virtualPollen

March 26, 2019

Type Package

Title Generation of virtual pollen curves.

Version 0.1.0

Author Blas M. Benito

Maintainer Blas M. Benito <blasbenito@gmail.com>

Description Provides tools to simulate pollen curves of annual resolution based on virtual taxa with different traits (life-span, fecundity, growth-rate, niche position and niche breadth) by using a simple individual-based model. It also provides the mean to generate a virtual sediment accumulation rate to aggregate simulated pollen curves into centimetres, to represent taphonomical processes.

License What license is it under?

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

R topics documented:

acfToDf *Applies* acf *to a vector and returns a dataframe for pretty plotting in* ggplot2.

Description

It reads a vector representing a time series, applies acf for a given number of lags

Usage

```
acfToDf(
  x = NULL,
  lag.max = 100,
  length.out = 10
  )
```

Arguments

x	numeric vector.
lag.max	integer, number of lags over which to compute temporal autocorrelation.
length.out	integer, total number of lags to consider for plotting. Should be a subset of lag.max.

Details

This function computes temporal autocorrelation in a given vector using acf, and returns a dataframe ready for easy plotting in ggplot2.

Value

A dataframe with the columns: #'

- lag: numeric, lag in the time units of x with a maximum determined by lag.max, and a number of unique values determined by length.out
- acf: Pearson correlation index returned by the acf for a given number of lags for the given lag.
- ci.max: Maximum value of the confidence interval of acf.
- ci.min: Minimum value of the confidence interval of acf.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

acf

Examples

```
#simulating driver
x <- simulateDriver(
  random.seed = 10,
  time = 1:1000,
  autocorrelation.length = 200,
  output.min = -10,
  output.max = 20,
  rescale = TRUE
  )

#computing temporal autocorrelations
x.df <- acfToDf(
  x = x,
  lag.max = 300,
  length.out = 100
  )
str(x.df)

#plotting output
plotAcf(x.df)
```

---

aggregateSimulation *Aggregates the output of* simulatePopulation.

Description

Takes the output of simulatePopulation and aggregates it into centimetres

Usage

```
aggregateSimulationn(
  simulation.output=NULL,
  accumulation.rate=NULL,
  sampling.intervals=1
  )
```

Arguments

simulation.output	list, output of simulatePopulation.
accumulation.rate	dataframe, output of simulateAccumulationRate.
sampling.intervals	numeric or numeric vector, in centimetres, depth interval or intervals between consecutive samples. If 1, all samples are returned, if 2, returned samples are separated by 1 cm.

Details

The function uses the values in the grouping column of the simulateAccumulationRate output to aggregate together (by computing the mean) as many samples as cases in grouping have the same identificator. Output samples are identified by the average age of the samples within the given centimetre.

Value

A list of dataframes with as many rows as virtual taxa were produced by simulatePopulation, and several columns. First column is the original data. Second column is the data aggregated into centimetres according to the result of simulateAccumulationRate. Each additional column is the data of the second column resampled as given by the sampling.intervals argument. See example below for more clarity.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulateAccumulationRate, simulatePopulation

Examples

```
#generating driver
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#simulating population dynamics
sim.output <- simulatePopulation(
 parameters=parameters,
 driver.A=driver,
 driver.B=NULL
 )

#generating accumulation rate
acc.rate <- simulateAccumulationRate(
 seed=50,
 time=1:1000,
 output.min=10,
```

```
 output.max=40,
 direction=1,
 plot=TRUE
 )

#aggregating simulated data
sim.output.aggregated <- aggregateSimulation(
 simulation.output=sim.output,
 accumulation.rate=acc.rate,
 sampling.intervals=3)

#comparing simulations
par(mfrow=c(3,1))
#notice the subsetting of the given column of the input list
plot(sim.output.aggregated[[1,1]]$Time,
 sim.output.aggregated[[1,1]]$Pollen,
 type="l",
 xlim = c(0, 1000),
 main="Annual"
 )
plot(sim.output.aggregated[[1,2]]$Time,
 sim.output.aggregated[[1,2]]$Pollen,
 type="l",
 xlim = c(0, 1000),
 main="1cm"
 )
plot(sim.output.aggregated[[1,3]]$Time,
 sim.output.aggregated[[1,3]]$Pollen,
 type="l",
 xlim = c(0, 1000),
 main="3cm"
 )

#check differences in nrow
nrow(sim.output.aggregated[[1,1]]) #original data
nrow(sim.output.aggregated[[1,2]]) #1cm
nrow(sim.output.aggregated[[1,3]]) #3cm intervals
```

`compareSimulations` *Compares different simulations produced by* `simulatePopulation`.

Description

Plots together the results of different taxa produced by a single run of `simulatePopulation`.

Usage

```
compareSimulations(
  simulation.output=NULL,
```

```
  species="all",
  filename=NULL,
  columns="Pollen",
  time.zoom=NULL,
  width=12,
  text.size=20,
  title.size=25,
  plot.title=""
  )
```

Arguments

simulation.output	output of simulatePopulation.
species	a number or vector or numbers representing rows in the parameters dataframe, or a string or vector of strings referencing to the "label" column of the parameters dataframe.
filename	character string, name of output pdf file. If NULL or empty, no pdf is produced.
columns	character string or vector of character strings with these possible values: "Pollen", "Population.mature", "Population.immature", "Population.viable.seeds", "Suitability", "Biomass.total", "Biomass.mature", "Biomass.immature", "Mortality.mature", "Mortality.immature", "Driver.A", "Driver.B".
time.zoom	vector of two numbers indicating the beginnign and end of the time interval to be plotted (i.e. "c(5000, 10000)")
width	plot width in inches.
text.size	text size of the plot.
title.size	plot title size.
plot.title	character string to use as plot title.
burnin	if FALSE, burn-in period is not considered in the model.

Details

The user can decide what virtual taxa to plot (argument species), and what information to show throught the columns argument. Output is plotted on screen by default, and printed to pdf if the filename argument is filled.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulatePopulation, plotSimulation

Examples

```
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters[2,] <- c("Species 1", 500, 100, 10, 0.02, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#simulating population dynamics
sim.output <- simulatePopulation(
 parameters=parameters,
 driver.A=driver
 )

#plot simulation
compareSimulations(simulation.output = sim.output)
```

`fixParametersTypes` *Fix data types in parameters dataframe.*

Description

It converts all columns (but the `label` one) of a parameters dataframe created by `parametersDataframe` into type numeric.

Usage

```
parametersDataframe(x)
```

Arguments

`x`	dataframe resulting from `parametersDataframe`.

Value

Same dataframe provided in argument `x` but with fixed data types.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

parametersDataframe

Examples

```
parameters <- parametersDataframe(rows=1)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)
```

---

parametersCheck *Plots main simulation parameters.*

Description

Plots the normal function/s, fecundity, growth curve, and maturity age, of each virtual taxa in parameters.

Usage

```
parametersCheck(
  parameters,
  species="all",
  driver.A=NULL,
  driver.B=NULL,
  drivers=NULL,
  filename=NULL
  )
```

Arguments

parameters	the parameters dataframe.
species	if "all" or "ALL", all species in "parameters" are plotted. It also accepts a vector of numbers representing the rows of the selected species, or a vector of names of the selected species.
driver.A	numeric vector with driver values.
driver.B	numeric vector with driver values.
drivers	dataframe with drivers
filename	character string, filename of the output pdf.

Details

The function prints the plot, can save it to a pdf file if filename is provided, and returns a ggplot2 object.

Value

A ggplot2 object.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

parametersDataframe, fixParametersTypes

Examples

```
#generating driver
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters[2,] <- c("Species 1", 500, 100, 10, 0.02, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#plotting parameters
parametersCheck(
 parameters=parameters,
 driver.A=driver,
 filename="Parameters.pdf"
 )
```

---

parametersDataframe *Generates a template dataframe to contain simulation parameters.*

Description

Generates the dataframe structure needed to contain the parameters used as input for the simulatePopulation function.

Usage

```
parametersDataframe(rows = 1)
```

Arguments

rows	integer, number of rows in the output dataframe.

Details

The resulting dataframe can either be filled manually through vectors, as shown in the example (but this requires to use the function fixParametersTypes once the dataframe is completed), or can be edited manually in Rstudio by installing the editData package.

Value

A dataframe filled with NA values and the columns:

- label: to store names (character string) of the virtual taxa.
- maximum.age: integer, maximum possible age of the individuals in years.
- reproductive.age: integer, age of sexual maturity in years.
- fecundity: integer, number of maximum viable seeds produced by a mature individual under fully suitable conditions.
- growth.rate: numeric, parameter of the logistic growth function.
- pollen.control: numeric in the interval [0, 1]. If 0, pollen productivity depends on environmental suitability only. The larger the number, biomass takes over environmental suitability in determining pollen productivity.
- maximum.biomass: integer, maximum biomass of the individuals.
- carrying.capacity: integer, maximum sum of biomass of the individuals. Very large carrying capacities plus a low maximum.biomass generates too many individuals for the simulation to remain efficient. Try to set carrying.capacity and maximum.biomass to carrying.capacity divided by biomass returns a number lower than 1000 (and even better if it is closer to 100).
- driver.A.weight: numeric in the interval [0, 1], represents the relative influence of the driver on environmental suitability.
- driver.B.weight: numeric in the interval [0, 1], represents the relative influence of the driver on environmental suitability. The sum of weights of drivers A and B should be 1.
- niche.A.mean: numeric, in the same units as driver A. It is the mean of the normal function defining the response of the virtual taxa to driver A.
- niche.A.sd: numeric, in the same units as driver A. It is the standard deviation of the normal function defining the response of the virtual taxa to driver A.
- niche.B.mean: as above, but for driver B.
- niche.B.sd: as above, but for driver B.
- autocorrelation.length.A: numeric, only useful if several drivers generated with different autocorrelation lengths are available (and identified by the column autocorrelation.length) in the drivers argument provided to the simulatePopulation function.
- autocorrelation.length.B: same as above.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulatePopulation, fixParametersTypes

Examples

```
parameters <- parametersDataframe(rows=1)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
```

---

plotAcf *Plots results of* acfToDf.

Description

Plots a dataframe resulting from acfToDf by using ggplot2 (and cowplot)

Usage

```
plotAcf(
  x = NULL,
  plot.title = ""
  )
```

Arguments

x	dataframe, output of acfToDf
plot.title	string, title for the plot.

Value

A ggplot object

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

acfToDf

Examples

```
#simulating driver
x <- simulateDriver(
  random.seed = 10,
  time = 1:1000,
  autocorrelation.length = 200,
  output.min = -10,
  output.max = 20,
  rescale = TRUE
  )

#computing temporal autocorrelation
x.df <- acfToDf(
  x = x,
  lag.max = 300,
  length.out = 100
  )

#plotting output
plotAcf(x.df)
```

---

plotSimulation *Plots results of* simulatePopulation.

Description

This function takes as input a list of dataframes or a single dataframe resulting from the execution of simulatePopulation, and plots the resulting time series of pollen abundance, number of individuals, biomass, driver, and environmnetal suitability.

Usage

```
plotSimulation(
  simulation.output=NULL,
  species="all",
  burnin=FALSE,
  filename=NULL,
  time.zoom=NULL,
  panels=c("Driver A",
           "Driver B",
           "Suitability",
           "Population",
           "Mortality",
           "Biomass",
           "Pollen"
           ),
```

```
 plot.title=NULL,
 width=12,
 text.size=20,
 title.size=25,
 line.size=1
 )
```

Arguments

simulation.output	output of simulatePopulation.
species	a number or vector or numbers representing rows in the parameters dataframe, or a string or vector of strings referencing to the "label" column of the parameters dataframe.
burnin	if FALSE, burn-in period is not considered in the model.
filename	character string, name of output pdf file. If NULL or empty, no pdf is produced.
time.zoom	vector of two numbers indicating the beginnign and end of the time interval to be plotted (i.e. "c(5000, 10000)")
panels	character string or vector of character strings with these possible values: "Driver A", "Driver B","Suitability", "Population", "Mortality", "Biomass", "Pollen".
plot.title	character string to use as plot title.
width	plot width in inches.
text.size	text size of the plot.
title.size	plot title size.

Details

The user can decide what virtual taxa to plot (argument species), and what information to show throught the panels argument. Output is plotted on screen by default, and printed to pdf if the filename argument is filled.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulatePopulation, compareSimulations

Examples

```
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
```

```
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters[2,] <- c("Species 1", 500, 100, 10, 0.02, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#simulating population dynamics
sim.output <- simulatePopulation(
 parameters=parameters,
 driver.A=driver
 )

#plot simulation
plotSimulation(simulation.output = sim.output)
```

`rescaleVector` *Rescales a vector within given bounds.*

Description

Takes a numeric vector `x` and rescales it within the values given by `new.min` and `new.max`.

Usage

```
rescaleVector(
  x = rnorm(100),
  new.min = 0,
  new.max = 100,
  integer = FALSE
  )
```

Arguments

`x`	numeric vector to be rescaled.
`new.min`	numeric, new minimum value for `x`. Default is 0.
`new.max`	numeric, new maximum value for `x`. Default is 100.
`integer`	boolean, if TRUE, output vector is returned as vector of integers. Default is FALSE.

Value

A vector of the same length as `x` rescaled between `output.min` and `output.max`.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

Examples

```
#generating example data
x = rnorm(100)

#as float
x.float <- rescaleVector(
  x = x,
  new.min = 0,
  new.max = 100,
  integer = FALSE
  )

#as integer
x.integer <- rescaleVector(
  x = x,
  new.min = 0,
  new.max = 100,
  integer = TRUE
  )
```

---

simulateAccumulationRate

Simulates a virtual sediment accumulation rate.

Description

Generates a virtual sediment accumulation rate to be applied to the results of `simulatePopulation`

Usage

```
simulateAccumulationRate(
  seed=50,
  time=1:1000,
  output.min=10,
  output.max=40,
  direction=1,
  plot=TRUE
  )
```

Arguments

seed	integer, seed to be used by `set.seed` to configure the state of the pseudo-random number generator. It defines the shape of the curve.

time	vector of time values (ideally the same used to generate the simulations).
output.min	numeric, in years per centimetre, minimum sediment accumulation rate (10 by default).
output.max	numeric, in years per centimetre, maximum sediment accumulation rate (40 bu default).
direction	integer, values 1 or -1, to invert the resulting accumulation rate.
plot	boolean, plots output accumulation rate if TRUE.

Details

The accumulation rate curve is generated through a random walk smoothed by a GAM model. The value of the seed argument changes the shape of the curve, but the user has no more control than trying different values to achieve a curve closer to the desired one. If plot is set to TRUE, the accumulation rate curve is printed on screen, but not exported to pdf.

Value

A dataframe with the following columns.

- *time* numeric, time or age of the given case.
- *accumulation.rate* numeric, in years per centimetre, simulated accumulation rate.
- *grouping* integer, grouping variable to aggregate together with aggregateSimulation samples deposited in the same centimetre according *accumulation.rate*.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

aggregateSimulation

Examples

```
acc.rate <- simulateAccumulationRate(
 seed=50,
 time=1:1000,
 output.min=10,
 output.max=40,
 direction=1,
 plot=TRUE
 )

str(acc.rate)
```

simulateDriver *Generates a random time series with temporal autocorrelation.*

Description

Generates a vector of the same legnth as the `time` argument, with a temporal autocorrelation length close to the defined by `autocorrelation.length`, and a range within `output.min` and `output.max`. The output of this function is intended to be used as an input to the function `rescaleVector`.

Usage

```
simulateDriver(
  random.seed = 50,
  time = 1:1000,
  autocorrelation.length = 100,
  output.min = 0,
  output.max = 100,
  rescale = TRUE
  )
```

Arguments

`random.seed`	integer, seed to be used by `set.seed()`. Default is 50.
`time`	integer, or numeric vector of integers with constant intervals. If a single integer is provided, a time sequence is generated from 0 to the given integer as *seq(0, time, by = 1)*. Default is 1:1000.
`autocorrelation.length`	integer, represents the length of the convolution filter to be used to impose a particular temporal structure on the time series. Default is 100, equivalent to a filter composed by a hundred of ones.
`output.min`	numeric, minimum value of the output time series. Used as input for **rescaleVector**. Default is 0.
`output.max`	numeric, maximum value of the output time series. Used as input for **rescaleVector**. Default is 100.
`rescale`	boolean. If FALSE, `output.min` and `output.max` are ignored, and the original data range provided by `rnorm` is preserved. Default is TRUE.

Details

It is recommended to use `time` vectors with a time step of 1 between consecutive values when the output is to be used as input for `simulatePopulation`, which considers annual time-steps while simulating virtual pollen curves. Initial random sequence is generated by `rnorm`. Desired temporal autocorrelation are approximate, but deviation becomes higher if `autocorrelation.length` is larger than half the length of `time`. Consequently, the function limits `autocorrelation.length` to `length(time)/2`.

Value

A vector of the same length as `time`.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

`rescaleVector`

Examples

```
x <- simulateDriver(
  random.seed = 10,
  time = 1:1000,
  autocorrelation.length = 200,
  output.min = -10,
  output.max = 20,
  rescale = TRUE
  )

#plots output
plot(x, type="l")

#checks temporal autocorrelation
acf(x, lag.max = 300)
```

---

simulatePopulation *Simulates population dynamics for virtual species with different traits.*

Description

This function takes as input a dataframe of parameters defining virtual taxa produced by `parametersDataframe` and `fixParametersTypes`, a driver or drivers generated with `simulateDriver`, and simulates population dynamics at yearly resolution for the time-length defined by the driver or drivers. The model relies on the following set of assumptions.

- The spatial structure of the population is not important to explain its pollen productivity. This is an operative assumption, to speed-up model execution.
- The environmental niche of the species follows a Gaussian distribution, characterized by a mean (niche optimum, also niche position) and a standard deviation (niche breadth or tolerance).
- Different drivers can have a different influence on the species dynamics, and that influence can be defined by the user by tuning the weights of each driver.

- Environmental suitability, expressed in the range [0, 1], is the result of an additive function of the species niches (normal function defined by the species' mean and standard deviation for each driver), the drivers' values, and the relative influence of each driver (driver weights).
- Pollen productivity is a function of the individual's biomass and environmental suitability, so under a hypothetical constant individual's biomass, its pollen production depends linearly on environmental suitability values.
- Effective fecundity is limited by environmental suitability. Low environmental suitability values limit recruitment, acting as an environmental filter. Therefore, even though the fecundity of the individuals is fixed by the fecundity parameter, the overall population fecundity is limited by environmental suitability.

Usage

```
simulatePopulation(
  parameters=NULL,
  species="all",
  driver.A=NULL,
  driver.B=NULL,
  drivers=NULL,
  burnin=TRUE
  )
```

Arguments

parameters	dataframe with parameters.
species	if "all" or "ALL", all species in "parameters" are simulated It also accepts a vector of numbers representing the rows of the selected species, or a vector of names of the selected species.
driver.A	numeric vector with driver values.
driver.B	numeric vector with driver values.
drivers	dataframe with drivers.
burnin	boolean, generates a warming-up period for the population model of a length of five times the maximum age of the virtual taxa.

Details

The model starts with a population of 100 individuals with random ages, in the range [1, maximum age], taken from a uniform distribution (all ages are equiprobable). For each environmental suitability value, including the burn-in period, the model performs the following operations:

- Aging: adds one year to the age of the individuals.
- Mortality due to senescence: individuals reaching the maximum age are removed from the simulation.
- Local extinction and immigration If the number of individuals drops to zero, the population is replaced by a "seed bank" of #' 100 individuals with age zero, and the simulation jumps to step 7.. This is intended to simulate the arrival of seeds from nearby regions, and will only lead to population growth if environmental suitability is higher than zero.

- Plant growth: Applies a plant growth equation to compute the biomass of every individual.
- Carrying capacity: If maximum population biomass is reached, individuals are iteratively selected for removal according to a mortality risk curve computed by the equation $P_m = 1 - sqrt(a/A)$, were *Pm* is the probability of mortality, *a* is the age of the given individual, and *A* is the maximum age reached by the virtual taxa. This curve gives removal preference to younger individuals, matching observed patterns in natural populations.
- Pollen productivity: In each time step the model computes the pollen productivity (in relative values) of the population using the equation $P_t = \sum x_{it} \times max(S_t, B)$, where *t* is time (a given simulation time step), *P* is the pollen productivity of the population at a given time, *x_i* represents the biomass of every adult individual, *S* is the environmental suitability at the given time, *B* is the contribution of biomass to pollen productivity regardless of environmental suitability (*pollen.control* parameter in the simulation, 0 by default). If *B* equals 1, *P* is equal to the total biomass sum of the adult population, regardless of the environmental suitability. If *B* equals 0, pollen productivity depends entirely on environmental suitability values.
- Reproduction: Generates as many seeds as reproductive individuals are available multiplied by the maximum fecundity and the environmental suitability of the given time.

The model returns a table with climatic suitability, pollen production, and population size (reproductive individuals only) per simulation year. Figure 10 shows the results of the population model when applied to the example virtual species.

Value

A list of dataframes, each one of them with the results of one simulation. Each dataframe has the columns:

- *Time* integer, ages in years. Negative ages indicate the burn-in period.
- *Pollen* numeric, pollen counts
- *Population.mature* numeric, number of mature individuals.
- *Population.immatre* numeric, number of immature individuals.
- *Population.viable.seeds* numeric, number of viable seeds generated each year.
- *Suitability* numeric, environmental suitability computed from the driver by the normal function/s defining the taxon niche.
- *Biomass.total* numeric, overall biomass of the population.
- *Biomass.mature* numeric, sum of biomass of mature individuals.
- *Biomass.immature* numeric, sum of biomass of immature individuals.
- *Mortality.mature* numeric, number of mature individuals dead each year.
- *Mortality.immature* numeric, same as above for immature individuals.
- *Driver.A* numeric, values of driver A.
- *Driver.B* numeric, values of driver B, if available, and NA otherwise.
- *Period* qualitative, with value "Burn-in" for burn-in period, and "Simulation" otherwise.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

parametersDataframe, fixParametersTypes, plotSimulation

Examples

```
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters[2,] <- c("Species 1", 500, 100, 10, 0.02, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#simulating population dynamics
sim.output <- simulatePopulation(
 parameters=parameters,
 driver.A=driver
 )

#checking output for Species 1
str(sim.output[[1]])
```

---

toRegularTime *Reinterpolates aggregated simulations into regular time.*

Description

Takes the output of aggregateSimulation, and interpolates it into a regular time grid.

Usage

```
toRegularTime(
  x,
  time.column="Time",
  interpolation.interval,
  columns.to.interpolate=c("Suitability", "Driver.A", "Pollen")
  )
```

Arguments

x	list, output of aggregateSimulation.
time.column	character string, default value is "Time".
interpolation.interval	integer, in years, time length encompassed by each sample.
columns.to.interpolate	character string or character vector, columns of simulation output to be interpolated. Any subset of: "Pollen", "Population.mature", "Population.immature", "Population.viable.seeds", "Suitability", "Biomass.total", "Biomass.mature", "Biomass.immature", "Mortality.mature", "Mortality.immature", "Driver.A", "Driver.B".

Details

The function takes the input list, and on each dataframe it fits a loess model of the form `y ~ x`, where `y` is any column given by `columns.to.interpolate` and `x` is the column given by the `time.column` argument. The model is used to interpolate column `y` on a regular time series of intervals equal to `interpolation.interval`.

Value

A list of dataframes with the same structure as the input list. Each dataframe has the columns "Time" (now regular), and any column listed in `columns.to.interpolate`.

Author(s)

Blas M. Benito <blasbenito@gmail.com>

See Also

simulateAccumulationRate, aggregateSimulation

Examples

```
#generating driver
driver <- simulateDriver(
 random.seed = 10,
 time = 1:1000,
 autocorrelation.length = 200,
 output.min = 0,
 output.max = 100,
 rescale = TRUE
 )

#preparing parameters
parameters <- parametersDataframe(rows=2)
parameters[1,] <- c("Species 1", 50, 20, 2, 0.2, 0, 100, 1000, 1, 0, 50, 10, 0, 0, NA, NA)
parameters <- fixParametersTypes(x=parameters)

#simulating population dynamics
sim.output <- simulatePopulation(
```

```
 parameters=parameters,
 driver.A=driver,
 driver.B=NULL
 )

#generating accumulation rate
acc.rate <- simulateAccumulationRate(
 seed=50,
 time=1:1000,
 output.min=10,
 output.max=40,
 direction=1,
 plot=TRUE
 )

#aggregating simulated data
sim.output.aggregated <- aggregateSimulation(
 simulation.output=sim.output,
 accumulation.rate=acc.rate,
 sampling.intervals=3
 )

#comparing simulations
sim.output.regular <- toRegularTime(
 x=sim.output.aggregated,
 time.column="Time",
 interpolation.interval=20,
 columns.to.interpolate=c("Driver.A", "Pollen")
 )
```

Index