Package ‘rmetasim’

September 19, 2017

Version 3.1.3
Author Allan Strand <stranda@cofc.edu>,
James Niehaus
Maintainer Allan Strand <stranda@cofc.edu>
Date 2017-9-3
Imports pegas, ade4, gtools, adegenet
Suggests magrittr, knitr, ggplot2, reshape2
Title An Individual-Based Population Genetic Simulation Environment
License GPL
Description An interface between R and the metasim simulation engine.
Please see the vignettes CreatingLandscapes and Simulating to get some ideas on how to use the packages.
See the rmetasim vignette to get an overview and to see important changes to the code in the most recent version.

VignetteBuilder knitr
NeedsCompilation yes
Repository CRAN
Date/Publication 2017-09-19 00:26:32 UTC

R topics documented:

  is.landscape .................................................. 2
  landscape.allelecount .................................. 3
  landscape.allelefreq ................................... 4
  landscape.amova ......................................... 4
  landscape.amova.locus .................................. 5
  landscape.amova.pairwise .............................. 6
  landscape.clean ......................................... 6
is.\text{landscape}

\textit{Test whether an object is a (fairly) legitimate landscape}

\textbf{Description}

Test whether a genuine landscape
landscape.allelecount

**Usage**

```r
is.landscape(Rland = NULL, verb = TRUE, exact = FALSE)
```

**Arguments**

- `Rland`: the Rmetasim landscape object
- `verb`: print why not a landscape
- `exact`: more strict

**Examples**

```r
exampleland <- landscape.new.example()
is.landscape(exampleland)
rm(exampleland)
```

---

**landscape.allelecount**

*Calculate allele numbers (frequency in the statistical sense) at each locus in each population*

**Description**

Calculate allele counts

**Usage**

```r
landscape.allelecount(Rland, tbl.out=FALSE)
```

**Arguments**

- `Rland`: the Rmetasim landscape object
- `tbl.out`: Create a table rather than data.frame output

**Value**

Depends on the value of `tbl.out`. See above.

**See Also**

- `landscape.allelefreq`, `landscape.obs.het`, `landscape.exp.het`, `landscape.Fwright`, `landscape.Fst`

**Examples**

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
landscape.allelefreq(exampleland, tbl.out=TRUE)
landscape.allelefreq(exampleland, tbl.out=FALSE)
rm(exampleland)
```
landscape.allelefreq  \hspace{1cm} \textit{Calculate allele frequencies at each locus in each population}

\textbf{Description}  
Calculate allele frequencies

\textbf{Usage}  
\begin{verbatim}
landscape.allelefreq(rland, tbl.out=FALSE)
\end{verbatim}

\textbf{Arguments}  
- \texttt{rland} \hspace{1cm} the Rmetasim landscape object
- \texttt{tbl.out} \hspace{1cm} return as a (three-dimensional) table if TRUE. If FALSE, return as a dataframe with categorical variables denoting the locus, population and allele.

\textbf{Value}  
Depends on the value of tbl.out. See above.

\textbf{See Also}  
landscape.obs.het, landscape.exp.het, landscape.Fwright, landscape.Fst

\textbf{Examples}  
\begin{verbatim}
exempleland <- landscape.new.example()
exempleland <- landscape.simulate(exempleland, 4)
landscape.allelefreq(exempleland, tbl.out=TRUE)
landscape.allelefreq(exempleland, tbl.out=FALSE)
rm(exempleland)
\end{verbatim}

landscape.amova  \hspace{1cm} \textit{calcuates \( \phi_{ST} \) for every locus in the landscape}

\textbf{Description}  
\( \phi_{ST} \) \hspace{1cm} \textit{for every locus in the landscape}

\textbf{Usage}  
\begin{verbatim}
landscape.amova(rland, np = NULL, ns = NULL)
\end{verbatim}
**Arguments**

- `rland` : landscape object
- `np` : max number of pops to include
- `ns` : max number of samples to collect

**Value**

vector of length equal to the number of loci

**See Also**

`landscape.amova.locus`, `landscape.amova.pairwise`

---

**Description**

Runs an amova on a locus. Does not include information about sequence similarity or ssr size in analysis.

**Usage**

```r
landscape.amova.locus(rland, l = 1)
```

**Arguments**

- `l` : locus number
- `rland` : landscape object

**Details**

Should be the same as Weir and Cockerham's $\theta$

**Value**

list of amova results for a locus

**See Also**

`landscape.amova`, `landscape.amova.pairwise`
landscape.amova.pairwise

 calculates pairwise phi-ST for a landscape

Description

pairwise $\phi_{ST}$ calculator. Kind of slow. use landscape.sample

Usage

landscape.amova.pairwise(rland)

Arguments

rland landscape object

See Also

landscape.amova, landscape.amova.locus

---

landscape.clean

 Function to resolve inconsistencies within a landscape

Description

Converts a landscape to internal format and back. This can resolve inconsistencies in a 'hand-built' landscape

Usage

landscape.clean(rland)

Arguments

rland the Rmetasim landscape object

Examples

exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
exampleland.clean <- landscape.clean(exampleland)
rm(exampleland)
**landscape.coalinput**

Add loci and individuals based upon output from SimCoal 2.0

**Description**

Take rmetasim object and replaces the locus and individual data based on the results of a SimCoal run stored in Arlequin format files.

**Usage**

```
## must be called AFTER integer, switch, and float params have been created
landscape.coalinput(rland, npp=200, arlseq = NULL, arlms = NULL, seqsitemut=1e-7, msmut = 5e-4, mut.rates = NULL)
```

**Arguments**

- `rland`: partially created landscape object, required
- `npp`: number per population. Scalar or vector of length equal to number of populations. If scalar, value replicated
- `arlseq`: name of the Arlequin format file containing a single locus of haploid sequence data for any number of populations
- `arlms`: name of the Arlequin format file containing a single locus of diploid microsatellite data for any number of populations
- `seqsitemut`: mutation rate for sequence data
- `msmut`: mutation rate for diploid genotypic data
- `mut.rates`: alternative means to specify mutation rates. Legal values are either NULL or a vector of rates equal to the number of loci to simulate. If NULL, SSR loci are assigned msmut as a mutation rate and sequence-based loci, seqsitemut. If a vector, overrides msmut and seqsitemut

**Details**

This function provides part of an interface between R and SimCoal, an environment for simulating sequences and microsatellite genotypes from coalescent trees. SimCoal can be used to simulate a standing crop of alleles and their relationships under a wide range of demographies. It returns haplotypes and genotypes of individuals in Arlequin format files.

If either `arlseq` or `arlms` are set to NULL, their corresponding data will not be included in the landscape (for example if arlseq=NULL, only diploid genotypes will be imported).

The genotypes in the Arlequin files are used to create `rland$loci` objects based upon their frequencies and states. These `rland$loci` sub-objects are then used to populate the `rland$individuals` sub-object.

The number of populations in the Arlequin files should be the same among genetic locus types (sequence versus microsatellite) and the `rland$intparam$habitats` parameter. The per-population frequency data will be used in creating individuals.
Value

an rmetasim object with new loci and individuals

Author(s)

Mark Bravington and Allan Strand

Examples

def exampleland <- landscape.new.empty()
def exampleland <- landscape.new.intparam(exampleland, s=2, h=2)
def exampleland <- landscape.new.floatparam(exampleland)
def exampleland <- landscape.new.switchparam(exampleland)

# exampleland <- landscape.coalinput(exampleland)
# exampleland$loci

landscape.compress

Function to resolve inconsistencies within a landscape, deprecated

Description

Deprecated, ‘clean.landscape()’ does the same.

Usage

landscape.compress(Rland)

Arguments

Rland the Rmetasim landscape object

Examples

def exampleland <- landscape.new.example()
def exampleland <- landscape.simulate(exampleland, 4)
def exampleland.clean <- landscape.compress(exampleland)
rm(exampleland)
landscape.democol  

Description

return largest demographic column from a landscape

Usage

landscape.democol()

Details

Useful to write functions that will be insensitive to some changes in the individuals object (mainly addition of non-genetic information)

Value

a scalar integer representing the largest column of demographic information in a landscape’s individuals object

See Also

landscape.locus

landscape.demography  Calculate demographic parameters

Description

Calculate demographic parameters from a landscape: CURRENTLY BROKEN!

Usage

landscape.demography(Rland)

Arguments

Rland      the Rmetasim landscape object

Value

A list of length populations+1. The first 1..populations elements are lists comprised of lambda, the equilibrium stage-structure, the actual stage structure, a χ² value for the test of difference between predicted and actual, and an estimate of significance for that test. The last element of the main list is the same as the previous ones except it refers to the entire landscape
landscape.exp.het Calculate expected heterozygosity

Description

Calculate expected heterozygosity from a landscape

Usage

landscape.exp.het(Rland)

Arguments

Rland the Rmetasim landscape object

Details

Calculates the expected heterozygosity in each population:

\[ 1 - \sum_i p_i^2 \]

where \( p \) is a vector of allele frequencies for a locus in a population.

Value

A matrix with num loci columns and num populations rows. Each element reflects the expected heterozygosity for that population x locus combination

See Also

landscape.obs.het, Fst.landscape

Examples

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
exphet <- landscape.exp.het(exampleland)
rm(exampleland)
```
landscape.freq.locnames

*Return a vector of locus IDs concatenated to Allele ids*

**Description**

Extracts locus and allele ids from landscape

**Usage**

`landscape.freq.locnames(Rland)`

**Arguments**

- **Rland**
  
  the Rmetasim landscape object

**Details**

This function returns a vector of reasonable column names for a frequency by individual matrix from an Rland object. It is mainly useful in creating genind objects from landscapes.

Genotypes are stored in a landscape object as columns representing allelic position. If the locus is haploid, there is a single column and if diploid, two. The allele indices are entered in each column for individual genotypes.

The ‘frequency per individual’ format has a column for every locusXallele combination; genotypes are represented as the frequency of particular alleles in an individuals genotype (for example freq = 1 in a column diploid means that the individual is homozygous for that column, 0.5 in two columns represents a heterozygote for the alleles represented by those two columns). This is essentially the genind format from adegenet.

**Value**

character vector

**See Also**

landscape.ind.freq, landscape.make.genind

**Examples**

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
landscape.freq.locnames(exampleland)
rm(exampleland)
```
landscape.Fst  

*Calculates population structure statistic for the entire landscape*

**Description**

Calculate Fst for each allele at each locus in the landscape. If verb is set to TRUE, the function prints average Fst for loci and overall.

**Usage**

```r
landscape.Fst(rland, verb=FALSE)
```

**Arguments**

- `rland`: the Rmetasim landscape object
- `verb`: determines whether there is verbose output

**Details**

Calculates Fst based upon the ratio of variance in allele frequency across subpopulations to the total variance in that allele’s frequency. Does not calculate Wright’s other statistics.

**Value**

A matrix with num alleles columns and num loci rows. Each element reflects the value of Fst for that allelexlocus combination. NA is assigned to alleles that are not present at a locus (either no longer or ever)

**See Also**

obs.het.landscape, exp.het.landscape, FWright.landscape

**Examples**

```r
eexampleland <- landscape.new.example()
eexampleland <- landscape.simulate(eexampleland, 4)
Fst <- landscape.Fst(eexampleland, verb=TRUE)
Fst
rm(eexampleland,Fst)
```
landscape.ind.freq  

Description
Converts a landscape into a format that can be used in creating genind objects

Usage
landscape.ind.freq(rlandL include.states=TRUE)

Arguments
rland  
the Rmetasim landscape object

include.states  
deprecated

Details
This function returns a frequency by individual matrix for the landscape given in rland. Genotypes are stored in a landscape object as columns representing allelic position. If the locus is haploid, there is a single column and if diploid, two. The allele indices are entered in each column for individual genotypes.

The 'frequency per individual' format has a column for every locusXallele combination; genotypes are represented as the frequency of particular alleles in an individual's genotype (for example freq = 1 in a column diploid means that the individual is homozygous for that column, 0.5 in two columns represents a heterozygote for the alleles represented by those two columns). This is essentially the genind format from adegenet.

Value
matrix

See Also
landscape.freq.locnames, landscape.make.genind

Examples
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 25)
head(landscape.ind.freq(exampleland))
landscape.locus  
return a matrix containing genotypes for a particular locus

Description

return a matrix containing genotypes for a particular locus

Usage

landscape.locus(Rland, lnum=1)

Arguments

lnum  the locus to return
Rland  the Rmetasim landscape object

Details

Returns a matrix with rows = dim(rland$individuals)[1]. The first three columns correspond to the class (and two placeholder variables) of an individual. Here rland is a landscape object. The remaining columns (1 if haploid, 2 if diploid) contain the allele indices for the various loci

Value

matrix

See Also

landscape.populations

Examples

exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
print("Allele frequencies at locus 1")
table(landscape.locus(exampleland,1)[,c(-1:-(landscape.democol()))])
rm(exampleland)
landscape.locus.states

*return a matrix containing actual allelic states and their indices*

**Description**

Convenience function to return a matrix containing the states of the alleles and their indices for a particular locus.

**Usage**

```r
landscape.locus.states(rland, lnum = 1)
```

**Arguments**

- `lnum`: the locus to return
- `rland`: the Rmetasim landscape object

**Value**

matrix

**See Also**

landscape.locus, landscape.states

---

landscape.locusvec

*return a vector with the locus ids for each column in the individuals component of a landscape*

**Description**

return a vector with the locus ids for each column in the individuals component of a landscape

**Usage**

```r
landscape.locusvec(rland)
```

**Arguments**

- `rland`: the Rmetasim landscape object

**Value**

vector
See Also

landscape.populations

Examples

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
landscape.locusvec(exampleland)
rm(exampleland)
```

```r
landscape.make.genind

Converting genetic marker data in a landscape into a `genind` format from adegenet

Description

Makes a genind object from a landscape. Populations can also be given names with an optional parameter.

Usage

```r
landscape.make.genind(Rland)
```

Arguments

- `Rland` the Rmetasim landscape object

Value

an adegenet `genind` object with optional population names

See Also

landscape.make.genpop
landscape.make.genpop

Converts genetic marker data in a landscape into a the genpop format from adegenet

Description
Makes a genpop object from a landscape.

Usage
landscape.make.genpop(rland)

Arguments
rland the Rmetasim landscape object

Value
an adegenet 'genpop' object.

See Also
landscape.make.genind

landscape.mig.matrix

Creates a Migration Matrix for All Life Stages

Description
Creates a binary matrix representing the migration between a set of 'h' populations containing 's' life stages each. This matrix can be based on a given migration model or on a custom matrix.

Usage
landscape.mig.matrix(h=3,s=2,mig.model="island",first.rep.s=s, h.dim=NULL, distance.fun=NULL, distance.factor=1, R.custom = NULL, ...)

Arguments
h habitats (default=3), the number of different subpopulations within the landscape
s stages (default=2), the number of stages in the life cycle of the organism
mig.model migration model (default="island"), the migration model to be used to make the matrix. Choices are "island", "stepping.stone.linear", "stepping.stone.circular", "twoD", "twoDwDiagonal", "distance","custom". See details.
first.rep.s first reproductive life stage (default=s), the life stage at which the organism starts to reproduce

h.dim rectangular arrangement of populations (default=NULL). vector of length 2 showing the distribution of populations in rows and columns when the model of evolution is equal to "twoD" or "twoDwDiagonal".

distance.fun function to calculate migration (default=NULL), an user created function that uses the distance between each population to calculate the migration rate between those two populations if the migration model is equal to "distance".

distance.factor distance factor (default=1), the distance between each adjacent population if the migration model is equal to "distance"

R.custom custom migration matrix (default=NULL), migration matrix with 'h' by 'h' dimensions to be used to create the larger 'h*s' by 'h*s' matrix if the migration model is equal to "custom"

... additional arguments passed to 'distance.fun'

Details

This function can work on three different ways:

1. With a given migration model This will take in consideration one of the predefined migration models to create the migration matrix.
   - "island" Migration occurs among all the populations in the model.
   - "stepping.stone.linear" The populations are distributed linearly and migration only occurs between the adjacent populations.
   - "stepping.stone.circular" Similar to "stepping.stone.linear", but the populations are distributed in a circle so there is migration between the first and the last population.
   - "twoD" The populations are distributed in two dimensions. It is necessary to provide the "h.dim" term in order to determine the distribution of the populations in rows and columns respectively. Migration only occurs between populations that are adjacent to each other
   - "twoDwDiagonal" Similar to "twoD", but within a square formed by four populations (two rows and two columns) there is migration in the diagonal

2. With a custom migration matrix This requires the user to provide the "R.custom" argument. In this case the function will expand the migration pattern given on "R.custom" to encompass all life stages. For the function to work this way the "mig.model" term must be equal to "custom".

3. With a distance functions This requires a function that shows how migration changes with changing distance. The "distance.fun" is very versatile and the use of "..." allows the functions to accept extra terms. The "distance.factor" term allows the user to change the distance between the populations to facilitate the use of distance functions that work on greater or smaller scales. It is necessary to provide the "h.dim". It is necessary to provide the "h.dim" term in order to determine the distribution of the populations in rows and columns respectively. It is possible to have a linear distribution of populations if one of the terms of "h.dim" is equal to 1. For the function to work this way the "mig.model" term must be equal to "distance".
Value

Matrix containing the final result from the function call. This should be a "h*s" by "h*s" matrix indicating what life stages from what populations migrate to the first life stage of what populations. When the "mig.model" is equal to distance this matrix will indicate the rate of migration between the populations instead of if it just occurs or not.

- **h**: the number of different subpopulations
- **s**: the number of stages in the life cycle of the organism
- **mig.model**: the migration model used to make the matrix
- **first.rep.s**: the life stage at which the organism starts to reproduce
- **R.int**: A "h" by "h" matrix indicating the migration pattern. If "mig.model" is equal to custom, "R.int" will be equal to "R.custom".

Author(s)

Artur Veloso and Allan Strand

Examples

```r
#Circular stepping stone migration model
landscape.mig.matrix(s=3,h=4,mig.model="stepping.stone.linear",first.rep.s=2)

#Two dimensions with diagonal migration model
landscape.mig.matrix(h=18,h.dim=c(3,6),s=2,mig.model="twoDwDiagonal")

#Using a custom migration matrix
R.custom <- matrix(c(0, 0, 1, 0,
                      1, 0, 1, 0,
                      1, 0, 0, 0,
                      1, 0, 1, 0), ncol=4,nrow=4,byrow=TRUE)
landscape.mig.matrix(s=3,h=4,first.rep.s=2,mig.model="custom",R.custom=R.custom)

#Using a distance function. Notice that the distance function requires
#the argument "lambda" that can be given in the "make.mig.matrix"
#function call.
my.dist <- function(distance,lambda) {exp(-distance*lambda)}
landscape.mig.matrix(h=18,h.dim=c(3,6),s=2,mig.model="distance",distance.fun=my.dist,lamba=1)
```

---

`landscape.mismatchdist`

**Calculate a mismatch distribution for a locus in a landscape**

Description

Calculate mismatch distribution from a landscape based upon the number of seqregating sites.
Usage

landscape.mismatchdist(Rland, lnum=1)

Arguments

lnum  
locus number to calculate mismatch upon

Rland  
the Rmetasim landscape object

Details

Calculates a mismatch distribution for DNA-sequence-based loci.

Value

A matrix with num loci columns and num populations rows. Each element reflects the estimated theta for that population x locus combination.

Examples

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
misdist <- landscape.mismatchdist(exampleland, 3) #will produce
misdist #ridiculous output
```

landscape.modify.epoch

Modifies one of the landscape’s epochs

Description

This function updates the demographic parameters in a landscape for a particular epoch.

Usage

landscape.modify.epoch(rland, epoch=1, S=NULL, R=NULL, M=NULL,
epochprob=NULL, startgen=NULL, extinct=NULL,
carry=NULL, localprob=NULL)

Arguments

rland  
landscape object, required

epoch  
the epoch to modify, default 1

S  
(default=NULL) Survivability matrix for epoch, NULL leaves unchanged

R  
(default=NULL) female Reproduction matrix for epoch, NULL leaves unchanged

M  
(default=NULL) Male reproduction matrix for epoch, NULL leaves unchanged

epochprob  
(default=NULL) probability of choosing this epoch, NULL leaves unchanged
**landscape.new.epoch**

Create an Epoch

### Description

Create an epoch for a Rmetasim landscape object

### Usage

```r
## must be called AFTER integer, switch, and float params have
## been created and after the demography has been created
## S, R, and M matrices must be square matrices of size X by X
## where X = rland$intparam$stages*rland$intparam$habitats

landscape.new.epoch(rland, S=NULL, R=NULL, M=NULL,
epochprob=1, startgen=0, extinct=NULL, carry=NULL,
localprob=NULL)
```

### Arguments

- **rland**: partially created landscape object, required
- **S**: (default=NULL) Survivability matrix for epoch, NULL gives no movement between subpopulations (0 matrix)
- **R**: (default=NULL) female Reproduction matrix for epoch, NULL gives no dispersal between subpopulations (0 matrix)
- **M**: (default=NULL) Male reproduction matrix for epoch, NULL gives no sperm or pollen movement between subpopulations (0 matrix)
- **epochprob**: (default=1) probability of choosing this epoch randomly if randepoch==1
- **startgen**: (default=0) generation in which this epoch starts
- **extinct**: (default=NULL) vector of extinction probabilities per generation for each subpopulation, must be rland$intparam$habitats in length, passing NULL gives a 0% probability of extinction to each subpopulation
- **carry**: (default=NULL) vector of carrying capacities for each subpopulation, must be rland$intparam$habitats in length, passing NULL gives a 1000 individual carrying capacity to each subpopulation
- **localprob**: (default=NULL) vector of probabilities for choosing local demographies, must be length(rland$demography$localdem) in length, passing NULL gives each demography an equal probability
## Examples

```r
exampleS <- matrix(c(0.1, 0, 0.5, 0.3), nrow = 2)
exampleR <- matrix(c(0, 1.1, 0, 0), nrow = 2)
exampleM <- matrix(c(0, 0, 0, 1), nrow = 2)

exampleland <- landscape.new.empty()
exampleland <- landscape.new.intparam(exampleland, s=2, h=2)
exampleland <- landscape.new.floatparam(exampleland)
exampleland <- landscape.new.local.demo(exampleland, exampleS, exampleR, exampleM)

## nonsense matrices
exampleS <- matrix(c(rep(0,4),
                      rep(1,4),
                      rep(0,4),
                      rep(1,4)), nrow = 4)
exampleR <- matrix(c(rep(0.5,4),
                      rep(0,4),
                      rep(0.5,4),
                      rep(0,4)), nrow = 4)
exampleM <- matrix(c(rep(0,4),
                      rep(.25,4),
                      rep(0,4),
                      rep(0,4)), nrow = 4)

## defaults
exampleland<- landscape.new.epoch(exampleland, exampleS, exampleR, exampleM)
exampleland$demography$epochs[[1]]
rm(exampleS)
rm(exampleR)
rm(exampleM)
rm(exampleland)
```

---

### landscape.new.example  
Create a Default Landscape

#### Description
Create a Rmetasim landscape with all default parameters.

#### Usage
```r
landscape.new.example()
```

#### Arguments
None
**landscape.new.floatparam**

*Create a set of floating point parameters*

**Description**

Create a set of floating point parameters for a Rmetasim landscape.

**Usage**

```
## must be called AFTER landscape.new.empty()
landscape.new.floatparam(rland, s=0)
```

**Arguments**

- `rland`: skeleton of landscape object, required
- `s`: selfing (default=0), the selfing rate of the species

**Examples**

```
## Defaults
exampleland <- landscape.new.empty()
exampleland <- landscape.new.floatparam(exampleland)
exexampleland$floatparam

## .5 selfing rate
exampleland <- landscape.new.empty()
exampleland <- landscape.new.floatparam(exampleland, s=0.5)
exexampleland$floatparam
rm(exampleland)
```

---

**landscape.new.individuals**

*Fill a landscape with individuals*

**Description**

Create a set of individuals for a Rmetasim landscape object.
landscape.new.individuals

Usage

```r
## must be called AFTER integer, switch, and float params, demography,
## epochs, and loci have been created

landscape.new.individuals(rland, Populationsizes)
```

Arguments

- `rland`: nearly complete landscape object, required
- `Populationsizes`: vector of integers denoting how many individuals are in which stage and in which subpopulation, vector is ordered as: (pop1 stage1, pop1 stage2, ..., pop2 stage1, pop2 stage2, ...), must be of length `rland$intparam$habitats * rland$intparam$stages`

Examples

```r
eexampleS <- matrix(c(0.1, 0, 0.5, 0.3), nrow = 2)
eexampleR <- matrix(c(0, 1.1, 0, 0), nrow = 2)
eexampleM <- matrix(c(0, 0, 0, 1), nrow = 2)

eexampleland <- landscape.new.empty()
eexampleland <- landscape.new.intparam(exampleland, s=2, h=2)
eexampleland <- landscape.new.floatparam(exampleland)
eexampleland <- landscape.new.switchparam(exampleland)
eexampleland <- landscape.new.local.demo(exampleland, exampleS, exampleR, exampleM)
```
landscape.new.intparam

Create a set of integer parameters

Description
Create a set of integer parameters for a Rmetasim landscape.

Usage

```r
## must be called AFTER landscape.new.empty()
landscape.new.intparam(rland, h=1, s=1, cg=0, ce=0, totgen=1000,
                        maxland=200000, xdim=0, ydim=0)
```

Arguments

- `rland`: skeleton of landscape object, required
- `h`: habitats (default=1), the number of different subpopulations within the landscape
- `s`: stages (default=1), the number of stages in the life cycle of the organism
- `cg`: currentgen (default=0), the current generation the simulation has reached
- `ce`: currentepoch (default=0), the current epoch the simulation has reached
- `totgen`: totalgens (default=1000), the total number of generations to simulate
- `maxland`: maxlandsize (default=200000), the maximum number of individuals that can exist in the simulation. Keeps large simulation from crashing computer
- `xdim`: if habitats are arrayed in rectangle, number of x grid cells
- `ydim`: if habitats are arrayed in rectangle, number of x grid cells

Examples

```r
## Defaults
exampleland <- landscape.new.empty()
exampleland <- landscape.new.intparam(exampleland)
exampleland$intparam

## 2 habitats, 3 stage lifecycle, 1000000 generations, maximum 1000000 individuals
exampleland <- landscape.new.empty()
exampleland <- landscape.new.intparam(exampleland, h=2, s=2, totgen=1000000, maxland=1000000)
exampleland$intparam

rm(exampleland)
```
landscape.new.landscape

Create a Skeletal Landscape

Description

Create a skeletal Rmetasim landscape ready to be configured

Usage

landscape.new.empty()

Arguments

None

Examples

## Only usage
landscape.new.empty()

landscape.new.local.demo

Create a Local Demography

Description

Create a local demography for an Rmetasim Landscape object

Usage

## must be called AFTER integer, switch, and float params have been created
## S, R, and M matrices must be square matrices of size
## rland$intparam$stages by rland$intparam$stages

landscape.new.local.demo(rland,S,R,M,k=0)

Arguments

rland  partially created landscape object, required
S  Survivablity matrix for demography, required
R  female Reproduction matrix for demography, required
M  Male reproduction matrix for demography, required
k  flag for type of matrix, 0=demography at zero population density, 1=demography at carrying capacity
Details

The local demography objects encapsulate demography within a particular region. Multiple such objects can be defined to account for different demographies across space. The flag, k, can indicate whether the matrices represent demography at zero population growth and at carrying capacity, if density-dependence is modeled.

Examples

```r
exampleS <- matrix(c(0.1, 0, 0.5, 0.3), nrow = 2)
exampleR <- matrix(c(0, 1.1, 0, 0), nrow = 2)
exampleM <- matrix(c(0, 0, 0, 1), nrow = 2)

exampleland <- landscape.new.empty()
exampleland <- landscape.new.intparam(exampleland, s=2)
exampleland <- landscape.new.floatparam(exampleland)
exampleland <- landscape.new.switchparam(exampleland)
exampleland <- landscape.new.local.demo(exampleland, exampleS, exampleR, exampleM)

exampleland$demography$localedem

rm(exampleS)
rm(exampleR)
rm(exampleM)
rm(exampleland)
```

---

**landscape.new.locus**  
*Add a locus*

Description

Add a locus to a Rmetasim landscape object.

Usage

```r
## must be called AFTER integer, switch, and float params have been created
landscape.new.locus(rland,type=0,ploidy=1,
                    mutationrate=0,transmission=1,numalleles=2,
                    allelesize=50,frequencies=NULL, states = NULL)
```

Arguments

- `rland`: partially created landscape object, required
- `type`: (default=0) type of locus, 0=Infinite Allele mutation model (Integer), 1=Stepwise mutation model (Integer) state, 2=DNA base (variable length string state)
- `ploidy`: (default=1) locus ploidy, 1 or 2
- `mutationrate`: (default=0) probability of mutation per generation, less than or equal to 1
transmission  (default=1) 1=uniparental inheritance, 0=biparental inheritance
numalleles   (default=2) number of different alleles at the time of creation
allelesize   (default=50) length of DNA strings if type=2
frequencies  (default=NULL) vector of frequencies for each allele, must be numalleles long
             and add up to 1, if NULL frequencies are equally distributed
states       States for each of the alleles specified. If NULL sequential numeric states used.

Examples

exampleland <- landscape.new.empty()
exampleland <- landscape.new.intparam(exampleland, s=2, h=2)
exampleland <- landscape.new.floatparam(exampleland)
exampleland <- landscape.new.switchparam(exampleland)

exampleland <- landscape.new.locus(exampleland, type=2, ploidy=2, mutationrate=0.001, numalleles=5, allelesize=100)

exampleland$loci

r(exampleland)

landscape.new.switchparam

*Create a set of boolean parameters*

Description

Create a set of boolean (1 or 0) parameters for a RmetaSim landscape.

Usage

```r
## must be called AFTER landscape.new.empty()
landscape.new.switchparam(rland, re=0, rd=0, mp=1, dd=0)
```

Arguments

- **rland**: skeleton of landscape object, required
- **re**: randepoch (default=0), 1=randomly pick a new epoch (from the epochs listed in the landscape) after an epoch completes, 0=epochs are chosen in order
- **rd**: randdemo (default=0), 1=randomly choose a demography (from the demographies listed in the landscape) for each subpopulation, 0=demographies are assigned in order
- **mp**: multip (default=1), 1=multiple paternity, 0=entire families from a single mating
- **dd**: density dependence. If dd=1, then two of each local demography matrix must be defined, the first set using new.local.demo with k=0 and representing demography at low density and again with k=1 for demography at high population density.
Examples

```r
## Defaults
exampleland <- landscape.new.empty()
exampleland <- landscape.new.switchparam(exampleland)
exampleland$switchparam

## Random epochs, random demographies, and no multiple paternity
exampleland <- landscape.new.empty()
exampleland <- landscape.new.switchparam(exampleland, re=1, rd=1, mp=0)
exampleland$switchparam

rm(exampleland)
```

---

`landscape.obs.het`  
*Calculate observed heterozygosity*

### Description

Calculate observed heterozygosity from a landscape

### Usage

```r
landscape.obs.het(Rland)
```

### Arguments

- `Rland`  
  the Rmetasim landscape object

### Value

A matrix with num loci columns and num populations rows. Each element reflects the observed heterozygosity for that population x locus combination

### See Also

`landscape.exp.het`, `landscape.Fst`

### Examples

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
obshet <- landscape.obs.het(exampleland)
rm(exampleland)
```
landscape.ploidy

Description

return a vector with the ploidy of each locus in the order they appear in the landscape

Usage

landscape.ploidy(Rland)

Arguments

Rland the Rmetasim landscape object

Value

vector

See Also

landscape.populations

Examples

eexampleland <- landscape.new.example()
landscape.ploidy(exampleland)
rm(exampleland)


landscape.populations

Description

return a vector of population IDs from a landscape

Usage

landscape.populations(Rland)

Arguments

Rland the Rmetasim landscape object
Details

Returns a vector of length \( \text{dim(rland$individuals)[1]} \) where \text{rland} is a landscape object. The vector classifies individuals into populations (or habitats).

Value

a vector

See Also

landscape.locus, landscape.ploidy

Examples

```r
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
plot(table(landscape.populations(exampleland)),
     main="Distribution of population size in landscape")
rm(exampleland)
```

```
landscape.sample simulates sampling for genetics on the landscape
```

Description

Randomly pulls a max of \text{ns} individuals from a max of \text{np} populations and returns a landscape object that could be used for further simulation, but is usually used for analyses and summary statistics calculations. If one needs a sample of specific populations/habitats, then these should be given in the vector \text{pvec}

Usage

```r
landscape.sample(Rland, np = NULL, ns=NULL, pvec=NULL, svec = NULL)
```

Arguments

- \texttt{Rland} landscape object
- \texttt{ns} number samples per population
- \texttt{np} number of populations to sample
- \texttt{pvec} a vector of populations to sample
- \texttt{svec} A vector of the stages sampled from a landscape. This provides good control over which populations and which stages within populations can be sampled.

Details

This function is intended to provide a lot of flexibility in sampling individuals from a rmetasim landscape.
landscape.simulate

Value
landscape object

Examples
1 <- landscape.new.example()
1 <- landscape.simulate(1,1)
1.samp <- landscape.sample(1, np=1, ns=24)
landscape.amova.pairwise(1.samp)
  1.samp2 <- landscape.sample(1, np=24, pvec=c(2))
landscape.amova.pairwise(1.samp2)

landscape.simulate Run a simulation for a single landscape through time

Description
Simulate a Rmetasim landscape for a number of generations.

Usage
landscape.simulate(Rland, numit, seed=-1, compress=FALSE, adj.lambda=0)

Arguments

Rland the Rmetasim landscape object
numit the number of generations/iterations to simulate, note that landscapes will not run past the rland$intparam$totalgens value
seed The default value of seed uses the seed set in the calling environment. Any other value for seed uses 'set.seed()' to reset the random number generator. landscape.simulate uses the RNG selected by the calling environment.
compress If true, landscape.simulate executes a survival and carrying capacity step before returning. In demographies with high reproductive potential, this can significantly reduce the size of R objects returned
adj.lambda Tries to apply a correction to population growth that makes the observed growth rate more closely approximate that predicted from standard analysis eigensystem of the sum of the survival and reproduction Lefkovitch matrices

Examples
exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
exampleland
rm(exampleland)
landscape.states

return a matrix containing actual genotypes for a particular locus

Description

return a matrix containing the states of the alleles in genotypes for a particular locus

Usage

landscape.states(Rland,lnum=1)

Arguments

lnum the locus to return
Rland the Rmetasim landscape object

Details

Returns a matrix with rows = dim(rland$individuals)[1]. The columns 1:landscape.democol() correspond to demographic variables for an individual. The columns are: state, placeholder, birthyear, id, mother's id, and father's id. Here rland is a landscape object. The remaining columns (1 if haploid, 2 if diploid) contain the states of the alleles for the selected loci

Value

matrix

See Also

landscape.locus

Examples

exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
print("Allele frequencies at locus 1")
table(landscape.states(exampleland,1)[,c(-1:-landscape.democol())])
rm(exampleland)
Calculate theta using heterozygosity

Description

Calculate theta from a landscape based upon heterozygosity.

Usage

landscape.theta.h(rland)

Arguments

rland the Rmetasim landscape object

Details

Uses routines in the package 'ape'

Value

A matrix with num loci columns and num populations rows. Each element reflects the estimated theta for that population x locus combination

See Also

landscape.theta.k, landscape.theta.s

Examples

exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)

theta.h.mat <- landscape.theta.h(exampleland)
theta.h.mat
### `landscape.theta.k`

*Calculate theta using the number of alleles*

**Description**

Calculate theta using number of alleles from a landscape.

**Usage**

```r
landscape.theta.k(rland)
```

**Arguments**

- `rland` the Rmetasim landscape object

**Details**

Uses routines in the package 'ape'

**Value**

A matrix with num loci columns and num populations rows. Each element reflects the estimated theta for that population x locus combination

**See Also**

`landscape.theta.h`, `landscape.theta.s`

**Examples**

```r
eempland <- landscape.new.example()
eempland <- landscape.simulate(eempland, 4)
theta.k.mat <- landscape.theta.k(eempland)
theta.k.mat
```

### `landscape.theta.s`

*Calculate theta using segregating sites*

**Description**

Calculate theta from a landscape based upon the number of segregating sites.

**Usage**

```r
landscape.theta.s(rland)
```
Arguments

rland the Rmetasim landscape object

Details

Uses routines in the package ‘ape’

Value

A matrix with num loci columns and num populations rows. Each element reflects the estimated theta for that population x locus combination

See Also

theta.k.landscape, theta.h.landscape

Examples

exampleland <- landscape.new.example()
exampleland <- landscape.simulate(exampleland, 4)
theta.s.mat <- landscape.theta.s(exampleland)
theta.s.mat

landscape.write.foreign

*Save a landscape to a file in a foreign format*

Description

Save a Rmetasim landscape object to a file in a suite of output formats. This function used to be supported with backend C++ functions. These functions have been removed from the codebase. Currently this function only supports writing genepop-format files to disk.

Usage

landscape.write.foreign(Rland, fn="foreign.genepop", fmt="genepop", ...)

Arguments

Rland the Rmetasim landscape object

fn the path and name of the file to save the landscape to

fmt the output format for the landscape: Can take the following values:"genepop"

... arguments passed to individual functions that actually do the work
Details

This is a wrapper for R-side functions that write landscape data to disk.

It is probably not necessary, because of the new landscape.make.genind() function. However, there is a single format implemented right now: GenePop. In addition to the parameters named above, a character string called 'title' can be passed to this function and it will give a title at the top of the resulting genepop file.

Former versions of this function had a "numi" parameter to specify how many individuals to sample per population. Now use landscape.sample() before running this one.

Value

None, run only for the side-effect of writing to disk

Examples

```r
## Needs write access to the current directory, files created!!
exampleland <- landscape.new.example()
landscape.write.foreign(exampleland, fn="exampleland.genepop", fmt="genepop")
rm(exampleland)
```

SimulationComponents  

Code components to simulate a landscape

Description

These functions can be used to construct custom simulations of landscapes. Each conducts only a single generations worth of change.

Usage

```r
landscape.advance(Rland, seed = -1)
landscape.carry(Rland, seed = -1)
landscape.extinct(Rland, seed = -1)
landscape.reproduce(Rland, seed = -1)
landscape.survive(Rland, seed = -1)
```

Arguments

- **Rland**: the Rmetasim landscape object
- **seed**: seed for random number generator

Details

`landscape.advance()` merely advances the generation counter and selects the new generations demographic conditions if such conditions can vary. The other functions implement carrying capacity, local extinction, reproduction, and survival/growth, respectively. The function `landscape.simulate()` bundles the functionality of these components into a single function (and executes it slightly faster all within linked C++ code).
See Also

landscape.simulate
Index

*Topic misc
  is.landscape, 2
  landscape.allelecount, 3
  landscape.allelefreq, 4
  landscape.amova, 4
  landscape.amova.locus, 5
  landscape.amova.pairwise, 6
  landscape.clean, 6
  landscape.coalinput, 7
  landscape.compress, 8
  landscape.democol, 9
  landscape.demography, 9
  landscape.exp.het, 10
  landscape.freq.locnames, 11
  landscape.Fst, 12
  landscape.ind.freq, 13
  landscape.locus, 14
  landscape.locus.states, 15
  landscape.locusvec, 15
  landscape.make.genind, 16
  landscape.make.genpop, 17
  landscape.mismatchdist, 19
  landscape.modify.epoch, 20
  landscape.new.epoch, 21
  landscape.new.example, 22
  landscape.new.floatparam, 23
  landscape.new.individuals, 23
  landscape.new.intparam, 25
  landscape.new.landscape, 26
  landscape.new.local.demo, 26
  landscape.new.locus, 27
  landscape.new.switchparam, 28
  landscape.obs.het, 29
  landscape.ploidy, 30
  landscape.populations, 30
  landscape.sample, 31
  landscape.simulate, 32
  landscape.states, 33
  landscape.theta.h, 34
  landscape.theta.k, 35
  landscape.theta.s, 35
  landscape.write.foreign, 36
  SimulationComponents, 37

is.landscape, 2

landscape.advance
  (SimulationComponents), 37
landscape.allelecount, 3
landscape.allelefreq, 4
landscape.amova, 4
landscape.amova.locus, 5, 5
landscape.amova.pairwise, 5, 6
landscape.carry (SimulationComponents), 37
landscape.clean, 6
landscape.coalinput, 7
landscape.compress, 8
landscape.democol, 9
landscape.demography, 9
landscape.exp.het, 10
landscape.extinct
  (SimulationComponents), 37
landscape.freq.locnames, 11
landscape.Fst, 12
landscape.ind.freq, 13
landscape.locus, 14
landscape.locus.states, 15
landscape.locusvec, 15
landscape.make.genind, 16
landscape.make.genpop, 17
landscape.mismatchdist, 19
landscape.modify.epoch, 20
landscape.new.empty
  (landscape.new.landscape), 26
landscape.new.epoch, 21
landscape.new.example, 22
landscape.new.floatparam, 23
landscape.new.individuals, 23
landscape.new.intparam, 25
landscape.new.landscape, 26
landscape.new.local.demo, 26
landscape.new.locus, 27
landscape.new.switchparam, 28
landscape.obs.het, 29
landscape.ploidy, 30
landscape.populations, 30
landscape.reproduce
  (SimulationComponents), 37
landscape.sample, 31
landscape.simulate, 32
landscape.states, 33
landscape.survive
  (SimulationComponents), 37
landscape.theta.h, 34
landscape.theta.k, 35
landscape.theta.s, 35
landscape.write.foreign, 36

SimulationComponents, 37