# Package ‘simSAC’

January 7, 2016

**Title** Simulate spatially autocorrelated (SAC) data

**Version** 0.0.0.9000

**Description** simSAC provides a function to simulate data on three different landscapes; a Gaussian, Bernoulli or zero-inflated Poisson distributed repsonse variable; and four different causes of SAC or reference data, i.e. no SAC. It further provides a function to readily extract data and attributes from a netCDF file.

**Depends** R (>= 3.2.1), ncdf4, RandomFields, raster, wordspace

**License** GPL

**LazyData** true

**Maintainer** 'Severin Hauenstein' <severin.hauenstein@saturn.uni-freiburg.de>

**Author** Severin Hauenstein <severin.hauenstein@saturn.uni-freiburg.de> [aut, cre], Carsten Dormann [aut]

**Repository** CRAN, R-Forge

**Additional_repositories** http://R-Forge.R-project.org

**RoxygenNote** 5.0.1

**NeedsCompilation** no

# R topics documented:



---

`extract.ncdf` *Extract data and attributes (e.g. model structure) from netCDF (.nc) file*

---

### Description

This function allows to readily extract data, simulated with simSAC::simData, and general information (attributes) from the produced netCDF (.nc) file.

### Usage

```
extract.ncdf(ncfile)
```

### Arguments

| | |
|---|---|
| ncfile | netCDF input file (character with extension .nc). |

### Value

A list containing [[1]] a list of attributes (information and instruction), and [[2]] a `data.frame` with the simulated data.

### See Also

`simData`

### Examples

```
## Not run:
# simulate data for smooth landscape, normally distributed y and no SAC
simData("110")
# extract attributes and data
SGR <- extract.ncdf(paste0(getwd(),"/dataset110.nc"))
SGR[[1]] # attributes
head(SGR[[2]]) # data

library(lattice)
levelplot(y~Lon+Lat,data=SGR[[2]]) # plot response varibale on Longitude-Latitude grid

## End(Not run)
```

---

## geo.to.num *Transform geographic coordinates to numeric values*

---

### Description

This function transforms geographic coordinates, such as `"5N24E"` or `"7S27E"` to numeric values, such as `c(5, 24)` or `c(-7, 27)`.

### Usage

```
geo.to.num(geo)
```

### Arguments

| | |
|---|---|
| geo | Geographic coordinate (Character). |

### Value

Returns the coordinates as numeric values. To be used for the creation of (`extent`) objects.

### See Also

Function is used in `simData`.

### Examples

```
## Not run:
# Transform "5N24E" or "7S27E" to numeric values
coords <- c("5N24E", "7S27E")
num.coords <- sapply(1:2, function(x) geo.to.num(coords[x]))

## End(Not run)
```

---

## `keep.asking` *Wait for specific keypress*

---

### Description

This function allows to ask a question and prompts to a specific answer.

### Usage

```
keep.asking(Q, A = c("y", "n"),
  Reminder = "Please enter: y for yes, n for no.")
```

### Arguments

| | |
|---|---|
| `Q` | Question to be asked (Character). |
| `A` | Vector of possible answers. Defaults to `c("y","n")` |
| `Reminder` | If wrong answer was entered, reminds of the possible answers. Defaults to `"Please enter: y for yes, n for no."` |

### Value

Returns the answer entered by the user (Character).

### See Also

Function is used in `simData`.

### Examples

```
# check with user if data really should be downloaded
## Not run:
check.download <- keep.asking(Q = "Do you want to download data from http://www.worldclim.org?")
y

## End(Not run)
```

simData *Simulate spatially autocorrelated (SAC) data*

## Description

Use this function to simulate data on three different landscapes; a Gaussian, Bernoulli or zero-inflated Poisson distributed repsonse variable; and four different causes of SAC or reference data, i.e. no SAC.

## Usage

```
simData(dataset,
        filename = "default",
        gridsize = c(50L, 50L),
        cvfold = 10L,
        cvblock.size = c(10,10),
        r.seed = 20151126,
        n.predictor = 7L,
        f.smooth = list(function() lon,
                        function() lat,
                        function() (lon - mean(lon))^2,
                        function() (lat - mean(lat))^2,
                        function() x3^x4 * x4^x3,
                        function() x1^x1 * x3^x4,
                        function() x2^x1 * x4^x3 * log(x5 + 1)),
        f.realistic = list(var = 0.1, scale = 0.1),
        f.real = list(resolution = 10L,
                      extent = c("5N24E", "7S37E"),
                  bio.vars = c("bio1", "bio19", "bio2", "bio12", "bio4", "bio18", "bio3")),
        f.response = c("x1", "x4", "x4^2", "x3*x4", "x3"),
        par.response = "default",
        f.sac1 = list(corCoef = -0.3,
                      sarFactor = 10),
        f.sac2 = "x1",
        f.sac3 = c("^","*"),
        f.sac4 = list(dispersal.max = 0.1,
                      dispersal.shape = 30),
        interactive = TRUE)
```

## Arguments

`dataset` Input character of the form "abc", with:

a predictor landscape:
  **1** smooth (linear and non-linear gradients without noise)
  **2** realistic (unconditional Gaussian random fields from exponential covariance models)
  **3** real (Real bio-climatic predictors from http://www.worldclim.org)

b distribution of the response variable:
  **1** Gaussian
  **2** Bernoulli

- **3** zero-inflated Poisson
- c SAC scenario:
  - **0** Reference, i.e. no SAC
  - **1** SAC onto response variable
  - **2** Omitted predictor
  - **3** Wrong functional form, e.g. intentionally miss quadratic term or interaction
  - **4** Dispersal

`filename` The destination file name (character). Defaults to "datasetdataset", e.g. "dataset110".

`gridsize` Vector defining [1] the number of cells in x direction (Longitude), and [2] the number of cells in y direction (Latitude).

`cvfold` Number of unique Cross-Validation (CV) IDs to be assigned blockwise to the data.

`cvblock.size` Number of cells in x, y direction in one CV block.

`r.seed` Randomisation value to be used in `set.seed` before any stochastic process. Defaults to `20151126`

`n.predictor` Number of predictors to be simulated.

`f.smooth` If `dataset = "1**"`: List of `n.predictor` linear and non-linear functions. Can be functions of Longitude (`lon`) and/or Latitude (`lat`).

`f.realistic` If `dataset = "2**"`: A list of `var` and `scale`, which are passed to `RMexp` to compute the exponential covariance model. If both arguments are of length `n.predictor` a new model is computed for every predictor.

`f.real` If `dataset = "3**"`: A list comprising:

- `resolution` [minutes of a degree] = 2.5, 5, and 10 (default). Defines the resoultion of the global interpolated climate data from `http://www.worldclim.org`;
- `extent` = numeric vector of two geogr. coordinates (diagonal corners).
- `bio.vars` = character string of length `n.predictor` defining which bioclimatic should variables be used.

`f.response` Character string of mathematical terms (based on predictors x1, x2,...) yielding the response varibale.

`par.response` Coefficients for the elements in f.response. By default this numeric vector contains an intercept (first element) and beta values for every element in f.response. If the distribution is set to Gaussian, an addtional (last) element is provided to set the standard deviation in `rnorm`. Defaults to

- Gaussian: 0.8, 0.2, -0.9, 0.8, -0.6, 0.5, 0.2
- Bernoulli: 0.2, 4.5, -1.2, -1.2, -1.1, 0.9
- Poisson: 0.2, 1.6, 0.9, 0.8, -0.8, 0.5

Remember: Poisson is zero-inflated and therefore requires a list of two numeric parameter vectors. First item is a numeric vector setting the Bernoulli coefficients, second the Poisson coefficients.

`f.sac1` If `dataset = "**1"`: List of `corCoef`, a coefficient impacting the correlation structure, and (only if `dataset = "*11"`) `sarFactor`, a factor determining the magnitude of SAC added to the existing response varibale.

`f.sac2` If `dataset = "**2"`: Name of the predictor(s) to be omitted in the model structure.

| | |
|---|---|
| f.sac3 | If dataset = "**3": Character string of "^" and/or "*" to filter (grep) and omit respective terms in the model structure. |
| f.sac4 | If dataset = "**4": A list of dispersal.max = maximum dispersal factor, and dispersal.shape = shape factor, the higher the more skewed the exponential curve. |
| interactive | Defaults to TRUE. If FALSE existing files are overwritten and data (if not existiing) downloaded automatically, i.e. without asking. |

## Details

Designed to run in default mode.

## Value

No R output. Data and instruction is saved as netCDF file (filename.nc).

## Note

- SAC cause 1, i.e. spatial error onto response (dataset = "**1"), is computationally burdensome because of the inversion of the distance matrix. This can be severe for large grids, i.e. gridsize > c(100,100).
- It may be necessary to change the parameter settings (par.response) for SAC cause 3. Particularly in the case of a Bernoully distributed response variable the magnitude of SAC is rather sensitive to the par.response values.

## Author(s)

Severin Hauenstein, <severin.hauenstein@saturn.uni-freiburg.de>

## See Also

extract.ncdf which allows you to readily extract data and attributes from the netCDF file.

## Examples

```
## Not run:

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# example structure                            #
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# landscape, distribution, SAC cause

# simulate data with simData

# extract data with extract.ncdf
# lattice::levelplot of the response variable

# build linear model: model structure in attributes of netCDF file

# compute residulas
# Uni- and multivariate spatial correlograms with ncf::correlog
# plot correlogram to check spatial autocorrelation

#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#
# 12 datasets with different settings          #
```

```
#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#

library(lattice)
library(ncf)
#----------------------------------------------#
# smooth landscape, Gaussian distribution, refrence data

simData("110")

d110 <- extract.ncdf("dataset110.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d110) # levelplot response

fm110 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d110, family = "gaussian")
summary(fm110)
res110 <- residuals(fm110) # calculate residuals
co110 <- correlog(d110$Lat, d110$Lon, res110, increment=0.02, resamp=1) # check autocorrleation
plot(co110$mean.of.class, co110$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset110")

#----------------------------------------------#
# smooth landscape, Gaussian distribution, SAC onto response

simData("111")

d111 <- extract.ncdf("dataset111.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d111) # levelplot response

fm111 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d111, family = "gaussian")
summary(fm111)
res111 <- residuals(fm111) # calculate residuals
co111 <- correlog(d111$Lat, d111$Lon, res111, increment=0.02, resamp=1) # check autocorrleation
plot(co111$mean.of.class, co111$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset111")

#----------------------------------------------#
# smooth landscape, Gaussian distribution, omitted predictor

simData("112")

d112 <- extract.ncdf("dataset112.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d112) # levelplot response

fm112 <- glm(y ~ x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d112, family = "gaussian") # omit x1
summary(fm112)
res112 <- residuals(fm112) # calculate residuals
co112 <- correlog(d112$Lat, d112$Lon, res112, increment=0.02, resamp=1) # check autocorrleation
plot(co112$mean.of.class, co112$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset112")

#----------------------------------------------#
# smooth landscape, Bernoulli distribution, refrence data

simData("120")
```

```
d120 <- extract.ncdf("dataset120.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d120) # levelplot response

fm120 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d120, family = "gaussian")
summary(fm120)
res120 <- residuals(fm120) # calculate residuals
co120 <- correlog(d120$Lat, d120$Lon, res120, increment=0.02, resamp=1) # check autocorrleation
plot(co120$mean.of.class, co120$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset120")

#---------------------------------------------#
# smooth landscape, Bernoulli distribution, SAC onto response

simData("121")

d121 <- extract.ncdf("dataset121.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d121) # levelplot response

fm121 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d121, family = "gaussian")
summary(fm121)
res121 <- residuals(fm121) # calculate residuals
co121 <- correlog(d121$Lat, d121$Lon, res121, increment=0.02, resamp=1) # check autocorrleation
plot(co121$mean.of.class, co121$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset121")

#---------------------------------------------#

simData("122")

d122 <- extract.ncdf("dataset122.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d122) # levelplot response

fm122 <- glm(y ~ x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d122, family = "gaussian") # omit x1
summary(fm122)
res122 <- residuals(fm122) # calculate residuals
co122 <- correlog(d122$Lat, d122$Lon, res122, increment=0.02, resamp=1) # check autocorrleation
plot(co122$mean.of.class, co122$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset122")

#---------------------------------------------#
# real landscape, Gaussian distribution, refrence data

simData("310")

d310 <- extract.ncdf("dataset310.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d310) # levelplot response

fm310 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d310, family = "gaussian")
summary(fm310)
res310 <- residuals(fm310) # calculate residuals
co310 <- correlog(d310$Lat, d310$Lon, res310, increment=0.16, resamp=1) # check autocorrleation
plot(co310$mean.of.class, co310$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset310")
```

```
#--------------------------------------------#
# real landscape, Gaussian distribution, SAC onto response

simData("311")

d311 <- extract.ncdf("dataset311.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d311) # levelplot response

fm311 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d311, family = "gaussian")
summary(fm311)
res311 <- residuals(fm311) # calculate residuals
co311 <- correlog(d311$Lat, d311$Lon, res311, increment=0.16, resamp=1) # check autocorrleation
plot(co311$mean.of.class, co311$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset311")

#--------------------------------------------#
# real landscape, Gaussian distribution, omitted predictor

simData("312")

d312 <- extract.ncdf("dataset312.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d312) # levelplot response

fm312 <- glm(y ~ x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d312, family = "gaussian") # omit x1
summary(fm312)
res312 <- residuals(fm312) # calculate residuals
co312 <- correlog(d312$Lat, d312$Lon, res312, increment=0.16, resamp=1) # check autocorrleation
plot(co312$mean.of.class, co312$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset312")

#--------------------------------------------#
# real landscape, Bernoulli distribution, refrence data

simData("320")

d320 <- extract.ncdf("dataset320.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d320) # levelplot response

fm320 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d320, family = "gaussian")
summary(fm320)
res320 <- residuals(fm320) # calculate residuals
co320 <- correlog(d320$Lat, d320$Lon, res320, increment=0.16, resamp=1) # check autocorrleation
plot(co320$mean.of.class, co320$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset320")

#--------------------------------------------#
# real landscape, Bernoulli distribution, SAC onto response

simData("321")

d321 <- extract.ncdf("dataset321.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d321) # levelplot response
```

```
fm321 <- glm(y ~ x1 + x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d321, family = "gaussian")
summary(fm321)
res321 <- residuals(fm321) # calculate residuals
co321 <- correlog(d321$Lat, d321$Lon, res321, increment=0.16, resamp=1) # check autocorrleation
plot(co321$mean.of.class, co321$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset321")

#---------------------------------------------#
# real landscape, Bernoulli distribution, omitted predictor

simData("322")

d322 <- extract.ncdf("dataset322.nc")[[2]] # extract data
levelplot(y~Lon+Lat,data=d322) # levelplot response

fm322 <- glm(y ~ x4 + I(x4^2) + x3*x4 + x3 + x2 + x5 + x6 + x7,
             data = d322, family = "gaussian") # omit x1
summary(fm322)
res322 <- residuals(fm322) # calculate residuals
co322 <- correlog(d322$Lat, d322$Lon, res322, increment=0.16, resamp=1) # check autocorrleation
plot(co322$mean.of.class, co322$correlation, type = "o", ylim = c(-1,1), # plot correlogram
     ylab="Moran Similarity", xlab="averaged distance class", main = "dataset322")

## End(Not run)
```

# Index