Package 'DHARMa'

January 16, 2022

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

Version 0.4.5

Date 2022-01-12

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 linear and generalized linear (mixed) models from 'lme4'
(classes 'lmerMod', 'glmerMod'), 'glmmTMB' 'GLMMadaptive' and 'spaMM', generalized addi-
tive models
('gam' from 'mgcv'), 'glm' (including 'negbin' from 'MASS', but excluding quasi-distributions) and
'lm' model classes. Moreover, 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, Matrix, parallel, gap,
lmtest, ape, qgam (>= 1.3.2), lme4

Suggests knitr, testthat, rmarkdown, KernSmooth, sfsmisc, MASS, mgcv,
mgcViz (>= 0.1.9), spaMM (>= 3.2.0), GLMMadaptive, glmmTMB (>=
1.1.2.3)

Enhances phyr, rstan, rjags, BayesianTools

License GPL (>= 3)

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

LazyData TRUE

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

RoxygenNote 7.1.1

VignetteBuilder knitr

Encoding UTF-8

NeedsCompilation no

Author Florian Hartig [aut, cre] (<<https://orcid.org/0000-0002-6255-9059>>),
     Lukas Lohse [ctb]

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

R **topics documented:**

---

createData *Simulate test data*

Description

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

Usage

```
createData(sampleSize = 100, 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, replicates = 1, hasNA = F)
```

Arguments

sampleSize	sample size of the dataset
intercept	intercept (linear scale)
fixedEffects	vector of fixed effects (linear scale)
quadraticFixedEffects	
	vector of quadratic fixed effects (linear scale)
numGroups	number of groups for the random effect
randomEffectVariance	
	variance of the random effect (intercept)
overdispersion	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.
family	family
scale	scale if the distribution has a scale (e.g. sd for the Gaussian)
cor	correlation between predictors
roundPoissonVariance	
	if set, this creates a uniform noise on the possion 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 temporalAutocorrelation

spatialAutocorrelation

                strength of spatial Autocorrelation

factorResponse   should the response be transformed to a factor (inteded to be used for 0/1 data)

replicates   number of datasets to create

hasNA   should an NA be added to the environmental predictor (for test purposes)

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)
```

---

createDHARMa   *Create a DHARMa object from hand-coded simulations or Bayesian*
                            posterior predictive simulations

Description

Create a DHARMa object from hand-coded simulations or Bayesian posterior predictive simulations

Usage

```
createDHARMa(simulatedResponse, observedResponse,
  fittedPredictedResponse = NULL, integerResponse = F, seed = 123,
  method = c("PIT", "traditional"), rotation = NULL)
```

Arguments

simulatedResponse
>   matrix of observations simulated from the fitted model - row index for observations and colum index for simulations

observedResponse
>   true observations

fittedPredictedResponse
>   optional fitted predicted response. For Bayesian posterior predictive simulations, using the median posterior prediction as fittedPredictedResponse is recommended. If not provided, the mean simulatedResponse will be used.

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

seed
>   the random seed to be used within DHARMa. The default setting, recommended for most users, is keep the random seed on a fixed value 123. This means that you will always get the same randomization and thus teh same result when running the same code. NULL = no new seed is set, but previous random state will be restored after simulation. FALSE = no seed is set, and random state will not be restored. The latter two options are only recommended for simulation experiments. See vignette for details.

method
>   the quantile randomization method used. The two options implemented at the moment are probability integral transform (PIT-) residuals (current default), and the "traditional" randomization procedure, that was used in DHARMa until version 0.3.0. For details, see getQuantile

rotation
>   optional rotation of the residual space to remove residual autocorrelation. See details in simulateResiduals, section *residual auto-correlation* for an extended explanation, and getQuantile for syntax.

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

Examples

```
## READING IN HAND-CODED SIMULATIONS

testData = createData(sampleSize = 50, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1, data = testData, family = "poisson")

# in DHARMA, using the simulate.glm function of glm
sims = simulateResiduals(fittedModel)
plot(sims, quantreg = FALSE)

# Doing the same with a handcoded simulate function.
# of course this code will only work with a 1-par glm model
simulateMyfit <- function(n=10, fittedModel){
  int = coef(fittedModel)[1]
  slo = coef(fittedModel)[2]
  pred = exp(int + slo * testData$Environment1)
  predSim = replicate(n, rpois(length(pred), pred))
  return(predSim)
}

sims = simulateMyfit(250, fittedModel)

dharmaRes <- createDHARMa(simulatedResponse = sims,
                          observedResponse = testData$observedResponse,
                    fittedPredictedResponse = predict(fittedModel, type = "response"),
                          integer = TRUE)
plot(dharmaRes, quantreg = FALSE)
```

DHARMa   *DHARMa - Residual Diagnostics for HierArchical (Multi-level / Mixed) Regression Models*

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 linear and generalized linear (mixed) models from 'lme4' (classes 'lmerMod', 'glmerMod'), 'glmmTMB' and 'spaMM', generalized additive models ('gam' from 'mgcv'), 'glm' (including 'negbin' from 'MASS', but excluding quasi-distributions) and 'lm' model classes. Moreover, 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.

Details

See index / vignette for details

See Also

simulateResiduals

Examples

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

| getFitted | *Get fitted / predicted values* |

Description

Wrapper to get the fitted / predicted response of model at the response scale

Usage

```
getFitted(object, ...)

## Default S3 method:
getFitted(object, ...)

## Default S3 method:
getResiduals(object, ...)

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

## S3 method for class 'HLfit'
getFitted(object, ...)

## S3 method for class 'MixMod'
getFitted(object, ...)
```

Arguments

object	a fitted model
...	additional parameters to be passed on, usually to the simulate function of the respective model class

Details

The purpose of this wrapper is to standardize extract the fitted values, which is implemented via predict(model, type = "response") for most model classes.

If you implement this function for a new model class, you should include an option to modifying which REs are included in the predictions. If this option is not available, it is essential that predictions are provided marginally / unconditionally, i.e. without the random effect estimates (because of https://github.com/florianhartig/DHARMa/issues/43), which corresponds to re-form = ~0 in lme4

Author(s)

Florian Hartig

See Also

[getObservedResponse](), [getSimulations](), [getRefit](), [getFixedEffects]()

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)

# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])

getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

getFixedEffects *Extract fixed effects of a supported model*

Description

A wrapper to extract fixed effects of a supported model

Usage

```
getFixedEffects(object, ...)

## Default S3 method:
getFixedEffects(object, ...)

## S3 method for class 'MixMod'
getFixedEffects(object, ...)
```

Arguments

object	a fitted model
...	additional parameters

See Also

getObservedResponse, getSimulations, getRefit, getFitted

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)

# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])

getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

getObservedResponse   *Get model response*

Description

Extract the response of a fitted model

Usage

```
getObservedResponse(object, ...)

## Default S3 method:
getObservedResponse(object, ...)

## S3 method for class 'HLfit'
getObservedResponse(object, ...)
```

Arguments

object	a fitted model
...	additional parameters

Details

The purpose of this function is to safely extract the observed response (dependent variable) of the fitted model classes

Author(s)

Florian Hartig

See Also

[getRefit](#), [getSimulations](#), [getFixedEffects](#), [getFitted](#)

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)

# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])

getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

getQuantile   *calculate quantiles*

Description

calculates residual quantiles from a given simulation

Usage

```
getQuantile(simulations, observed, integerResponse, method = c("PIT",
  "traditional"), rotation = NULL)
```

Arguments

`simulations`	a matrix with simulations from a fitted model. Rows = observations, columns = replicate simulations
`observed`	a vector with the observed data
`integerResponse`	
	is the response integer-valued. Only has an effect for method = "traditional"
`method`	the quantile randomization method used. See details
`rotation`	optional rotation of the residuals, either provided as a covariance matrix, or specify "estimated", in which case the residual covariance will be approximated by simulations. See comments in details

Details

The function calculates residual quantiles from the simulated data. For continuous distributions, this will simply the the value of the ecdf.

Randomization procedure for discrete data

For discrete data, there are two options implemented.

The current default (available since DHARMa 0.3.1) are probability integral transform (PIT-) residuals (Smith, 1985; Dunn & Smyth, 1996; see also see also Warton, et al., 2017).

Before DHARMa 0.3.1, a different randomization procedure was used, in which the a U(-0.5, 0.5) distribution was added on observations and simulations for discrete distributions. For a completely discrete distribution, the two procedures should deliver equivalent results, but the second method has the disadvantage that a) one has to know if the distribution is discrete (DHARMa tries to recognize this automatically), and b) that it leads to inefficiencies for some distributions such as the the Tweedie, which are partly continuous, partly discrete (see e.g. https://github.com/florianhartig/DHARMa/issues/168).

Rotation (optional)

The getQuantile function includes an additional option to rotate residuals. The purpose is to de-correlated residuals in case of residual autocorrelation. If the expected residual autocorrelation is known (e.h. when fitting gls type models), it can be provided as a covariance matrix. If that is note the case, the option "estimated" will try to estimate the covariance from the data simulated by the model. Note, however, that this approximation will tend to have considerable error and may be slow to compute for high-dimensional data.

References

Smith, J. Q. "Diagnostic checks of non-standard time series models." Journal of Forecasting 4.3 (1985): 283-291.

Dunn, P.K., & Smyth, G.K. (1996). Randomized quantile residuals. Journal of Computational and Graphical Statistics 5, 236-244.

Warton, David I., Loïc Thibaut, and Yi Alice Wang. "The PIT-trap—A "model-free" bootstrap procedure for inference about regression models with discrete, multivariate responses." PloS one 12.7 (2017)

---

getRandomState   *Record and restore a random state*

Description

The aim of this function is to record, manipulate and restore a random state

Usage

```
getRandomState(seed = NULL)
```

Arguments

seed   seed argument to set.seed(), typically a number. Additional options: NULL = no
                    seed is set, but return includes function for restoring random seed. F = function
                    does nothing, i.e. neither seed is changed, nor does the returned function do
                    anything

Details

This function is intended for two (not mutually exclusive tasks)

a) record the current random state

b) change the current random state in a way that the previous state can be restored

Value

a list with various infos about the random state that after function execution, as well as a function
to restore the previous state before the function execution

Author(s)

Florian Hartig

Examples

```
set.seed(13)
runif(1)

# testing the function in standard settings
currentSeed = .Random.seed
x = getRandomState(123)
runif(1)
x$restoreCurrent()
all(.Random.seed == currentSeed)

# if no seed was set in env, this will also be restored
```

```
rm(.Random.seed) # now, there is no random seed
x = getRandomState(123)
exists(".Random.seed")  # TRUE
runif(1)
x$restoreCurrent()
exists(".Random.seed") # False
runif(1) # re-create a seed

# with seed = false
currentSeed = .Random.seed
x = getRandomState(FALSE)
runif(1)
x$restoreCurrent()
all(.Random.seed == currentSeed)

# with seed = NULL
currentSeed = .Random.seed
x = getRandomState(NULL)
runif(1)
x$restoreCurrent()
all(.Random.seed == currentSeed)
```

---

getRefit *Get model refit*

Description

Wrapper to refit a fitted model

checks if the fitted model excluded NA values

Usage

```
getRefit(object, newresp, ...)

## Default S3 method:
getRefit(object, newresp, ...)

## S3 method for class 'lm'
getRefit(object, newresp, ...)

## S3 method for class 'glmmTMB'
getRefit(object, newresp, ...)

## S3 method for class 'HLfit'
getRefit(object, newresp, ...)

## S3 method for class 'MixMod'
getRefit(object, newresp, ...)
```

Arguments

object	a fitted model
newresp	the new response that should be used to refit the model
...	additional parameters to be passed on to the refit or update class that is used to refit the model

Details

The purpose of this wrapper is to standardize the refit of a model. The behavior of this function depends on the supplied model. When available, it uses the refit method, otherwise it will use update. For glmmTMB: since version 1.0, glmmTMB has a refit function, but this didn't work, so I switched back to this implementation, which is a hack based on the update function.

Checks if the fitted model excluded NA values

Author(s)

Florian Hartig

See Also

[getObservedResponse](#), [getSimulations](#), [getFixedEffects](#)

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)

# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])

getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

getResiduals   *Get model residuals*

Description

Wrapper to get the residuals of a fitted model

Usage

```
getResiduals(object, ...)

## S3 method for class 'MixMod'
getResiduals(object, ...)
```

Arguments

object	a fitted model
...	additional parameters to be passed on, usually to the residual function of the respective model class

Details

The purpose of this wrapper is to standardize extract the model residuals. Similar to some other functions, a key question is whether to calculate those conditional or unconditional on the fitted REs.

Author(s)

Florian Hartig

See Also

[getObservedResponse](), [getSimulations](), [getRefit](), [getFixedEffects](), [getFitted]()

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)
```

```
# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])

getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

getSimulations   *Get model simulations*

Description

Wrapper to simulate from a fitted model

Usage

```
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## Default S3 method:
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## S3 method for class 'negbin'
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## S3 method for class 'gam'
getSimulations(object, nsim = 1, type = c("normal", "refit"),
  mgcViz = TRUE, ...)

## S3 method for class 'lmerMod'
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## S3 method for class 'glmmTMB'
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## S3 method for class 'HLfit'
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)

## S3 method for class 'MixMod'
getSimulations(object, nsim = 1, type = c("normal", "refit"), ...)
```

Arguments

object	a fitted model
nsim	number of simulations
type	if simulations should be prepared for getQuantile or for refit
...	additional parameters to be passed on, usually to the simulate function of the respective model class
mgcViz	whether simulations should be created with mgcViz (if mgcViz is available)

Details

The purpose of this wrapper for for the simulate function is to return the simulations from a model in a standardized way

The purpose of this function is to wrap or implement the simulate function of different model classes and thus return simulations from fitted models in a standardized way.

Note: GLMM and other regression packages often differ in how simulations are produced, and which parameters can be used to modify this behavior.

One important difference is how to modifiy which hierarchical levels are held constant, and which are re-simulated. In lme4, this is controlled by the re.form argument (see lme4::simulate.merMod). For other packages, please consort the help.

If the model was fit with weights and the respective model class does not include the weights in the simulations, getSimulations will throw a warning. The background is if weights are used on the likelihood directly, then what is fitted is effectively a pseudo likelihood, and there is no way to directly simulate from the specified likelihood. Whether or not residuals can be used in this case depends very much on what is tested and how weights are used. I'm sorry to say that it is hard to give a general recommendation, you have to consult someone that understands how weights are processed in the respective model class.

Value

a matrix with simulations

Author(s)

Florian Hartig

See Also

getObservedResponse, getRefit, getFixedEffects, getFitted

Examples

```
testData = createData(sampleSize = 400, family = gaussian())

fittedModel <- lm(observedResponse ~ Environment1 , data = testData)

# response that was used to fit the model
getObservedResponse(fittedModel)

# predictions of the model for these points
getFitted(fittedModel)

# extract simulations from the model as matrix
getSimulations(fittedModel, nsim = 2)

# extract simulations from the model for refit (often requires different structure)
x = getSimulations(fittedModel, nsim = 2, type = "refit")

getRefit(fittedModel, x[[1]])
```

```
getRefit(fittedModel, getObservedResponse(fittedModel))
```

---

hist.DHARMa   *Histogram of DHARMa residuals*

Description

The function produces a histogram from a DHARMa output

Usage

```
## S3 method for class 'DHARMa'
hist(x, breaks = seq(-0.02, 1.02, len = 53),
  col = c("red", rep("lightgrey", 50), "red"),
  main = "Hist of DHARMa residuals",
  xlab = "Residuals (outliers are marked red)", cex.main = 1, ...)
```

Arguments

x	a DHARMa simulation output (class DHARMa)
breaks	breaks for hist() function
col	col for hist bars
main	plot main
xlab	plot xlab
cex.main	plot cex.main
...	other arguments to be passed on to hist

See Also

[plotSimulatedResiduals](), [plotResiduals]()

Examples

```
testData = createData(sampleSize = 200, family = poisson(),
                      randomEffectVariance = 1, numGroups = 10)
fittedModel <- glm(observedResponse ~ Environment1,
                   family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

######### main plotting function #############

# for all functions, quantreg = T will be more
# informative, but slower

plot(simulationOutput, quantreg = FALSE)
```

```
############# Distribution #####################

plotQQunif(simulationOutput = simulationOutput,
           testDispersion = FALSE,
           testUniformity = FALSE,
           testOutliers = FALSE)

hist(simulationOutput )

############# residual plots ###############

# rank transformation, using a simulationOutput
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE)

# smooth scatter plot - usually used for large datasets, default for n > 10000
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE, smoothScatter = TRUE)

# residual vs predictors, using explicit values for pred, residual
plotResiduals(simulationOutput, form = testData$Environment1,
              quantreg = FALSE)

# if pred is a factor, or if asFactor = T, will produce a boxplot
plotResiduals(simulationOutput, form = testData$group)

# All these options can also be provided to the main plotting function

# If you want to plot summaries per group, use
simulationOutput = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput, quantreg = FALSE)
# we see one residual point per RE
```

---

hurricanes	*Hurricanes*

Description

A data set on hurricane strength and fatalities in the US between 1950 and 2012. The data originates from the study by Jung et al., PNAS, 2014, who claim that the masculinity / femininity of a hurricane name has a causal effect on fatalities, presumably through a different perception of danger caused by the names.

Format

A 'data.frame': 93 obs. of 14 variables

Year Year of the hurricane (1950-2012)

Name Name of the hurricane

MasFem Masculinity-femininity rating of the hurricane's name in the range 1 = very masculine, 11 = very feminine

MinPressure_before Minimum air pressure (909-1002)

Minpressure_Updated_2014 Updated minimum air pressure (909-1003)

Gender_MF Binary gender (male/female) based on MasFem (male = 0, female = 1)

Category Strength of the hurricane in categories (1:7). (1 = not at all, 7 = very intense)

alldeaths Deaths occured (1:256)

NDAM normalized damage in millions (1:75.000). The raw (dollar) amounts of property damage caused by hurricanes were obtained, and the unadjusted dollar amounts were normalized to 2013 monetary values by adjusting them to inflation, wealth and population density

Elapsed_Yrs elapsed since the occurrence of hurricanes (1:63)

Source MWR/wikipedia ()

ZMasFem scaled (MasFem)

ZMinPressure_A scaled (Minpressure_Updated_2014)

ZNDAM scaled (NDAM) ...

References

Jung, K., Shavitt, S., Viswanathan, M., & Hilbe, J. M. (2014). Female hurricanes are deadlier than male hurricanes. Proceedings of the National Academy of Sciences, 111(24), 8782-8787.

Examples

```
## Not run:
# Loading hurricanes dataset

library(DHARMa)

data(hurricanes)
str(hurricanes)

# this is the model fit by Jung et al.
library(glmmTMB)
originalModelGAM = glmmTMB(alldeaths ~ scale(MasFem) *
                              (scale(Minpressure_Updated_2014) + scale(NDAM)),
                          data = hurricanes, family = nbinom2)

# no significant deviation in the general DHARMa plot
res <- simulateResiduals(originalModelGAM)
plot(res)

# but residuals ~ NDAM looks funny, which was pointed
# out by Bob O'Hara in a blog post after publication of the paper
plotResiduals(res, hurricanes$NDAM)

# we also find temporal autocorrelation
res2 = recalculateResiduals(res, group = hurricanes$Year)
testTemporalAutocorrelation(res2, time = unique(hurricanes$Year))
```

```
# task: try to address these issues - in many instances, this will
# make the MasFem predictor n.s.

## End(Not run)
```

---

outliers   *Return outliers*

Description

Returns the outliers of a DHARMa object

Usage

```
outliers(object, lowerQuantile = 0, upperQuantile = 1,
  return = c("index", "logical"))
```

Arguments

object	an object with simulated residuals created by simulateResiduals
lowerQuantile	lower threshold for outliers. Default is zero = outside simulation envelope
upperQuantile	upper threshold for outliers. Default is 1 = outside simulation envelope
return	wheter to return an indices of outliers or a logical vector

Details

First of all, note that the standard definition of outlier in the DHARMa plots and outlier tests is an observation that is outside the simulation envelope. How far outside that is depends a lot on how many simulations you do. If you have 100 data points and to 100 simulations, you would expect to have one "outlier" on average, even with a perfectly fitting model. This is in fact what the outlier test tests.

Thus, keep in mind that for a small number of simulations, outliers are mostly a technical term: these are points that are outside our simulations, but we don't know how far away they are.

If you are seriously interested in HOW FAR outside the expected distribution a data point is, you should increase the number of simulations in simulateResiduals to be sure to get the tail of the data distribution correctly. In this case, it may make sense to adjust lowerQuantile and upperQuantile, e.g. to 0.025, 0.975, which would define outliers as values outside the central 95% of the distribution.

Also, note that outliers are particularly concerning if they have a strong influence on the model fit. One could test the influence, for example, by removing them from the data, or by some meausures of leverage, e.g. generalisations for Cook's distance as in Pinho, L. G. B., Nobre, J. S., & Singer, J. M. (2015). Cook's distance for generalized linear mixed models. Computational Statistics & Data Analysis, 82, 126–136. doi:10.1016/j.csda.2014.08.008. At the moment, however, no such function is provided in DHARMa.

---

plot.DHARMa   *DHARMa standard residual plots*

Description

This S3 function creates standard plots for the simulated residuals contained in an object of class DHARMa, using [plotQQunif](left panel) and [plotResiduals](right panel)

Usage

```
## S3 method for class 'DHARMa'
plot(x, title = "DHARMa residual", ...)
```

Arguments

x   an object of class DHARMa with simulated residuals created by [simulateResiduals](link)

title   The title for both panels (plotted via mtext, outer = T)

...   further options for [plotResiduals](link). Consider in particular parameters quantreg,
             rank and asFactor. xlab, ylab and main cannot be changed when using plot.DHARMa,
             but can be changed when using plotResiduals.

Details

The function creates a plot with two panels. The left panel is a uniform qq plot (calling [plotQQunif](link)), and the right panel shows residuals against predicted values (calling [plotResiduals](link)), with outliers highlighted in red.

Very briefly, we would expect that a correctly specified model shows:

a) a straight 1-1 line, as well as n.s. of the displayed tests in the qq-plot (left) -> evidence for an the correct overall residual distribution (for more details on the interpretation of this plot, see [plotQQunif](link))

b) visual homogeneity of residuals in both vertical and horizontal direction, as well as n.s. of quantile tests in the res ~ predictor plot (for more details on the interpretation of this plot, see [plotResiduals](link))

Deviations from these expectations can be interpreted similar to a linear regression. See the vignette for detailed examples.

Note that, unlike [plotResiduals](link), plot.DHARMa command uses the default rank = T.

See Also

[plotResiduals](link), [plotQQunif](link)

Examples

```
testData = createData(sampleSize = 200, family = poisson(),
                      randomEffectVariance = 1, numGroups = 10)
fittedModel <- glm(observedResponse ~ Environment1,
                   family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

######### main plotting function #############

# for all functions, quantreg = T will be more
# informative, but slower

plot(simulationOutput, quantreg = FALSE)

#############  Distribution  #####################

plotQQunif(simulationOutput = simulationOutput,
           testDispersion = FALSE,
           testUniformity = FALSE,
           testOutliers = FALSE)

hist(simulationOutput )

#############  residual plots  ###############

# rank transformation, using a simulationOutput
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE)

# smooth scatter plot - usually used for large datasets, default for n > 10000
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE, smoothScatter = TRUE)

# residual vs predictors, using explicit values for pred, residual
plotResiduals(simulationOutput, form = testData$Environment1,
              quantreg = FALSE)

# if pred is a factor, or if asFactor = T, will produce a boxplot
plotResiduals(simulationOutput, form = testData$group)

# All these options can also be provided to the main plotting function

# If you want to plot summaries per group, use
simulationOutput = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput, quantreg = FALSE)
# we see one residual point per RE
```

---

plot.DHARMaBenchmark   *Plots DHARMa benchmarks*

Description

The function plots the result of an object of class DHARMaBenchmark, created by runBenchmarks

Usage

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

Arguments

x   object of class DHARMaBenchmark, created by runBenchmarks

...   parameters to pass to the plot function

Details

The function will create two types of plots, depending on whether the run contains only a single value (or no value) of the control parameter, or whether a vector of control values was provided.

If a single or no value of the control parameter was provided, the function will create box plots of the estimated p-values, with the number of significant p-values plotted to the left

If a control parameter was provided, the function will plot the proportion of significant p-values against the control parameter, with 95% CIs based based on the performed replicates displayed as confidence bands

See Also

runBenchmarks

```
plotConventionalResiduals
```
                    Conventional residual plot

Description

Convenience function to draw conventional residual plots

Usage

```
plotConventionalResiduals(fittedModel)
```

Arguments

fittedModel   a fitted model object

---

plotQQunif *Quantile-quantile plot for a uniform distribution*

Description

The function produces a uniform quantile-quantile plot from a DHARMa output

Usage

```
plotQQunif(simulationOutput, testUniformity = T, testOutliers = T,
  testDispersion = T, ...)
```

Arguments

simulationOutput
: a DHARMa simulation output (class DHARMa)

testUniformity
: if T, the function [testUniformity](#) will be called and the result will be added to the plot

testOutliers
: if T, the function [testOutliers](#) will be called and the result will be added to the plot

testDispersion
: if T, the function [testDispersion](#) will be called and the result will be added to the plot

...
: arguments to be passed on to [qqunif](#)

Details

the function calls qqunif from the R package gap to create a quantile-quantile plot for a uniform distribution, and overlays tests for particular distributional problems as specified.

See Also

[plotSimulatedResiduals](#), [plotResiduals](#)

Examples

```
testData = createData(sampleSize = 200, family = poisson(),
                      randomEffectVariance = 1, numGroups = 10)
fittedModel <- glm(observedResponse ~ Environment1,
                   family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

######### main plotting function #############

# for all functions, quantreg = T will be more
# informative, but slower

plot(simulationOutput, quantreg = FALSE)
```

```
############# Distribution ######################

plotQQunif(simulationOutput = simulationOutput,
           testDispersion = FALSE,
           testUniformity = FALSE,
           testOutliers = FALSE)

hist(simulationOutput )

############# residual plots ###############

# rank transformation, using a simulationOutput
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE)

# smooth scatter plot - usually used for large datasets, default for n > 10000
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE, smoothScatter = TRUE)

# residual vs predictors, using explicit values for pred, residual
plotResiduals(simulationOutput, form = testData$Environment1,
              quantreg = FALSE)

# if pred is a factor, or if asFactor = T, will produce a boxplot
plotResiduals(simulationOutput, form = testData$group)

# All these options can also be provided to the main plotting function

# If you want to plot summaries per group, use
simulationOutput = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput, quantreg = FALSE)
# we see one residual point per RE
```

---

plotResiduals   *Generic res ~ pred scatter plot with spline or quantile regression on*
                                 top

Description

The function creates a generic residual plot with either spline or quantile regression to highlight patterns in the residuals. Outliers are highlighted in red.

Usage

```
plotResiduals(simulationOutput, form = NULL, quantreg = NULL, rank = T,
  asFactor = NULL, smoothScatter = NULL, quantiles = c(0.25, 0.5, 0.75),
  ...)
```

Arguments

simulationOutput	on object, usually a DHARMa object, from which residual values can be extracted. Alternatively, a vector with residuals or a fitted model can be provided, which will then be transformed into a DHARMa object.
form	optional predictor against which the residuals should be plotted. Default is to used the predicted(simulationOutput)
quantreg	whether to perform a quantile regression based on [testQuantiles](#) or a smooth spline around the mean. Default NULL chooses T for nObs < 2000, and F otherwise.
rank	if T, the values provided in form will be rank transformed. This will usually make patterns easier to spot visually, especially if the distribution of the predictor is skewed. If form is a factor, this has no effect.
asFactor	should a numeric predictor provided in form be treated as a factor. Default is to choose this for < 10 unique values, as long as enough predictions are available to draw a boxplot.
smoothScatter	if T, a smooth scatter plot will plotted instead of a normal scatter plot. This makes sense when the number of residuals is very large. Default NULL chooses T for nObs < 10000, and F otherwise.
quantiles	for a quantile regression, which quantiles should be plotted
...	additional arguments to plot / boxplot.

Details

The function plots residuals against a predictor (by default against the fitted value, extracted from the DHARMa object, or any other predictor).

Outliers are highlighted in red (for information on definition and interpretation of outliers, see [testOutliers](#)).

To provide a visual aid in detecting deviations from uniformity in y-direction, the plot function calculates an (optional) quantile regression of the residuals, by default for the 0.25, 0.5 and 0.75 quantiles. As the residuals should be uniformly distributed for a correctly specified model, the theoretical expectations for these regressions are straight lines at 0.25, 0.5 and 0.75, which are displayed as dashed black lines on the plot. Some deviations from these expectations are to be expected by chance, however, even for a perfect model, especially if the sample size is small. The function therefore tests if deviation of the fitted quantile regression from the expectation is significant, using [testQuantiles](#). If so, the significant quantile regression will be highlighted as red, and a warning will be displayed in the plot.

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. This is default for n > 2000.

If form is a factor, a boxplot will be plotted instead of a scatter plot. The distribution for each factor level should be uniformly distributed, so the box should go from 0.25 to 0.75, with the median line at 0.5 (within-group). To test if deviations from those expecations are significant, KS-tests per group and a Levene test for homogeneity of variances is performed. See [testCategorical](#) for details.

Value

if quantile tests are performed, the function returns them invisibly.

Note

if nObs > 10000, the scatter plot is replaced by graphics::smoothScatter

See Also

plotQQunif, testQuantiles, testOutliers

Examples

```
testData = createData(sampleSize = 200, family = poisson(),
                      randomEffectVariance = 1, numGroups = 10)
fittedModel <- glm(observedResponse ~ Environment1,
                   family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

######### main plotting function #############

# for all functions, quantreg = T will be more
# informative, but slower

plot(simulationOutput, quantreg = FALSE)

#############  Distribution  #####################

plotQQunif(simulationOutput = simulationOutput,
           testDispersion = FALSE,
           testUniformity = FALSE,
           testOutliers = FALSE)

hist(simulationOutput )

#############  residual plots  ###############

# rank transformation, using a simulationOutput
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE)

# smooth scatter plot - usually used for large datasets, default for n > 10000
plotResiduals(simulationOutput, rank = TRUE, quantreg = FALSE, smoothScatter = TRUE)

# residual vs predictors, using explicit values for pred, residual
plotResiduals(simulationOutput, form = testData$Environment1,
              quantreg = FALSE)

# if pred is a factor, or if asFactor = T, will produce a boxplot
plotResiduals(simulationOutput, form = testData$group)

# All these options can also be provided to the main plotting function
```

```
# If you want to plot summaries per group, use
simulationOutput = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput, quantreg = FALSE)
# we see one residual point per RE
```

---

plotSimulatedResiduals

DHARMa standard residual plots

Description

DEPRECATED, use plot() instead

Usage

```
plotSimulatedResiduals(simulationOutput, ...)
```

Arguments

simulationOutput

an object with simulated residuals created by [simulateResiduals](#)

...   further options for [plotResiduals](#). Consider in particular parameters quantreg, rank and asFactor. xlab, ylab and main cannot be changed when using plotSimulatedResiduals, but can be changed when using plotResiduals.

Note

This function is deprecated. Use [plot.DHARMa](#)

See Also

[plotResiduals](#), [plotQQunif](#)

---

print.DHARMa   *Print simulated residuals*

Description

Print simulated residuals

Usage

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

Arguments

x   an object with simulated residuals created by simulateResiduals

...   optional arguments for compatibility with the generic function, no function im-
                 plemented

---

recalculateResiduals   *Recalculate residuals with grouping*

Description

The purpose of this function is to recalculate scaled residuals per group, based on the simulations
done by simulateResiduals

Usage

```
recalculateResiduals(simulationOutput, group = NULL, aggregateBy = sum,
  sel = NULL, seed = 123, method = c("PIT", "traditional"),
  rotation = NULL)
```

Arguments

simulationOutput

                 an object with simulated residuals created by simulateResiduals

group   group of each data point

aggregateBy   function for the aggregation. Default is sum. This should only be changed if
                 you know what you are doing. Note in particular that the expected residual
                 distribution might not be flat any more if you choose general functions, such as
                 sd etc.

sel   an optional vector for selecting the data to be aggregated

seed   the random seed to be used within DHARMa. The default setting, recommended
                 for most users, is keep the random seed on a fixed value 123. This means that
                 you will always get the same randomization and thus teh same result when run-
                 ning the same code. NULL = no new seed is set, but previous random state
                 will be restored after simulation. FALSE = no seed is set, and random state will
                 not be restored. The latter two options are only recommended for simulation
                 experiments. See vignette for details.

method   the quantile randomization method used. The two options implemented at the
                 moment are probability integral transform (PIT-) residuals (current default), and
                 the "traditional" randomization procedure, that was used in DHARMa until ver-
                 sion 0.3.0. For details, see getQuantile

rotation   optional rotation of the residual space to remove residual autocorrelation. See
                 details in simulateResiduals, section *residual auto-correlation* for an extended
                 explanation, and getQuantile for syntax.

Value

an object of class DHARMa, similar to what is returned by [simulateResiduals](), but with additional outputs for the new grouped calculations. Note that the relevant outputs are 2x in the object, the first is the grouped calculations (which is returned by $name access), and later another time, under identical name, the original output. Moreover, there is a function 'aggregateByGroup', which can be used to aggregate predictor variables in the same way as the variables calculated here

Examples

```
library(lme4)

testData = createData(sampleSize = 100, overdispersion = 0.5, family = poisson())
fittedModel <- glmer(observedResponse ~ Environment1 + (1|group),
                     family = "poisson", data = testData)

simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# standard plot
plot(simulationOutput)

# one of the possible test, for other options see ?testResiduals / vignette
testDispersion(simulationOutput)

# the calculated residuals can be accessed via
residuals(simulationOutput)

# transform residuals to other pdf, see ?residuals.DHARMa for details
residuals(simulationOutput, quantileFunction = qnorm, outlierValues = c(-7,7))

# get residuals that are outside the simulation envelope
outliers(simulationOutput)

# calculating aggregated residuals per group
simulationOutput2 = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput2, quantreg = FALSE)

# calculating residuals only for subset of the data
simulationOutput3 = recalculateResiduals(simulationOutput, sel = testData$group == 1 )
plot(simulationOutput3, quantreg = FALSE)
```

---

residuals.DHARMa   *Return residuals of a DHARMa simulation*

Description

Return residuals of a DHARMa simulation

Usage

```
## S3 method for class 'DHARMa'
residuals(object, quantileFunction = NULL, outlierValues = NULL, ...)
```

Arguments

object   an object with simulated residuals created by simulateResiduals

quantileFunction

                  optional - a quantile function to transform the uniform 0/1 scaling of DHARMa to another distribution

outlierValues   if a quantile function with infinite support (such as dnorm) is used, residuals that are 0/1 are mapped to -Inf / Inf. outlierValues allows to convert -Inf / Inf values to an optional min / max value.

...   optional arguments for compatibility with the generic function, no function implemented

Details

the function accesses the slot $scaledResiduals in a fitted DHARMa object, and optionally transforms the standard DHARMa quantile residuals (which have a uniform distribution) to a particular pdf.

Note

some of the papers on simulated quantile residuals transforming the residuals (which are natively uniform) back to a normal distribution. I presume this is because of the larger familiarity of most users with normal residuals. Personally, I never considered this desirable, for the reasons explained in https://github.com/florianhartig/DHARMa/issues/39, but with this function, I wanted to give users the option to plot normal residuals if they so wish.

Examples

```
library(lme4)

testData = createData(sampleSize = 100, overdispersion = 0.5, family = poisson())
fittedModel <- glmer(observedResponse ~ Environment1 + (1|group),
                     family = "poisson", data = testData)

simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# standard plot
plot(simulationOutput)

# one of the possible test, for other options see ?testResiduals / vignette
testDispersion(simulationOutput)

# the calculated residuals can be accessed via
residuals(simulationOutput)
```

```
# transform residuals to other pdf, see ?residuals.DHARMa for details
residuals(simulationOutput, quantileFunction = qnorm, outlierValues = c(-7,7))

# get residuals that are outside the simulation envelope
outliers(simulationOutput)

# calculating aggregated residuals per group
simulationOutput2 = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput2, quantreg = FALSE)

# calculating residuals only for subset of the data
simulationOutput3 = recalculateResiduals(simulationOutput, sel = testData$group == 1 )
plot(simulationOutput3, quantreg = FALSE)
```

---

runBenchmarks *Benchmark calculations*

Description

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

Usage

```
runBenchmarks(calculateStatistics, controlValues = NULL, nRep = 10,
  alpha = 0.05, parallel = FALSE, ...)
```

Arguments

calculateStatistics

> the statistics to be benchmarked. Should return one value, or a vector of values. If controlValues are given, must accept a parameter control

controlValues optionally, a vector with a control parameter (e.g. to vary the strength of a problem the test should be specific to). See help for an example

nRep number of replicates per level of the controlValues

alpha significance level

parallel whether to use parallel computations. Possible values are F, T (sets the cores automatically to number of available cores -1), or an integer number for the number of cores that should be used for the cluster

... additional parameters to calculateStatistics

Value

A object with list structure of class DHARMaBenchmark. Contains an entry simulations with a matrix of simulations, and an entry summaries with an list of summaries (significant (T/F), mean, p-value for KS-test uniformity). Can be plotted with plot.DHARMaBenchmark

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.

Author(s)

Florian Hartig

See Also

plot.DHARMaBenchmark

Examples

```
# define a function that will run a simulation and return a number of statistics, typically p-values
returnStatistics <- function(control = 0){
  testData = createData(sampleSize = 20, family = poisson(), overdispersion = control,
                        randomEffectVariance = 0)
  fittedModel <- glm(observedResponse ~ Environment1, data = testData, family = poisson())
  res <- simulateResiduals(fittedModel = fittedModel, n = 250)
 out <- c(testUniformity(res, plot = FALSE)$p.value, testDispersion(res, plot = FALSE)$p.value)
  return(out)
}

# testing a single return
returnStatistics()

# running benchmark for a fixed simulation, increase nRep for sensible results
out = runBenchmarks(returnStatistics, nRep = 5)

# plotting results depend on whether a vector or a single value is provided for control
plot(out)

# running benchmark with varying control values
# out = runBenchmarks(returnStatistics, controlValues = c(0,0.5,1), nRep = 100)
# plot(out)

# running benchmark can be done using parallel cores
# out = runBenchmarks(returnStatistics, nRep = 100, parallel = TRUE)
# out = runBenchmarks(returnStatistics, controlValues = c(0,0.5,1), nRep = 10, parallel = TRUE)

# Alternative plot function using vioplot, provides nicer pictures

# plot.DHARMaBenchmark <- function(x, ...){
#
#   if(length(x$controlValues)== 1){
#   vioplot::vioplot(x$simulations[,x$nSummaries:1], las = 2, horizontal = T, side = "right",
#                    areaEqual = F,
#                    main = "p distribution under H0",
#                    ylim = c(-0.15,1), ...)
#     abline(v = 1, lty = 2)
```

```
#      abline(v = c(0.05, 0), lty = 2, col = "red")
#      text(-0.1, x$nSummaries:1, labels = x$summaries$propSignificant[-1])
#
#   }else{
#      res = x$summaries$propSignificant
#      matplot(res$controlValues, res[,-1], type = "l",
#              main = "Power analysis", ylab = "Power", ...)
#      legend("bottomright", colnames(res[,-1]),
#              col = 1:x$nSummaries, lty = 1:x$nSummaries, lwd = 2)
#
#   }
# }
```

---

simulateResiduals   *Create simulated residuals*

Description

The function creates scaled residuals by simulating from the fitted model. Residuals can be extracted with `residuals.DHARMa`. See `testResiduals` for an overview of residual tests, `plot.DHARMa` for an overview of available plots.

Usage

```
simulateResiduals(fittedModel, n = 250, refit = F,
  integerResponse = NULL, plot = F, seed = 123, method = c("PIT",
  "traditional"), rotation = NULL, ...)
```

Arguments

fittedModel	a fitted model of a class supported by DHARMa
n	number of simulations. The smaller the number, the higher the stochastic error on the residuals. Also, for very small n, discretization artefacts can influence the tests. Default is 250, which is a relatively safe value. You can consider increasing to 1000 to stabilize the simulated values.
refit	if FALSE, new data will be simulated and scaled residuals will be created by comparing observed data with new data. If TRUE, 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 TRUE, 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 TRUE, `plotResiduals` will be directly run after the residuals have been calculated

seed   the random seed to be used within DHARMa. The default setting, recommended
                  for most users, is keep the random seed on a fixed value 123. This means that
                  you will always get the same randomization and thus the same result when run-
                  ning the same code. NULL = no new seed is set, but previous random state
                  will be restored after simulation. FALSE = no seed is set, and random state will
                  not be restored. The latter two options are only recommended for simulation
                  experiments. See vignette for details.

method   for refit = F, the quantile randomization method used. The two options imple-
                  mented at the moment are probability integral transform (PIT-) residuals (cur-
                  rent default), and the "traditional" randomization procedure, that was used in
                  DHARMa until version 0.3.0. Refit = T will always use "traditional", respec-
                  tively of the value of method. For details, see getQuantile

rotation   optional rotation of the residual space prior to calculating the quantile residu-
                  als. The main purpose of this is to remove residual autocorrelation. See details
                  below, section *residual auto-correlation*, and help of getQuantile.

...   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, e.g. via re.form. Note that not all models support
                  syntax to specify conditional or unconditional simulations. See also details

Details

There are a number of important considerations when simulating from a more complex (hierarchi-
cal) model:

Re-simulating random effects / hierarchical structure: in a hierarchical model, we have several
stochastic processes 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 state-space models, similar considerations apply.

In such a situation, 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. This is often referred to as a conditional
simuation. For controlling how many levels should be re-simulated, the simulateResidual 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 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.
In particular dispersion tests may produce different results when switching from conditional to
unconditional simulations, and often the conditional simulation is more sensitive.

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 = FALSE (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 = TRUE, 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. I advise against using this method per default (see more comments in the vignette), unless you are really sure that you need it.

Residuals per group: In many situations, it can be useful to look at residuals per group, e.g. to see how much the model over / underpredicts per plot, year or subject. To do this, use `recalculateResiduals`, together with a grouping variable (see also help)

Transformation to other distributions: DHARMa calculates residuals for which the theoretical expectation (assuming a correctly specified model) is uniform. To transform this residuals to another distribution (e.g. so that a correctly specified model will have normal residuals) see `residuals.DHARMa`.

Integer responses: this is only relevant if method = "traditional", in which case it activates the randomization of the residuals. Usually, this does not need to be changed, as DHARMa will try to automatically if the fitted model has an integer or discrete distribution via the family argument. However, in some cases the family does not allow to uniquely identify the distribution type. For example, a tweedie distribution can be inter or continuous. Therefore, DHARMa will additionally check the simulation results for repeated values, and will change the distribution type if repeated values are found (a message is displayed in this case).

Residual auto-correlation: a common problem is residual autocorrelation. Spatial, temporal and phylogenetic autocorrelation can be tested with testSpatialAutocorrelation and testTemporalAutocorrelation. If simulations are unconditional, residual correlations will be maintained, even if the autocorrelation is addressed by an appropriate CAR structure. This may be a problem, because autocorrelation may create apparently systematic patterns in plots or tests such as testUniformity. To reduce this problem, either simulate conditional on fitted correlated REs, or you could try to rotate residuals via the rotation parameter (the latter will likely only work in approximately linear models). See getQuantile for details on the rotation.

Value

An S3 class of type "DHARMa". Implemented S3 functions include `plot.DHARMa`, `print.DHARMa` and `residuals.DHARMa`. For other functions that can be used on a DHARMa object, see section "See Also" below.

See Also

`testResiduals`, `plotResiduals`, `recalculateResiduals`, `outliers`

Examples

```
library(lme4)

testData = createData(sampleSize = 100, overdispersion = 0.5, family = poisson())
fittedModel <- glmer(observedResponse ~ Environment1 + (1|group),
                     family = "poisson", data = testData)

simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# standard plot
```

```
plot(simulationOutput)

# one of the possible test, for other options see ?testResiduals / vignette
testDispersion(simulationOutput)

# the calculated residuals can be accessed via
residuals(simulationOutput)

# transform residuals to other pdf, see ?residuals.DHARMa for details
residuals(simulationOutput, quantileFunction = qnorm, outlierValues = c(-7,7))

# get residuals that are outside the simulation envelope
outliers(simulationOutput)

# calculating aggregated residuals per group
simulationOutput2 = recalculateResiduals(simulationOutput, group = testData$group)
plot(simulationOutput2, quantreg = FALSE)

# calculating residuals only for subset of the data
simulationOutput3 = recalculateResiduals(simulationOutput, sel = testData$group == 1 )
plot(simulationOutput3, quantreg = FALSE)
```

---

testCategorical   *Test for categorical dependencies*

Description

This function tests if there are probles in a res ~ group structure. It performs two tests: test for within-group uniformity, and test for between-group homogeneity of variances

Usage

```
testCategorical(simulationOutput, catPred, quantiles = c(0.25, 0.5, 0.75),
  plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via simulateResiduals for supported models or by createDHARMa for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

catPred   a categorical predictor with the same dimensions as the residuals in simulationOutput

quantiles   whether to draw the quantile lines.

plot   if T, the function will create an additional plot

Details

The function tests for two common problems: are residuals within each group distributed according to model assumptions, and is the variance between group heterogeneous.

The test for within-group uniformity is performed via multipe KS-tests, with adjustment of p-values for multiple testing. If the plot is drawn, problematic groups are highlighted in red, and a corresponding message is displayed in the plot.

The test for homogeneity of variances is done with a Levene test. A significant p-value means that group variances are not constant. In this case, you should consider modelling variances, e.g. via ~dispformula in glmmTMB.

Author(s)

Florian Hartig

See Also

[testResiduals](), [testUniformity](), [testOutliers](), [testDispersion](), [testZeroInflation](), [testGeneric](), [testTemporalAutocorrelation](), [testSpatialAutocorrelation](), [testQuantiles](), [testCategorical]()

Examples

```
testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# the plot function runs 4 tests
# i) KS test i) Dispersion test iii) Outlier test iv) quantile test
plot(simulationOutput, quantreg = TRUE)

# testResiduals tests distribution, dispersion and outliers
# testResiduals(simulationOutput)

####### Individual tests #######

# KS test for correct distribution of residuals
testUniformity(simulationOutput)

# KS test for correct distribution within and between groups
testCategorical(simulationOutput, testData$group)

# Dispersion test - for details see ?testDispersion
testDispersion(simulationOutput) # tests under and overdispersion

# Outlier test (number of observations outside simulation envelope)
# Use type = "boostrap" for exact values, see ?testOutliers
testOutliers(simulationOutput, type = "binomial")

# testing zero inflation
testZeroInflation(simulationOutput)
```

```
# testing generic summaries
countOnes <- function(x) sum(x == 1)  # testing for number of 1s
testGeneric(simulationOutput, summary = countOnes) # 1-inflation
testGeneric(simulationOutput, summary = countOnes, alternative = "less") # 1-deficit

means <- function(x) mean(x) # testing if mean prediction fits
testGeneric(simulationOutput, summary = means)

spread <- function(x) sd(x) # testing if mean sd fits
testGeneric(simulationOutput, summary = spread)
```

---

testDispersion   *DHARMa dispersion tests*

Description

This function performs simulation-based tests for over/underdispersion. If type = "DHARMa" (default and recommended), simulation-based dispersion tests are performed. Their behavior differs depending on whether simulations are done with refit = F, or refit = T, and whether data is simulated conditional (e.g. re.form ~0 in lme4) (see below). If type = "PearsonChisq", a chi2 test on Pearson residuals is performed.

Usage

```
testDispersion(simulationOutput, alternative = c("two.sided", "greater",
  "less"), plot = T, type = c("DHARMa", "PearsonChisq"), ...)
```

Arguments

simulationOutput

> an object of class DHARMa, either created via [simulateResiduals](#) for supported models or by [createDHARMa](#) for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

alternative   a character string specifying whether the test should test if observations are "greater", "less" or "two.sided" compared to the simulated null hypothesis. Greater corresponds to testing only for overdispersion. It is recommended to keep the default setting (testing for both over and underdispersion)

plot   whether to provide a plot for the results

type   which test to run. Default is DHARMa, other options are PearsonChisq (see details)

...   arguments to pass on to [testGeneric](#)

Details

Over / underdispersion means that the observed data is more / less dispersed than expected under the fitted model. There is no unique way to test for dispersion problems, and there are a number of different dispersion tests implemented in various R packages. This function implements several dispersion tests.

Simulation-based dispersion tests (type == "DHARMa")

If type = "DHARMa" (default and recommended), simulation-based dispersion tests are performed. Their behavior differs depending on whether simulations are done with refit = F, or refit = T, and whether data is simulated conditional (e.g. re.form ~0 in lme4)

If refit = F, the function uses `testGeneric` to compare the variance of the observed raw residuals (i.e. var(observed - predicted), displayed as a red line) against the variance of the simulated residuals (i.e. var(observed - simulated), histogram). The variances are scaled to the mean simulated variance. A significant ratio > 1 indicates overdispersion, a significant ratio < 1 underdispersion.

If refit = T, the function compares the approximate deviance (via squared pearson residuals) with the same quantity from the models refitted with simulated data. Applying this is much slower than the previous alternative. Given the computational cost, I would suggest that most users will be satisfied with the standard dispersion test.

Important: for either refit = T or F, the results of type = "DHARMa" dispersion test will differ depending on whether simulations are done conditional (= conditional on fitted random effects) or unconditional (= REs are re-simulated). How to change between conditional or unconditional simulations is discussed in `simulateResiduals`. The general default in DHARMa is to use unconditional simulations, because this has advantages in other situations, but dispersion tests for models with strong REs specifically may increase substantially in power / sensitivity when switching to conditional simulations. I therefore recommend checking dispersion with conditional simulations if supported by the used regression package.

Analytical dispersion tests (type == "PearsonChisq")

This is the test described in https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#overdispersion, identical to performance::check_overdispersion. Works only if the fitted model provides df.residual and Pearson residuals.

The test statistics is biased to lower values under quite general conditions, and will therefore tend to test significant for underdispersion. It is recommended to use this test only for overdispersion, i.e. use alternative == "greater". Also, obviously, it requires that Pearson residuals are available for the chosen model, which will not be the case for all models / packages.

Note

For particular model classes / situations, there may be more powerful and thus preferable over the DHARMa test. The advantage of the DHARMa test is that it directly targets the spread of the data (unless other tests such as dispersion/df, which essentially measure fit and may thus be triggered by problems other than dispersion as well), and it makes practically no assumptions about the fitted model, other than the availability of simulations.

Author(s)

Florian Hartig

See Also

[testResiduals](#), [testUniformity](#), [testOutliers](#), [testDispersion](#), [testZeroInflation](#), [testGeneric](#), [testTemporalAutocorrelation](#), [testSpatialAutocorrelation](#), [testQuantiles](#), [testCategorical](#)

Examples

```
library(lme4)
set.seed(123)

testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 1)
fittedModel <- glmer(observedResponse ~ Environment1 + (1|group),
                     family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# default DHARMa dispersion test - simulation-based
testDispersion(simulationOutput)
testDispersion(simulationOutput, alternative = "less", plot = FALSE) # only underdispersion
testDispersion(simulationOutput, alternative = "greater", plot = FALSE) # only oversispersion

# for mixed models, the test is usually more powerful if residuals are calculated
# conditional on fitted REs
simulationOutput <- simulateResiduals(fittedModel = fittedModel, re.form = NULL)
testDispersion(simulationOutput)

# DHARMa also implements the popular Pearson-chisq test that is also on the glmmWiki by Ben Bolker
# The issue with this test is that it requires the df of the model, which are not well defined
# for GLMMs. It is biased towards underdispersion, with bias getting larger with the number
# of RE groups. In doubt, only test for overdispersion
testDispersion(simulationOutput, type = "PearsonChisq", alternative = "greater")

# if refit = T, a different test on simulated Pearson residuals will calculated (see help)
simulationOutput2 <- simulateResiduals(fittedModel = fittedModel, refit = TRUE, seed = 12, n = 20)
testDispersion(simulationOutput2)

# often useful to test dispersion per group (in particular for binomial data, see vignette)
simulationOutputAggregated = recalculateResiduals(simulationOutput2, group = testData$group)
testDispersion(simulationOutputAggregated)
```

---

testGeneric	*Generic simulation test of a summary statistic*

Description

This function tests if a user-defined summary differs when applied to simulated / observed data.

Usage

```
testGeneric(simulationOutput, summary, alternative = c("two.sided",
  "greater", "less"), plot = T,
  methodName = "DHARMa generic simulation test")
```

Arguments

simulationOutput

an object of class DHARMa, either created via `simulateResiduals` for sup-
ported models or by `createDHARMa` for simulations created outside DHARMa,
or a supported model. Providing a supported model directly is discouraged,
because simulation settings cannot be changed in this case.

summary   a function that can be applied to simulated / observed data. See examples below

alternative   a character string specifying whether the test should test if observations are
                "greater", "less" or "two.sided" compared to the simulated null hypothesis

plot   whether to plot the simulated summary

methodName   name of the test (will be used in plot)

Details

This function tests if a user-defined summary differs when applied to simulated / observed data.
the function can easily be remodeled to apply summaries on the residuals, by simply defining f =
function(x) summary (x - predictions), as done in `testDispersion`

Note

The function that you supply is applied on the data as it is represented in your fitted model, which
may not always correspond to how you think. This is important in particular when you use k/n
binomial data, and want to test for 1-inflation. As an example, if have k/20 observations, and you
provide your data via cbind (y, y-20), you have to test for 20-inflation (because this is how the data
is represented in the model). However, if you provide data via y/20, and weights = 20, you should
test for 1-inflation. In doubt, check how the data is internally represented in model.frame(model),
or via simulate(model)

Author(s)

Florian Hartig

See Also

`testResiduals`, `testUniformity`, `testOutliers`, `testDispersion`, `testZeroInflation`, `testGeneric`,
`testTemporalAutocorrelation`, `testSpatialAutocorrelation`, `testQuantiles`, `testCategorical`

Examples

```
testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)
```

```
# the plot function runs 4 tests
# i) KS test i) Dispersion test iii) Outlier test iv) quantile test
plot(simulationOutput, quantreg = TRUE)

# testResiduals tests distribution, dispersion and outliers
# testResiduals(simulationOutput)

####### Individual tests #######

# KS test for correct distribution of residuals
testUniformity(simulationOutput)

# KS test for correct distribution within and between groups
testCategorical(simulationOutput, testData$group)

# Dispersion test - for details see ?testDispersion
testDispersion(simulationOutput) # tests under and overdispersion

# Outlier test (number of observations outside simulation envelope)
# Use type = "boostrap" for exact values, see ?testOutliers
testOutliers(simulationOutput, type = "binomial")

# testing zero inflation
testZeroInflation(simulationOutput)

# testing generic summaries
countOnes <- function(x) sum(x == 1)  # testing for number of 1s
testGeneric(simulationOutput, summary = countOnes) # 1-inflation
testGeneric(simulationOutput, summary = countOnes, alternative = "less") # 1-deficit

means <- function(x) mean(x) # testing if mean prediction fits
testGeneric(simulationOutput, summary = means)

spread <- function(x) sd(x) # testing if mean sd fits
testGeneric(simulationOutput, summary = spread)
```

---

testOutliers   *Test for outliers*

Description

This function tests if the number of observations outside the simulatio envelope are larger or smaller than expected

Usage

```
testOutliers(simulationOutput, alternative = c("two.sided", "greater",
  "less"), margin = c("both", "upper", "lower"), type = c("default",
  "bootstrap", "binomial"), nBoot = 100, plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via [simulateResiduals](#) for supported models or by [createDHARMa](#) for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

alternative   a character string specifying whether the test should test if observations are "greater", "less" or "two.sided" (default) compared to the simulated null hypothesis

margin   whether to test for outliers only at the lower, only at the upper, or both sides (default) of the simulated data distribution

type   either default, bootstrap or binomial. See details

nBoot   number of boostrap replicates. Only used ot type = "bootstrap"

plot   if T, the function will create an additional plot

Details

DHARMa residuals are created by simulating from the fitted model, and comparing the simulated values to the observed data. It can occur that all simulated values are higher or smaller than the observed data, in which case they get the residual value of 0 and 1, respectively. I refer to these values as simulation outliers, or simply outliers.

Because no data was simulated in the range of the observed value, we don't know "how strongly" these values deviate from the model expectation, so the term "outlier" should be used with a grain of salt. It is not a judgment about the magnitude of the residual deviation, but simply a dichotomous sign that we are outside the simulated range. Moreover, the number of outliers will decrease as we increase the number of simulations.

To test if the outliers are a concern, testOutliers implements 2 options (bootstrap, binomial), which can be chosen via the parameter "type". The third option (default) chooses bootstrap for integer-valued distribubtions with nObs < 500, and else binomial.

The binomial test considers that under the null hypothesis that the model is correct, and for continuous distributions (i.e. data and the model distribution are identical and continous), the probability that a given observation is higher than all simulations is 1/(nSim +1), and binomial distributed. The testOutlier function can test this null hypothesis via type = "binomial". In principle, it would be nice if we could extend this idea to integer-valued distributions, which are randomized via the PIT procedure (see [simulateResiduals](#)), the rate of "true" outliers is more difficult to calculate, and in general not 1/(nSim +1). The testOutlier function implements a small tweak that calculates the rate of residuals that are closer than 1/(nSim+1) to the 0/1 border, which roughly occur at a rate of nData /(nSim +1). This approximate value, however, is generally not exact, and may be particularly off non-bounded integer-valued distributions (such as Poisson or neg binom).

For this reason, the testOutlier function implements an alternative procedure that uses the bootstrap to generate a simulation-based expectation for the outliers. It is recommended to use the bootstrap for integer-valued distributions (and integer-valued only, because it has no advantage for continuous distributions, ideally with reasonably high values of nSim and nBoot (I recommend at least 1000 for both). Because of the high runtime, however, this option is switched off for type = default when nObs > 500.

Both binomial or bootstrap generate a null expectation, and then test for an excess or lack of outliers. Per default, testOutliers() looks for both, so if you get a significant p-value, you have to check if you have to many or too few outliers. An excess of outliers is to be interpreted as too many values outside the simulation envelope. This could be caused by overdispersion, or by what we classically call outliers. A lack of outliers would be caused, for example, by underdispersion.

Author(s)

Florian Hartig

See Also

[testResiduals](#), [testUniformity](#), [testOutliers](#), [testDispersion](#), [testZeroInflation](#), [testGeneric](#), [testTemporalAutocorrelation](#), [testSpatialAutocorrelation](#), [testQuantiles](#), [testCategorical](#)

---

testOverdispersion   *Simulated overdisperstion tests*

Description

Simulated overdisperstion tests

Usage

```
testOverdispersion(simulationOutput, ...)
```

Arguments

simulationOutput

an object of class DHARMa with simulated quantile residuals, either created via [simulateResiduals](#) or by [createDHARMa](#) for simulations created outside DHARMa

...   additional arguments to [testDispersion](#)

Details

Deprecated, switch your code to using the [testDispersion](#) function

---

testOverdispersionParametric

Parametric overdisperstion tests

Description

Parametric overdisperstion tests

Usage

```
testOverdispersionParametric(...)
```

Arguments

...	arguments will be ignored, the parametric tests is no longer recommend

Details

Deprecated, switch your code to using the testDispersion function.

---

testPDistribution   *Plot distribution of p-values*

Description

Plot distribution of p-values

Usage

```
testPDistribution(x, plot = T,
  main = "p distribution \n expected is flat at 1", ...)
```

Arguments

x	vector of p values
plot	should the values be plotted
main	title for the plot
...	additional arguments to hist

Author(s)

Florian Hartig

---

testQuantiles *Test for quantiles*

Description

This function tests

Usage

```
testQuantiles(simulationOutput, predictor = NULL, quantiles = c(0.25, 0.5,
  0.75), plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via [simulateResiduals](#) for supported models or by [createDHARMa](#) for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

predictor   an optional predictor variable to be used, instead of the predicted response (default)

quantiles   the quantiles to be tested

plot   if T, the function will create an additional plot

Details

The function fits quantile regressions (via package qgam) on the residuals, and compares their location to the expected location (because of the uniform distributionm, the expected location is 0.5 for the 0.5 quantile).

A significant p-value for the splines means the fitted spline deviates from a flat line at the expected location (p-values of intercept and spline are combined via Benjamini & Hochberg adjustment to control the FDR)

The p-values of the splines are combined into a total p-value via Benjamini & Hochberg adjustment to control the FDR.

Author(s)

Florian Hartig

See Also

[testResiduals](#), [testUniformity](#), [testOutliers](#), [testDispersion](#), [testZeroInflation](#), [testGeneric](#), [testTemporalAutocorrelation](#), [testSpatialAutocorrelation](#), [testQuantiles](#), [testCategorical](#)

Examples

```
testData = createData(sampleSize = 200, overdispersion = 0.0, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1, family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# run the quantile test
x = testQuantiles(simulationOutput)
x # the test shows a combined p-value, corrected for multiple testing

## Not run:
# accessing results of the test
x$pvals # pvalues for the individual quantiles
x$qgamFits # access the fitted quantile regression
summary(x$qgamFits[[1]]) # summary of the first fitted quantile

# possible to test user-defined quantiles
testQuantiles(simulationOutput, quantiles = c(0.7))

#  example with missing environmental predictor
fittedModel <- glm(observedResponse ~ 1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)
testQuantiles(simulationOutput, predictor = testData$Environment1)

plot(simulationOutput)
plotResiduals(simulationOutput)

## End(Not run)
```

---

testResiduals   *DHARMa general residual test*

Description

Calls both uniformity and dispersion test

Usage

```
testResiduals(simulationOutput, plot = T)
```

Arguments

simulationOutput

> an object of class DHARMa, either created via simulateResiduals for supported models or by createDHARMa for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

plot   if T, plots functions of the tests are called

Details

This function is a wrapper for the various test functions implemented in DHARMa. Currently, this function calls the testUniformity and the testDispersion functions. All other tests (see list below) have to be called by hand.

Author(s)

Florian Hartig

See Also

testResiduals, testUniformity, testOutliers, testDispersion, testZeroInflation, testGeneric, testTemporalAutocorrelation, testSpatialAutocorrelation, testQuantiles, testCategorical

Examples

```
testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# the plot function runs 4 tests
# i) KS test i) Dispersion test iii) Outlier test iv) quantile test
plot(simulationOutput, quantreg = TRUE)

# testResiduals tests distribution, dispersion and outliers
# testResiduals(simulationOutput)


####### Individual tests #######

# KS test for correct distribution of residuals
testUniformity(simulationOutput)

# KS test for correct distribution within and between groups
testCategorical(simulationOutput, testData$group)

# Dispersion test - for details see ?testDispersion
testDispersion(simulationOutput) # tests under and overdispersion

# Outlier test (number of observations outside simulation envelope)
# Use type = "boostrap" for exact values, see ?testOutliers
testOutliers(simulationOutput, type = "binomial")

# testing zero inflation
testZeroInflation(simulationOutput)

# testing generic summaries
countOnes <- function(x) sum(x == 1)  # testing for number of 1s
testGeneric(simulationOutput, summary = countOnes) # 1-inflation
testGeneric(simulationOutput, summary = countOnes, alternative = "less") # 1-deficit

means <- function(x) mean(x) # testing if mean prediction fits
testGeneric(simulationOutput, summary = means)
```

```
spread <- function(x) sd(x) # testing if mean sd fits
testGeneric(simulationOutput, summary = spread)
```

---

testSimulatedResiduals

Residual tests

Description

Residual tests

Usage

```
testSimulatedResiduals(simulationOutput)
```

Arguments

simulationOutput

an object of class DHARMa, either created via `simulateResiduals` for supported models or by `createDHARMa` for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

Details

Deprecated, switch your code to using the `testResiduals` function

Author(s)

Florian Hartig

---

testSpatialAutocorrelation

Test for distance-based (spatial, phylogenetic or similar) autocorrelation

---

Description

This function performs a Moran's I test for distance-based (spatial, phylogenetic or similar) autocorrelation on the calculated quantile residuals

Usage

```
testSpatialAutocorrelation(simulationOutput, x = NULL, y = NULL,
  distMat = NULL, alternative = c("two.sided", "greater", "less"),
  plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via simulateResiduals for supported models or by createDHARMa for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

x   the x coordinate, in the same order as the data points. Must be specified unless distMat is provided.

y   the y coordinate, in the same order as the data points. Must be specified unless distMat is provided.

distMat   optional distance matrix. If not provided, euclidean distances based on x and y will be calculated. See details for explanation

alternative   a character string specifying whether the test should test if observations are "greater", "less" or "two.sided" compared to the simulated null hypothesis

plot   whether to plot output

Details

The function performs Moran.I test from the package ape on the DHARMa residuals. If a distance matrix (distMat) is provided, calculations will be based on this distance matrix, and x,y coordinates will only used for the plotting (if provided). If distMat is not provided, the function will calculate the euclidean distances between x,y coordinates, and test Moran.I based on these distances.

Testing for spatial autocorrelation requires unique x,y values - if you have several observations per location, either use the recalculateResiduals function to aggregate residuals per location, or extract the residuals from the fitted object, and plot / test each of them independently for spatially repeated subgroups (a typical scenario would repeated spatial observation, in which case one could plot / test each time step separately for temporal autocorrelation). Note that the latter must be done by hand, outside testSpatialAutocorrelation.

Note

Standard DHARMa simulations from models with (temporal / spatial / phylogenetic) conditional autoregressive terms will still have the respective temporal / spatial / phylogenetic correlation in the DHARMa residuals, unless the package you are using is modelling the autoregressive terms as explicit REs and is able to simulate conditional on the fitted REs. This has two consequences

1. If you check the residuals for such a model, they will still show significant autocorrelation, even if the model fully accounts for this structure.

2. Because the DHARMa residuals for such a model are not statistically independent any more, other tests (e.g. dispersion, uniformity) may have inflated type I error, i.e. you will have a higher likelihood of spurious residual problems.

There are three (non-exclusive) routes to address these issues when working with spatial / temporal / other autoregressive models:

1. Simulate conditional on the fitted CAR structures (see conditional simulations in the help of simulateResiduals)

2. Rotate simulations prior to residual calculations (see parameter rotation in simulateResiduals)

3. Use custom tests / plots that explicitly compare the correlation structure in the simulated data to the correlation structure in the observed data.

Author(s)

Florian Hartig

See Also

testResiduals, testUniformity, testOutliers, testDispersion, testZeroInflation, testGeneric, testTemporalAutocorrelation, testSpatialAutocorrelation, testQuantiles, testCategorical

Examples

```
testData = createData(sampleSize = 40, family = gaussian())
fittedModel <- lm(observedResponse ~ Environment1, data = testData)
res = simulateResiduals(fittedModel)

# Standard use
testSpatialAutocorrelation(res, x =  testData$x, y = testData$y)

# Alternatively, one can provide a distance matrix
dM = as.matrix(dist(cbind(testData$x, testData$y)))
testSpatialAutocorrelation(res, distMat = dM)

# You could add a spatial variogram via
# library(gstat)
# dat = data.frame(res = residuals(res), x = testData$x, y = testData$y)
# coordinates(dat) = ~x+y
# vario = variogram(res~1, data = dat, alpha=c(0,45,90,135))
# plot(vario, ylim = c(-1,1))

# if there are multiple observations with the same x values,
# create first ar group with unique values for each location
# then aggregate the residuals per location, and calculate
# spatial autocorrelation on the new group

# modifying x, y, so that we have the same location per group
# just for completeness
testData$x = as.numeric(testData$group)
testData$y = as.numeric(testData$group)

# calculating x, y positions per group
groupLocations = aggregate(testData[, 6:7], list(testData$group), mean)

# calculating residuals per group
res2 = recalculateResiduals(res, group = testData$group)

# running the spatial test on grouped residuals
testSpatialAutocorrelation(res2, groupLocations$x, groupLocations$y)
```

```
# careful when using REs to account for spatially clustered (but not grouped)
# data. this originates from https://github.com/florianhartig/DHARMa/issues/81

# Assume our data is divided into clusters, where observations are close together
# but not at the same point, and we suspect that observations in clusters are
# autocorrelated

clusters = 100
subsamples = 10
size = clusters * subsamples

testData = createData(sampleSize = size, family = gaussian(), numGroups = clusters )
testData$x  = rnorm(clusters)[testData$group] + rnorm(size, sd = 0.01)
testData$y  = rnorm(clusters)[testData$group] + rnorm(size, sd = 0.01)

# It's a good idea to use a RE to take out the cluster effects. This accounts
# for the autocorrelation within clusters

library(lme4)
fittedModel <- lmer(observedResponse ~ Environment1 + (1|group), data = testData)

# DHARMa default is to re-simulted REs - this means spatial pattern remains
# because residuals are still clustered

res = simulateResiduals(fittedModel)
testSpatialAutocorrelation(res, x =  testData$x, y = testData$y)

# However, it should disappear if you just calculate an aggregate residuals per cluster
# Because at least how the data are simulated, cluster are spatially independent

res2 = recalculateResiduals(res, group = testData$group)
testSpatialAutocorrelation(res2,
                           x =  aggregate(testData$x, list(testData$group), mean)$x,
                           y = aggregate(testData$y, list(testData$group), mean)$x)

# For lme4, it's also possible to simulated residuals conditional on fitted
# REs (re.form). Conditional on the fitted REs (i.e. accounting for the clusters)
# the residuals should now be indepdendent. The remaining RSA we see here is
# probably due to the RE shrinkage

res = simulateResiduals(fittedModel, re.form = NULL)
testSpatialAutocorrelation(res, x =  testData$x, y = testData$y)
```

---

testTemporalAutocorrelation

Test for temporal autocorrelation

Description

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

Usage

```
testTemporalAutocorrelation(simulationOutput, time,
  alternative = c("two.sided", "greater", "less"), plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via simulateResiduals for supported models or by createDHARMa for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

time   the time, in the same order as the data points.

alternative   a character string specifying whether the test should test if observations are "greater", "less" or "two.sided" compared to the simulated null hypothesis

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 be 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.

Testing for temporal autocorrelation requires unique time values - if you have several observations per time value, either use recalculateResiduals function to aggregate residuals per time step, or extract the residuals from the fitted object, and plot / test each of them independently for temporally repeated subgroups (typical choices would be location / subject etc.). Note that the latter must be done by hand, outside testTemporalAutocorrelation.

Note

Standard DHARMa simulations from models with (temporal / spatial / phylogenetic) conditional autoregressive terms will still have the respective temporal / spatial / phylogenetic correlation in the DHARMa residuals, unless the package you are using is modelling the autoregressive terms as explicit REs and is able to simulate conditional on the fitted REs. This has two consequences

1. If you check the residuals for such a model, they will still show significant autocorrelation, even if the model fully accounts for this structure.

2. Because the DHARMa residuals for such a model are not statistically independent any more, other tests (e.g. dispersion, uniformity) may have inflated type I error, i.e. you will have a higher likelihood of spurious residual problems.

There are three (non-exclusive) routes to address these issues when working with spatial / temporal / other autoregressive models:

1. Simulate conditional on the fitted CAR structures (see conditional simulations in the help of simulateResiduals)

2. Rotate simulations prior to residual calculations (see parameter rotation in simulateResiduals)

3. Use custom tests / plots that explicitly compare the correlation structure in the simulated data to the correlation structure in the observed data.

Author(s)

Florian Hartig

See Also

testResiduals, testUniformity, testOutliers, testDispersion, testZeroInflation, testGeneric, testTemporalAutocorrelation, testSpatialAutocorrelation, testQuantiles, testCategorical

Examples

```
testData = createData(sampleSize = 40, family = gaussian(),
                      randomEffectVariance = 0)
fittedModel <- lm(observedResponse ~ Environment1, data = testData)
res = simulateResiduals(fittedModel)

# Standard use
testTemporalAutocorrelation(res, time =  testData$time)

# If you have several observations per time step, e.g.
# because you have several locations, you will have to
# aggregate

timeSeries1 = createData(sampleSize = 40, family = gaussian(),
                         randomEffectVariance = 0)
timeSeries1$location = 1
timeSeries2 = createData(sampleSize = 40, family = gaussian(),
                         randomEffectVariance = 0)
timeSeries2$location = 2
testData = rbind(timeSeries1, timeSeries2)

fittedModel <- lm(observedResponse ~ Environment1, data = testData)
res = simulateResiduals(fittedModel)

# Will not work because several residuals per time
# testTemporalAutocorrelation(res, time = testData$time)

# aggregating residuals by time
res = recalculateResiduals(res, group = testData$time)
testTemporalAutocorrelation(res, time = unique(testData$time))

# testing only subgroup location 1, could do same with loc 2
res = recalculateResiduals(res, sel = testData$location == 1)
testTemporalAutocorrelation(res, time = unique(testData$time))
```

```
# example to demonstrate problems with strong temporal correlations and
# how to possibly remove them by rotating residuals

## Not run:

set.seed(1)
C <- exp(-as.matrix(dist(seq(0,50,by=.5))))
obs <- as.numeric(mvtnorm::rmvnorm(1,sigma=C))

opar <- par(mfrow = c(1,2))
image(C, main = "Specified autocorrelation (covariance)")
plot(obs, type = "l", main = "Time series")
par(opar)

# calculate standard DHARMa residuals

## simulations from the model:
x = replicate(1000, as.numeric(mvtnorm::rmvnorm(1,sigma=C)))

res <- createDHARMa(x, obs, integerResponse = F)
plot(res)
testTemporalAutocorrelation(res, time = 1:length(res$scaledResiduals))

# calculated rotated DHARMa residuals to remove temporal correlation
# this only works if the autocorrelation is homogeneous / stationary
res <- createDHARMa(x, obs, integerResponse = F, rotation = C)
testUniformity(res)
testTemporalAutocorrelation(res, time = 1:length(res$scaledResiduals))

# the same, but with a covariance based on simulations
res <- createDHARMa(x, obs, integerResponse = F, rotation = "estimated")
testUniformity(res)
testTemporalAutocorrelation(res, time = 1:length(res$scaledResiduals))



## End(Not run)
```

---

testUniformity   *Test for overall uniformity*

Description

This function tests the overall uniformity of the simulated residuals in a DHARMa object

Usage

```
testUniformity(simulationOutput, alternative = c("two.sided", "less",
  "greater"), plot = T)
```

Arguments

simulationOutput

an object of class DHARMa, either created via [simulateResiduals](#) for supported models or by [createDHARMa](#) for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

alternative   a character string specifying whether the test should test if observations are "greater", "less" or "two.sided" compared to the simulated null hypothesis. See [ks.test](#) for details

plot   if T, plots calls [plotQQunif](#) as well

Details

The function applies a [ks.test](#) for uniformity on the simulated residuals.

Author(s)

Florian Hartig

See Also

[testResiduals](#), [testUniformity](#), [testOutliers](#), [testDispersion](#), [testZeroInflation](#), [testGeneric](#), [testTemporalAutocorrelation](#), [testSpatialAutocorrelation](#), [testQuantiles](#), [testCategorical](#)

Examples

```
testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# the plot function runs 4 tests
# i) KS test i) Dispersion test iii) Outlier test iv) quantile test
plot(simulationOutput, quantreg = TRUE)

# testResiduals tests distribution, dispersion and outliers
# testResiduals(simulationOutput)

####### Individual tests #######

# KS test for correct distribution of residuals
testUniformity(simulationOutput)

# KS test for correct distribution within and between groups
testCategorical(simulationOutput, testData$group)

# Dispersion test - for details see ?testDispersion
testDispersion(simulationOutput) # tests under and overdispersion

# Outlier test (number of observations outside simulation envelope)
# Use type = "boostrap" for exact values, see ?testOutliers
```

```
testOutliers(simulationOutput, type = "binomial")

# testing zero inflation
testZeroInflation(simulationOutput)

# testing generic summaries
countOnes <- function(x) sum(x == 1)  # testing for number of 1s
testGeneric(simulationOutput, summary = countOnes) # 1-inflation
testGeneric(simulationOutput, summary = countOnes, alternative = "less") # 1-deficit

means <- function(x) mean(x) # testing if mean prediction fits
testGeneric(simulationOutput, summary = means)

spread <- function(x) sd(x) # testing if mean sd fits
testGeneric(simulationOutput, summary = spread)
```

---

testZeroInflation   *Tests for zero-inflation*

Description

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

Usage

```
testZeroInflation(simulationOutput, ...)
```

Arguments

simulationOutput

> an object of class DHARMa, either created via `simulateResiduals` for supported models or by `createDHARMa` for simulations created outside DHARMa, or a supported model. Providing a supported model directly is discouraged, because simulation settings cannot be changed in this case.

...   further arguments to `testGeneric`

Details

The plot shows the expected distribution of zeros against the observed values, the ratioObsSim shows observed vs. simulated zeros. A value < 1 means that the observed data has less zeros than expected, a value > 1 means that it has more zeros than expected (aka zero-inflation). Per default, the function tests both sides.

Some notes about common problems / questions:

- Zero-inflation tests after fitting the model are crucial to see if you have zero-inflation. Just because there are a lot of zeros doesn't mean you have zero-inflation, see Warton, D. I. (2005). Many zeros does not mean zero inflation: comparing the goodness-of-fit of parametric models to multivariate abundance data. Environmetrics 16(3), 275-289.

- That being said, zero-inflation tests are often not a reliable guide to decide wheter to add a zi term or not. In general, model structures should be decided on ideally a priori, if that is not possible via model selection techniques (AIC, BIC, WAIC, Bayes Factor). A zero-inflation test should only be run after that decision, and to validate the decision that was taken.

Note

This function is a wrapper for testGeneric, where the summary argument is set to function(x) sum(x == 0)

Author(s)

Florian Hartig

See Also

testResiduals, testUniformity, testOutliers, testDispersion, testZeroInflation, testGeneric, testTemporalAutocorrelation, testSpatialAutocorrelation, testQuantiles, testCategorical

Examples

```
testData = createData(sampleSize = 100, overdispersion = 0.5, randomEffectVariance = 0)
fittedModel <- glm(observedResponse ~ Environment1 , family = "poisson", data = testData)
simulationOutput <- simulateResiduals(fittedModel = fittedModel)

# the plot function runs 4 tests
# i) KS test i) Dispersion test iii) Outlier test iv) quantile test
plot(simulationOutput, quantreg = TRUE)

# testResiduals tests distribution, dispersion and outliers
# testResiduals(simulationOutput)

####### Individual tests #######

# KS test for correct distribution of residuals
testUniformity(simulationOutput)

# KS test for correct distribution within and between groups
testCategorical(simulationOutput, testData$group)

# Dispersion test - for details see ?testDispersion
testDispersion(simulationOutput) # tests under and overdispersion

# Outlier test (number of observations outside simulation envelope)
# Use type = "boostrap" for exact values, see ?testOutliers
testOutliers(simulationOutput, type = "binomial")

# testing zero inflation
testZeroInflation(simulationOutput)

# testing generic summaries
countOnes <- function(x) sum(x == 1)  # testing for number of 1s
```

```
testGeneric(simulationOutput, summary = countOnes) # 1-inflation
testGeneric(simulationOutput, summary = countOnes, alternative = "less") # 1-deficit

means <- function(x) mean(x) # testing if mean prediction fits
testGeneric(simulationOutput, summary = means)

spread <- function(x) sd(x) # testing if mean sd fits
testGeneric(simulationOutput, summary = spread)
```

---

transformQuantiles   *Transform quantiles to pdf (deprecated)*

Description

The purpose of this function was to transform the DHARMa quantile residuals (which have a uniform distribution) to a particular pdf. Since DHARMa 0.3.0, this functionality is integrated in the [residuals.DHARMa](#) function. Please switch to using this function.

Usage

```
transformQuantiles(res, quantileFunction = qnorm, outlierValue = 7)
```

Arguments

res   an object with simulated residuals created by [simulateResiduals](#)

quantileFunction

                 optional - a quantile function to transform the uniform 0/1 scaling of DHARMa
                 to another distribution

outlierValue   if a quantile function with infinite support (such as dnorm) is used, residuals that
                 are 0/1 are mapped to -Inf / Inf. outlierValues allows to convert -Inf / Inf values
                 to an optional min / max value.

Index