication 2024-03-12 16:50:02 UTC

R topics documented:

landsepi-package .............................................. 4
AgriLand ......................................................... 9
allocateCroptypeCultivars ..................................... 12
allocateCultivarGenes .......................................... 13
allocateLandscapeCroptypes .................................... 14
antideriv_verhulst ............................................. 17
checkCroptypes ................................................... 17
checkCultivars ................................................. 18
checkCultivarsGenes ........................................... 18
checkDispersalHost ............................................ 19
checkDispersalPathogen ......................................... 19
checkGenes ....................................................... 20
checkInoculum .................................................. 20
checkLandscape .................................................. 21
checkOutputs .................................................... 21
checkPathogen .................................................... 22
checkP0_mat ...................................................... 22
checkSimulParams ............................................... 23
checkTime ......................................................... 23
checkTreatment .................................................. 24
compute_audpc100S ............................................. 24
createSimulParams .............................................. 25
Cultivars_list .................................................. 26
demo_landsepi .................................................. 27
dispP ............................................................ 28
epid_output ....................................................... 29
evol_output ....................................................... 33
R topics documented:

- getMatrixCroptypePatho ........................................ 35
- getMatrixCultivarPatho ........................................ 36
- getMatrixGenePatho ........................................... 37
- getMatrixPolyPatho ............................................ 38
- initialize.LandsepiParams-method .............................. 39
- inoculumToMatrix .............................................. 41
- invlogit ........................................................ 42
- is.in.01 ......................................................... 43
- is.positive ..................................................... 44
- is.strict.positive ............................................. 44
- is.wholenumber ................................................ 45
- landscapeTEST ................................................ 45
- LandsepiParams ............................................... 46
- loadCroptypes ................................................. 47
- loadCultivar ................................................... 48
- loadDispersalHost ............................................. 49
- loadDispersalPathogen ....................................... 50
- loadGene ....................................................... 51
- loadInoculum .................................................. 52
- loadLandscape ................................................ 54
- loadOutputs .................................................... 55
- loadPathogen ................................................... 56
- loadSimulParams .............................................. 57
- loadTreatment ................................................ 58
- logit .......................................................... 59
- model_landsepi ............................................... 60
- multiN .......................................................... 65
- periodic_cov .................................................. 67
- plotland ......................................................... 67
- plot_allocation ............................................... 69
- plot_freqPatho ............................................... 70
- price_reduction ............................................... 71
- print .......................................................... 72
- resetCultivarsGenes ......................................... 72
- runShinyApp ................................................. 73
- runSimul ........................................................ 73
- saveDeploymentStrategy ..................................... 76
- setCroptypes .................................................. 78
- setCultivars ................................................... 79
- setDispersalHost ............................................. 81
- setDispersalPathogen ....................................... 82
- setGenes ....................................................... 83
- setInoculum .................................................. 84
- setLandscape .................................................. 85
- setOutputs ..................................................... 86
- setPathogen ................................................... 88
- setSeed ......................................................... 89
- setSeedValue .................................................. 90
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

<table>
<thead>
<tr>
<th>Package:</th>
<th>landsepi</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type:</td>
<td>Package</td>
</tr>
<tr>
<td>Version:</td>
<td>1.4.0</td>
</tr>
<tr>
<td>Date:</td>
<td>2024-02-12</td>
</tr>
<tr>
<td>License:</td>
<td>GPL (&gt;=2)</td>
</tr>
</tbody>
</table>

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 also helps investigate the effect of landscape organisation, the considered pathosystem and the epidemi-evolutionary context on the performance of a given strategy.

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 (via mutation, recombination through sexual reproduction, selection and drift) 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 (allocateCroptypeCultivars()) but it is possible to use your own cultivar allocation if it is included in the shapefile containing the landscape. Additionally, any cultivar may be treated with contact pesticides, which reduce the pathogen infection rate with an efficiency gradually decreasing with time and host growth.

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 aggressiveness on a resistant host (i.e. adaptation to the corresponding resistance genes) can be penalised by a fitness cost, either on all hosts, or only on susceptible hosts (in the latter case, pathogen genotypes adapted to a resistance gene have a reduced aggressiveness on hosts that do not carry this gene, and a ‘relative advantage’ on host that do carry such 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.

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). A parameterisation to downy mildew of grapevine (Plasmopara viticola) and black sigatoka of banana (Pseudocercospora fijiensis) are also available. 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 (exposed and 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 individuals (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 (i.e. sown/planted and harvested), 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. Cultivars may be treated with chemicals which reduce the pathogen infection rate (contact treatment). Treatment efficiency decreases with host growth (i.e. new biomass is not protected by treatments) and time (i.e. pesticide degradation). Cultivars to be treated and dates of chemical applications are fixed prior to simulations but only polygons where disease severity exceeds a given threshold (possibly 0) are treated.

9. Components of a mixture are independent each other (i.e. there is neither plant-plant interaction nor competition for space, and harvests are segregated). If one component is treated with a chemical, it does not affect other components.

10. The pathogen is haploid.

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

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

13. **Boundaries of the landscape are reflective: propagules stay in the system as if it was closed.**

14. Pathogen reproduction can be purely clonal, purely sexual, or mixed (alternation of clonal and sexual reproduction).

15. If there is sexual reproduction (or gene recombination), it occurs only between parental infections located in the same polygon and the same host genotype. At that scale, the pathogen population is panmictic (i.e. all pairs of parents have the same probability to occur). The propagule production rate of a parental pair is the sum of the propagule production rates of the parents. For a given parental pair, the genotype of each propagule is issued from random loci segregation of parental qualitative resistance genes. For each quantitative resistance gene, the value of each propagule trait is issued from a normal distribution around the average of the parental traits, following the infinitesimal model (Fisher 1919).

16. All types of propagules (i.e. clonal and sexual) share the same pathogenicity parameters (e.g. infection rate, latent period duration, etc.) but each of them has their own dispersal and survival abilities (see after).

17. At the end of each cropping season, pathogens experience a bottleneck representing the off-season and then propagules are produced (either via clonal or sexual reproduction). The **probability of survival is the same every year and in every polygon.** Clonal propagules are released during the following season only, either altogether at the first day of the season, or progressively (in that case the day of release of each propagule is sampled from a uniform distribution). Sexual propagules are gradually released during several of the following seasons (between-season release). The season of release of each propagule is sampled from an exponential distribution, truncated by a maximum viability
limit. Then, the day of release in a given season is sampled from a uniform distribution (within-season release).

18. Pathogenicity genes mutate independently from each other.

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

20. If a fitness cost penalises pathogen adaptation to a given resistance gene, this cost is paid on hosts that do not carry this gene, or, alternatively, on all hosts with possibly a relative advantage on hosts carrying the resistance gene. It consists in a reduction in the same aggressiveness component as the one targeted by the resistance gene.

21. When there is a delay for activation of a given resistance gene (APR), the age of activation is the same for all hosts carrying this gene and located in the same polygon.

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

**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).

5. the yearly contribution of pathogen genotypes to LIR dynamics on every host as well as the whole landscape.

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 (depends on initial host density and planting cost) and pesticide treatments (depends on the number of applications and the cost of a single application) in monetary units per hectare.
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. Cucumber mosaic virus on pepper, potato virus Y on pepper).
- An updated version of the shiny interface.

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

- g++
- libgsl2
- libgsl-dev

and the following R packages:

- Rcpp
- sp
- stats
- Matrix
- mvtnorm
- fields
- splancs
- sf
- DBI
- RSQLite
- foreach
- parallel
- doParallel
- deSolve

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>
Marta Zaffaroni <marta.zaffaroni@inrae.fr>
Jean-Loup Gaussen <jean-loup-thomas.gaussen@inrae.fr>
Manon Couty <manon.couty@insa-lyon.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.biosp.inrae.fr/landsepi/](https://csiro-inra.pages.biosp.inrae.fr/landsepi/)
- [https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi](https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi)
- Report bugs at [https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi/-/issues](https://gitlab.paca.inrae.fr/CSIRO-INRA/landsepi/-/issues)

Examples

```r
## Not run:
library("landsepi")

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

```r
AgriLand(
  landscape,
  Nyears,
  rotation_period = 0,
  rotation_sequence = list(c(0, 1, 2)),
  rotation_realloc = FALSE,
)```
prop = list(c(1/3, 1/3, 1/3)),
aggreg = list(1),
algo = "periodic",
croptype_names = c(),
graphic = FALSE,
outputDir = "./"
)

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 be 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


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
```

allocateCroptypeCultivars

Allocate cultivars to one croptype

Description
Updates a given croptype by allocating cultivars composing it.

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.
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(""),
  force.clean = FALSE
)

Arguments

  params a LandsepiParams object.
  cultivarName the name of the cultivar to be allocated.
  listGenesNames the names of the genes the cultivar carries
  force.clean force to clean previous allocated genes to all cultivars

Value

a LandsepiParams object
allocateLandscapeCroptypes

Allocate croptypes to the landscape

Description

Updates the landscape of a LandsepiParams object with croptype allocation in every polygon of the landscape and every year of simulation. Allocation is based on an algorithm which controls croptype proportions (in surface) and spatio-temporal aggregation. Note that time, landscape and croptype parameters must be set before allocating landscape croptypes.

Usage

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

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.
- **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.
- **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, 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 polygons. Next, the croptypes are allocated to different polygons 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
allocateLandscapeCroptypes

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

Value

a LandsepiParams object with Landscape updated with the layer "croptypeID". It contains croptype allocation in every polygon 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)
```
Antiderivative of the Verhulst logistic function

Description
Give the antiderivative of the logistic function from the Verhulst model.

Usage
antideriv_verhulst(x, initial_density, max_density, growth_rate)

Arguments
- x: timestep up to which antiderivative must be computed
- initial_density: initial density
- max_density: maximal density
- growth_rate: growth rate

Details
The Verhulst model (used to simulate host growth) is defined by \( f(x) = \frac{\text{max\_density}}{1 + (\text{max\_density}/\text{initial\_density}) \times e^{-\text{growth\_rate}x}} \). See https://en.wikipedia.org/wiki/Logistic_function for details.

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

Examples
antideriv_verhulst(119, 0.1, 2, 0.1) / 120

Check croptypes

Description
checks croptypes validity

Usage
checkCroptypes(params)
checkCultivarsGenes

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       Check genes

Description
checks Genes data.frame validity

Usage
checkGenes(params)

Arguments
params        a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise

checkInoculum       Check inoculum

Description
Checks inoculum validity.

Usage
checkInoculum(params)

Arguments
params        a LandsepiParams object.

Value
a boolean, TRUE if OK, FALSE otherwise
checkLandscape  

**Description**
Checks landscape validity

**Usage**
checkLandscape(params)

**Arguments**
params  a LandsepiParams Object.

**Value**
TRUE if Ok, FALSE otherwise

checkOutputs  

**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

checkPI0_mat

Description
Checks validity of the array.

Usage
checkPI0_mat(mat, params)

Arguments
mat a 3D array of dimensions (1:Nhost,1:Npatho,1:Npoly)
params a LandsepiParams object.

Value
the same array at mat, possibly corrected if incompatibility has been detected
checkSimulParams  Check simulation parameters

Description

Checks validity of a LandsepiParams object.

Usage

checkSimulParams(params)

Arguments

params  a LandsepiParams Object.

Value

TRUE if OK for simulation, FALSE otherwise

checkTime  Check time

Description

Checks time parameters validity

Usage

checkTime(params)

Arguments

params  a LandsepiParams Object.

Value

a boolean TRUE if times are setted.
checkTreatment  \hspace{1cm} Check treatment

Description
Checks treatment validity

Usage
checkTreatment(params)

Arguments
params \hspace{1cm} a LandsepiParams Object.

Value
a boolean, TRUE if OK, FALSE otherwise

compute_audpc100S  \hspace{1cm} Compute AUDPC in a single 100% susceptible field

Description
Compute AUDPC in a single field cultivated with a susceptible cultivar.

Usage
compute_audpc100S(
  disease = "rust",
  hostType = "growingHost",
  nTSpY = 120,
  area = 1e+06,
  seed = 12345
)

Arguments
\begin{itemize}
  \item disease \hspace{1cm} a disease name, among "rust" (default), "mildew" and "sigatoka"
  \item hostType \hspace{1cm} cultivar type, among: "growingHost" (default), "nongrowingHost", "grapevine".
  \item nTSpY \hspace{1cm} number to time steps per cropping season
  \item area \hspace{1cm} area of the field (must be in square meters).
  \item seed \hspace{1cm} an integer used as seed value (for random number generator).
\end{itemize}
createSimulParams

Details

audpc100S is the average AUDPC computed in a non-spatial simulation.

Value

The AUDPC value (numeric)

See Also

loadOutputs

Examples

## Not run:
compute_audpc100S("rust", "growingHost", area=1E6)
compute_audpc100S("mildew", "grapevine", area=1E6)
compute_audpc100S("sigatoka", "banana", area=1E6, nTspY=182)

## End(Not run)

createSimulParams

Create a LandsepiParams object.

Description

Creates a default object of class LandsepiParams.

Usage

createSimulParams(outputDir = "./"

Arguments

outputDir output 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-polygon dispersal (neither pathogen nor host)
- no pathogen introduction
• no resistance gene
• no chemical treatment
• no output to generate.

Examples

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

## End(Not run)
```

<table>
<thead>
<tr>
<th>Cultivars list</th>
<th>Cultivars Type list</th>
</tr>
</thead>
</table>

Description

A set of configurated cultivars types

Usage

Cultivars_list

Format

A list of list indexed by type name

- 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,
- 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).
Description

run a simulation demonstration with landsepi

Usage

demo_landsepi(
  seed = 5,
  strat = "MO",
  Nyears = 10,
  nTSpY = 120,
  videoMP4 = FALSE
)

Arguments

seed an integer used as seed for Random Number Generator.
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 on rust fungi of cereal crops, 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 10-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: 

\[
f(x) = \left(\frac{(b-2) \times (b-1)}{2 \times \pi \times a^2}\right) \times (1 + x/a)^{-b}
\]

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 \(2a/(b-3)=20\) m.

## References

**Examples**

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

**Description**

Generates epidemiological and economic outputs from model simulations.

**Usage**

```r
epid_output(
  types = "all",
  time_param,
  Npatho,
  area,
  rotation,
  croptypes,
  cultivars_param,
  eco_param,
  treatment_param,
  pathogen_param,
  audpc100S = 0.76,
  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)
  - "contrib": contribution of pathogen genotypes to LIR dynamics
epid_output

- "HILR_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, ‘cul-
  tivarID’ 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).

treatment_param
  list of parameters related to pesticide treatments:
  - treatment_degradation_rate = degradation rate (per time step) of chemical
    concentration,
  - treatment_efficiency = maximal efficiency of chemical treatments (i.e. frac-
    tional reduction of pathogen infection rate at the time of application),
  - treatment_timesteps = vector of time-steps corresponding to treatment ap-
    plication dates,
  - treatment_cultivars = vector of indices of the cultivars that receive treat-
    ments,
• treatment_cost = cost of a single treatment application (monetary units/ha)
• treatment_application_threshold = vector of thresholds (i.e. disease severity, one for each treated cultivar) above which the treatment is applied in a polygon

pathogen_param = a list of i. pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance and ii. sexual reproduction parameters:
• 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_mean = minimal expected duration of the latent period,
• latent_period_var = variance of the latent period duration,
• infectious_period_mean = 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,
• sex_propagule_viability_limit = maximum number of cropping seasons up to which a sexual propagule is viable
• sex_propagule_release_mean = average number of seasons after which a sexual propagule is released,
• clonal_propagule_gradual_release = whether or not clonal propagules surviving the bottleneck are gradually released along the following cropping season.

audpc100S = the audpc in a fully susceptible landscape (used as reference value for graphics).
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).

5. Contribution of pathogen genotypes: for every year and every host (as well as for the whole landscape and the whole simulation duration), fraction of cumulative LIR infections attributed to each pathogen genotype.

**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). Note that when disease = "mildew" a price reduction between 0% and 5% is applied to the market value depending on disease severity.

3. Operational crop costs: yearly costs associated with crop planting (depends on initial host density and planting cost) and pesticide treatments (depends on the number of applications and the cost of a single application) in monetary units per hectare.

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**


**See Also**

evol_output

**Examples**

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

## End(Not run)
```
evol_output

---

**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": Evolution of pathogen genotypes
    - "evol_aggr": Evolution of pathogen aggressiveness (i.e. phenotype)
    - "durability": Durability of resistance genes
    - "all": compute all these outputs (default)

- **time_param**
  - list of simulation parameters:
    - `Nyears` = number of 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:
    - `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 (evol_patho) or phenotype (evol_aggr, note that different pathogen genotypes may lead to the same phenotype on a resistant host), several computations are performed based on pathogen genotype frequencies:

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

The value Nyears + 1 time step is used if the genotype or phenotype never appeared/infected/invaded. Durability is defined as the time to invasion of completely adapted pathogen individuals.

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
evol_output

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

## End(Not run)```
getMatrixCroptypePatho

Get the "croptype/pathogen genotype" compatibility matrix.

Description
Build the matrix indicating if infection is possible at the beginning of the season for every combination of croptype (rows) and pathogen genotype (columns).

Usage
getMatrixCroptypePatho(params)

Arguments
params a LandsepiParams object.

Details
For each croptype, there is either possibility of infection by the pathogen genotype (value of 1), either complete protection (value of 0)

Value
an interaction matrix composed of 0 and 1 values.

See Also
getMatrixGenePatho, getMatrixCultivarPatho, getMatrixPolyPatho

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 = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivar4 <- loadCultivar(name = "Pyramid", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3, cultivar4)
  , stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
simul_params <- allocateCultivarGenes(simul_params, "Resistant1", c("MG 1"))
simul_params <- allocateCultivarGenes(simul_params, "Resistant2", c("MG 2"))
simul_params <- allocateCultivarGenes(simul_params, "Pyramid", c("MG 1", "MG 2"))
croptypes <- loadCroptypes(simul_params,
getMatrixCultivarPatho

Get the "cultivar/pathogen genotype" compatibility matrix.

Description

Build the matrix indicating if infection is possible at the beginning of the season for every combination of cultivar (rows) and pathogen genotype (columns).

Usage

getMatrixCultivarPatho(params)

Arguments

params a LandsepiParams object.

Details

For each cultivar, there is either possibility of infection by the pathogen genotype (value of 1), or complete protection (value of 0).

Value

an interaction matrix composed of 0 and 1 values.

See Also

getMatrixGenePatho, getMatrixCroptypePatho, getMatrixPolyPatho
getMatrixGenePatho

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)
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "monoResistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "monoResistant2", type = "growingHost")
cultivar4 <- loadCultivar(name = "Pyramid", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3, cultivar4)
, stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
simul_params <- allocateCultivarGenes(simul_params, "monoResistant1", c("MG 1"))
simul_params <- allocateCultivarGenes(simul_params, "monoResistant2", c("MG 2"))
simul_params <- allocateCultivarGenes(simul_params, "Pyramid", c("MG 1", "MG 2"))
getMatrixCultivarPatho(simul_params)
## End(Not run)
```

getMatrixGenePatho

Get the "resistance gene/pathogen genotype" compatibility matrix.

Description

Build the matrix indicating if infection is possible at the beginning of the season for every combination of plant resistance gene (rows) and pathogen genotype (columns).

Usage

```r
getMatrixGenePatho(params)
```

Arguments

- `params` a `LandsepiParams` object.

Details

For hosts carrying each resistance gene, there is either possibility of infection by the pathogen genotype (value of 1), either complete protection (value of 0). Complete protection only occurs if the resistance gene targets the infection rate, has a complete efficiency, and is expressed from the beginning of the cropping season (i.e. this is not an APR).

Value

an interaction matrix composed of 0 and 1 values.
getMatrixPolyPatho

Get the "polygon/pathogen genotype" compatibility matrix.

### Description

Build the matrix indicating if infection is possible at the beginning of the season for every combination of polygon (rows) and pathogen genotype (columns).

### Usage

```r
getMatrixPolyPatho(params)
```

### Arguments

- **params**: a LandsepiParams object.

### Details

For each polygon, there is either possibility of infection by the pathogen genotype (value of 1), either complete protection (value of 0)

### Value

An interaction matrix composed of 0 and 1 values.

### See Also

- `getMatrixGenePatho`
- `getMatrixCultivarPatho`
- `getMatrixCroptypePatho`
initialize, LandsepiParams-method

Description

Creates and initialises a LandsepiParams object with default parameters.

Usage

## S4 method for signature 'LandsepiParams'
initialize(}

Examples

```R
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears = 1, nTSpY = 80)
simul_params <- setLandscape(simul_params, loadLandscape(id = 1))
gene1 <- loadGene(name = "MG 1", type = "majorGene")
gene2 <- loadGene(name = "MG 2", type = "majorGene")
gen <- data.frame(rbind(gene1, gene2), stringsAsFactors = FALSE)
simul_params <- setGenes(simul_params, gene)
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant1", type = "growingHost")
cultivar3 <- loadCultivar(name = "Resistant2", type = "growingHost")
cultivar4 <- loadCultivar(name = "Pyramid", type = "growingHost")
cultivars <- data.frame(rbind(cultivar1, cultivar2, cultivar3, cultivar4)
, stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)
simul_params <- allocateCultivarGenes(simul_params, "Resistant1", c("MG 1"))
simul_params <- allocateCultivarGenes(simul_params, "Resistant2", c("MG 2"))
simul_params <- allocateCultivarGenes(simul_params, "Pyramid", c("MG 1", "MG 2"))
croptypes <- loadCroptypes(simul_params,
, names = c("Susceptible crop",
, "Resistant crop 1",
, "Mixture S+R",
, "Mixture R1+R2",
, "Pyramid crop"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop 1", "Resistant1")
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture S+R", c("Susceptible", "Resistant1"))
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture R1+R2", c("Resistant1", "Resistant2"))
croptypes <- allocateCroptypeCultivars(croptypes, "Pyramid crop", c("Pyramid"))
simul_params <- setCroptypes(simul_params, croptypes)
simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = 0,
prop=rep(1/5, 5), aggreg=3, rotation_sequence = croptypes$croptypeID)
getMatrixPolyPatho(simul_params)
## End(Not run)
```
initialize.LandsepiParams-method

> .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_mean = 0,
> latent_period_var = 0, infectious_period_mean = 0, infectious_period_var = 0,
> sigmoid_kappa = 0, sigmoid_sigma = 0, sigmoid_plateau = 1,
> sex_propagule_viability_limit = 0, sex_propagule_release_mean = 0,
> clonal_propagule_gradual_release = 0),
> PI0 = 0,
> DispHost = vector(),
> DispPathoClonal = vector(),
> DispPathoSex = vector(),
> Treatment = list(treatment_degradation_rate = 0.1, treatment_efficiency = 0,
> treatment_timesteps = vector(), treatment_cultivars = vector(), treatment_cost = 0,
> treatment_application_threshold = vector()),
> OutputDir = normalizePath(character(getwd())),
> OutputGPKG = "landsepi_landscape.gpkg",
> Outputs = list(epid_outputs = "", evol_outputs = "", thres_breakdown = NA, audpc100S =
> NA),
> TimeParam = list(Nyears = 0, nTSpY = 0),
> Seed = NULL,
> ...
>
> Arguments

> .Object a LandsepiParam 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 dataframe 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 vector of length NpolyNhostNpatho filled with the initial probabilities for hosts
to be infectious (i.e. state I), for each pathogen genotype, at the beginning of the simulation.
DispHost a vectorized matrix giving the probability of host dispersal from any polygon of the landscape to any other polygon.

DispPathoClonal a vectorized matrix giving the probability of pathogen dispersal from any polygon of the landscape to any other polygon.

DispPathoSex a vectorized matrix giving the probability of pathogen dispersal from any polygon of the landscape to any other polygon (sexual propagule).

Treatment a list of chemical treatment parameters (indices of treated cultivars, times of application, efficiency and degradation rate).

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

---

inoculumToMatrix  
**Inoculum To Matrix**

**Description**

Transform the inoculum pI0 (1D vector of length NhostNpathoNpoly) into a 3D array (for visualization purpose).

**Usage**

inoculumToMatrix(params)

**Arguments**

params a LandsepiParams object.

**Details**

After defining the inoculum with setInoculum(), this function returns the inoculum as a 3D array.

**Value**

a 3D array of structure (1:Nhost,1:Npatho,1:Npoly)

**See Also**

setInoculum
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears = 1, nTSpY = 80)
simul_params <- setPathogen(simul_params, loadPathogen(disease = "rust"))
simul_params <- setLandscape(simul_params, loadLandscape(id = 1))
simul_params <- setDispersalPathogen(simul_params, loadDispersalPathogen(id = 1)[[1]])
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"))
croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop", "Resistant crop"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop", c("Resistant"))
simul_params <- setCroptypes(simul_params, croptypes)
simul_params@Croptypes

simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = 0
, rotation_sequence = croptypes$croptypeID
, prop = c(1/2,1/2), aggreg = 1, graphic = FALSE)
pI0 <- loadInoculum(simul_params, pI0_patho=c(1E-3,1E-4,1E-4,1E-5), pI0_host=c(1,1))
simul_params <- setInoculum(simul_params, pI0)
inoculumToMatrix(simul_params)[,1:5]

## End(Not run)
```

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` function is defined by $\exp(x)/(1 + \exp(x))$. Values in `x` of `-Inf` or `Inf` 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

invlogit(10)

description

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

Usage

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

Examples

is.in.01(-5)
is.in.01(0)
is.in.01(1)
is.in.01(0, exclude0 = TRUE)
is.in.01(2.5)
is.in.01(matrix(5:13/10, nrow=3))
is.positive

Description
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

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
is.wholenumber

Examples

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

is.wholenumber

Description
Tests if a number or vector is a whole number

Usage

is.wholenumber(x, tol = .Machine$double.eps^0.5)

Arguments

x a number or vector or matrix
tol double tolerance

Value

a logical of the same format as x

Examples

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

landscapeTEST

Landscapes

Description

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

Usage

landscapeTEST1
landscapeTEST2
landscapeTEST3
landscapeTEST4
landscapeTEST5
Format

Landspős 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

Landspős simulation parameters

Details

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

Slots

Landscape a landscape as sf object. See 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 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 data.frame of parameters associated with each resistance gene and with the evolution of each corresponding pathogenicity gene. See loadGene and setGenes
Pathogen a list of the pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance and ii. sexual reproduction parameters. See loadPathogen and setPathogen
ReproSexProb a vector of size TimeParam$nTSpY + 1 (end of season) of the probabilities for an infectious host to reproduce via sex rather than via cloning at each step.

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

DispPathoClonal a vectorized matrix giving the probability of pathogen dispersal (clonal propagules) from any field of the landscape to any other field. See loadDispersalPathogen and setDispersalPathogen

DispPathoSex a vectorized matrix giving the probability of pathogen dispersal (sexual propagules) from any field of the landscape to any other field. See loadDispersalPathogen and setDispersalPathogen

Treatment a list of parameters to simulate the effect of chemical treatments on the pathogen, see loadTreatment and setTreatment

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 loadOutputs and 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 start at 0 and match with croptype IDs in the landscape)
- **names** a vector containing the names of all croptypes

**Details**

Croptypes need to be later updated with allocateCroptypeCultivars. If neither croptypeIDs nor names are given, it will automatically generate 1 croptype per cultivar.
loadCultivar

Load a cultivar

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, among: "growingHost" (default), "nongrowingHost", "grapevine", "banana" or "nonCrop".

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 (as typified by cereal crops).
- "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.
- "grapevine" corresponds to parameters for grapevine (including host growth).
- "banana" corresponds to parameters for banana (including host growth).
- "nonCrop" is not planted, does not cost anything and does not yield anything (e.g. forest, fallow).

Value
a data.frame with croptype parameters

See Also
setCroptypes

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

## End(Not run)

loadCultivar
loadDispersalHost

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 = "nonCrop")
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 polygons in the landscape, the landscape must be defined before calling loadDispersalHost.

Value

A vectorised dispersal matrix.

See Also

setDispersalHost
Examples

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

---

### loadDispersalPathogen

**Load pathogen dispersal matrices**

**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. And set a vectorized identity matrix for sexual reproduction dispersal.

**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:

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

**Value**

a vectorised dispersal matrix representing the dispersal of clonal propagules, and a vectorised dispersal identity matrix for sexual propagules. All by columns.

**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 on its type.

**Usage**

`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 non-host).

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
```
loadInoculum

Description

Loads an inoculum for the beginning of the simulation (t=0), with controlled localisation (polygons), infected cultivars and pathogen genotypes. Note that landscape, gene, cultivar and croptype parameters must be set before loading the inoculum.

Usage

loadInoculum(
  params,
  pI0_all = NULL,
  pI0_host = NULL,
  pI0_patho = NULL,
  pI0_poly = NULL,
  pI0_mat = NULL
)

Arguments

params a LandsepiParams object.
pI0_all a numeric indicating the (same) probability to infect a host for all pathogen genotypes, all cultivars and in all polygons
pI0_host a vector of length Nhost indicating the probabilities to infect an host, for each cultivar (for all pathogen genotypes and all polygons).
pI0_patho a vector of length Npatho indicating the probabilities to infect an host, for each pathogen genotype (for all cultivars and all polygons).
pI0_poly a vector of length Npoly indicating the probabilities to infect an host, for each polygon (for all pathogen genotypes and all cultivars).
pI0_mat a 3D array of dimensions (1:Nhost,1:Npatho,1:Npoly) indicating the probability to infect an host, for each crop, cultivar, pathogen genotype and polygon. Note that pI0_all, pI0_host, pI0_patho and pI0_poly are not accounted if pI0_mat is filled.

Details

The different options enable different types of inoculum (localisation, infected cultivars and pathogen genetic diversity, see different options in Examples).

Unless the array pI0_mat is filled, the probability for a host to be infected at the beginning of the simulation is computed in every polygon (poly), cultivar (host) and pathogen genotype (patho) with pI0[host, patho, poly] = pI0_all * pI0_patho[patho] * pI0_host[host] * pI0_poly[poly]. Before loading the inoculum, one can use getMatrixGenePatho(), getMatrixCultivarPatho() and getMatrixCroptypePatho() to acknowledge which pathogen genotypes are adapted to which genes, cultivars and croptypes.
Once `setInoculum()` is used, one can call `inoculumToMatrix()` to get the inoculum as a 3D array 
(1:Nhost,1:Npatho,1:Npoly)

Value

a 3D array of dimensions (1:Nhost,1:Npatho,1:Npoly)

See Also

`inoculumToMatrix`, `getMatrixGenePatho`, `getMatrixCultivarPatho`, `getMatrixCrotypePatho`, `setInoculum`

Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears = 1, nTSpY = 80)
basic_patho_param <- loadPathogen(disease = "rust")
simul_params <- setPathogen(simul_params, patho_params = basic_patho_param)
simul_params <- setLandscape(simul_params, loadLandscape(id = 1))
simul_params <- setDispersalPathogen(simul_params, loadDispersalPathogen(id = 1)[[1]])
gen1 <- loadGene(name = "MG 1", type = "majorGene")
gen2 <- loadGene(name = "MG 2", type = "majorGene")
genes <- data.frame(rbind(gen1, gen2), 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"))
croptypes <- loadCroptypes(simul_params, names = c("Susceptible crop", "Resistant crop"))
croptypes <- allocateCrotypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCrotypeCultivars(croptypes, "Resistant crop", c("Resistant"))
simul_params <- setCrotypeCultivars(simul_params, croptypes)
simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = 0
, rotation_sequence = croptypes$croptypeID
, prop = c(1/2,1/2), aggreg = 1, graphic = FALSE)

#### Definition of the inoculum ####

#### Scenario 1. Only the avirulent pathogen on the susceptible cultivar ####
# In this situation, the susceptible cultivar must be entered
# at the first line of the table cultivars

## Global inoculum (i.e. in the whole landscape)
# Option 1: simply use the default parameterisation
simul_params <- setInoculum(simul_params, 5E-4)

# Option 2: use `loadInoculum()`
Npatho <- prod(simul_params$Genes$Nlevels_aggressiveness)
Nhost <- nrow(simul_params$Cultivars)
pI0 <- loadInoculum(simul_params, 
  pI0_all=5E-4, 
  pI0_host=c(1,rep(0, Nhost-1)), 
  pI0_patho=c(1,rep(0, Npatho-1)))

simul_params <- setInoculum(simul_params, pI0)
inoculumToMatrix(simul_params)

## Local inoculum (i.e. in some random polygons only)
Npatho <- prod(simul_params@Genes$Nlevels_aggressiveness)
Nhost <- nrow(simul_params@Cultivars)
Npoly <- nrow(simul_params@Landscape)
Npoly_inoc <- 5  ## number of inoculated polygons

## whether the avr pathogen can infect the polygons
compatible_poly <- getMatrixPolyPatho(simul_params)[,1]

## random polygon picked among compatible ones
id_poly <- sample(grep(1, compatible_poly), Npoly_inoc)

pI0_poly <- as.numeric(1:Npoly %in% id_poly)

pI0 <- loadInoculum(simul_params, 
  pI0_all=5E-4, 
  pI0_host=c(1,rep(0, Nhost-1)), 
  pI0_patho=c(1,rep(0, Npatho-1)), 
  pI0_poly=pI0_poly)
simul_params <- setInoculum(simul_params, pI0)
inoculumToMatrix(simul_params)

### Scenario 2. Diversity of pathogen genotypes in the inoculum ###

# in this example, Nhost=2 cultivars, Npatho=4

## Global inoculum (i.e. in all polygons of the landscape)
pI0 <- loadInoculum(simul_params, pI0_patho=c(1E-3,1E-4,1E-4,1E-5), pI0_host=c(1,1))
simul_params <- setInoculum(simul_params, pI0)
inoculumToMatrix(simul_params)[,,1:5]

## Local inoculum (i.e. in some polygons only) ##
Npoly <- nrow(simul_params@Landscape)
Npoly_inoc <- 5  ## number of inoculated polygons

## random polygon
id_poly <- sample(1:Npoly, Npoly_inoc)

pI0_poly <- as.numeric(1:Npoly %in% id_poly)

pI0 <- loadInoculum(simul_params, pI0_patho=c(1E-3,1E-4,1E-4,1E-5), 
  pI0_host=c(1,1), pI0_poly=pI0_poly)
simul_params <- setInoculum(simul_params, pI0)
inoculumToMatrix(simul_params)

## End(Not run)
loadLandscape

Description

Loads one of the five built-in landscapes simulated using a T-tesselation algorithm and composed of 155, 154, 152, 153 and 156 polygons, 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)
• "contrib": contribution of pathogen genotypes to LIR dynamics
• "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, compute_audpc100S

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")
loadSimulParams

Arguments

disease a disease name, among "rust" (default), "mildew", "sigatoka" and "no pathogen"

Details

Available diseases:

- "no pathogen"
- "rust" (genus *Puccinia*, e.g. stripe rust, stem rust and leaf rust of wheat and barley)
- "mildew" (*Plasmopara viticola*, downy mildew of grapevine)
- "sigatoka" (*Pseudocercospora fijiensis*, black sigatoka of banana) Note that when disease = "mildew" a price reduction between 0% and 5% is applied to the market value according to disease severity.

Value

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

See Also

setPathogen

Examples

```r
basic_patho_params <- loadPathogen()
basic_patho_params
```

---

loadSimulParams Load simulation parameters

Description

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

Usage

```r
loadSimulParams(inputGPKG = "")
```

Arguments

inputGPKG name of the GPKG file.

Details

See `saveDeploymentStrategy`.

Value

a LandsepiParams object.
Description

Loads a list of treatment parameters for a specific disease (initialised at 0, i.e. absence of treatments)

Usage

loadTreatment(disease = "no pathogen")

Arguments

disease a disease name, among "mildew", "sigatoka" and "no pathogen"

Details

Chemical treatment is applied in a polygon only if disease severity (i.e. I/N) in this polygon exceeds the threshold given by treatment_application_threshold. Treatment efficiency is maximum (i.e. equal to the parameter treatment_efficiency) at the time of treatment application (noted t*); then it decreases with time (i.e. natural pesticide degradation) and host growth (i.e. new biomass is not protected by treatments): protected by treatments): Efficiency of the treatment at time t after the application date is given by:

\[
\text{efficiency}(t) = \frac{\text{treatment}_\text{efficiency}}{1 + \exp(a - b \cdot C(t))}
\]

with \(C(t) = C_1 \cdot C_2\):

- \(C_1 = \exp(-\text{treatment}_\text{degradation}_\text{rate} \cdot \Delta t)\) is the reduction of fungicide concentration due to time (e.g. natural degradation, volatilization, weathering), with \(\Delta t = t - t^*\) the timelag passed since the time of treatment application.
- \(C_2 = \min(1, N(t^*)/N(t))\) is the reduction of fungicide concentration due to plant growth, since new plant tissue is not covered by fungicide. \(N(t^*)\) and \(N(t)\) being the number of host individuals at the time of treatment \(t^*\) and at time \(t\), respectively.
- \(a \in [3.5; 4.5]\) and \(b \in [8; 9]\) are shape parameters.

Value

a list of treatment parameters:

- treatment_degradation_rate = degradation rate (per time step) of chemical concentration,
- treatment_efficiency = maximal efficiency of chemical treatments (i.e. fractional reduction of pathogen infection rate at the time of application),
- treatment_timesteps = vector of time steps corresponding to treatment application dates,
- treatment_cultivars = vector of indices of the cultivars that receive treatments,
- treatment_cost = cost of a single treatment application (monetary units/ha)
- treatment_application_threshold = vector of thresholds (i.e. disease severity, one for each treated cultivar) above which the treatment is applied in a polygon.
logit

See Also

setTreatment

Examples

treat <- loadTreatment("sigatoka")
treat

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

```r
model_landsepi(
  time_param,
  area_vector,
  rotation_matrix,
  croptypes_cultivars_prop,
  dispersal,
  inits,
  seed,
  cultivars_param,
  basic_patho_param,
  genes_param,
  treatment_param
)
```

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_clonal = vectorised dispersal matrix of the pathogen (clonal propagules),
  • disp_patho_sex = vectorised dispersal matrix of the pathogen (sexual propagules),
  • disp_host = vectorised dispersal matrix of the host.

inits  list of initial conditions:
• $pI_0$ = vector of length $N_{poly}N_{patho}N_{host}$ giving the probability to be infectious (i.e. state $I$) at $t=0$ for each polygon, pathogen genotype and host.

$seed$ (for random number generation).

$seed$ (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,
• $relative\_yield\_H$ = Yield of $H$ individuals relative to $H$ individuals (100%),
• $relative\_yield\_L$ = Yield of $L$ individuals relative to $H$ individuals
• $relative\_yield\_I$ = Yield of $I$ individuals relative to $H$ individuals
• $relative\_yield\_R$ = Yield of $R$ individuals relative to $H$ individuals
• $sigmoid\_kappa\_host$ = 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 i. pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance and ii. sexual reproduction parameters:
• $infection\_rate$ = maximal expected infection rate of a propagule on a healthy host,
• $propagule\_prod\_rate$ = maximal expected reproduction rate of an infectious host per timestep,
• $latent\_period\_mean$ = minimal expected duration of the latent period,
• $latent\_period\_var$ = variance of the latent period duration,
• $infectious\_period\_mean$ = maximal expected duration of the infectious period,
• $infectious\_period\_var$ = variance of the infectious period duration,
• $survival\_prob$ = matrix giving the probability for a propagule to survive the off-season, for each croptype (rows) and each year (columns)
• $repro\_sex\_prob$ = vector of probabilities for an infectious host to reproduce via sex rather than via cloning for each timestep,
• $sigmoid\_kappa$ = 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,
• sex_propagule_viability_limit = maximum number of cropping seasons up to which a sexual propagule is viable
• sex_propagule_release_mean = average number of cropping seasons after which a sexual propagule is released.
• clonal_propagule_gradual_release = whether or not clonal propagules surviving the bottleneck are gradually released along the following cropping season.

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),
• age_of_activ_mean = vector of expected delays to resistance activation (for APRs),
• age_of_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),
• adaptation_cost = vector of adaptation penalties paid by pathogen genotypes fully adapted to the considered resistance genes on all hosts,
• relative_advantage = vector of fitness advantages of a pathogen genotype fully adapted to the resistance genes on hosts carrying these genes, relative to those that do not carry these genes,
• 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.

treatment_param list of parameters related to pesticide treatments:
• treatment_degradation_rate = degradation rate (per time step) of chemical concentration,
• treatment_efficiency = maximal efficiency of chemical treatments (i.e. fractional reduction of pathogen infection rate at the time of application),
• treatment_timesteps = vector of time-steps corresponding to treatment application dates,
• treatment_cultivars = vector of indices of the cultivars that receive treatments,
• treatment_cost = cost of a single treatment application (monetary units/ha),
• treatment_application_threshold = vector of thresholds (i.e. disease severity, one for each treated cultivar) above which the treatment is applied
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 (via mutation, recombination through sexual reproduction, selection and drift) 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 (possibly including chemical treatments) 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 (for host reproduction),
- 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). Additionally, a binary file called TFI is generated and gives the Treatment Frequency Indicator (expressed as the number of treatment applications per polygon).

References


Examples

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

```
model_landsepi

, relative_yield_I = as.numeric(cultivars$yield_I / yield0)
, relative_yield_R = as.numeric(cultivars$yield_R / yield0)
, 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"))
treatment=list(treatment_degradation_rate=0.1,
treatment_efficiency=0,
treatment_timesteps=logical(0),
treatment_cultivars=logical(0),
treatment_cost=0,
treatment_application_threshold = logical(0))

## run simulation
model_landsepi(seed=1,
time_param = time_param,
basic_patho_param = basic_patho_param,
inits = list(pI0=c(0.1, rep(0, 7))),
area_vector = area,
dispersal = list(disp_patho_clonal=c(0.99,0.01,0.01,0.99),
disp_patho_sex=c(1,0,0,1),
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,
treatment_param = treatment)

## Compute outputs
eco_param <- list(yield_perHa = cbind(H = as.numeric(cultivars$relative_yield_H),
L = as.numeric(cultivars$relative_yield_L),
I = as.numeric(cultivars$relative_yield_I),
R = as.numeric(cultivars$relative_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_res <- epid_output(, time_param, Npatho, area, rotation , croptypes_cultivars_prop, cultivars, eco_param, treatment, basic_patho_param)

#### 1-year simulation of a rust epidemic in pure susceptible crop in a single 1-km2 patch ####
## Simulation and pathogen parameters
time_param <- list(Nyears=1, nTSpY=120)
area <- c(1E6)
basic_patho_param = loadPathogen(disease = "rust")
basic_patho_param$repro_sex_prob <- rep(0, time_param$nTSpY+1)
## croptypes, cultivars and genes
rotation <- data.frame(year_1=c(0), year_2=c(0))
croptypes_cultivars_prop <- data.frame(croptypeID=c(0), cultivarID=c(0), proportion=c(1))
cultivars <- as.list(rbind(loadCultivar(name="Susceptible", type="growingHost")))

### 1-year simulation of a rust epidemic in pure susceptible crop in a single 1-km2 patch ###
## Simulation and pathogen parameters
time_param <- list(Nyears=1, nTSpY=120)
area <- c(1E6)
basic_patho_param = loadPathogen(disease = "rust")
basic_patho_param$repro_sex_prob <- rep(0, time_param$nTSpY+1)
## croptypes, cultivars and genes
rotation <- data.frame(year_1=c(0), year_2=c(0))
croptypes_cultivars_prop <- data.frame(croptypeID=c(0), cultivarID=c(0), proportion=c(1))
cultivars <- as.list(rbind(loadCultivar(name="Susceptible", type="growingHost")))
multiN <- list(cropName = character(0), adaptation_cost = numeric(0), relative_advantage = numeric(0), mutation_prob = numeric(0), efficiency = numeric(0), tradeoff_strength = numeric(0), Nlevels_aggressiveness = numeric(0), age_of_activ_mean = numeric(0), age_of_activ_var = numeric(0), target_trait = character(0), recombination_sd = numeric(0), treatment_degradation_rate = 0.1, treatment_efficiency = 0, treatment_timesteps = logical(0), treatment_cultivars = logical(0), treatment_cost = 0, treatment_application_threshold = logical(0))

## run simulation
model_landsepi(seed=1, time_param = time_param, basic_patho_param = basic_patho_param, inits = list(pI0=5E-4), area_vector = area, dispersal = list(disp_patho_clonal=c(1), disp_patho_sex=c(1), disp_host=c(1)), rotation_matrix = as.matrix(rotation), treatment_param = treatment, croptypes_cultivars_prop = as.matrix(croptypes_cultivars_prop), cultivars_param = cultivars, genes_param = genes)

## End(Not run)

### multiN Allocation of cultivars

**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).

See Also
AgriLand, allocateLandscapeCroptypes

Examples
## 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)
**periodic_cov**

*Periodic covariance function*

---

**Description**

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

**Usage**

```r
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 \( \exp(-2 \sin\left(d \pi / (2 \times \text{range})\right)^2 / \phi^2) \). It is used to generate highly fragmented or highly aggregated landscapes.

**Value**

An object of the same type as `d`.

**See Also**

`multiN`

**Examples**

```r
periodic_cov(10, range = 5)
```

---

**plotland**

*Plotting the landscape*

---

**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

## 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 croptypes in a landscape

Description

Plots croptype allocation in the landscape at a given year of the simulation

Usage

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

Arguments

- `landscape`: a SpatialPolygonsDataFrame
- `year`: year to be plotted
- `croptype_names`: croptype 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
```
plot_freqPatho(
  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
)
```

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
```
## 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)
```
price_reduction

Price reduction function

Description

Give the price reduction rate associated with the infection on the (grapevine) fruits

Usage

```r
price_reduction(
    I_host,
    N_host,
    Nhost,
    Nyears,
    nTSpY,
    severity_thresh = 0.075,
    price_penalty = 0.3
)
```

Arguments

- `I_host` number of infected individuals for each cultivar and timestep
- `N_host` total number of individuals for each cultivar and timestep
- `Nhost` total number of cultivars considered in the simulation
- `Nyears` number of simulated cropping seasons
- `nTSpY` number of timesteps (e.g. days) per cropping season
- `severity_thresh` disease severity threshold above which the price reduction is applied
- `price_penalty` percentage of price reduction

Value

A matrix with the price reduction rate per cultivar and per year of simulation

References

Description
Prints a LandsepiParams object.

Usage
## S4 method for signature 'LandsepiParams'
print(x, ...)

Arguments
x a LandsepiParams object
... print options

resetCultivarsGenes Reset cultivars genes

Description
Resets the lists of genes carried by all cultivars.

Usage
resetCultivarsGenes(params)

Arguments
params a LandsepiParams object.

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**
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 Landsepi-Params object. Briefly, the model is stochastic, spatially explicit (the basic spatial unit is an individual field or polygon), 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 (via mutation, recombination through sexual reproduction, selection and drift) 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 (possibly including chemical treatments) 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 (for host reproduction),
- 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 polygon, and when appropriate for each host and pathogen genotype. Additionally, a binary file called TFI is generated and gives the Treatment Frequency Indicator (expressed as the number of treatment applications per polygon).

References


See Also
demo_landsepi
Examples

## 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 parameters
simul_params <- setPathogen(simul_params, loadPathogen("rust"))
## 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")
gen1s <- data.frame(rbind(gene1, gene2), stringsAsFactors = FALSE)
simul_params <- setGenes(simul_params, gen1s)
## 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",
    "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)
Set the inoculum
simul_params <- setInoculum(simul_params, 5e-4)
## 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
runSimul(simul_params)

### Simulation of rust epidemics in a 1-km^2 patch cultivated with a susceptible wheat cultivar
seed=10
Nyears=5
disease="rust"
hostType="growingHost"
simul_params <- createSimulParams(outputDir = getwd())

## Seed and time parameters
simul_params <- setSeed(simul_params, seed)
simul_params <- setTime(simul_params, Nyears, nTSpY=120)

simul_params <- setPathogen(simul_params, loadPathogen(disease))
myLand <- Polygons(list(Polygon(matrix(c(0,0,1,1,0,1,1,0)*1000, nrow=4))), "ID1")
myLand <- SpatialPolygons(list(myLand))
simul_params <- setLandscape(simul_params, myLand)

## Simulation, pathogen, landscape and dispersal parameters
simul_params <- setDispersalPathogen(simul_params, c(1))

## Cultivars
simul_params <- setCultivars(simul_params, loadCultivar(name = "Susceptible", type = hostType))

croptype <- data.frame(croptypeID = 0, croptypeName = c("Fully susceptible crop"), Susceptible = 1)
simul_params <- setCroptypes(simul_params, croptype)
simul_params <- allocateLandscapeCroptypes(simul_params, rotation_period = 0, rotation_sequence = list(c(0)), rotation_realloc = FALSE, prop = 1, aggreg = 1)

## Inoculum
simul_params <- setInoculum(simul_params, 5e-4)

## list of outputs to be generated
outputlist <- loadOutputs(epid_outputs = "all", evol_outputs = ")
simul_params <- setOutputs(simul_params, outputlist)

## Check, save and run simulation
checkSimulParams(simul_params)
runSimul(simul_params, graphic = TRUE)

## End(Not run)
saveDeploymentStrategy

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")
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"))
setCroptypes

Set croptypes

Description

Updates a LandsepiParams object with croptypes and their composition with regard to cultivar proportions. Note that landscape and cultivar parameters may be required if not all information is present to set croptypes.

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.

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;cropt1&quot;</td>
<td>1</td>
<td>0</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>&quot;cropt2&quot;</td>
<td>0.5</td>
<td>0.5</td>
<td>...</td>
</tr>
</tbody>
</table>
croptypeIDs must start at 0 and match with 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)
```
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,
- **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 dataframe 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>yield_H</th>
<th>yield_L</th>
<th>yield_I</th>
<th>yield_R</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>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Resistant1</td>
<td>0.1</td>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Resistant2</td>
<td>0.1</td>
<td>2.0</td>
<td>0.1</td>
<td>0.0</td>
<td>2.5</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Value

a LandsepiParams object

See Also

loadCultivar

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)
```
**setDispersalHost**

```
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. Note that landscape parameters must be set before updating setting dispersal.

### Usage

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

### Arguments

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

### Details

The dispersal matrix gives the probability for a host individual in a polygon i (row) to migrate to polygon 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)
```
setDispersalPathogen  

Set pathogen dispersal

Description

Updates a LandsepiParams object with a pathogen dispersal matrix. Note that landscape parameters must be set before updating setting dispersal.

Usage

setDispersalPathogen(params, mat_clonal, mat_sex = NULL)

Arguments

- **params**: a LandsepiParams Object.
- **mat_clonal**: a square matrix giving the probability of pathogen dispersal (clonal propagules) from any polygon of the landscape to any other polygon. It can be generated manually, or, alternatively, via `loadDispersalPathogen`. The size of the matrix must match the number of polygons in the landscape, and lines of the matrix may sum to 1 (reflecting boundaries) or be <1 (absorbing boundaries).
- **mat_sex**: a square matrix giving the probability of pathogen dispersal (sexual propagules) from any polygon of the landscape to any other polygon (default identity matrix). It can be generated manually, or, alternatively, via `loadDispersalPathogen`. The size of the matrix must match the number of polygons in the landscape, and lines of the matrix may sum to 1 (reflecting boundaries) or be <1 (absorbing boundaries).

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 polygons in the landscape and whose elements are, for each line i and each column i’ the probability that propagules migrate from polygon i to polygon i’. Lines of the matrix can be normalised to sum to 1 (reflective boundaries); otherwise propagules dispersing outside the landscape are lost (absorbing boundaries).

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[[1]], d[[2]])
simul_params@DispPathoClonal

## End(Not run)
```

---

### setGenes

**Set genes**

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),
- **age_of_activ_mean**: expected delays to resistance activation (for APRs),
- **age_of_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),
- adaptation_cost: fitness penalties paid by pathogen genotypes fully adapted to the considered resistance genes on all hosts,
- relative_advantage: fitness advantages of pathogen genotypes fully adapted to the resistance genes on hosts carrying these genes, relative to those that do not carry these 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.
- recombination sd: standard deviation of the normal distribution used for recombination of quantitative traits during sexual reproduction (infinitesimal model)

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

<table>
<thead>
<tr>
<th>geneName</th>
<th>efficiency</th>
<th>age_of_activ_mean</th>
<th>age_of_activ_var</th>
<th>mutation_prob</th>
<th>Nlevels_aggressiveness</th>
<th>adaptation_cost</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>
</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>
</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)
```

---

**setInoculum**

**Set inoculum**

**Description**

Updates a LandsepiParams object with the initial probability for an individual host to be infectious (i.e. state I) at the beginning of the simulation (i.e. t=0).

**Usage**

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

params  
a LandsepiParams object.

val  
a numeric value (default = 5e-4) indicating the probability for the first cultivar to be infected by the first pathogen genotype in all polygons of the landscape (must be between 0 and 1). The parameter can also be entered as a 3D array of dimensions (1:Nhost,1:Npatho,1:Npoly) indicating the initial probability to be infectious, for each cultivar, pathogen genotype and polygon (independently from the possible presence of cultivars carrying resistance genes). It can be generated manually or, alternatively, via `loadInoculum`.

**Details**

Before setting the inoculum, one can use `getMatrixGenePatho()`, `getMatrixCultivarPatho()`, `getMatrixCroptypePatho()` and `getMatrixPolyPatho()` to acknowledge which pathogen genotypes are compatible to which genes, cultivars, croptypes and polygons. Once `setInoculum()` is used, one can call `inoculumToMatrix()` to get the inoculum as a 3D array (1:Nhost,1:Npatho,1:Npoly)

**Value**

a LandsepiParams object

**See Also**

`inoculumToMatrix`, `loadInoculum`

**Examples**

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

---

### setLanscape

**Set the landscape**

**Description**

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

**Usage**

`setLandscape(params, land)`
setOutputs

Description

Updates a LandsepiParams object with a list of output parameters.

Usage

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).
- 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)
- "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)
- "contrib": contribution of pathogen genotypes to LIR dynamics
- "HLIR dynamics", "H dynamics", "L dynamics", "IR dynamics", "HLL 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

```r
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_mean` = minimal expected duration of the latent period,
  - `latent_period_var` = variance of the latent period duration,
  - `infectious_period_mean` = 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,
setSeed

- sigmoid_plateau = plateau parameter of the sigmoid contamination function,
- sex_propagule_viability_limit = maximum number of cropping seasons up to which a sexual propagule is viable
- sex_propagule_release_mean = average number of seasons after which a sexual propagule is released.
- clonal_propagule_gradual_release = whether or not clonal propagules surviving the bottleneck are gradually released along the following cropping season.

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

Details

a set of parameters representative of rust fungi, downy mildew or black sigatoka can be loaded via `loadPathogen`.

Value

a `LandsepiParams` object

See Also

`loadPathogen`

Examples

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

## End(Not run)
```

---

**setSeed**

*Set the seed*

Description

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

Usage

`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 <- setSeed(simul_params, 100)
simul_params@Seed

## End(Not run)
```

---

### Description

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

### Usage

```r
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

```r
setSeedValue(seed = 10)
```
### setTime

**Description**

Updates a LandsepiParams object with time parameters: `Nyears` and `nTSpY`.

**Usage**

```r
callTime(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)
```

### setTreatment

**Description**

Updates a LandsepiParams object with treatment parameters.

**Usage**

```r
callTreatment(params, treatment_params)
```

**Examples**

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

- params: a LandsepiParams object.
- treatment_params: list of parameters related to pesticide treatments:
  - treatment_degradation_rate: degradation rate (per time step) of chemical concentration.
  - treatment_efficiency: maximal efficiency of chemical treatments (i.e. fractional reduction of pathogen infection rate at the time of application).
  - treatment_timesteps: vector of time steps corresponding to treatment application dates.
  - treatment_cultivars: vector of indices of the cultivars that receive treatments.
  - treatment_cost: cost of a single treatment application ( monetary units/ha).
  - treatment_application_threshold: vector of thresholds (i.e. disease severity, one for each treated cultivar) above which the treatment is applied in a polygon.

Details

Chemical treatment is applied in a polygon only if disease severity (i.e. I/N) in this polygon exceeds the threshold given by treatment_application_threshold. Treatment efficiency is maximum (i.e. equal to the parameter treatment_efficiency) at the time of treatment application (noted t*); then it decreases with time (i.e. natural pesticide degradation) and host growth (i.e. new biomass is not protected by treatments); protected by treatments): Efficiency of the treatment at time t after the application date is given by: 
\[
  \text{efficiency}(t) = \frac{\text{treatment\_efficiency}}{1 + \exp(a - b \cdot C(t))}
\]
with 
\[
  C(t) = C_1 \cdot C_2
\]

- \(C_1 = \exp(-\text{treatment\_degradation\_rate} \cdot \Delta t)\) is the reduction of fungicide concentration due to time (e.g. natural degradation, volatilization, weathering), with \(\Delta t = t - t^*\) the timelag passed since the time of treatment application.
- \(C_2 = \min(1, N(t^*)/N(t))\) is the reduction of fungicide concentration due to plant growth, since new plant tissue is not covered by fungicide. \(N(t^*)\) and \(N(t)\) being the number of host individuals at the time of treatment \(t^*\) and at time \(t\), respectively.
- \(a \in [3.5; 4.5]\) and \(b \in [8; 9]\) are shape parameters.

An empty list of treatments (i.e. absence of application) can be loaded using loadPathogen.

Value

a LandsepiParams object

See Also

loadTreatment
Examples

```r
## Not run:
t <- loadTreatment()
simul_params <- setTreatment(simul_params, t)
simul_params@Treatment
## End(Not run)
```

Description

Shows a LandsepiParams 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 = 5, nTSpY = 120),
  croptype_names = c("Susceptible crop"),
  croptypes_cultivars_prop = data.frame(croptypeID = 0, cultivarID = 0, proportion = 1),
  cultivars = data.frame(cultivarName = "Susceptible", initial_density = 0.1, max_density = 2, growth_rate = 0.1, reproduction_rate = 0, yield_H = 2.5, yield_L = 0, yield_I = 0, yield_R = 0, planting_cost = 225, market_value = 200),
  cultivars_genes_list = list(numeric(0)),
  genes = data.frame(geneName = character(0), mutation_prob = numeric(0), efficiency = numeric(0), tradeoff_strength = numeric(0), Nlevels_aggressiveness = numeric(0)),
  ```
adaptation_cost = numeric(0), relative_advantage = numeric(0), age_of_activ_mean = numeric(0), age_of_activ_var = numeric(0), target_trait = character(0), recombination_sd = numeric(0)),
landscape = NULL,
area = 1e+06,
rotation = data.frame(year_1 = c(0), year_2 = c(0), year_3 = c(0), year_4 = c(0), year_5 = c(0), year_6 = c(0)),
basic_patho_param = list(name = "rust", survival_prob = 1e-04, repro_sex_prob = 0, infection_rate = 0.4, propagule_prod_rate = 3.125, latent_period_mean = 10, latent_period_var = 9, infectious_period_mean = 24, infectious_period_var = 105, sigmoid_kappa = 5.333, sigmoid_sigma = 3, sigmoid_plateau = 1, sex_propagule_viability_limit = 1, sex_propagule_release_mean = 1, clonal_propagule_gradual_release = 0),
disp_patho_clonal = c(1),
disp_patho_sex = c(1),
disp_host = c(1),
treatment = list(treatment_degradation_rate = 0.1, treatment_efficiency = 0, treatment_timesteps = logical(0), treatment_cultivars = logical(0), treatment_cost = 0, treatment_application_threshold = logical(0)),
pI0 = c(5e-04),
epid_outputs = "all",
evotl_outputs = "all",
thres_breakdown = 50000,
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,
- 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),
  - age_of_activ_mean: expected delays to resistance activation (for APRs),
  - age_of_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),
  - adaptation_cost: fitness penalties paid by pathogen genotypes fully adapted to the considered resistance genes on all hosts,
  - relative_advantage: fitness advantages of pathogen genotype fully adapted to the considered resistance genes on hosts carrying these genes, relative to those that do not carry these 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,
  - recombination_sd: standard deviation of the normal distribution used for recombination of quantitative traits during sexual reproduction (infinitesimal model)

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 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.

basic_patho_param

  a list of i. pathogen aggressiveness parameters on a susceptible host for a pathogen genotype not adapted to resistance and ii. sexual reproduction parameters:

  • 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_mean = minimal expected duration of the latent period,
  • latent_period_var = variance of the latent period duration,
  • infectious_period_mean = 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 (can be entered as a matrix to give a different probability for every year (rows) and every croptype (columns)),
  • repro_sex_prob = probability for an infectious host to reproduce via sex rather than via cloning (can be entered as a vector of size `time_param$nSTpY+1` to give a different probability for every time step),
  • 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,
  • sex_propagule_viability_limit = maximum number of cropping seasons up to which a sexual propagule is viable
  • sex_propagule_release_mean = average number of seasons after which a sexual propagule is released,
  • clonal_propagule_gradual_release = Whether or not clonal propagules surviving the bottleneck are gradually released along the following cropping season.

disp_patho_clonal

  a vectorized matrix giving the probability of pathogen dispersal from any field of the landscape to any other field.

disp_patho_sex

  a vectorized matrix giving the probability of pathogen dispersal for sexual propagules 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

treatment

  list of parameters related to pesticide treatments:

  • treatment_degradation_rate = degradation rate (per time step) of chemical concentration,
  • treatment_efficiency = maximal efficiency of chemical treatments (i.e. fractional reduction of pathogen infection rate at the time of application),
• treatment_timesteps = vector of time-steps corresponding to treatment application dates,
• treatment_cultivars = vector of indices of the cultivars that receive treatments,
• treatment_cost = cost of a single treatment application (monetary units/ha)
• treatment_application_threshold = vector of thresholds (i.e. disease severity, one for each treated cultivar) above which the treatment is applied in a polygon

$pI\theta$

probability for the first cultivar to be infected (and infectious, i.e. state I) by the first pathogen genotype in all polygons of the landscape at $t=0$ (i.e. the beginning of the simulation). It can also be entered as a vector of length $N_{host}N_{patho}N_{poly}$ giving the probability for each cultivar, pathogen genotype and polygon (independently from the possible presence of cultivars carrying resistance genes).

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)
• "contrib": contribution of pathogen genotypes to LIR dynamics
• "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).

audpc100S

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

writeTXT

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

graphic

a logical indicating if graphics must be generated (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 epid_outputs="audpc_rel" (or epid_outputs="all").

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 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 (via mutation, recombination through sexual reproduction, selection and drift) 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 (possibly including chemical treatments) 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).

Value

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 (for host reproduction),
- 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 genotype. Additionally, a binary file called TFI is generated and gives the Treatment Frequency Indicator (expressed as the number of treatment applications per polygon).
References

See Also
model_landsepi, epid_output, evol_output, video, runSimul

Examples
```r
## Not run:
#### Spatially-implicit simulation with a single 1-km^2 patch 100% cultivated with a susceptible cultivar
simul_landsepi()

#### 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),
cropID = c(0, 1),
cultivarID = c(0, 1),
proportion = c(1, 1))
cultivars <- rbind(loadCultivar(name = "Susceptible", type = "growingHost"),
loadCultivar(name = "Resistant", type = "growingHost"))
genes <- 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,
基本_patho_param = loadPathogen(disease = "rust"),
disp_patho_clonal = c(0.99, 0.01, 0.01, 0.99),
disp_patho_sex = c(0.99, 0.01, 0.01, 0.99),
disp_host = c(1, 0, 0, 1),
pI0 = c(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
nTSpY <- 120
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 1",
"Resistant crop 2",
"Resistant crop 3")
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)
Nhost <- nrow(cultivars)
genes <- data.frame(rbind(
loadGene(name = "MG 1", type = "majorGene"),
loadGene(name = "MG 2", type = "majorGene"),
loadGene(name = "MG 3", type = "majorGene")
), stringsAsFactors = FALSE)
cultivars_genes_list <- list(numeric(0), 0, 1, 2)
Npatho <- prod(genes$Nlevels_aggressiveness)

## Run simulation

simul_landsepi(
seed = 12345, time_param = list(Nyears = Nyears, nTSpY = nTSpY),
croptype_names, croptypes_cultivars_prop, cultivars,
cultivars_genes_list, genes, landscape, area, rotation,
basic_patho_param = loadPathogen(disease = "rust"),
disp_patho_clonal = loadDispersalPathogen(1)[[1]],
disp_patho_sex = as.numeric(diag(Npoly)),
disp_host = as.numeric(diag(Npoly)),
pI0 = c(5E-4)
)

## End(Not run)
survivalProbToMatrix

Description
Prints the summary of a LandsepiParams object.

Usage

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

Arguments

object a LandsepiParams object.

survivalProbToMatrix Survival probability To Matrix

Description
Transform the off-season survival probability of the pathogen (1D vector of length Nyears*Ncroptypes) into a matrix (for visualization purpose)

Usage

survivalProbToMatrix(params)

Arguments

params a LandsepiParams object.

Details
After updating the off-season survival probability with updateSurvivalProb(), this function returns the probability as a matrix for every year (rows) and croptypes (columns) as well as, if croptypes have been previously allocated to a landscape, a matrix for every polygon (rows) and year (columns).

Value

a list containing a matrix of dimensions (Nyears, Ncroptypes) as well as a matrix of dimensions (Npoly, Nyears)

See Also

updateSurvivalProb
Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears=10, nTSpY=120)
simul_params <- setPathogen(simul_params, loadPathogen("rust"))

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)
croptypes <- loadCroptypes(simul_params
  , names = c("Susceptible crop", "Resistant crop", "Mixture"))
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop", "Resistant")
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture", c("Susceptible", "Resistant"))
simul_params <- setCroptypes(simul_params, croptypes)

Ncroptypes <- nrow(simul_params@Croptypes)
Nyears <- simul_params@TimeParam$Nyears

landscape <- loadLandscape(1)
simul_params <- setLandscape(simul_params, landscape)
simul_params <- allocateLandscapeCroptypes(simul_params,
  rotation_period = 0, rotation_sequence = croptypes$croptypeID,
  rotation_realloc = FALSE, 
  prop = rep(1/Ncroptypes, Ncroptypes),
  aggreg = 0.05, graphic = FALSE)

## One probability per year and per croptype:
simul_params <- updateSurvivalProb(simul_params
  , mat=matrix(runif(Nyears*Ncroptypes), ncol=Ncroptypes))
simul_params@Pathogen
survivalProbToMatrix(simul_params)

## End(Not run)
```

`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))
```

---

**updateReproSexProb**  
*Update the probability of sexual reproduction*

Description

set the probabilities for an infectious host to reproduce via sex rather than via cloning at every time step. Note that time parameters must be set before updating sexual reproduction probabilities.

Usage

```
updateReproSexProb(params, vec)
```

Arguments

- **params**: a LandsepiParams object
- **vec**: a vector of size `TimeParam$nTSpY + 1` (season end) with the probabilities for an infectious host to reproduce via sex rather than via cloning at each time step.

Value

a LandsepiParams object updated

Examples

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears=10, nTSpY=120)
simul_params <- setPathogen(simul_params, loadPathogen("rust"))
repro_sex_probs <- c(rep(0.0, 120), 1.0)
simul_params <- updateReproSexProb(simul_params, repro_sex_probs)
simul_params@Pathogen

## End(Not run)
```
updateSurvivalProb  
*Update pathogen survival probability during the off-season*

**Description**

update survival probability of the pathogen with a probability value for every simulated year (number of years = Nyears) and every croptype (number of croptypes = Ncroptypes). Note that time parameters, pathogen and croptypes must be set before updating survival probabilities.

**Usage**

```r
updateSurvivalProb(params, mat_year = NULL, mat_croptype = NULL, mat = NULL)
```

**Arguments**

- `params`  
a LandsepiParams object

- `mat_year`  
a vector of size Nyear, giving survival probabilities for every year (replicated for every croptype).

- `mat_croptype`  
a vector of size Ncroptypes, giving survival probabilities for every croptype (replicated for every year).

- `mat`  
a matrix of dimension (Nyears, Ncroptypes) giving survival probabilities for every year (rows) and every croptype (columns).

**Details**

Unless the matrix `mat` is filled, the matrix containing the survival probability during the off-season is computed for every year and croptype with `mat[year, croptype] = mat_year[year] * mat_croptype[croptype]`.

**Value**

a LandsepiParams object updated.

**See Also**

- `survivalProbToMatrix`

**Examples**

```r
## Not run:
simul_params <- createSimulParams()
simul_params <- setTime(simul_params, Nyears=10, nTSpY=120)
simul_params <- setPathogen(simul_params, loadPathogen("rust"))
cultivar1 <- loadCultivar(name = "Susceptible", type = "growingHost")
cultivar2 <- loadCultivar(name = "Resistant", type = "growingHost")
```
video <- data.frame(rbind(cultivar1, cultivar2), stringsAsFactors = FALSE)
simul_params <- setCultivars(simul_params, cultivars)

croptypes <- loadCroptypes(simul_params)
  , names = c("Susceptible crop", "Resistant crop", "Mixture")
croptypes <- allocateCroptypeCultivars(croptypes, "Susceptible crop", "Susceptible")
croptypes <- allocateCroptypeCultivars(croptypes, "Resistant crop", "Resistant")
croptypes <- allocateCroptypeCultivars(croptypes, "Mixture", c("Susceptible", "Resistant"))
simul_params <- setCroptypes(simul_params, croptypes)
Ncroptypes <- nrow(simul_params@Croptypes)
Nyears <- simul_params@TimeParam$Nyears

## Same probability in every croptype:
simul_params <- updateSurvivalProb(simul_params, mat_year=1:Nyears/100)
simul_params@Pathogen
## Same probability every year:
simul_params <- updateSurvivalProb(simul_params, mat_croptype=1:Ncroptypes/10)
simul_params@Pathogen
## specific probability for different croptypes and years:
simul_params <- updateSurvivalProb(simul_params
  , mat_year=1:Nyears/100, mat_croptype=1:Ncroptypes/10)
simul_params@Pathogen
## One probability per year and per croptype:
simul_params <- updateSurvivalProb(simul_params
  , mat=matrix(runif(Nyears*Ncroptypes), ncol=Ncroptypes))
simul_params@Pathogen
survivalProbToMatrix(simul_params)

## End(Not run)

---

**video**

*Generation of a video*

**Description**

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

**Usage**

```r
video(
  audpc,
  time_param,
  Npatho,
  landscape,
  area,
  rotation,
  croptypes,
  croptype_names = c(),
)```

```r
```
cultivars_param,  
keyDates = NULL,  
nMapPY = 10,  
path = getwd()
)

Arguments

audpc A dataframe containing audpc outputs (generated through epid_output). 1 line per year 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 webM

Examples

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

## End(Not run)
```
Index

* SEIR
  landsepi-package, 4
* datasets
  Cultivars_list, 26
dispP, 28
landscapeTEST, 45
* 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

AgriLand, 9, 66
allocateCroptypeCultivars, 12, 46, 47, 78
allocateCultivarGenes, 13, 46
allocateLandscapeCroptypes, 11, 14, 66, 86
antideriv_verhulst, 17

checkCroptypes, 17
checkCultivars, 18
checkCultivarGenes, 18
checkDispersalHost, 19
checkDispersalPathogen, 19
checkGenes, 20
checkInoculum, 20
checkLandscape, 21
checkOutputs, 21
checkPathogen, 22
checkPI0_mat, 22
checkSimulParams, 23

checkTime, 23
checkTreatment, 24
compute_audpc100S, 24, 56
createSimulParams, 25, 46, 73
Cultivars_list, 26
demo_landsepi, 27, 74
dispP, 28, 50
dispP_1(dispP), 28
dispP_2(dispP), 28
dispP_3(dispP), 28
dispP_4(dispP), 28
dispP_5(dispP), 28
epid_output, 29, 34, 99
evol_output, 32, 33, 99

getMatrixCroptypePatho, 35, 36, 38, 53
getMatrixCultivarPatho, 35, 36, 38, 53
getMatrixGenePatho, 35, 36, 37, 38, 53
getMatrixPolyPatho, 35, 36, 38, 38

initialize, LandsepiParams-method, 39
inoculumToMatrix, 41, 53, 85
invlogit, 42
is.in.01, 43
is.positive, 44
is.strict.positive, 44
is.wholenumber, 45

landscapeTEST, 45, 55
landscapeTEST1(landscapeTEST), 45
landscapeTEST2(landscapeTEST), 45
landscapeTEST3(landscapeTEST), 45
landscapeTEST4(landscapeTEST), 45
landscapeTEST5(landscapeTEST), 45
landsepi (landsepi-package), 4
landsepi-package, 4
LandsepiParams, 46
LandsepiParams-class (LandsepiParams), 46
INDEX

loadCroptypes. 12, 46, 47, 78, 79
loadCultivar. 46, 48, 79, 80
loadDispersalHost. 47, 49, 81
loadDispersalPathogen. 47, 50, 82
loadGene. 46, 51, 83, 84
loadInoculum. 52, 85
loadLandscape. 46, 50, 54, 86
loadOutputs. 25, 47, 55, 87, 88
loadPathogen. 46, 56, 89, 92
loadSimulParams. 57
loadTreatment. 47, 58, 92
logit. 59

model_landsepi. 60, 99
multiN. 11, 65, 67
periodic_cov. 11, 67
plot_allocation. 69
plot_freqPatho. 70
plotland. 67, 69
price_reduction. 71
print. 72
print, LandsepiParams-method (print), 72
resetCultivarsGenes. 72
runShinyApp. 27, 73
runSimul. 27, 73, 99

saveDeploymentStrategy. 57, 76
setCroptypes. 13, 46, 48, 78
setCultivars. 13, 14, 46, 49, 79
setDispersalHost. 47, 49, 81
setDispersalPathogen. 47, 50, 82
setGenes. 14, 46, 51, 83
setInoculum. 41, 47, 53, 84
setLandscape. 55
setLandscape (setLansdcape). 85
setLandscape. 85
setOutputs. 47, 56, 86
setPathogen. 46, 57, 88
setSeed. 89
setSeedValue. 90
setTime. 47, 91
setTreatment. 47, 59, 91
show. 93
show, LandsepiParams-method (show), 93
simul_landsepi. 93
summary. 100
summary, LandsepiParams-method (summary). 100

loadOutputs. 25, 47, 55, 87, 88
loadLandscape. 46, 50, 54, 86
loadSimulParams. 57
loadTreatment. 47, 58, 92
logit. 59

model_landsepi, 60, 99
multiN, 11, 65, 67

loadCroptypes, 12, 46, 47, 78, 79
loadCultivar, 46, 48, 79, 80
loadDispersalHost, 47, 49, 81
loadDispersalPathogen, 47, 50, 82
loadGene, 46, 51, 83, 84
loadInoculum, 52, 85
loadLandscape, 46, 50, 54, 86
loadOutputs, 25, 47, 55, 87, 88
loadPathogen, 46, 56, 89, 92
loadSimulParams, 57
loadTreatment, 47, 58, 92
logit, 59

model_landsepi, 60, 99
multiN, 11, 65, 67

loadCroptypes, 12, 46, 47, 78, 79
loadCultivar, 46, 48, 79, 80
loadDispersalHost, 47, 49, 81
loadDispersalPathogen, 47, 50, 82
loadGene, 46, 51, 83, 84
loadInoculum, 52, 85
loadLandscape, 46, 50, 54, 86
loadOutputs, 25, 47, 55, 87, 88
loadPathogen, 46, 56, 89, 92
loadSimulParams, 57
loadTreatment, 47, 58, 92
logit, 59

model_landsepi, 60, 99
multiN, 11, 65, 67

periodic_cov, 11, 67
plot_allocation, 69
plot_freqPatho, 70
plotland, 67, 69
price_reduction, 71
print, 72
print, LandsepiParams-method (print), 72
resetCultivarsGenes, 72
runShinyApp, 27, 73
runSimul, 27, 73, 99

saveDeploymentStrategy, 57, 76
setCroptypes, 13, 46, 48, 78
setCultivars, 13, 14, 46, 49, 79
setDispersalHost, 47, 49, 81
setDispersalPathogen, 47, 50, 82
setGenes, 14, 46, 51, 83
setInoculum, 41, 47, 53, 84
setLandscape, 55
setLandscape (setLansdcape), 85
setLandscape, 85
setOutputs, 47, 56, 86
setPathogen, 46, 57, 88
setSeed, 89
setSeedValue, 90
setTime, 47, 91
setTreatment, 47, 59, 91
show, 93
show, LandsepiParams-method (show), 93
simul_landsepi, 93
summary, 100
summary, LandsepiParams-method (summary), 100

survivalProbToMatrix, 101, 104
switch_patho_to_aggr, 102
updateReproSexProb, 103
updateSurvivalProb, 101, 104
video, 99, 105