Package ‘DHARMa’

March 11, 2017

Title Residual Diagnostics for Hierarchical (Multi-Level / Mixed) Regression Models

Version 0.1.5

Date 2017-03-12

Description The 'DHARMa' package uses a simulation-based approach to create readily interpretable scaled (quantile) residuals for fitted generalized linear mixed models. Currently supported are generalized linear mixed models from 'lme4' (classes 'lmerMod', 'glmerMod'), generalized additive models ('gam' from 'mgcv'), 'glm' (including 'negbin' from 'MASS', but excluding quasi-distributions) and 'lm' model classes. Alternatively, externally created simulations, e.g. posterior predictive simulations from Bayesian software such as 'JAGS', 'STAN', or 'BUGS' can be processed as well. The resulting residuals are standardized to values between 0 and 1 and can be interpreted as intuitively as residuals from a linear regression. The package also provides a number of plot and test functions for typical model misspecification problems, such as over/underdispersion, zero-inflation, and residual spatial and temporal autocorrelation.

Depends R (>= 3.0.2)

Imports stats, graphics, utils, grDevices, gap, qrnn, lmtest, ape, sfsmisc, MASS, doParallel, foreach, lme4, mgcv

Suggests knitr, testthat

License GPL (>= 3)

URL http://florianhartig.github.io/DHARMa/

BugReports https://github.com/florianhartig/DHARMa/issues

LazyData true

RoxygenNote 6.0.0

VignetteBuilder knitr

NeedsCompilation no

Author Florian Hartig [aut, cre]

Maintainer Florian Hartig <florian.hartig@biologie.uni-regensburg.de>

Repository CRAN

Date/Publication 2017-03-11 00:03:57
### R topics documented:

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>benchmarkOverdispersion</td>
<td>2</td>
</tr>
<tr>
<td>benchmarkP</td>
<td>3</td>
</tr>
<tr>
<td>benchmarkUniformity</td>
<td>4</td>
</tr>
<tr>
<td>createData</td>
<td>5</td>
</tr>
<tr>
<td>createDHARMa</td>
<td>7</td>
</tr>
<tr>
<td>DHARMa</td>
<td>8</td>
</tr>
<tr>
<td>fitted.gam</td>
<td>8</td>
</tr>
<tr>
<td>if predictor is a factor, a boxplot will be plotted instead of a scatter plot</td>
<td>9</td>
</tr>
<tr>
<td>plot.DHARMa</td>
<td>10</td>
</tr>
<tr>
<td>plotConventionalResiduals</td>
<td>10</td>
</tr>
<tr>
<td>plotSimulatedResiduals</td>
<td>11</td>
</tr>
<tr>
<td>print.DHARMa</td>
<td>12</td>
</tr>
<tr>
<td>simulateResiduals</td>
<td>12</td>
</tr>
<tr>
<td>testOverdispersion</td>
<td>14</td>
</tr>
<tr>
<td>testOverdispersionParametric</td>
<td>15</td>
</tr>
<tr>
<td>testSimulatedResiduals</td>
<td>16</td>
</tr>
<tr>
<td>testSpatialAutocorrelation</td>
<td>17</td>
</tr>
<tr>
<td>testTemporalAutocorrelation</td>
<td>18</td>
</tr>
<tr>
<td>testUniformity</td>
<td>19</td>
</tr>
<tr>
<td>testZeroInflation</td>
<td>19</td>
</tr>
</tbody>
</table>

### Index

- benchmarkOverdispersion
  
  **Overdispersion benchmarks**

### Description

This function runs Power / Type I error simulations for overdispersion tests in DHARMa.

### Usage

```r
benchmarkOverdispersion(dispersionValues = 0, nRep = 10, alpha = 0.05, plot = T, parallel = F, ...)
```

### Arguments

- `dispersionValues`: amount of overdispersion
- `nRep`: replicates
- `alpha`: significance level
- `plot`: whether to do a plot
benchmarkP

```
parallel whether to use parallel computations. Possible values are F, T (parallel cores set to number of cores in the computer -1), or an integer number for the number of cores that should be used
```

```
parameters to pass on to `simulateResiduals`
```

Details

This function runs Type I error / power comparisons for overdispersion tests in DHARMa. Compared are a) the omnibus test `testUniformity` b) the parameteric dispersion test `testOverdispersionParametric`, and the nonparametric dispersion test `testOverdispersion`.

Note

The benchmark function in DHARMa are intended for development purposes, and for users that want to test / confirm the properties of functions in DHARMa. If you are running an applied data analysis, they are probably of little use.

See Also

`benchmarkUniformity`

`benchmarkP`  
General Type I/II benchmarks

Description

This function runs Power / Type I error simulations for an arbitrary test with a control parameter

Usage

```
benchmarkP(controlValues = 0, getP, nRep = 10, alpha = 0.05, plot = T, parallel = F)
```

Arguments

```
controlValues a vector with a control parameter (e.g. to vary the strength of a problem the test should be specific to)
getP the test to be benchmarked - this should be a function that takes one of the controlValues as an input, and returns a p-value
nRep number of replicates per level of the controlValues
alpha significance level
plot whether to do a plot
parallel whether to use parallel computations. Possible values are F, T (parallel cores set to number of cores in the computer -1), or an integer number for the number of cores that should be used
```
**Note**

The benchmark function in DHARMa are intended for development purposes, and for users that want to test / confirm the properties of functions in DHARMa. If you are running an applied data analysis, they are probably of little use.

**See Also**

`benchmarkUniformity`

---

**benchmarkUniformity**  
*Uniformity benchmarks*

**Description**

This function runs simulation to confirm uniformity of residuals under H0

**Usage**

`benchmarkUniformity(dataModelCreator, nSim = 100, plot = T, ...)`

**Arguments**

- `dataModelCreator`: a function that returns a list with two elements, data and model. See help for details
- `nSim`: number of simulations
- `plot`: should a plot be created
- `...`: parameters to pass on to `simulateResiduals`

**Details**

This function runs repeated simulations to test if residuals are really uniform if the data-creating process and the model that is used to fit the data are identical

**Note**

The benchmark function in DHARMa are intended for development purposes, and for users that want to test / confirm the properties of functions in DHARMa. If you are running an applied data analysis, they are probably of little use.

**See Also**

`benchmarkOverdispersion`
createData

Examples

library(lme4)

dataModelCreator <- function(){
  data = createData(sampleSize = 100, overdispersion = 0, family = poisson())
  model <- glmer(observedResponse ~ Environment1 + (1|group) +
                 (1|ID), data = data, family = "poisson")
  return(list(data=data, model = model))
}

## Not run: benchmarkUniformity(dataModelCreator = dataModelCreator, nSim = 4)

createData **Simulate test data**

Description

This function creates synthetic dataset with various problems such as overdispersion, zero-inflation, etc.

Usage

createData(replicates = 1, sampleSize = 10, intercept = 0,
            fixedEffects = 1, quadraticFixedEffects = NULL, numGroups = 10,
            randomEffectVariance = 1, overdispersion = 0, family = poisson(),
            scale = 1, cor = 0, roundPoissonVariance = NULL, pZeroInflation = 0,
            binomialTrials = 1, temporalAutocorrelation = 0,
            spatialAutocorrelation = 0, factorResponse = F)

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>replicates</td>
<td>number of datasets to create</td>
</tr>
<tr>
<td>sampleSize</td>
<td>sample size of the dataset</td>
</tr>
<tr>
<td>intercept</td>
<td>intercept (linear scale)</td>
</tr>
<tr>
<td>fixedEffects</td>
<td>vector of fixed effects (linear scale)</td>
</tr>
<tr>
<td>quadraticFixedEffects</td>
<td>vector of quadratic fixed effects (linear scale)</td>
</tr>
<tr>
<td>numGroups</td>
<td>number of groups for the random effect</td>
</tr>
<tr>
<td>randomEffectVariance</td>
<td>variance of the random effect (intercept)</td>
</tr>
<tr>
<td>overdispersion</td>
<td>if this is a numeric value, it will be used as the sd of a random normal variate that is added to the linear predictor. Alternatively, a random function can be provided that takes as input the linear predictor.</td>
</tr>
<tr>
<td>family</td>
<td>family</td>
</tr>
<tr>
<td>scale</td>
<td>scale if the distribution has a scale (e.g. sd for the Gaussian)</td>
</tr>
</tbody>
</table>
createData

cor correlation between predictors
roundPoissonVariance
    if set, this creates a uniform noise on the posssion response. The aim of this is to
create heteroscedasticity
pZeroInflation probability to set any data point to zero
binomialTrials Number of trials for the binomial. Only active if family == binomial
temporalAutocorrelation
    strength of temporal Autocorrelation
spatialAutocorrelation
    strength of spatial Autocorrelation
factorResponse should the response be transformed to a factor (intended to be used for 0/1 data)

Examples

testData = createData(sampleSize = 500, intercept = 2, fixedEffects = c(1),
    overdispersion = 0, family = poisson(), quadraticFixedEffects = c(-3),
    randomEffectVariance = 0)

par(mfrow = c(1,2))
plot(testData$Environment1, testData$observedResponse)
hist(testData$observedResponse)

# with zero-inflation

testData = createData(sampleSize = 500, intercept = 2, fixedEffects = c(1),
    overdispersion = 0, family = poisson(), quadraticFixedEffects = c(-3),
    randomEffectVariance = 0, pZeroInflation = 0.6)

par(mfrow = c(1,2))
plot(testData$Environment1, testData$observedResponse)
hist(testData$observedResponse)

# binomial with multiple trials

testData = createData(sampleSize = 40, intercept = 2, fixedEffects = c(1),
    overdispersion = 0, family = binomial(), quadraticFixedEffects = c(-3),
    randomEffectVariance = 0, binomialTrials = 20)

plot(observedResponse1 / observedResponse0 ~ Environment1, data = testData, ylab = "Proportion 1")

# spatial / temporal correlation

testData = createData(sampleSize = 100, family = poisson(), spatialAutocorrelation = 3,
    temporalAutocorrelation = 3)

plot(log(observedResponse) ~ time, data = testData)
plot(log(observedResponse) ~ x, data = testData)
Create DHARMa

Convert simulated residuals to a DHARMa object

Description

Convert simulated residuals to a DHARMa object

Usage

createDHARMa(scaledResiduals = NULL, simulatedResponse = NULL,
observedResponse = NULL, fittedPredictedResponse = NULL,
integerResponse = F)

Arguments

- scaledResiduals
  - optional scaled residuals from a simulation, e.g. Bayesian p-values. If those are not provided, simulated and true observations have to be provided.
- simulatedResponse
  - matrix of observations simulated from the fitted model - row index for observations and column index for simulations
- observedResponse
  - true observations
- fittedPredictedResponse
  - fitted predicted response. Optional, but will be necessary for some plots. If scaled residuals are Bayesian p-values, using the median posterior prediction as fittedPredictedResponse is recommended.
- integerResponse
  - if T, noise will be added at to the residuals to maintain a uniform expectations for integer responses (such as Poisson or Binomial). Unlike in simulateResiduals, the nature of the data is not automatically detected, so this MUST be set by the user appropriately

Details

The use of this function is to convert simulated residuals (e.g. from a point estimate, or Bayesian p-values) to a DHARMa object, to make use of the plotting / test functions in DHARMa

Note

Either scaled residuals or (simulatedResponse AND observed response) have to be provided
Description

The DHARMa package uses a simulation-based approach to create readily interpretable scaled residuals from fitted generalized linear mixed models. Currently supported are generalized linear mixed models from 'lme4' (classes 'lmerMod', 'glmerMod'), generalized additive models ('gam' from 'mgcv'), 'glm' (including 'negbin' from 'MASS', but excluding quasi-distributions) and 'lm' model classes. Alternatively, externally created simulations, e.g. posterior predictive simulations from Bayesian software such as 'JAGS', 'STAN', or 'BUGS' can be processed as well. The resulting residuals are standardized to values between 0 and 1 and can be interpreted as intuitively as residuals from a linear regression. The package also provides a number of plot and test functions for typical model misspecification problems, such as over/underdispersion, zero-inflation, and spatial / temporal autocorrelation.

Details

See index / vignette for details

See Also

simulateResiduals

Examples

vignette("DHARMa", package="DHARMa")

fitted.gam

This function overwrites the standard fitted function for GAM

Description

This function overwrites the standard fitted function for GAM

Usage

## S3 method for class 'gam'
fitted(object, ...)

Arguments

object fitted model
... arguments to be passed on to stats::fitted
Note

See explanation at

if predictor is a factor, a boxplot will be plotted instead of a scatter plot.

Generic residual plot with either spline or quantile regression

Description

The function creates a generic residual plot with either spline or quantile regression

Usage

plotResiduals(pred, residual, quantreg = T, ...)

Arguments

pred  predictor variable
residual  residual variable
quantreg  should a quantile regression be performed. If F, a smooth spline will be plotted
...  additional arguments to plot

Details

For a correctly specified model, we would expect uniformity in y direction when plotting against any predictor.

To provide a visual aid in detecting deviations from uniformity in y-direction, the plot of the residuals against the predicted values also performs an (optional) quantile regression, which provides 0.25, 0.5 and 0.75 quantile lines across the plots. These lines should be straight, horizontal, and at y-values of 0.25, 0.5 and 0.75. Note, however, that some deviations from this are to be expected by chance, even for a perfect model, especially if the sample size is small.

The quantile regression can take some time to calculate, especially for larger datasets. For that reason, quantreg = F can be set to produce a smooth spline instead.

If the predictor is a factor (categorical), a boxplot will be created - with a uniform distribution, the box should go from 0.25 to 0.75, with the median line at 0.5. Again, chance deviations from this will increase when the sample size is smaller.

See Also

plotSimulatedResiduals
plotConventionalResiduals

Description

Plot simulated residuals

Usage

```
## S3 method for class 'DHARMa'
plot(x, ...)
```

Arguments

- `x`: an object with simulated residuals created by `simulateResiduals`
- `...`: optional arguments to pass on to `plotSimulatedResiduals`

plotConventionalResiduals

Conventional residual plot

Description

Convenience function to draw conventional residual plots

Usage

```
plotConventionalResiduals(fittedModel)
```

Arguments

- `fittedModel`: a fitted model object
plotSimulatedResiduals

DHARMa standard residual plots

Description

This function creates standard plots for the simulated residuals

Usage

```
plotSimulatedResiduals(simulationOutput, quantreg = T)
```

Arguments

- `simulationOutput`: an object with simulated residuals created by `simulateResiduals`
- `quantreg`: whether to perform a quantile regression on 0.25, 0.5, 0.75. If F, a spline will be created instead

Details

The function creates two plots. To the left, a qq-uniform plot to detect deviations from overall uniformity of the residuals, and to the right a plot of residuals against predicted values. For a correctly specified model we would expect

a) a uniform (flat) distribution of the overall residuals, evidenced by a straight line in the qq-plot
b) uniformity in y direction if we plot against any predictor, including the predicted value.

To provide a visual aid in detecting deviations from uniformity in y-direction, the plot of the residuals against the predicted values also performs an (optional) quantile regression, which provides 0.25, 0.5 and 0.75 quantile lines across the plots. These lines should be straight, horizontal, and at y-values of 0.25, 0.5 and 0.75. Note, however, that some deviations from this are to be expected by chance, even for a perfect model, especially if the sample size is small.

The quantile regression can take some time to calculate, especially for larger datasets. For that reason, quantreg = F can be set to produce a smooth spline instead.

See Also

```
plotResiduals
```
**simulateResiduals**

### Description

Print simulated residuals

### Usage

```r
## S3 method for class 'DHARMa'
print(x, ...)
```

### Arguments

- `x`  
  an object with simulated residuals created by `simulateResiduals`

- `...`  
  optional arguments to pass on to `plotSimulatedResiduals`

---

**simulateResiduals**

### Description

The function creates scaled residuals by simulating from the fitted model

### Usage

```r
simulateResiduals(fittedModel, n = 250, refit = F, integerResponse = NULL, plot = F, ...)
```

### Arguments

- `fittedModel`  
  fitted model object. Supported are generalized linear mixed models from 'lme4' (classes 'lmerMod', 'glmerMod'), generalized additive models ('gam' from 'mgcv', excluding extended families from 'mgcv'), 'glm' (including 'negbin' from 'MASS', but excluding quasi-distributions) and 'lm' model classes.

- `n`  
  integer number > 1, number of simulations to run. If possible, set to at least 250, better 1000. See also details

- `refit`  
  if `F`, new data will be simulated and scaled residuals will be created by comparing observed data with new data. If `T`, the model will be refit on the simulated data (parametric bootstrap), and scaled residuals will be created by comparing observed with refitted residuals.

- `integerResponse`  
  if `T`, noise will be added at to the residuals to maintain a uniform expectations for integer responses (such as Poisson or Binomial). Usually, the model will automatically detect the appropriate setting, so there is no need to adjust this setting.
plot

if T, `plotSimulatedResiduals` will be directly run after the simulations have terminated

... parameters to pass to the simulate function of the model object. An important use of this is to specify whether simulations should be conditional on the current random effect estimates. See details.

**Details**

There are a number of important considerations when simulating from a more complex (hierarchical) model.

**Resimulating random effects / hierarchical structure**: the first is that in a hierarchical model, several layers of stochasticity are aligned on top of each other. Specifically, in a GLMM, we have a lower level stochastic process (random effect), whose result enters into a higher level (e.g. Poisson distribution). For other hierarchical models such as space-time models, similar considerations apply. When simulating, we have to decide if we want to re-simulate all stochastic levels, or only a subset of those. For example, in a GLMM, it is common to only simulate the last stochastic level (e.g. Poisson) conditional on the fitted random effects.

For controlling how many levels should be re-simulated, the `simulateResiduals` function allows to pass on parameters to the `simulate` function of the fitted model object. Please refer to the help of the different `simulate` functions (e.g. `?simulate.merMod`) for details. For `merMod` (`lme4`) model objects, the relevant parameters are `use.u`, and `re.form`.

If the model is correctly specified, the simulated residuals should be flat regardless how many hierarchical levels we re-simulate. The most thorough procedure would therefore be to test all possible options. If testing only one option, I would recommend to re-simulate all levels, because this essentially tests the model structure as a whole. This is the default setting in the DHARMA package. A potential drawback is that re-simulating the lower-level random effects creates more variability, which may reduce power for detecting problems in the upper-level stochastic processes.

**Integer responses**: a second complication is the treatment of integer responses. Imaging we have observed a 0, and we predict 30% zeros - what is the quantile that we should display for the residual? To deal with this problem and maintain a uniform response, the option `integerResponse` adds a uniform noise from -0.5 to 0.5 on the simulated and observed response. Note that this works because the expected distribution of this is flat - you can see this via `hist(ecdf(runif(10000))(runif(10000)))`.

**Refitting or not**: a third issue is how residuals are calculated. `simulateResiduals` has two options that are controlled by the `refit` parameter:

1. if `refit = F` (default), new data is simulated from the fitted model, and residuals are calculated by comparing the observed data to the new data
2. if `refit = T`, a parametric bootstrap is performed, meaning that the model is refit on the new data, and residuals are created by comparing observed residuals against refitted residuals

The second option is much slower, and only important for running tests that rely on comparing observed to simulated residuals, e.g. the `testOverdispersion` function.

**How many simulations**: about the choice of `n`: my simulations didn’t show major problems with a small `n` (if you get down to the order of a few 10, you will start seeing discretization artifacts from the empirical cumulative density estimates though). The default of 250 seems safe to me. If you want to be on the safe side, choose a high value (e.g. 1000) for producing your definite results.
testOverdispersion

Value
A list with various objects. The most important are scaledResiduals, which contain the scaled residuals, and scaledResidualsNormal, which are the scaled residuals transformed to a normal distribution.

See Also
testSimulatedResiduals, plotSimulatedResiduals

Examples

```r
library(lme4)

testData = createData(sampleSize = 200, overdispersion = 0.5, family = poisson())
fittedModel <- glmer(observedResponse ~ Environment1 + (1|group),
  family = "poisson", data = testData,
  control=glmerControl(optCtrl=list(maxfun=20000)))

simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# plot residuals, quantreg = T is better but costs more time
plotSimulatedResiduals(simulationOutput = simulationOutput, quantreg = FALSE)

# create simulations with refitting, n=5 is very low, set higher when using this
simulationOutput <- simulateResiduals(fittedModel = fittedModel, n = 10, refit = TRUE)
plotSimulatedResiduals(simulationOutput = simulationOutput, quantreg = FALSE)
```

---

testOverdispersion  Test for over/underdispersion

Description
This function performs a simulation-based test for over/underdispersion.

Usage
testOverdispersion(simulationOutput, alternative = "overdispersion",
  plot = F)

Arguments

- `simulationOutput`: an object with simulated residuals created by `simulateResiduals`
- `alternative`: whether to test for "overdispersion", "underdispersion", or "both" (both reduces power)
- `plot`: whether to plot output
Details

The function implements two tests, depending on whether it is applied on a simulation with refit = F, or refit = T.

If refit = F (not recommended), the function tests if the IQR of the scaled residuals deviate from the null hypothesis of a uniform distribution. Simulations show that this option is not properly calibrated and much less powerful than the parametric alternative testOverdispersionParametric and even the simple testUniformity, and therefore it’s use is not recommended. A warning will be returned if the function is called.

If refit = T, the function compares the approximate deviance (via squared pearson residuals) with the same quantity from the models refitted with simulated data. It is much slower than the parametric alternative testOverdispersionParametric, but simulations show that it is slightly more powerful than the latter, and more powerful than any other non-parametric test in DHARMa, and it doesn’t make any parametric assumptions. However, given the computational cost, I would suggest that most users will be satisfied with the parametric overdispersion test.

See Also

  * testSimulatedResiduals, testSimulatedResiduals, testZeroInflation, testTemporalAutocorrelation, testSpatialAutocorrelation, testOverdispersionParametric

---

testOverdispersionParametric

*Parametric overdispersion test*

Description

This function implements a parametric dispersion test based on comparing the residual deviance to the residual degrees of freedom that is commonly used, with the purpose of benchmarking against the nonparametric tests of DHARMa.

Usage

  * testOverdispersionParametric(model)

Arguments

  * model  
    a fitted model object. See details for possible models

Details

The general idea of such as test is to copy GLM wisdom that we can define a dispersion parameter as residual deviance / residual degrees of freedom. For a model with correct dispersion, this parameter

1. Should be on average 1 2. Be chi2 distributed with df = rdf

For GL(M)Ms, we have to answer three questions

1. What is the residual deviance 2. What are the rdf 3. Is the distribution still chisq

The implementation here follows the suggestion in http://glmm.wikidot.com/faq, which is based on dividing the pearson residuals by the (probably not completely accurate) rdf, and testing this against a chi2 distribution with df = rdf.

See Also

`testSimulatedResiduals, testSimulatedResiduals, testZeroInflation, testTemporalAutocorrelation, testSpatialAutocorrelation`

---

testSimulatedResiduals

*Residual tests*

**Description**

This is intended as a wrapper for the various test functions. Currently, this function calls only the `testUniformity` function. Other tests (see below) have to be called by hand.

**Usage**

testSimulatedResiduals(simulationOutput)

**Arguments**

- `simulationOutput`
  
an object with simulated residuals created by `simulateResiduals`

**Details**

Currently, this function calls only the `testUniformity` function. All other tests (see below) have to be called by hand.

**See Also**

`testUniformity, testZeroInflation, testTemporalAutocorrelation, testSpatialAutocorrelation, testOverdispersion, testOverdispersionParametric`
testSpatialAutocorrelation

Test for spatial autocorrelation

Description

This function performs a standard test for spatial autocorrelation on the simulated residuals.

Usage

testSpatialAutocorrelation(simulationOutput, x, y, plot = T)

Arguments

- simulationOutput: an object with simulated residuals created by simulateResiduals.
- x: the x coordinate, in the same order as the data points. If set to "random", random values will be created.
- y: the x coordinate, in the same order as the data points. If set to "random", random values will be created.
- plot: whether to plot output.

Details

performs Moran.I from the package ape to test against euclidean distance and plots the residuals against space.

Note

It is possible to not specify x and y. In this case, random x and y values are created. The sense of this option is to test the rate of false positives under the current residual structure (random x/y corresponds to H0: no spatial autocorrelation). This may be useful because it may be that the test doesn’t have nominal error rates due to some problem in the residual structure that is different from spatial autocorrelation.

See Also

testUniformity, testZeroInflation, testTemporalAutocorrelation, testSimulatedResiduals, testOverdispersion, testOverdispersionParametric
testTemporalAutocorrelation

*Test for temporal autocorrelation*

**Description**

This function performs a standard test for temporal autocorrelation on the simulated residuals.

**Usage**

```r
testTemporalAutocorrelation(simulationOutput, time, plot = T)
```

**Arguments**

- `simulationOutput`:
  - an object with simulated residuals created by `simulateResiduals`.

- `time`:
  - the time, in the same order as the data points. If set to "random", random values will be created.

- `plot`:
  - whether to plot output.

**Details**

The function performs a Durbin-Watson test on the uniformly scaled residuals, and plots the residuals against time. The DB test was originally designed for normal residuals. In simulations, I didn’t see a problem with this setting though. The alternative is to transform the uniform residuals to normal residuals and perform the DB test on those.

**Note**

It is possible to not specify x and y. In this case, random x and y values are created. The sense of this option is to test the rate of false positives under the current residual structure (random x/y corresponds to H0: no spatial autocorrelation). This may be useful because it may be that the test doesn’t have nominal error rates due to some problem in the residual structure that is different from spatial autocorrelation.

**See Also**

- `testUniformity`, `testZeroInflation`, `testSimulatedResiduals`, `testSpatialAutocorrelation`, `testOverdispersion`, `testOverdispersionParametric`
**testUniformity**  
*Test for overall uniformity*

---

**Description**
This function tests the overall uniformity of the residuals.

**Usage**
```r
testUniformity(simulationOutput)
```

**Arguments**
- `simulationOutput`: an object with simulated residuals created by `simulateResiduals`.

**Details**
Tests residuals against a uniform distribution with the KS test.

**See Also**
- `testSimulatedResiduals`, `testZeroInflation`, `testTemporalAutocorrelation`, `testSpatialAutocorrelation`, `testOverdispersion`, `testOverdispersionParametric`.

---

**testZeroInflation**  
*Tests for zero-inflation*

---

**Description**
This function compares the observed number of zeros with the zeros expected from simulations.

**Usage**
```r
testZeroInflation(simulationOutput, plot = T, alternative = "more")
```

**Arguments**
- `simulationOutput`: an object with simulated residuals created by `simulateResiduals`.
- `plot`: whether to plot output.
- `alternative`: whether to test for 'more', 'less', or 'both' more or less zeros in the observed data.
Details

shows the expected distribution of zeros against the observed

See Also

testUniformity, testSimulatedResiduals, testTemporalAutocorrelation, testSpatialAutocorrelation, testOverdispersion, testOverdispersionParametric
Index

benchmarkOverdispersion, 2, 4
benchmarkP, 3
benchmarkUniformity, 3, 4, 4

createData, 5
createDHARMa, 7

DHARMa, 8
DHARMa-package (DHARMa), 8

fitted.gam, 8

if predictor is a factor, a boxplot will be plotted instead of a scatter plot., 9

plot.DHARMa, 10
plotConventionalResiduals, 10
plotResiduals, 11
plotResiduals (if predictor is a factor, a boxplot will be plotted instead of a scatter plot.), 9
plotSimulatedResiduals, 9, 10, 11, 12–14
print.DHARMa, 12

simulateResiduals, 3, 4, 7, 8, 10–12, 12, 14, 16–19

testOverdispersion, 3, 13, 14, 16–20
testOverdispersionParametric, 3, 15, 15, 16–20
testSimulatedResiduals, 14–16, 16, 17–20
testSpatialAutocorrelation, 15, 16, 17, 18–20
testTemporalAutocorrelation, 15–17, 18, 19, 20
testUniformity, 3, 15–18, 19, 20
testZeroInflation, 15–19, 19