Package: colorednoise (via r-universe)

October 26, 2024

Type Package

Title Simulate Temporally Autocorrelated Populations

Version 1.1.2

Date 2024-02-23

Maintainer July Pilowsky <pilowskyj@caryinstitute.org>

Description Temporally autocorrelated populations are correlated in their vital rates (growth, death, etc.) from year to year. It is very common for populations, whether they be bacteria, plants, or humans, to be temporally autocorrelated. This poses a challenge for stochastic population modeling, because a temporally correlated population will behave differently from an uncorrelated one. This package provides tools for simulating populations with white noise (no temporal autocorrelation), red noise (positive temporal autocorrelation), and blue noise (negative temporal autocorrelation). The algebraic formulation for autocorrelated noise comes from Ruokolainen et al. (2009) <doi:10.1016/j.tree.2009.04.009>. Models for unstructured populations and for structured populations (matrix models) are available.

License GPL-3

Language en-US

Depends R (>= 3.3.0)

Imports stats (>= 3.3.2), purrr (>= 0.2.3), Rcpp (>= 1.0.5), data.table (>= 1.12.8)

LinkingTo Rcpp, RcppArmadillo

RoxygenNote 7.3.1

Encoding UTF-8

BugReports https://github.com/japilo/colorednoise/issues

Suggests ggplot2 (>= 2.2.1), knitr (>= 1.17), rmarkdown (>= 1.6), testthat (>= 1.0.2), covr (>= 3.0.0), pkgdown (>= 1.1.0)

VignetteBuilder knitr

NeedsCompilation yes

Author July Pilowsky [aut, cre]
(<https://orcid.org/0000-0002-6376-2585>)

Repository CRAN

Date/Publication 2024-02-28 21:50:02 UTC

Contents

`autocorrelation` *Estimate the Temporal Autocorrelation of a Numeric Vector*

Description

A wrapper for the `acf` function from the stats package that extracts only the temporal autocorrelation at a lag of one timestep (which is the type of temporal autocorrelation that this package simulates). The function omits NA values in the time series.

Usage

```
autocorrelation(x, biasCorrection = TRUE)
```

Arguments

`x`	A numeric vector.
`biasCorrection`	Autocorrelation estimates are biased for short time series. The function can correct for this bias in the manner proposed by Quenouille (1949). Set to TRUE by default.

Value

A single numeric value: the estimate of the temporal autocorrelation with a lag of 1.

Examples

```
rednoise <- colored_noise(timesteps = 50, mean = 0.5, sd = 0.2, phi = 0.3)
autocorrelation(rednoise)
```

`autocorr_sim` *Simulate Temporally Autocorrelated Populations for Every Combination of Parameters*

Description

Essentially a loop of `unstructured_pop`, this function simulates a population with temporally autocorrelated vital rates for every combination of parameters you specify, with as many replicates as desired. It also estimates the sample mean survival and fertility for each simulated population. Please be advised that this function can be very computationally intensive if you provide many possible parameter values and/or ask for many replicates.

Usage

```
autocorr_sim(
  timesteps,
  start,
  survPhi,
  fecundPhi,
  survMean,
  survSd,
  fecundMean,
  fecundSd,
  replicates
)
```

Arguments

`timesteps`	The number of timesteps you want to simulate. Individuals are added and killed off every timestep according to the survival and fertility rates. Can be a scalar or a vector of values to loop over.
`start`	The starting population size. Can be a scalar or vector.
`survPhi`	The temporal autocorrelation of survival. 0 is white noise (uncorrelated), positive values are red noise (directly correlated) and negative values are blue noise (inversely correlated). Can be a scalar or a vector.
`fecundPhi`	The temporal autocorrelation of fecundity. As above.
`survMean`	The mean survival from timestep to timestep. Must be a value between 0 (all individuals die) and 1 (all individuals live). Can be a scalar or a vector.
`survSd`	The standard deviation of the survival from timestep to timestep. Must be a value between 0 and 1. Can be a scalar or a vector.
`fecundMean`	The mean fertility: mean offspring produced by each individual per timestep. Can be a scalar or a vector.
`fecundSd`	The standard deviation of the fertility. Can be a scalar or a vector of values.
`replicates`	How many replicates you would like of each possible combination of parameters.

Value

A list of data frames, each with fourteen variables: timestep, newborns (new individuals added this timestep), survivors (individuals alive last year who survived this timestep), population (total individuals alive), growth (the increase or decrease in population size from last year), estimated survival in the timestep, estimated fecundity in the timestep, and the seven parameters used to generate the simulation.

Examples

```
survival_range <- autocorr_sim(timesteps = 30, start = 200, survPhi = 0.3, fecundPhi = 0.1,
                               survMean = c(0.2, 0.3, 0.4, 0.5, 0.6), survSd = 0.5,
                               fecundMean = 1.1, fecundSd = 0.5, replicates = 50)
head(survival_range[[1]])
```

---

colored_multi_rnorm *Generate Multiple Cross-Correlated & Autocorrelated Variables*

Description

Generates random variables that are correlated to each other and temporally autocorrelated.

Usage

```
colored_multi_rnorm(timesteps, mean, sd, phi, covMatrix)
```

Arguments

timesteps	The number of temporally autocorrelated random numbers (one per timestep) you want.
mean	A vector giving the mean of each variable.
sd	A vector giving the standard deviation of each variable.
phi	A vector giving the temporal autocorrelation of each variable.
covMatrix	A valid covariance matrix. The number of rows/columns must match the length of the mu, sigma, and phi vectors.

Value

A matrix with as many rows as timesteps and as many columns as mu/sigma/phi values.

Examples

```
cov <- matrix(c(1, 0.53, 0.73, 0.53, 1, 0.44, 0.73, 0.44, 1), nrow = 3)
test <- colored_multi_rnorm(100, c(0, 3, 5), c(1, 0.5, 1), c(0.5, -0.3, 0), cov)
var(test)
library(data.table)
as.data.table(test)[, .(V1_mean = mean(V1), V2_mean = mean(V2), V3_mean = mean(V3),
V1_sd = sd(V1), V2_sd = sd(V2), V3_sd = sd(V3),
V1_autocorrelation = autocorrelation(V1), V2_autocorrelation = autocorrelation(V2),
V3_autocorrelation = autocorrelation(V3))]
```

colored_noise *Generate Autocorrelated Noise*

Description

Generates temporally autocorrelated random numbers with a mean, standard deviation, and autocorrelation you specify.

Usage

```
colored_noise(timesteps, mean, sd, phi)
```

Arguments

timesteps	The number of temporally autocorrelated random numbers (one per timestep) you want.
mean	The mean of the temporally autocorrelated random numbers.
sd	The standard deviation of the temporally autocorrelated random numbers.
phi	The temporal autocorrelation. 0 is white noise (uncorrelated), positive values are red noise (directly correlated) and negative values are blue noise (inversely correlated).

Value

A vector of temporally autocorrelated random numbers.

Examples

```
rednoise <- colored_noise(timesteps = 30, mean = 0.5, sd = 0.2, phi = 0.3)
rednoise
```

cor2cov *Convert from Correlation Matrix to Covariance Matrix*

Description

Convert a correlation matrix to a covariance matrix.

Usage

```
cor2cov(sigma, corrMatrix)
```

Arguments

sigma	A vector of standard deviations for the variables you're describing. Length must be the same as the number of rows/columns of CorrMatrix.
corrMatrix	A valid correlation matrix.

Value

A covariance matrix with the same dimensions as corrMatrix.

Examples

```
corr <- matrix(c(1, 0.53, 0.73, 0.53, 1, 0.44, 0.73, 0.44, 1), nrow = 3)
sigmas <- c(2, 0.3, 1.2)
covar <- cor2cov(sigmas, corr)
cov2cor(covar)
```

---

matrix_model *Temporally Autocorrelated Matrix Population Models*

Description

Simulate a structured population with temporal autocorrelation using standard Leslie matrices. Each element in the Leslie matrix has a specified mean, variance, and temporal autocorrelation value. The matrix can have arbitrary dimensions and can have transitions besides linear survival. This model includes environmental stochasticity with colored noise. Density dependence and demographic stochasticity not currently supported.

Usage

```
matrix_model(
  data,
  initialPop,
  timesteps,
  covMatrix = NULL,
  colNames = NULL,
  matrixStructure = NULL,
  repeatElements = NULL,
  survivalOverflow = "scale"
)
```

Arguments

data	The input data can be one of two formats: a list of three matrices, or a data frame with three columns. If it is a list of three matrices, they must be standard Leslie matrices: the first a matrix of mean values for each matrix element, the second a matrix of standard deviations, and the third a matrix of temporal autocorrelations.

If it is a data frame, there must be three columns, one for mean vital rates, one for standard deviations, and one labeled 'autocorrelation.'
If the population has n stages, the first n rows of the data frame must be the matrix elements for the first stage, and the next n*(1-n) rows must be the transition probabilities, each row of the matrix from first to last transposed vertically.
If you want to run a matrix population model without temporal autocorrelation, simply set all autocorrelation values to zero.

`initialPop` An initial population vector. The length must be the same as the number of classes in the matrices.

`timesteps` The number of timesteps you would like to simulate the population.

`covMatrix` Optional: Add a covariance matrix describing within-year covariances between matrix elements. The matrix elements must be in the same order as they are in the data frame format above: a Leslie matrix turned into a vector row-wise. There should be as many columns as matrix elements, excluding repeat elements (see below) or structural zeros.

`colNames` Optional: If the mean, sd, and autocorrelation columns of your data frame input are not named 'mean', 'sd', and 'autocorrelation', provide their names here in a character vector, e.g., 'c(mean = 'Mean', sd = 'Standard Deviation', autocorrelation = 'phi')'

`matrixStructure`
Optional: By default, the function assumes that the first row of the matrix gives fecundities while the rest of the matrix gives transition or survival probabilities. However, these assumptions do not apply to many plant matrices. If your matrix has transition probabilities in the first row or fecundities beyond the first row (e.g., clonal reproduction), provide a character matrix here with the same dimensions as your matrix that gives in strings whether each element is 'fecundity' or 'transition'.

`repeatElements` Optional: Sometimes not all matrix elements can be measured, and some transitions or fertilities are generalized across classes. If you have any matrix elements that are copies of other matrix elements (e.g., stage 3 is assumed to have the same fertility as stage 4) indicate them here with a matrix of *rowwise* (not column-wise) indices that show which elements are repeats and which are unique. For example in a 2x2 matrix where both classes are assumed to have the same fertility, input 'matrix(c(1, 1, 3, 4), byrow = T, ncol = 2)'. If you indicate repeat elements and you include a covariance matrix, the covariance matrix must only have as many columns as *unique matrix elements*. Structural zeros should *not* be included here as repeats, as they are automatically detected in the function.

`survivalOverflow`
If the survival for a stage is very high or very variable, the function may sometimes generate projection matrices with survival that exceeds 1 for that stage. The function has two methods of dealing with this problem: either discard all projection matrices and generate new ones until the survival falls within acceptable bounds ("redraw") or divide all the non-fertility matrix elements for that stage by the survival such that they add to 1 ("scale"). The default is "scale".

Value

A data frame with n + 2 columns, where n is the number of stages in the matrix. One column indicates the timestep, there is one column with the population size for each stage, and one column for total population size.

Examples

```
meanMat <- matrix(c(0.55, 0.6, 0.24, 0.4), byrow = TRUE, ncol = 2)
sdMat <- matrix(c(0.3, 0.35, 0.05, 0.1), byrow = TRUE, ncol = 2)
phiMat <- matrix(c(-0.2, -0.2, 0, 0), byrow = TRUE, ncol = 2)
initialPop <- c(100, 100)
sim <- matrix_model(list(meanMat, sdMat, phiMat), initialPop, 50)
head(sim)
```

---

multi_rnorm *Generate Correlated Normal Random Numbers*

Description

Generate random numbers from a multivariate normal distribution. It can be used to create correlated random numbers.

Usage

```
multi_rnorm(n, mean, sd)
```

Arguments

n	The number of samples desired for each variable.
mean	A vector giving the mean of each variable.
sd	A valid covariance matrix.

Value

A matrix with n rows and as many columns as mean values.

Examples

```
mus <- c(0, 3, 5)
sigmas <- matrix(c(1, 0.265, 2.19, 0.265, 0.25, 0.66, 2.19, 0.66, 9), ncol = 3)
mat <- multi_rnorm(100, mus, sigmas)
var(mat)
```

stdev_transform — *Translate Standard Deviation from the Natural Scale to the Log or Logit Scale*

Description

This function changes a given standard deviation so that when a vector of samples is drawn from the given distribution, the original standard deviation will be recovered once it is back-transformed from the log or logit scale. In effect, the function "translates" a standard deviation from the natural scale to the log or logit scale for the purposes of random draws from a probability distribution.

Usage

```
stdev_transform(mu, sigma, dist)
```

Arguments

mu	The mean of the distribution on the natural scale.
sigma	The standard deviation of the distribution on the natural scale.
dist	The distribution to which the standard deviation should be transformed.

Value

The standard deviation translated to the log or logit scale.

Examples

```
mean <- 10
stdev <- 2
mean_trans <- log(mean)
stdev_trans <- stdev_transform(mean, stdev, "log")
draws <- rnorm(50, mean_trans, stdev_trans)
natural_scale <- exp(draws)
mean(draws)
sd(draws)
```

unstructured_pop — *Simulated Time Series of an Unstructured Temporally Autocorrelated Population*

Description

This function simulates an unstructured population with temporally autocorrelated vital rates (survival and fertility). In other words, this function will show you the dynamics over time of a population whose survival and fertility is stochastic, but also correlated to the survival and fertility in the previous year, respectively. The assumptions of the simulation are that the population is asexually reproducing or female-only, survival and fertility are the same at all ages / stages, and that individuals continue to be reproductively capable until they die. The function includes demographic stochasticity as well as environmental stochasticity, and does not support density dependence at this time.

Usage

```
unstructured_pop(
  start,
  timesteps,
  survPhi,
  fecundPhi,
  survMean,
  survSd,
  fecundMean,
  fecundSd
)
```

Arguments

`start`	The starting population size.
`timesteps`	The number of timesteps you want to simulate. Individuals are added and killed off every timestep according to the survival and fertility rates. In ecological applications, timesteps are usually years, but theoretically they can be any length of time.
`survPhi`	The temporal autocorrelation of survival. 0 is white noise (uncorrelated), positive values are red noise (directly correlated) and negative values are blue noise (inversely correlated).
`fecundPhi`	The temporal autocorrelation of fecundity. As above.
`survMean`	The mean survival from timestep to timestep. Must be a value between 0 (all individuals die) and 1 (all individuals live).
`survSd`	The standard deviation of the survival from timestep to timestep. Must be a value between 0 and 1.
`fecundMean`	The mean fertility: mean offspring produced by each individual per timestep.
`fecundSd`	The standard deviation of the fertility.

Details

Be advised that not all combinations of values will work. If you set survival and fertility unrealistically high, the population size will tend toward infinity and the simulation will fail because the numbers are too large to handle. Use your common sense as a demographer / population biologist.

Value

A data frame with four variables: timestep, population (total individuals alive at the start of the timestep), newborns (new individuals born this timestep), and survivors (individuals who survive this timestep).

Examples

```
series1 <- unstructured_pop(start = 20, timesteps = 10, survPhi = 0.7, fecundPhi = -0.1,
survMean = 0.6, survSd = 0.52, fecundMean = 1.2, fecundSd = 0.7)
head(series1)
```

Index