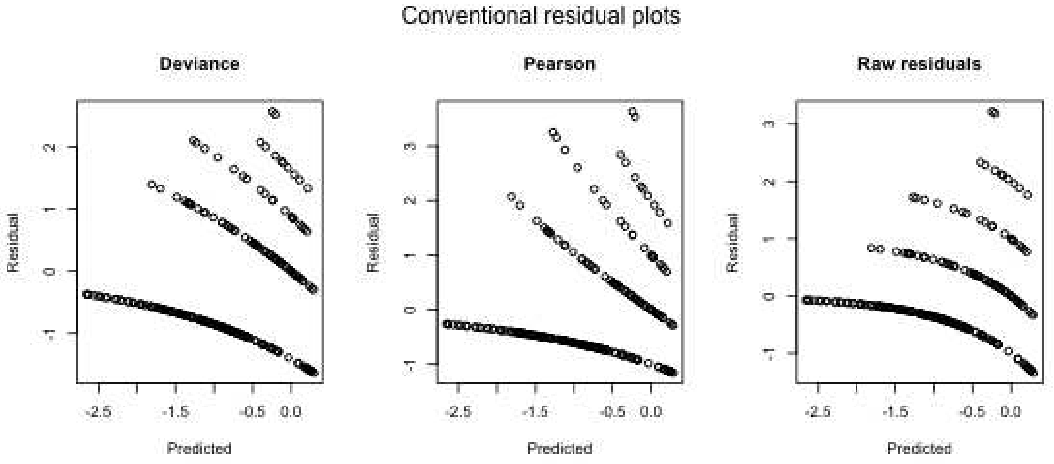
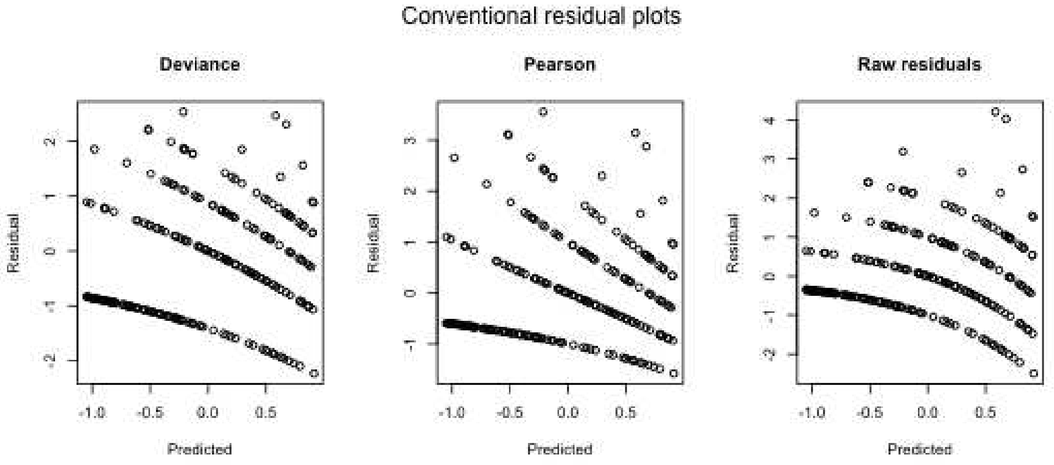
```
## Note that, since v0.1.6.2, DHARMa includes support for glmm
```

Motivation

Residual interpretation for generalized linear mixed models (GLMMs) is often problematic. As an example, here two Poisson GLMMs, one that is lacking a quadratic effect, and one that fits the data perfectly. I show three standard residuals diagnostics each. Which is the misspecified model?





Just for completeness - it was the first one. But don't get too excited if you got it right. Either you were lucky, or you noted that the first model seems a bit overdispersed (range of the Pearson residuals). But even when noting that, would you have added a quadratic effect, instead of adding an overdispersion correction? The point here is that misspecifications in GL(M)Ms cannot reliably be diagnosed with standard residual plots, and GLMMs are thus often not as thoroughly checked as LMs.

One reason why GL(M)Ms residuals are harder to interpret is that the expected distribution of the data changes with the fitted values. Reweighting with the expected variance, as done in Pearson residuals, or using deviance residuals, helps a bit, but does not lead to visually homogenous residuals even if the model is correctly specified. As a result, standard residual plots, when interpreted in the same way as for linear models, seem to show all kind of problems, such as non-normality, heteroscedasticity, even if the model is correctly specified. Questions on the R mailing lists and forums show that practitioners are regularly confused about whether such patterns in GL(M)M residuals are a problem or not.

But even experienced statistical analysts currently have few options to diagnose misspecification problems in GLMMs. In my experience, the current standard practice is to eyeball the residual plots for major misspecifications, potentially have a look at the random effect distribution, and then run a test for overdispersion, which is usually positive, after which the model is modified towards an overdispersed / zero-inflated distribution. This approach, however, has a number of problems, notably:

- Overdispersion often comes from missing or misspecified predictors. Standard residual plots make it difficult to test for residual patterns against the predictors to check for candidates.

- Not all overdispersion is the same. For count data, the negative binomial creates a different distribution than adding observation-level random effects to the Poisson. Once overdispersion is corrected, such violations of distributional assumptions are not detectable with standard overdispersion tests (because the tests only looks at total dispersion), and nearly impossible to see visually from standard residual plots.

- Dispersion frequently varies with predictors (heteroscedasticity). This can have a significant effect on the inference. While it is standard to tests for heteroscedasticity in linear regressions, heteroscedasticity is currently hardly ever tested for in GLMMs, although it is likely as frequent and influential.

- Moreover, if residuals are checked, they are usually checked conditional on the fitted random effect estimates. Thus, standard checks only check the final level of the random structure in a GLMM. One can perform extra checks on the random effects, but it is somewhat unsatisfactory that there is no check on the entire model structure.

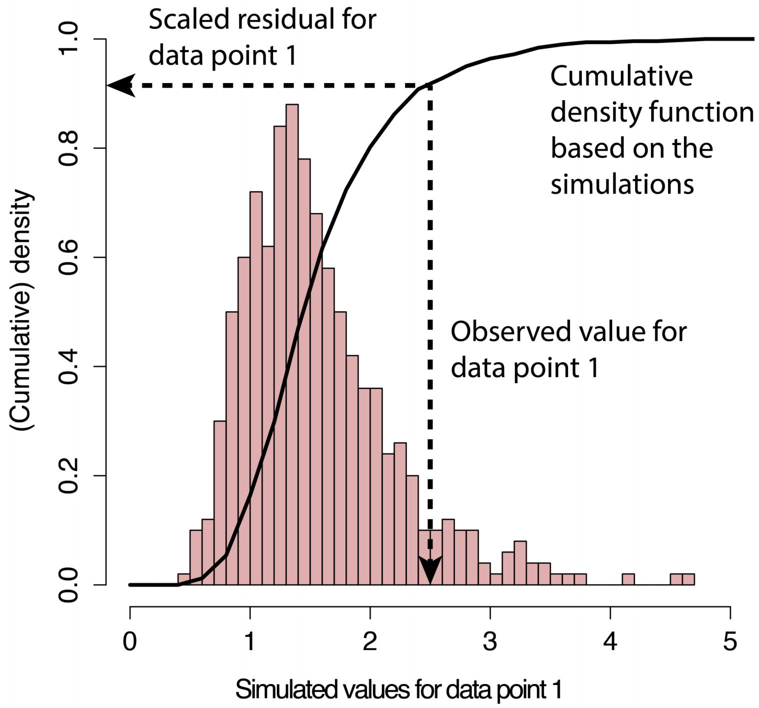
DHARMa aims at solving these problems by creating readily interpretable residuals for generalized linear (mixed) models that are standardized to values between 0 and 1, and that can be interpreted as intuitively as residuals for the linear model. This is achieved by a simulation-based approach, similar to the Bayesian p-value or the parametric bootstrap, that transforms the residuals to a standardized scale. The basic steps are:

1. Simulate new data from the fitted model for each observation.

2. For each observation, calculate the empirical cumulative density function for the simulated observations, which describes the possible values (and their probability) at the predictor combination of the observed value, assuming the fitted model is correct.

3. The residual is then defined as the value of the empirical density function at the value of the observed data, so a residual of 0 means that all simulated values are larger than the observed value, and a residual of 0.5 means half of the simulated values are larger than the observed value.

These steps are visualized in the following figure



The key idea for this definition is that, if the model is correctly specified, then the observed data should look like as if it was created from the fitted model. Hence, for a correctly specified model, all values of the cumulative distribution should appear with equal probability. That means we expect the distribution of the residuals to be flat, regardless of the model structure (Poisson, binomial, random effects and so on).

I currently prepare a more exact statistical justification for the approach in an accompanying paper, but if you must provide a reference in the meantime I would suggest citing

- Dunn, K. P., and Smyth, G. K. (1996). Randomized quantile residuals. Journal of Computational and Graphical Statistics 5, 1-10.
- Gelman, A. & Hill, J. Data analysis using regression and multilevel/hierarchical models Cambridge University Press, 2006

p.s.: DHARMa stands for “Diagnostics for HierArchical Regression Models” – which, strictly speaking, would make DHARM. But in German, Darm means intestines; plus, the meaning of DHARMa in Hinduism makes the current abbreviation so much more suitable for a package that tests whether your model is in harmony with your data:

> From Wikipedia, 28/08/16: In Hinduism, dharma signifies behaviours that are considered to be in accord with rta, the order that makes life and universe possible, and includes duties, rights, laws, conduct, virtues and ‘‘right way of living’’.

Workflow in DHARMa

Installing, loading and citing the package

If you haven't installed the package yet, either run

```
install.packages("DHARMa")
```

Or follow the instructions on https://github.com/florianhartig/DHARMa to install a development version.

Loading and citation

```
library(DHARMa)
citation("DHARMa")
```

```
##
## To cite package 'DHARMa' in publications use:
##
##   Florian Hartig (2018). DHARMa: Residual Diagnostics for
##   Hierarchical (Multi-Level / Mixed) Regression Models. R p
##   version 0.2.0. http://florianhartig.github.io/DHARMa/
##
## A BibTeX entry for LaTeX users is
##
##   @Manual{,
##     title = {DHARMa: Residual Diagnostics for Hierarchical
##     author = {Florian Hartig},
##     year = {2018},
##     note = {R package version 0.2.0},
##     url = {http://florianhartig.github.io/DHARMa/},
##   }
```

Calculating scaled residuals

The scaled (quantile) residuals are calculated with the simulateResiduals() function. The default number of simulations to run is 250, which proved to be a reasonable compromise between computation time and precision, but if high precision is desired, n should be raised to 1000 at least.

```
simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

What the function does is a) creating n new synthetic datasets by simulating from the fitted model, b) calculates the cumulative distribution of simulated values for each observed value, and c) returning the quantile value that corresponds to the observed value.

For example, a scaled residual value of 0.5 means that half of the simulated data are higher than the observed value, and half of them lower. A value of 0.99 would mean that nearly all simulated data are lower than the observed value. The minimum/maximum values for the residuals are 0 and 1.

The calculated residuals are stored in

```
simulationOutput$scaledResiduals
```

As discussed above, for a correctly specified model we would expect

- a uniform (flat) distribution of the overall residuals
- uniformity in y direction if we plot against any predictor.

Note: the expected uniform distribution is the only differences to the linear regression that one has to keep in mind when interpreting DHARMa residuals. If you cannot get used to this and you must have residuals that behave exactly like a linear regression, you can access a normal transformation of the residuals via

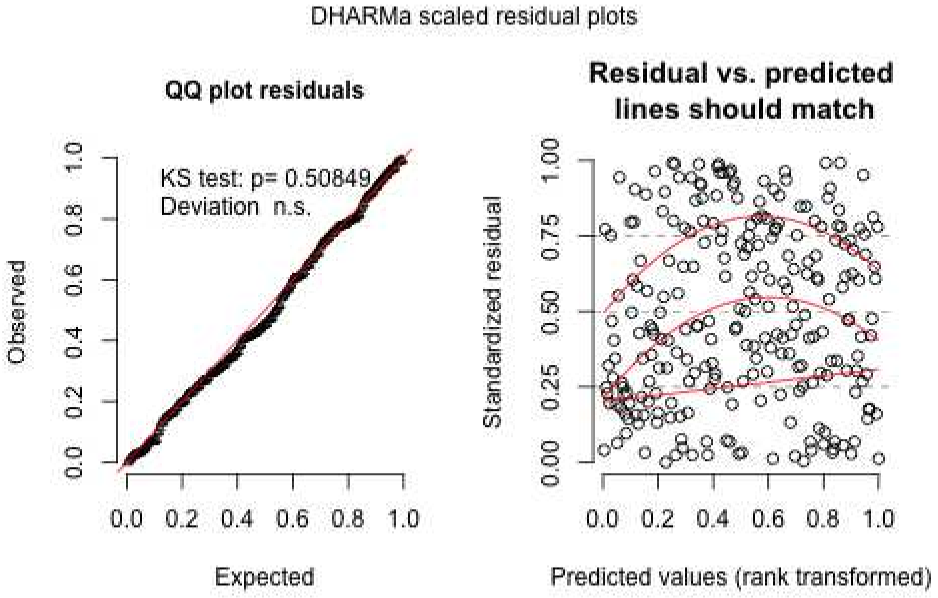
```
simulationOutput$scaledResidualsNormal
```

These normal residuals will behave exactly like the residuals of a linear regression. However, for reasons of a) numeric stability with low number of simulations and b) my conviction that it is much easier to visually detect deviations from uniformity than normality, I would STRONGLY advice against using this transformation.

Plotting the scaled residuals

We can get a visual impression of these properties with the plot.DHARMa() function

```
plot(simulationOutput)
```



which creates a qq-plot to detect overall deviations from the expected distribution, and a plot of the residuals against the predicted value.

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

The quantile regression can be very slow for large datasets. You can chose to use a simpler method with the option quantreg = F.

If you want to plot the residuals against other predictors (highly recommend), you can use the function

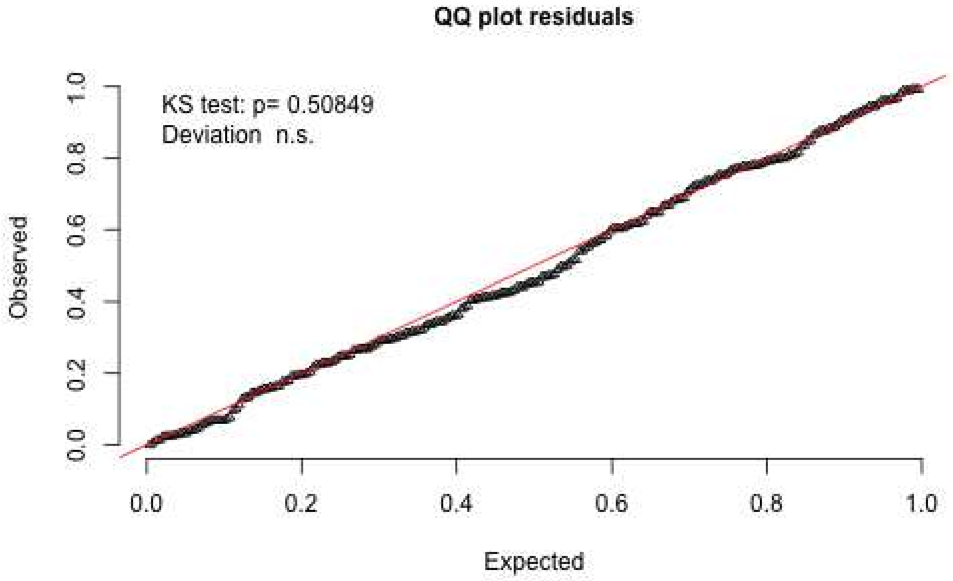
```
plotResiduals(YOURPREDICTOR, simulationOutput$scaledResiduals)
```

which does the same quantile plot as the main plotting function.

Formal goodness-of-fit tests on the scaled residuals

To support the visual inspection of the residuals, the DHARMa package provides a number of specialized goodness-of-fit tests on the simulated residuals. For example, the function

```
testUniformity(simulationOutput = simulationOutput)
```



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.052, p-value = 0.5085
## alternative hypothesis: two-sided
```

runs a KS test to test for overall uniformity of the residuals. There are a number of further tests

- testOverdispersion()
- testZeroinflation()
- testTemporalAutocorrelation()
- testSpatialAutocorrelation()

that basically do what they say. See the help of the functions and further comments below for a more detailed description.

Simulation options

There are a few important technical details regarding how the simulations are performed, in particular regarding the treatments of random effects and integer responses. I would therefore strongly recommend to read the help of

```
?simulateResiduals
```

The short summary is this: apart from the number of simulations, there are three important options in the simulateResiduals function

Refit

```
simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

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

The second option is much much slower, and also seemed to have lower power in some tests I ran. It is therefore not recommended for standard residual diagnostics! I only recommend using it in two situations

1. For running tests that rely on comparing observed to simulated residuals, e.g. the testOverdispersion function (see below),
2. Or, and this was my original motivation for introducing this option, if one expects that the tested model is biased. A bias could, for example, arise in small data situations, or when estimating models with shrinkage estimators that include a purposeful bias, such as ridge/lasso, random effects or the splines in GAMs. My idea was then that simulated data would not fit to the observations, but that residuals for model fits on simulated data would have the same patterns/bias than model fits on the observed data.

Note also that refit = T can sometimes run into numerical problems, if the fitted model does not converge on the newly simulated data.

Random effect simulations

The second option is the treatment of the stochastic hierarchy. In a hierarchical model, several layers of stochasticity are placed 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, but the hierarchy can be more complex. When simulating, we have to decide if we want to re-simulate all stochastic levels, or only a subset of those. For example, in a GLMM, it is common to only simulate the last stochastic level (e.g. Poisson) conditional on the fitted random effects, meaning that the random effects are set on the fitted values.

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 "use.u", and "re.form", as, e.g., in

```
simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

If the model is correctly specified and the fitting procedure is unbiased (disclaimer: GLMM estimators are not always unbiased), the simulated residuals should be flat regardless how many hierarchical levels we re-simulate. The most thorough procedure would be therefore 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 random effects creates more variability, which may reduce power for detecting problems in the upper-level stochastic processes.

Integer treatment / randomization

A third option is the treatment of integer responses. The background of this option is that, for integer-valued variables, some additional steps are neccessary to make sure that the residual distribution becomes flat (essentially, we have to smoothen away the integer nature of the data). The idea is explained in

- Dunn, K. P., and Smyth, G. K. (1996). Randomized quantile residuals. Journal of Computational and Graphical Statistics 5, 1-10.

The simulateResiduals function will automatically check if the family is integer valued, and apply randomization if that is the case. I see no reason why one would not want to randomize for an integer-valued function, so the parameter should usually not be changed.

Calculating 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 the recalculateResiduals() function, together with a grouping variable

```
simulationOutput = recalculateResiduals(simulationOutput, grou
```

you can keep using the simulation output as before. Note, hover, that items such as simulationOutput$scaledResiduals now have as many entries as you have groups, so if you perform plots by hand, you have to aggregate predictors in the same way. For the latter purpose, recalculateResiduals adds a function aggregateByGroup to the output.

Reproducibility notes, random seed and random state

As DHARMa uses simulations to calculate the residuals, a naive implementation of the algorithm would mean that residuals would look slightly different each time a DHARMa calculation is executed. This might only be confusing to a user, but it also bears the danger that one might run the simulation several times and take the result that looks better (which would amount to multiple testing / p-hacking).

By default, DHARMa therefore fixes the random seed to the same value every time a simulation is run, and afterwards restores the random state to the old value. This means that you will get exactly the same residual plot each time. If you want to avoid this behavior, for example for simulation experiments on DHARMa, use seed = NULL -> no seed set, but random state will be restored, or seed = F -> no seed set, and random state will not be restored. Whether or not you fix the seed, the setting for the random seed and the random state are stored in

```
simulationOutput$randomState
```

If you want to reproduce simualtions for such a run, set the variable .Random.seed by hand, and simulate with seed = NULL.

Moreover (general advice), to ensure reproducibility, it's advisable to add a set.seed() at the beginning, and a session.info() at the end of your script. The latter will lists the version number of R and all loaded packages.

Visual diagnostics and tests of common misspecification problems

In all plots / tests that were shown so far, the model was correctly specified, resulting in “perfect” residual plots. In this section, we discuss how to recognize and interpret model misspecifications in the scaled residuals.

Overdispersion / underdispersion

The most common concern for GLMMs is overdispersion, underdispersion and zero-inflation.

Over/underdispersion refers to the phenomenon that residual variance is larger/smaller than expected under the fitted model. Over/underdispersion can appear for any distributional family with fixed variance, in particular for Poisson and binomial models.

A few general rules of thumb

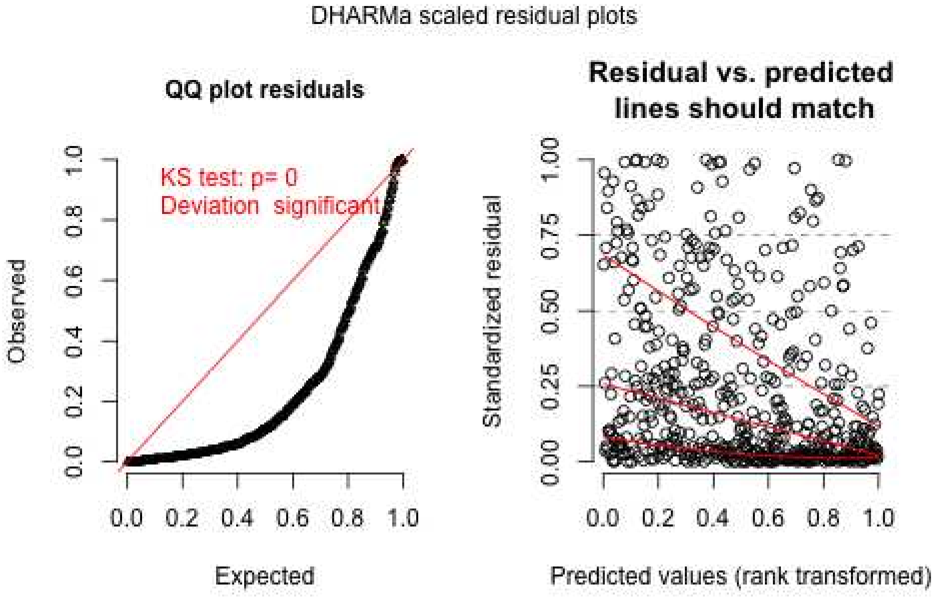
- You can detect overdispersion / zero-inflation only AFTER fitting the model
- Overdispersion is more common than underdispersion
- If overdispersion is present, confidence intervals tend to be too narrow, and p-values to small. The opposite is true for underdispersion
- A common reason for overdispersion is a misspecified model. When overdispersion is detected, one should therefore first search for problems in the model specification (e.g. by plotting residuals against predictors

with DHARMa), and only if this doesn't lead to success, overdispersion corrections such as individual-level random effects or changes in the distribution should be applied

An example of overdispersion

This this is how **overdispersion** looks like in the DHARMa residuals

```
testData = createData(sampleSize = 500, overdispersion = 2, fa
fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```



Note that we get more residuals around 0 and 1, which means that more residuals are in the tail of distribution than would be expected under the fitted model.

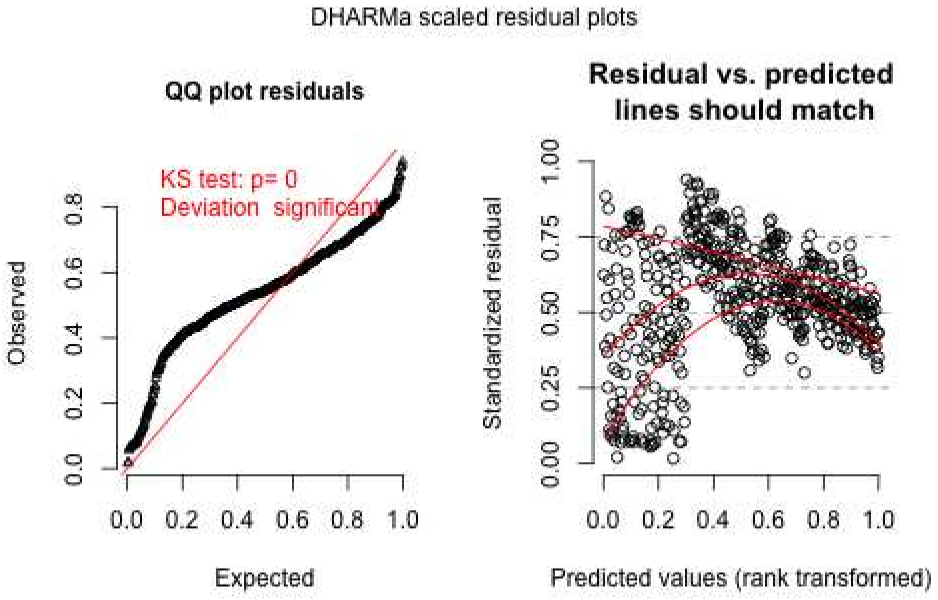
An example of underdispersion

This is an example of underdispersion

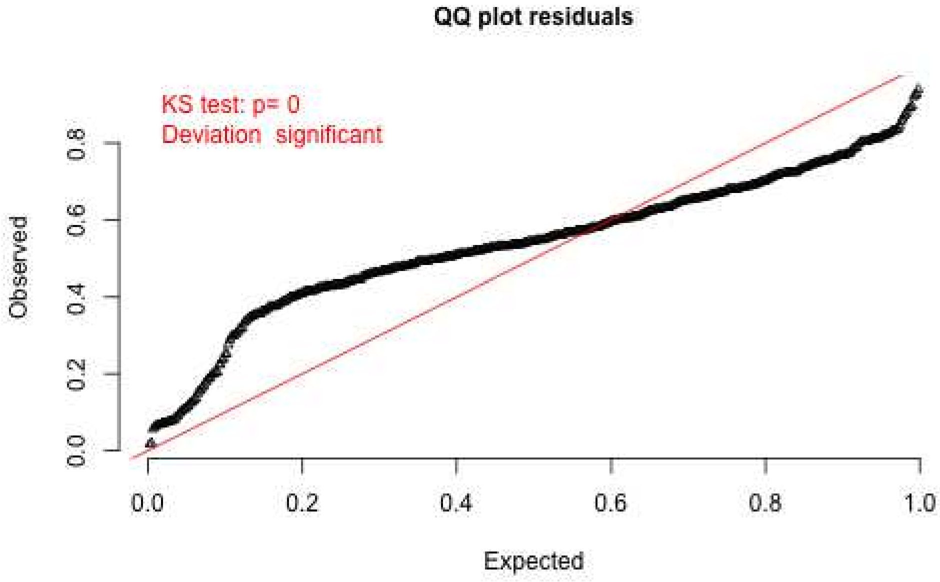
```
testData = createData(sampleSize = 500, intercept=0, fixedEffe
fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

summary(fittedModel)
```

```
## Generalized linear mixed model fit by maximum likelihood (L
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: observedResponse ~ Environment1 + (1 | group)
##    Data: testData
##
##      AIC      BIC   logLik deviance df.resid
##   1031.1   1043.8   -512.6   1025.1      497
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -0.64083 -0.35390 -0.05813  0.22834  0.91703
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  group  (Intercept) 0        0
## Number of obs: 500, groups:  group, 10
##
## Fixed effects:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.13024    0.05831  -2.233   0.0255 *
## Environment1  2.19567    0.08519  25.772   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
##

## Correlation of Fixed Effects:
##             (Intr)
## Environmnt1 -0.818
```

```
# plotConventionalResiduals(fittedModel)

simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```



```
testUniformity(simulationOutput = simulationOutput)
```



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.22, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

Here, we get too many residuals around 0.5, which means that we are not getting as many residuals as we would expect in the tail of the distribution than expected from the fitted model.

Testing for over/underdispersion

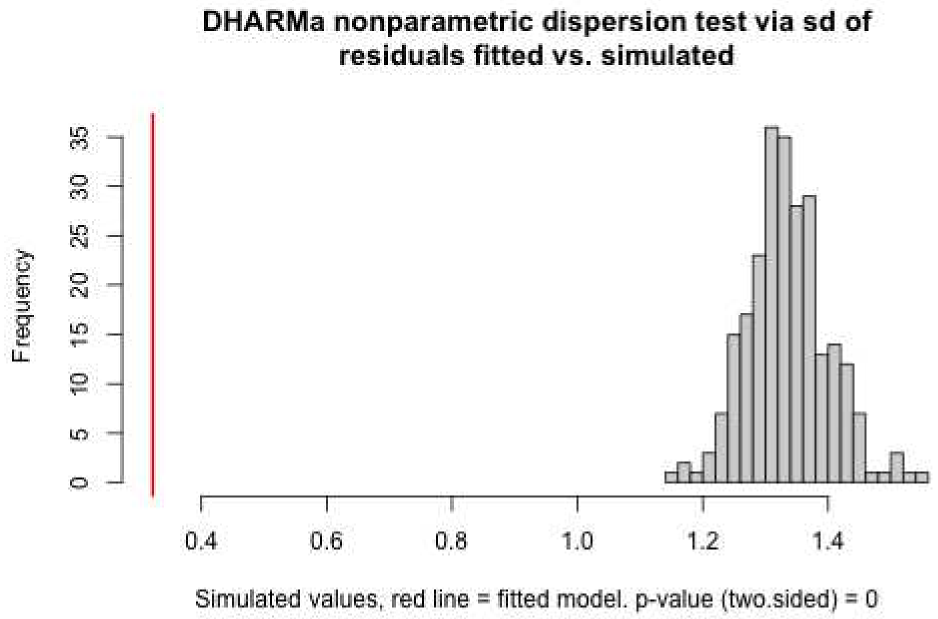
Although, as discussed above, over/underdispersion will show up in the residuals, and it's possible to detect it with the testUniformity function, simulations show that this test is less powerful than more targeted tests.

DHARMa therefore contains two overdispersion tests that compares the dispersion of simulated residuals to the observed residuals.

1. A non-parametric test on the simulated residuals
2. A non-parametric overdispersion test on the re-fitted residuals.

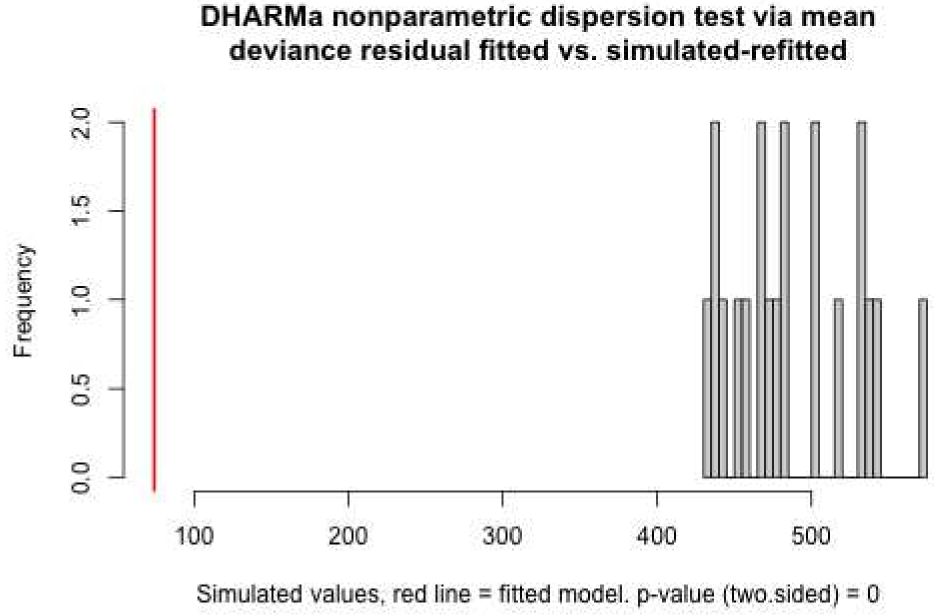
You can call these tests as follows:

```
# Option 2
testDispersion(simulationOutput)
```



```
##
##  DHARMa nonparametric dispersion test via sd of residuals f
##  vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 0.24135, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

```
# Option 3
simulationOutput2 <- simulateResiduals(fittedModel = fittedMod
testDispersion(simulationOutput2)
```



```
##
##  DHARMa nonparametric dispersion test via mean deviance res
##  fitted vs. simulated-refitted
##
## data:  simulationOutput2
## dispersion = 0.15184, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

Note: previous versions of DHARMa (< 0.2.0) discouraged the simulated overdispersion test in favor of the refitted and parametric tests. I have since changed the test function, and simulations show that it as powerful as the refitted or parametric test. Because of the generality and speed of this option, I see no good reason for either refitting or running parametric tests. Therefore

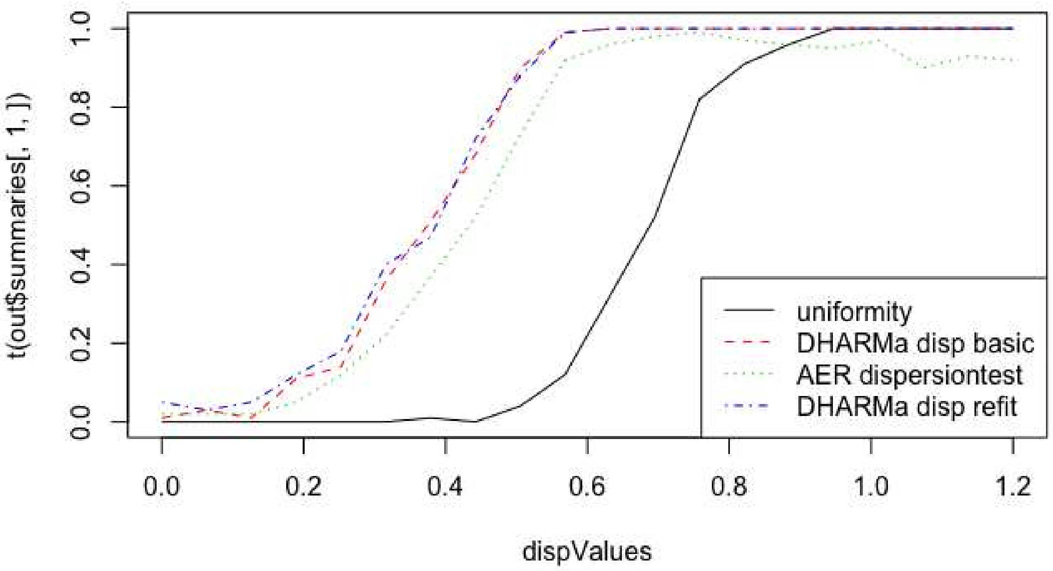
1. My **recommendation** for testing dispersion is to simply use the standard dispersion test, based on the simulated residuals

2. It's not clear to if the refitted test is better … but it's available.

3. In my simulations, parametric tests, such as AER::dispersiontest didn't provide higher power. Because of that, and because of the higher generality of the simulated tests, I no longer provide parametric tests in

DHARMa. However, you can see various implementions of the parametric tests in the DHARMa GitHub repo under Code/DHARMaPerformance/Power).

Below and example from there, which compares the four options to test for overdispersion (2 options to use DHARMa::testDispersoin, AER::dispersiontest, and DHARMa::testUniformity) for a Poisson glm



A word of warning that applies also to all other tests that follow: significance in hypothesis tests depends on at least 2 ingredients: strenght of the signal, and number of data points. Hence, the p-value alone is not a good indicator of the extent to which your residuals deviate from assumptions. Specifically, if you have a lot of data points, residual diagnostics will nearly inevitably become significant, because having a perfectly fitting model is very unlikely. That, however, doesn't neccessarily mean that you need to change your model. The p-values confirm that there is a deviation from your null hypothesis. It is, however, in your discretion to decide whether this deviation is worth worrying about. If you see a dispersion parameter of 1.01, I would not worry, even if the test is significant. A significant value of 5, however, is clearly a reason to move to a model that accounts for overdispersion.

Zero-inflation / k-inflation or deficits

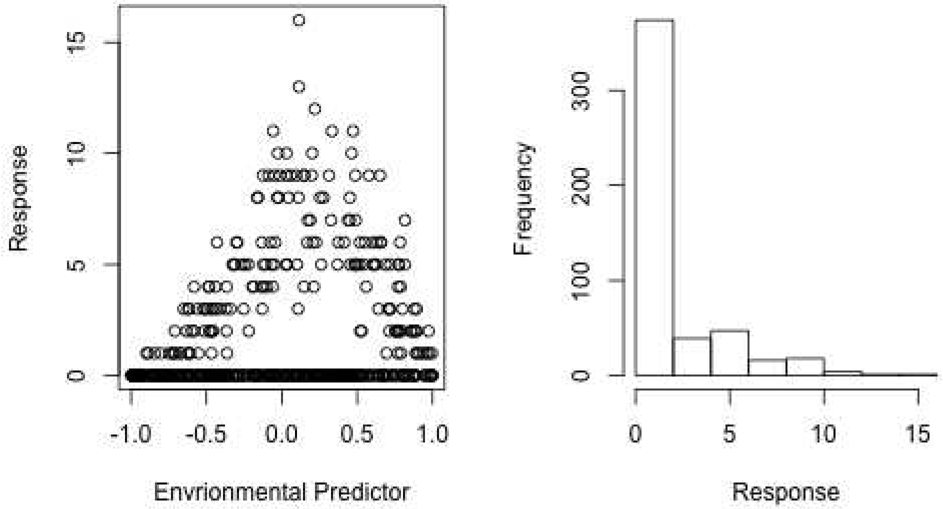
A common special case of overdispersion is zero-inflation, which is the situation when more zeros appear in the observation than expected under the fitted model. Zero-inflation requires special correction steps.

More generally, we can also have too few zeros, or too much or too few of any other values. We'll discuss that at the end of this section

An example of zero-inflation

Here an example of a typical zero-inflated count dataset, plotted against the environmental predictor

```
testData = createData(sampleSize = 500, intercept = 2, fixedEf

par(mfrow = c(1,2))
plot(testData$Environment1, testData$observedResponse, xlab =
hist(testData$observedResponse, xlab = "Response", main = "")
```

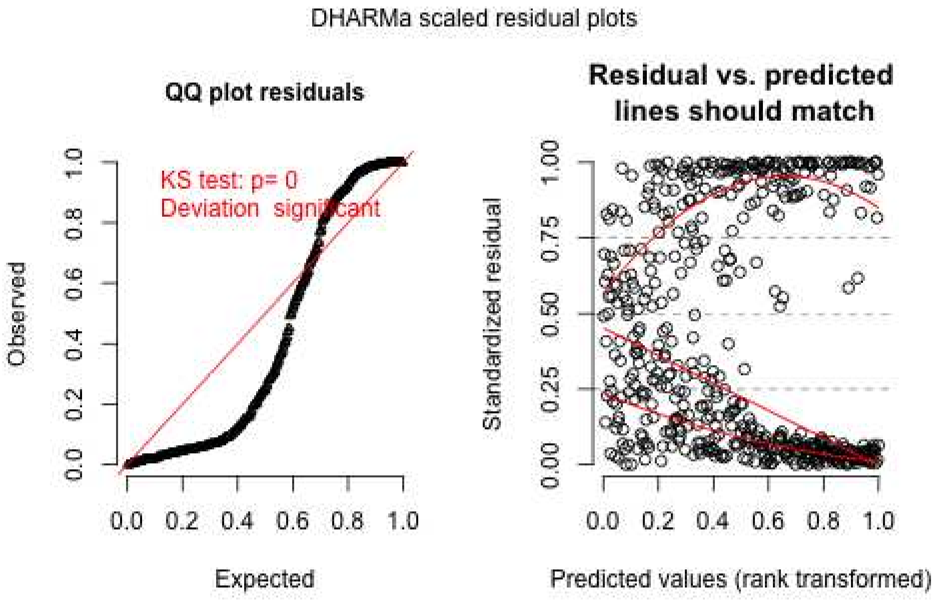


We see a hump-shaped dependence of the environment, but with too many zeros.

Zero-inflation in the scaled residuals

In the normal DHARMa residual, plots, zero-inflation will look pretty much like overdispersion

```
fittedModel <- glmer(observedResponse ~ Environment1 + I(Envir

simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```

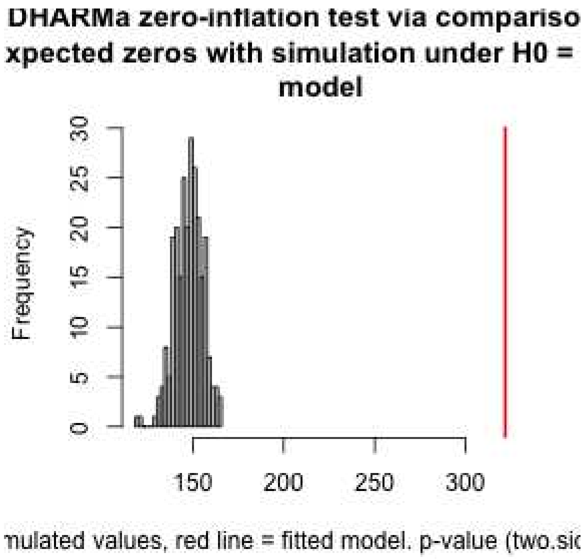


The reason is that the model will usually try to find a compromise between the zeros, and the other values, which will lead to excess variance in the residuals.

Test for zero-inflation

DHARMa has a special test for zero-inflation, which compares the distribution of expected zeros in the data against the observed zeros

```
testZeroInflation(simulationOutput)
```



```
##
##  DHARMa zero-inflation test via comparison to expected zero
##  simulation under H0 = fitted model
##
## data:  simulationOutput
## ratioObsSim = 2.1744, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

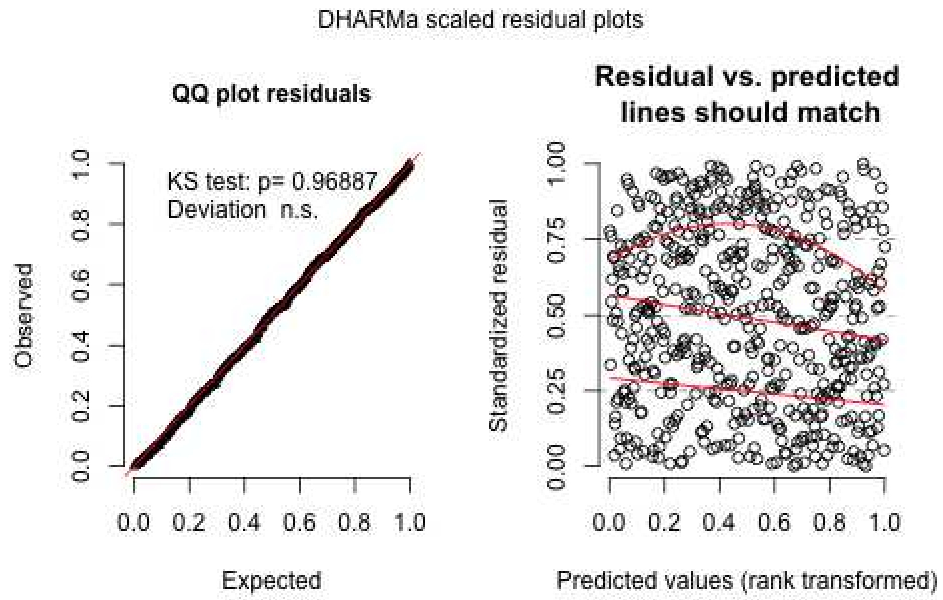
This test is likely better suited for detecting zero-inflation than the standard plot, but note that also overdispersion will lead to excess zeros, so only seeing too many zeros is not a reliable diagnostics for moving towards a zero-inflated model. A reliable differentiation between overdispersion and zero-inflation will usually only be possible when directly comparing alternative models, e.g. through residual comparison / model selection of a model with / without zero-inflation, or by simply fitting a model with zero-inflation and looking at the parameter estimate for the zero-inflation.

A good option is the R package glmmTMB, which is also supported by DHARMa. We can use this to fit

```
library(glmmTMB)
fittedModel <- glmmTMB(observedResponse ~ Environment1 + I(Env
summary(fittedModel)
```

```
##  Family: poisson  ( log )
## Formula:
## observedResponse ~ Environment1 + I(Environment1^2) + (1 |
## Zero inflation:                    ~1
## Data: testData
##
##      AIC      BIC   logLik deviance df.resid
##   1288.7   1309.7   -639.3   1278.7      495
##
## Random effects:
##
## Conditional model:
##  Groups Name        Variance Std.Dev.
##  group  (Intercept) 8.05e-10 2.837e-05
## Number of obs: 500, groups:  group, 10
##
## Conditional model:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)         2.00497    0.04648   43.13   <2e-16 ***
## Environment1        1.08342    0.10810   10.02   <2e-16 ***
## I(Environment1^2) -2.92000    0.19383  -15.06   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
##
## Zero-inflation model:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)   0.2990     0.1036   2.885  0.00392 **


## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
```

```
simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```

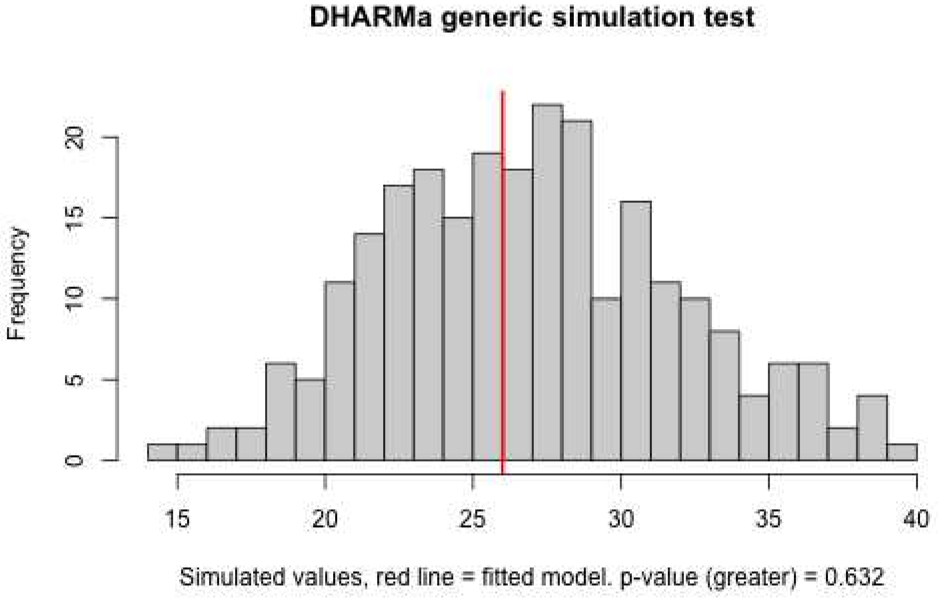


General zero/k-inflation or deficits

To test for generic excess / deficits of particular values, we have the function testGeneric, which compares the values of a generic, user-provided summary statistics

Choose one of alternative = c("greater", "two.sided", "less") to test for inflation / deficit or both. Default is "greater" = inflation.

```
countOnes <- function(x) sum(x == 1)  # testing for number of
testGeneric(simulationOutput, summary = countOnes, alternative
```



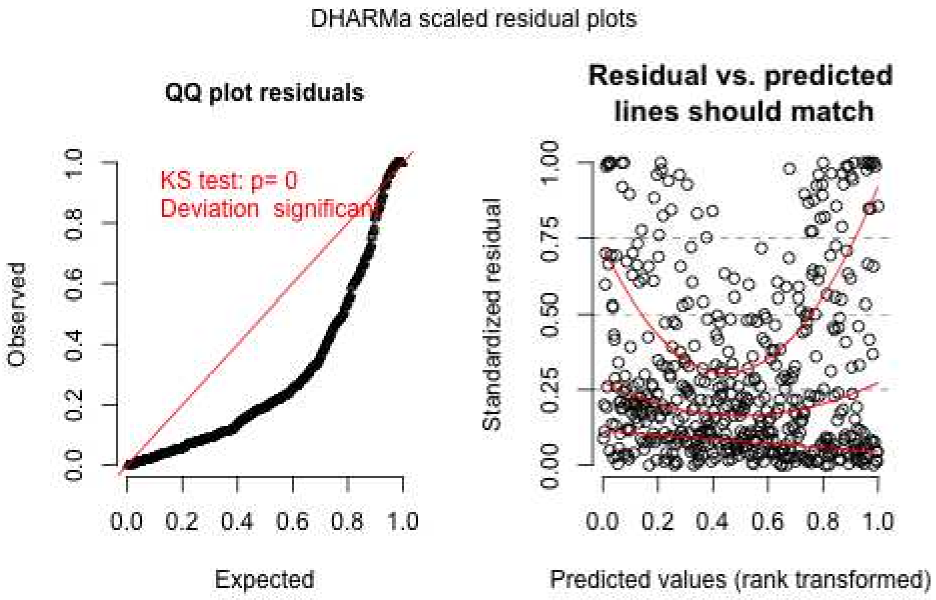
```
##
##  DHARMa generic simulation test
##
## data:  simulationOutput
## ratioObsSim = 0.94807, p-value = 0.632
## alternative hypothesis: greater
```

Heteroscedasticity

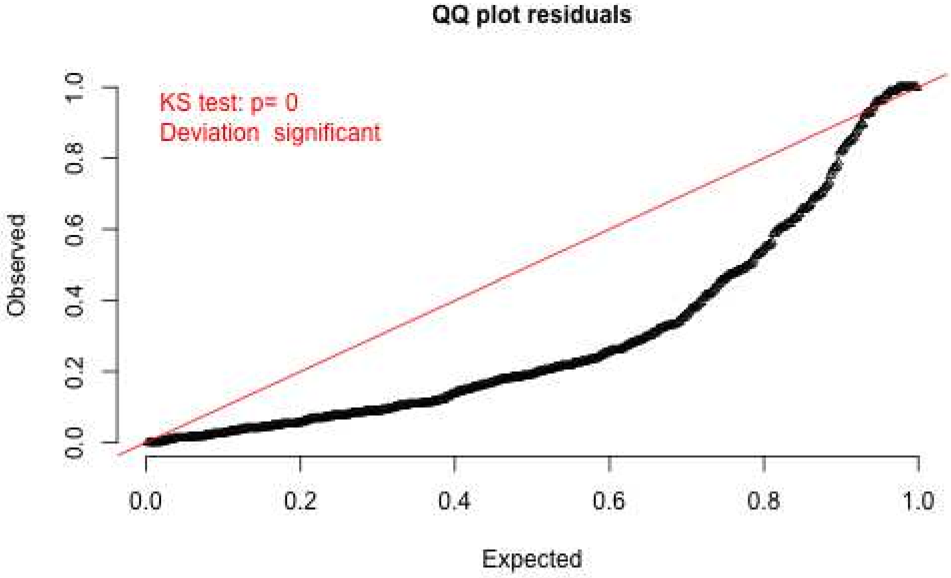
So far, most of the things that we have tested could also have been detected with parametric tests. Here, we come to the first issue that is difficult to detect with current tests, and that is usually neglected.

Heteroscedasticity means that there is a systematic dependency of the dispersion / variance on another variable in the model. It is not sufficiently appreciated that also binomial or Poisson models can show heteroscedasticity. Basically, it means that the level of over/underdispersion depends on another parameter. Here an example where we create such data

```
testData = createData(sampleSize = 500, intercept = 0, overdis
fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```



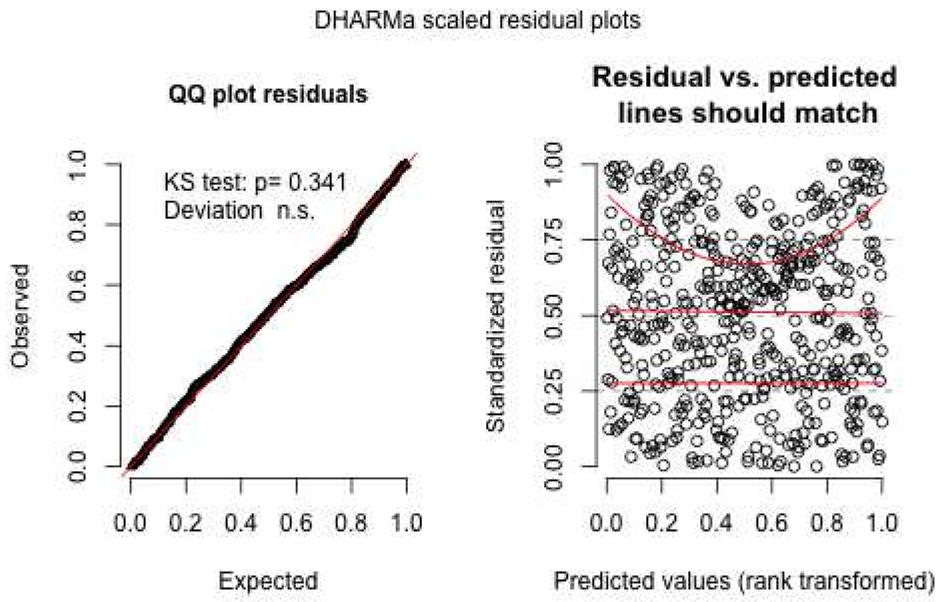
```
testUniformity(simulationOutput = simulationOutput)
```



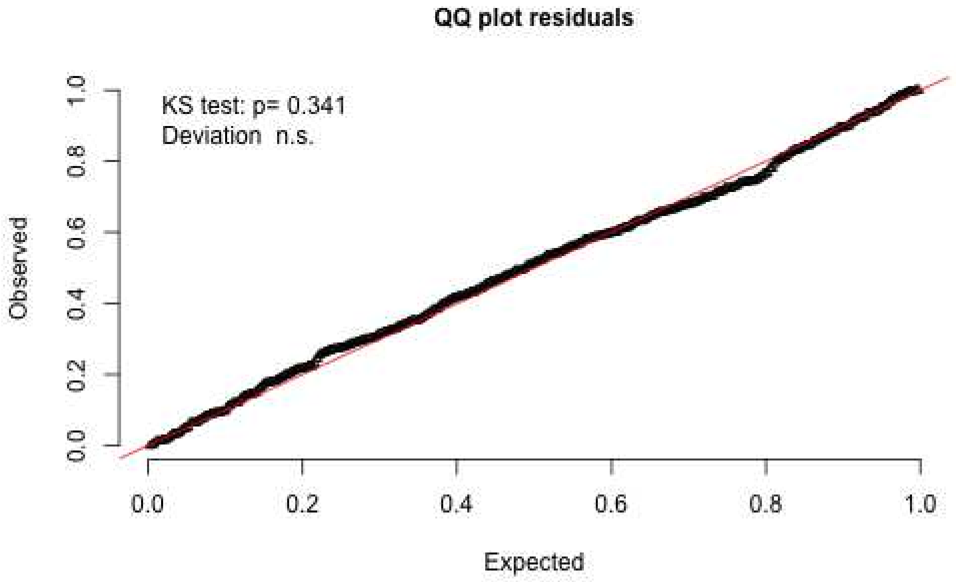
```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.354, p-value < 2.2e-16
## alternative hypothesis: two-sided
```

Adding a simple overdispersion correction will try to find a compromise between the different levels of dispersion in the model. The qq plot looks better now, but there is still a pattern in the residuals

```
testData = createData(sampleSize = 500, intercept = 0, overdis
fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

# plotConventionalResiduals(fittedModel)

simulationOutput <- simulateResiduals(fittedModel = fittedMode
plot(simulationOutput)
```



```
testUniformity(simulationOutput = simulationOutput)
```



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.042, p-value = 0.341
## alternative hypothesis: two-sided
```

To remove this pattern, you would need to make the dispersion parameter dependent on a predictor (e.g. in JAGS), or apply a transformation on the data.

Missing predictors or quadratic effects

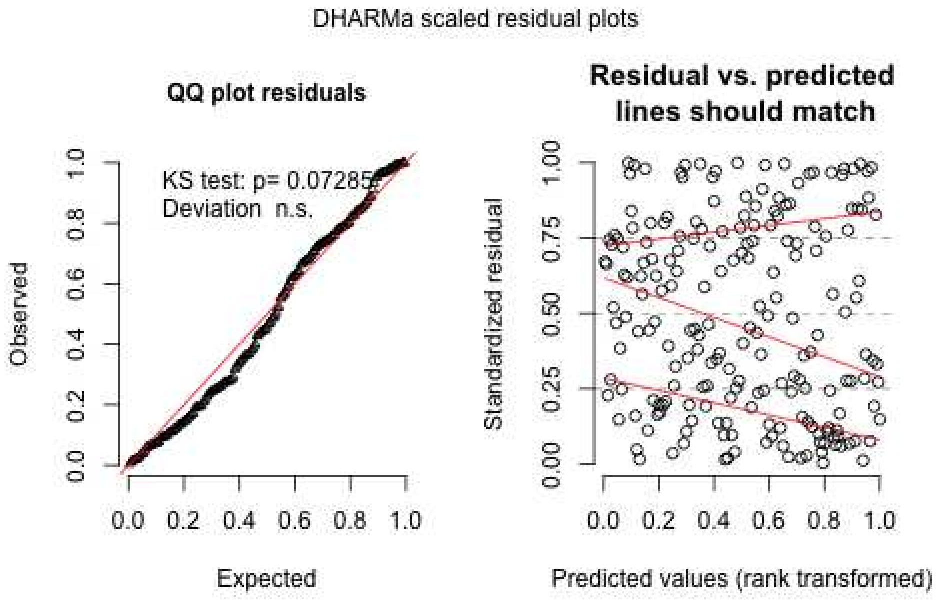
A second test that is typically run for LMs, but not for GL(M)Ms is to plot residuals against the predictors in the model (or potentially predictors that were not in the model) to detect possible misspecifications. Doing this is *highly recommended*. For that purpose, you can retrieve the residuals via

```
simulationOutput$scaledResiduals
```

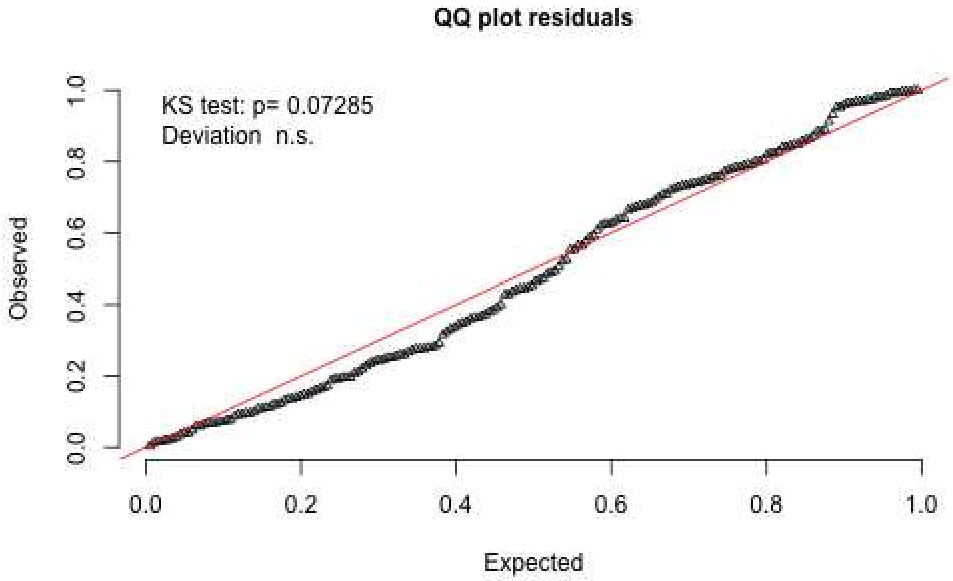
Note again that the residual values are scaled between 0 and 1. If you plot the residuals against predictors, space or time, the resulting plots should not only show no systematic dependency of those residuals on the covariates, but they should also again be flat for each fixed situation. That means that if you have, for example, a categorical predictor: treatment / control, the distribution of residuals for each predictor alone should be flat as well.

Here an example with a missing quadratic effect in the model and 2 predictors

```
testData = createData(sampleSize = 200, intercept = 1, fixedEf
fittedModel <- glmer(observedResponse ~ Environment1 + Environ
simulationOutput <- simulateResiduals(fittedModel = fittedMode
# plotConventionalResiduals(fittedModel)
plot(simulationOutput, quantreg = T)
```



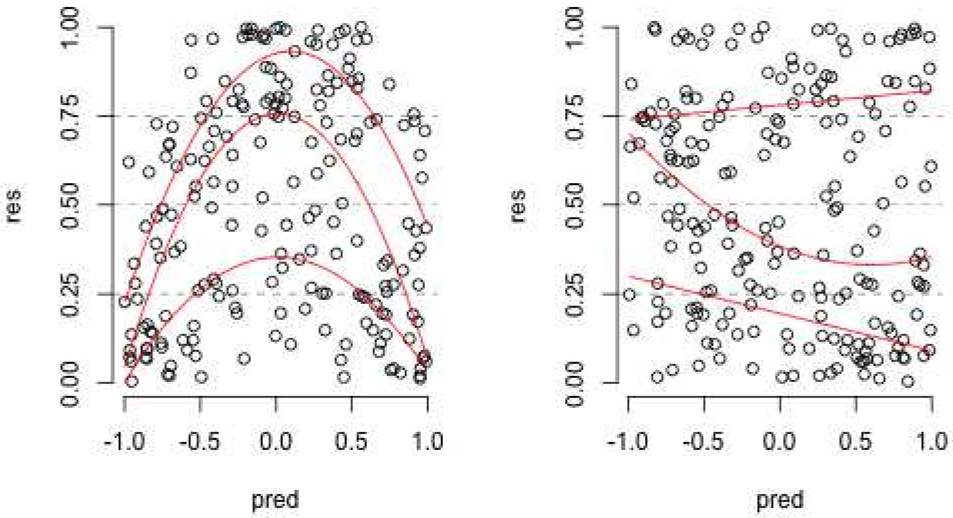
```
testUniformity(simulationOutput = simulationOutput)
```



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.091, p-value = 0.07285
## alternative hypothesis: two-sided
```

It is difficult to see that there is a problem at all in the general plot, but it becomes clear if we plot against the environment

```
par(mfrow = c(1,2))
plotResiduals(testData$Environment1,  simulationOutput$scaledR
plotResiduals(testData$Environment2,  simulationOutput$scaledR
```



Temporal autocorrelation

A special case of plotting residuals against predictors is the plot against time and space, which should always be performed if those variables are present in the model. Let's create some temporally autocorrelated data

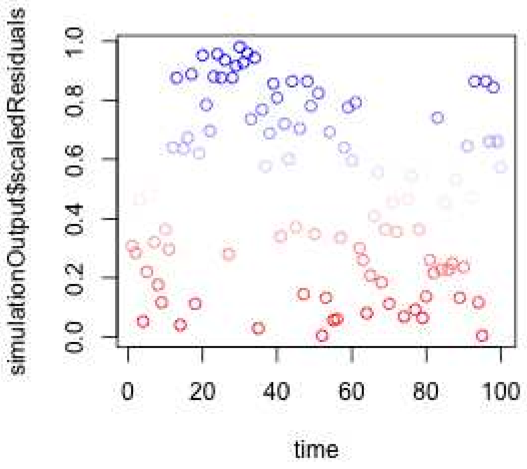
```
testData = createData(sampleSize = 100, family = poisson(), te

fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

Test and plot for temporal autocorrelation

The function testTemporalAutocorrelation performs a Durbin-Watson test from the package lmtest on the uniform residuals to test for temporal autocorrelation in the residuals, and additionally plots the residuals against time.

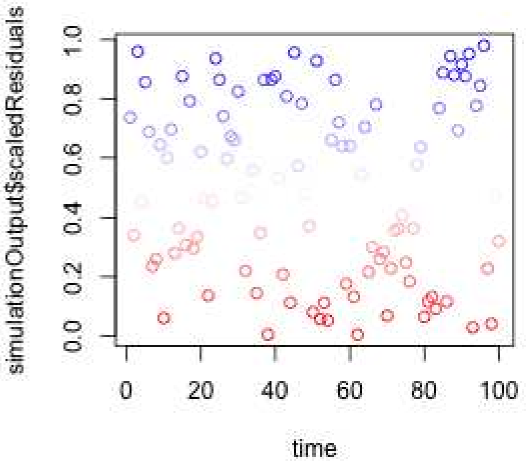
The function also has an option to perform the test against randomized time (H0) - the sense of this is to be able to run simulations for testing if the test has correct error rates in the respective situation, i.e. is not oversensitive (too high sensitivity has sometimes been reported for Durbin-Watson).

```
testTemporalAutocorrelation(simulationOutput = simulationOutpu
```



```
##
##  Durbin-Watson test
##
## data:  simulationOutput$scaledResiduals ~ 1
## DW = 1.4646, p-value = 0.006846
## alternative hypothesis: true autocorrelation is not 0
```

```
testTemporalAutocorrelation(simulationOutput = simulationOutpu
```



```
##
##  Durbin-Watson test
##
## data:  simulationOutput$scaledResiduals ~ 1
## DW = 1.8733, p-value = 0.5223
## alternative hypothesis: true autocorrelation is not 0
```

Note general caveats mentioned about the DW test in the help of testTemporalAutocorrelation(). In general, as for spatial autocorrelation, it is difficult to specify one test, because temporal and spatial autocorrelation can appear in many flavors, short-scale and long scale, homogenous or not, and so on. The pre-defined functions in DHARMa are a starting point, but they are not something you should rely on blindly.

Spatial autocorrelation

Here an example with spatial autocorrelation

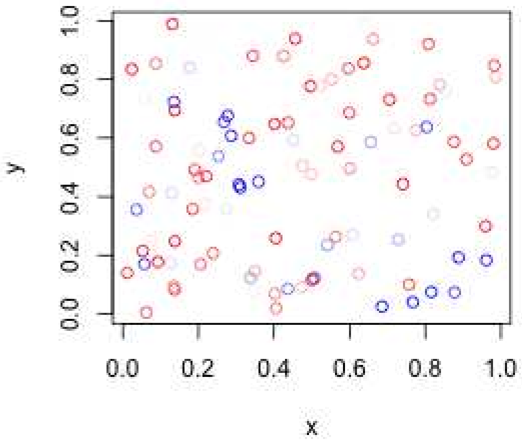
```
testData = createData(sampleSize = 100, family = poisson(), sp

fittedModel <- glmer(observedResponse ~ Environment1 + (1|grou

simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

Test and plot for spatial autocorrelation

The spatial autocorrelation test performs the Moran.I test from the package ape and plots the residuals against space.

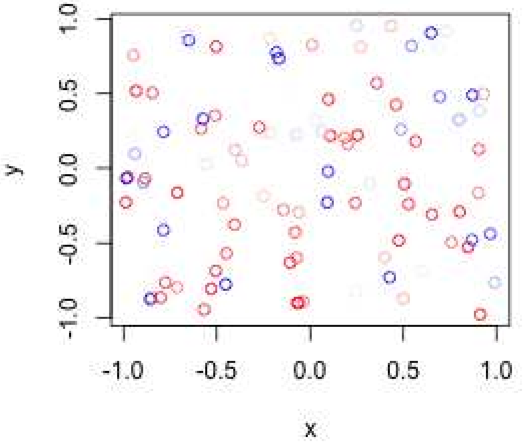
An additional test against randomized space (H0) can be performed, for the same reasons as explained above.

```
testSpatialAutocorrelation(simulationOutput = simulationOutput
```



```
##
##  DHARMa Moran's I test for spatial autocorrelation
##
## data:  simulationOutput
## observed = 0.036095, expected = -0.010101, sd = 0.016982, p
## = 0.006523
## alternative hypothesis: Spatial autocorrelation
```

```
testSpatialAutocorrelation(simulationOutput = simulationOutput
```



```
##
##  DHARMa Moran's I test for spatial autocorrelation
##
## data:  simulationOutput
## observed = -0.0025567, expected = -0.0101010, sd = 0.020928
## p-value = 0.7185
## alternative hypothesis: Spatial autocorrelation
```

The usual caveats for Moran.I apply, in particular that it may miss non-local and heterogeneous (non-stationary) spatial autocorrelation. The former should be better detectable visually in the spatial plot, or via regressions on the pattern.

Custom tests

A big advantage of the simulations is that you can test any problem that you think you may have. For example, you think you have an excess of tens in your count data? Maybe a faulty measurement instrument that returns too many tens? Just compare the observed with the expected tens from the simulations.

You think your random effect estimates look weird? Run the model with the refit = T option and see how typical random effect estimates look for your problem.

Supported packages

lm and glm

lm and glm are fully supported as specified in the help

lme4

lme 4 is fully supported as specified in the help

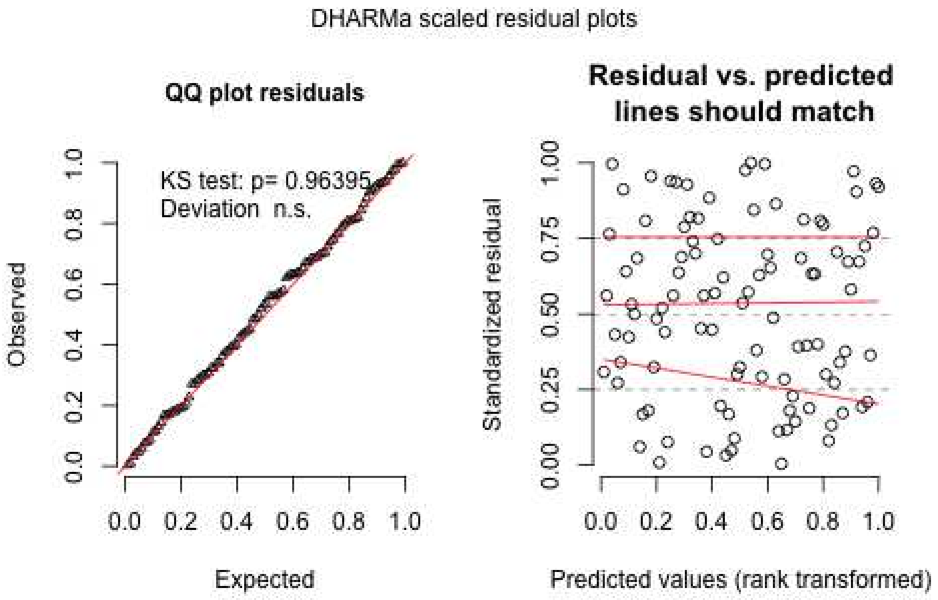
glmmTMB

Current limitations of glmmTMB

- residual function doesn't work with factor response (glmmTMB bug)
- refit doesn't work with n/k data, because update doesnt' work
- pearson residuals don't work with zi terms (not implemented), which limits some of the overdispersion tests
- Currently, glmmTMB doesn't support the reform argument.
    - predict() is conditional on all random effects, corresponding to lme4 re.form = NULL
    - simulate() is unconditional, i.e. all random effects will be resimulated, corresponding to lme4 re.form = 0

- That means that all predictions and simulations are conditional on REs, which can sometimes create a positive correlation between res and predicted , see https://github.com/florianhartig/DHARMa/issues/43

```
library(glmmTMB)
testData = createData(sampleSize = 100, fixedEffects = 2, fami

m <- glmmTMB(observedResponse ~ Environment1 , data=testData,
res = simulateResiduals(m)
plot(res)
```



DHARMa scaled residual plots

QQ plot residuals

Residual vs. predicted lines should match

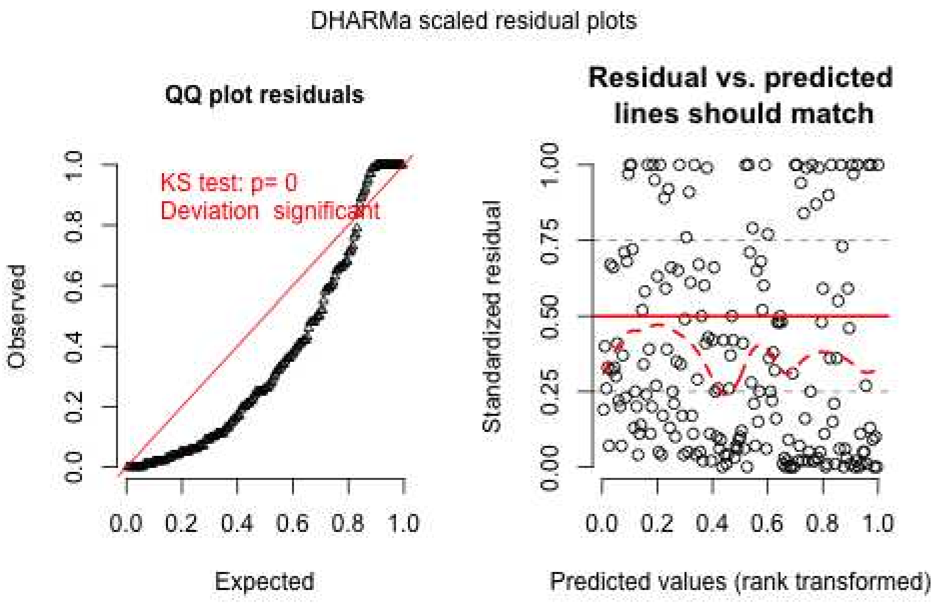
Unsupported packages

See my general comments about adding new R packages to DHARMa

As noted there, if you want to use DHARMa for a specific case, you could write a custom simulate function for the specific model you are working with. This will usually involve using the predict function and adding the random distribution, plus potentially drawing new data for the random effects or other hierarchical levels.

As an example, for an poisson glm, a simulate function could be programmed as in the following example, which also shows how the results are read into DHARMa and plotted (see also following section)

```
testData = createData(sampleSize = 200, overdispersion = 0.5,
fittedModel <- glm(observedResponse ~ Environment1, family = "

simulatePoissonGLM <- function(fittedModel, n){
  pred = predict(fittedModel, type = "response")
  nObs = length(pred)
  sim = matrix(nrow = nObs, ncol = n)
  for(i in 1:n) sim[,i] = rpois(nObs, pred)
  return(sim)
}

sim = simulatePoissonGLM(fittedModel, 100)

DHARMaRes = createDHARMa(simulatedResponse = sim, observedResp
             fittedPredictedResponse = predict(fittedModel))
plot(DHARMaRes, quantreg = F)
```



Importing external simulations (e.g. from Bayesian software or unsupported packages)

As mentioned earlier, the quantile residuals defined in DHARMa are the frequentist equivalent of the so-called “Bayesian p-values”, i.e. residuals created from posterior predictive simulations in a Bayesian analysis.

To make the plots and tests in DHARMa also available for Bayesian analysis, DHARMa provides the option to convert externally created posterior predictive simulations into a DHARMa object

```
res = createDHARMa(scaledResiduals = posteriorPredictiveSimula
```

What is provided as simulatedResponse is up to the user, but median posterior predictions seem most sensible to me. After the conversion, all DHARMa plots can be used, however, note that Bayesian p-values != DHARMA residuals, because in the Bayesian analysis, parameters are varied as well.

Important: as DHARMa doesn't know the distribution fitted model, it is vital to specify the integerResponse option by hand (see above / ?simulateResiduals for details).

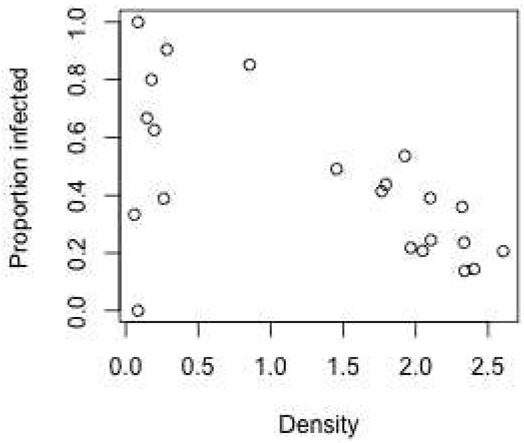
Case studies and examples

Note: More real-world examples on the DHARMa GitHub repository here

Budworm example (count-proportion n/k binomial)

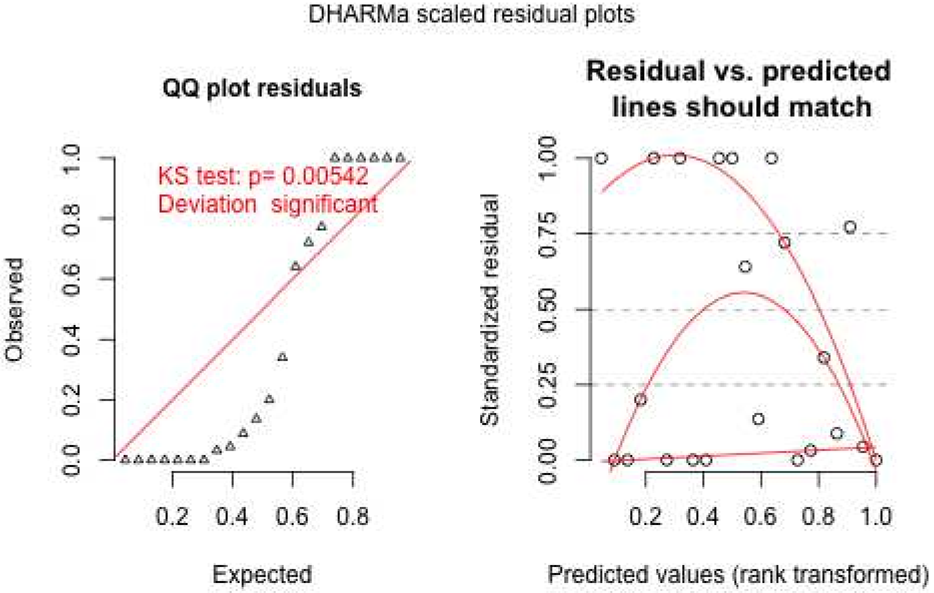
This example comes from Jochen Fründ. Measured are the number of parasitized observations, with population density as a covariate

```
plot(N_parasitized / (N_adult + N_parasitized ) ~ logDensity,
```



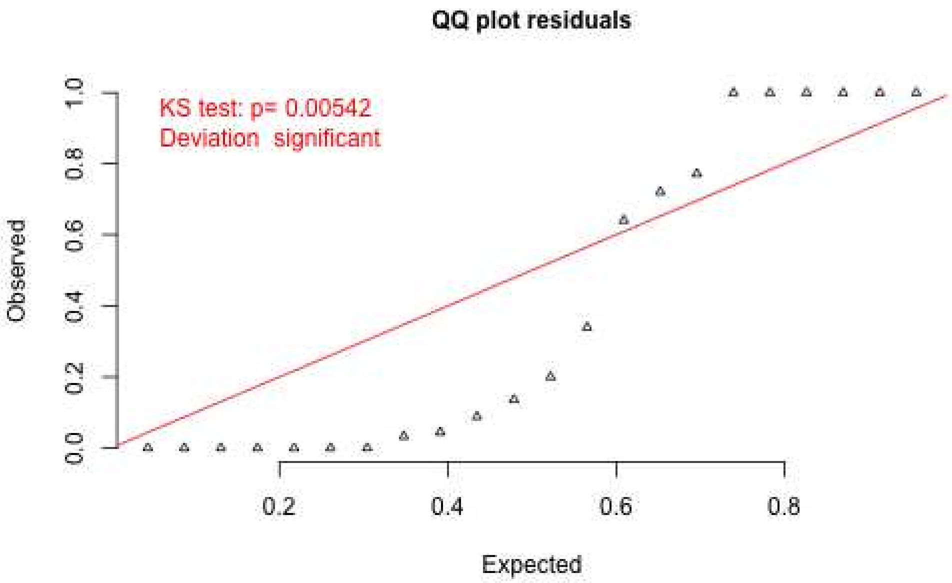
Let's fit the data with a regular binomial n/k glm

```
mod1 <- glm(cbind(N_parasitized, N_adult) ~ logDensity, data =
simulationOutput <- simulateResiduals(fittedModel = mod1)
plot(simulationOutput)
```



The residuals look clearly overdispersed. We can confirm that with the omnibus test

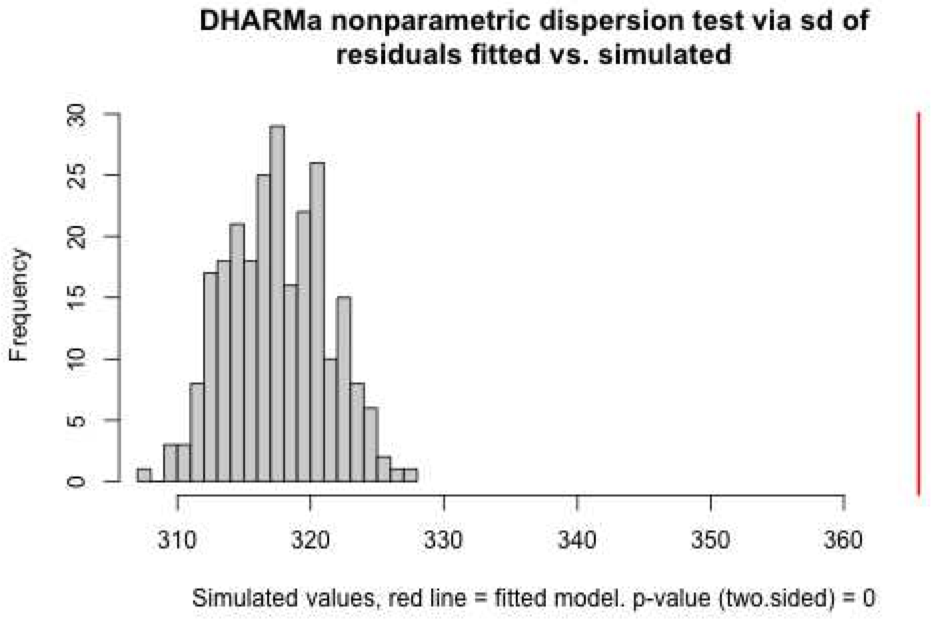
```
testUniformity(simulationOutput = simulationOutput)
```



```
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.36655, p-value = 0.005415
## alternative hypothesis: two-sided
```

Or with the more powerful overdispersion test

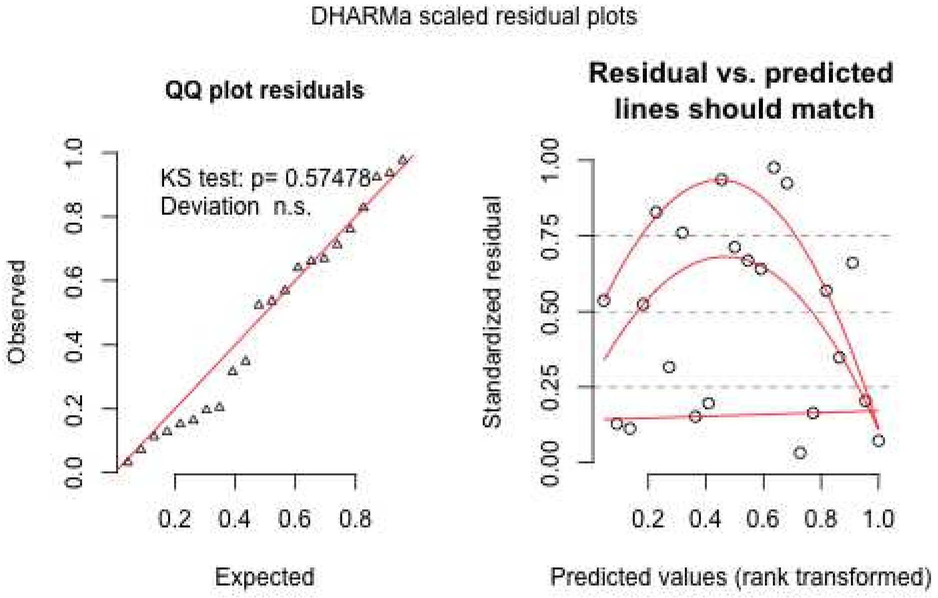
```
testOverdispersion(simulationOutput = simulationOutput)
```



```
##
##  DHARMa nonparametric dispersion test via sd of residuals f
##  vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 1.1516, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

OK, so let's add overdispersion through an individual-level random effect

```
mod2 <- glmer(cbind(N_parasitized, N_adult) ~ logDensity + (1|
simulationOutput <- simulateResiduals(fittedModel = mod2)
plot(simulationOutput)
```



The overdispersion looks better, but you can see that the residuals look a bit irregular.

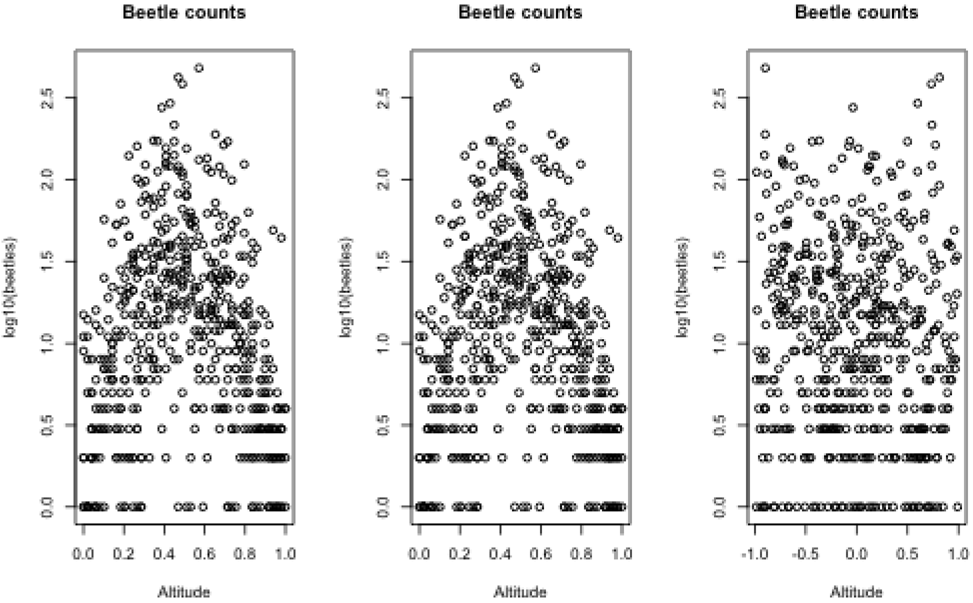
Likely, the reason is the steep increase in the beginning that one can see in the raw data plot. One would probably need to apply another transformation or a nonlinear function to completely fit this away.

Beetlecount / Poisson example

Dataset

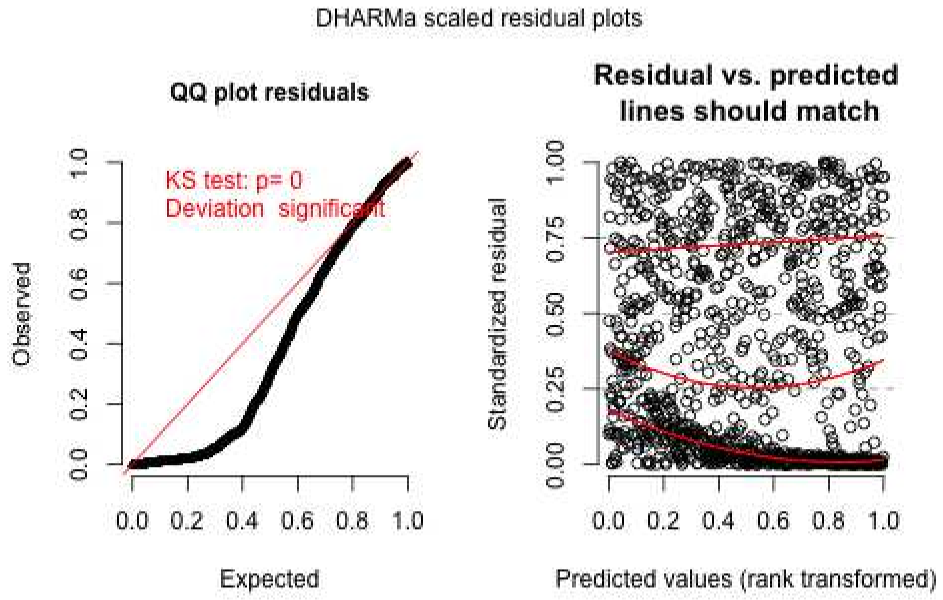
This example is a synthetic dataset of measured beetle counts over 50 plots across an altitudinal gradient that are yearly sampled over 20 years. The following plot shows the observed number of beetles (log10) vs. altitude. Additional variables in the data are soil moisture and the amount of deadwood on the plots.

```
par(mfrow = c(1,3))
plot(log10(beetles) ~ altitude + I(altitude) + moisture, data
```



Our question is: what is the effect of altitude on the abundance of the beetle? Let's start with a linear and quadratic term for altitude, linear effect of soil moisture, and random intercepts on plot and year

```
mod <- glmer(beetles ~ altitude + I(altitude^2) + moisture + (
simulationOutput <- simulateResiduals(fittedModel = mod)
plot(simulationOutput)
```

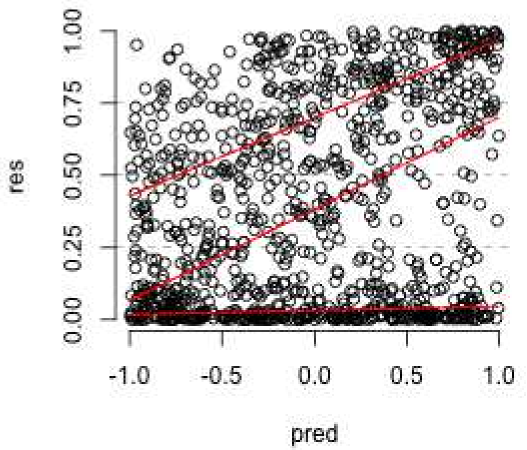

DHARMa scaled residual plots
QQ plot residuals
KS test: p= 0
Deviation significant
Observed
0.0
0.2
0.4
0.6
0.8
1.0
Expected
0.0
0.2
0.4
0.6
0.8
1.0
Residual vs. predicted
lines should match
Standardized residual
0.00
0.25
0.50
0.75
1.00
Predicted values (rank transformed)
0.0
0.2
0.4
0.6
0.8
1.0

```
summary(mod)
```

```
## Generalized linear mixed model fit by maximum likelihood (L
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: beetles ~ altitude + I(altitude^2) + moisture + (1
##     (1 | year)
##    Data: data
## Control: glmerControl(optCtrl = list(maxfun = 10000))
##
##      AIC      BIC   logLik deviance df.resid
##  20493.9  20523.4 -10241.0  20481.9      994
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -12.4093  -2.1837  -0.7425   1.8807  21.6505
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  plot   (Intercept) 0.1906   0.4366
##  year   (Intercept) 1.0850   1.0416
## Number of obs: 1000, groups:  plot, 50; year, 20
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.1998     0.2987  -0.669    0.504
## altitude       12.2749     0.8595  14.282   <2e-16 ***
## I(altitude^2) -12.5505     0.8337 -15.054   <2e-16 ***
## moisture       -0.2125     0.0161 -13.196   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
##
## Correlation of Fixed Effects:
##             (Intr) altitd I(l^2)
## altitude    -0.539
## I(altitd^2)  0.461 -0.967
## moisture     0.008 -0.006  0.006
```

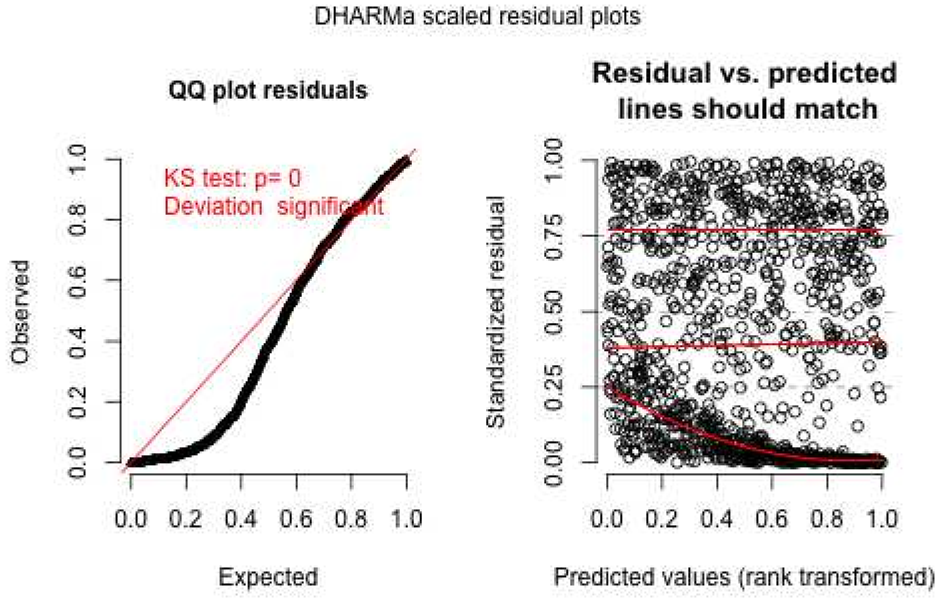
We see that we have a problem when we plot residuals against deadwood

```
plotResiduals(data$deadwood, simulationOutput$scaledResiduals)
```



so let's add this term as well

```
mod <- glmer(beetles ~ altitude + I(altitude^2) + moisture + d
simulationOutput <- simulateResiduals(fittedModel = mod)
plot(simulationOutput)
```

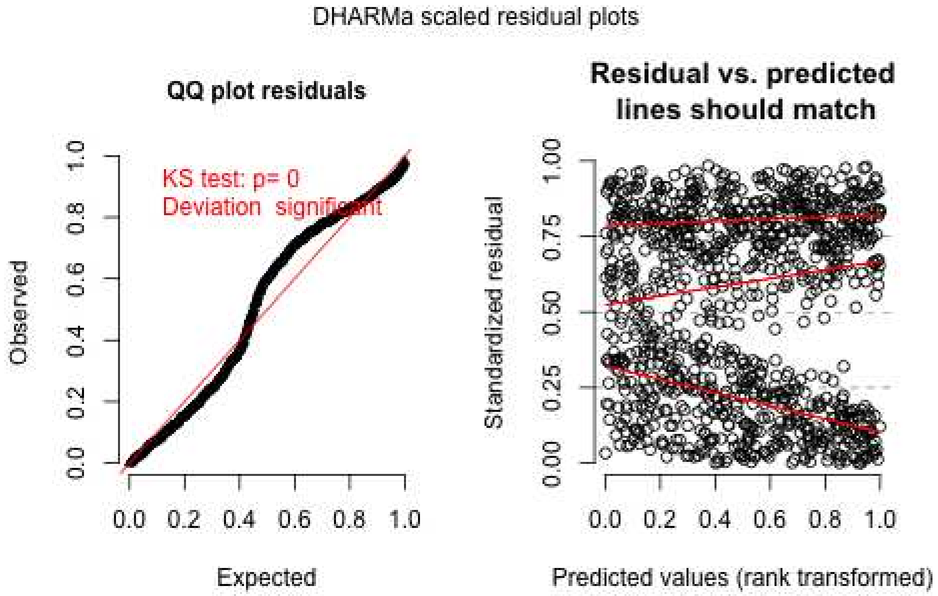

DHARMa scaled residual plots
QQ plot residuals
KS test: p= 0
Deviation significant
Observed
Expected
0.0
0.2
0.4
0.6
0.8
1.0
Residual vs. predicted
lines should match
Standardized residual
0.00
0.25
0.50
0.75
1.00
Predicted values (rank transformed)

```
summary(mod)
```

```
## Generalized linear mixed model fit by maximum likelihood (L
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: beetles ~ altitude + I(altitude^2) + moisture + de
##     plot) + (1 | year)
##    Data: data
## Control: glmerControl(optCtrl = list(maxfun = 10000))
##
##      AIC      BIC   logLik deviance df.resid
##  15162.1  15196.4  -7574.0  15148.1      993
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -11.3157  -1.6833  -0.0885   1.9843  14.3762
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  plot   (Intercept) 0.1778   0.4217
##  year   (Intercept) 1.1115   1.0543
## Number of obs: 1000, groups:  plot, 50; year, 20
##
## Fixed effects:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.47665    0.29746   -1.60    0.109
## altitude       12.42714    0.83225   14.93   <2e-16 ***
## I(altitude^2) -12.57408    0.80740  -15.57   <2e-16 ***
## moisture       -0.27695    0.01638  -16.91   <2e-16 ***
## deadwood        1.18974    0.01715   69.38   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '
##
## Correlation of Fixed Effects:
##             (Intr) altitd I(l^2) moistr
## altitude    -0.525
## I(altitd^2)  0.449 -0.967
## moisture     0.010 -0.007  0.006
## deadwood    -0.023  0.003 -0.002 -0.055
```

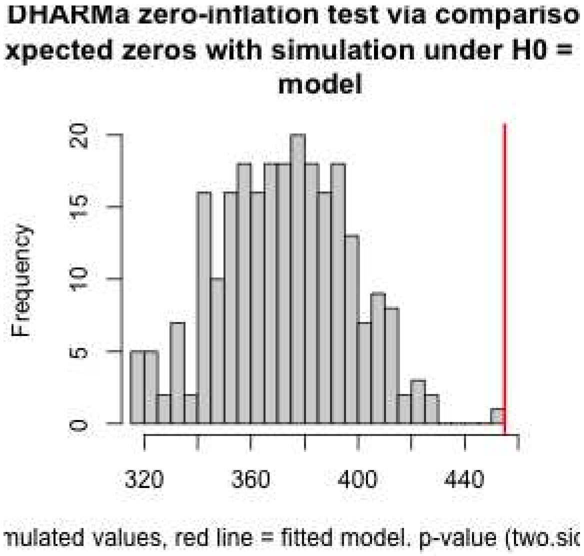
still, there is obviously overdispersion in the data, so we'll add an individual level random effect to account for overdispersion

```
mod <- glmer(beetles ~ altitude + I(altitude^2) + moisture + d
simulationOutput <- simulateResiduals(fittedModel = mod)
plot(simulationOutput)
```



The data still looks overdispersed. The reason is that there is in fact no standard overdispersion, but zero-inflation in the data. We can look at the excess zeros via

```
testZeroInflation(simulationOutput)
```



```
##
##  DHARMa zero-inflation test via comparison to expected zero
##  simulation under H0 = fitted model
##
## data:  simulationOutput
## ratioObsSim = 1.2203, p-value < 2.2e-16
## alternative hypothesis: two.sided
```

which shows that we have too many zeros. We need a GLMM with zero-inflation. The easiest option is to do this in a Bayesian framework, e.g. in JAGS as in this example

To be honest, however, if I hadn't created the data with zero-inflation myself, I would be hard pressed to say with certainty that zero-inflation is the cause of these residual patterns. The reason is that, if a model is presented with zero-inflated data, the mean model predictions will be drawn towards the zeros, which in turn means that mean model-predictions underestimate the "normal" data. Hence zero-inflation often looks like overdispersion, and seeing such a pattern after accounting for overdispersion might as well mean that the distributional assumptions of the chosen model are not fulfilled (for example because overdispersion is more heavy-tailed than in the chose distribution). The best way to test if the problem is really due to zero-inflation is probably to run model selections (e.g. simulated LRTs) between a number of alternative models, e.g. a zero-inflated GLMM vs. a number of different overdispersed GLMMs.

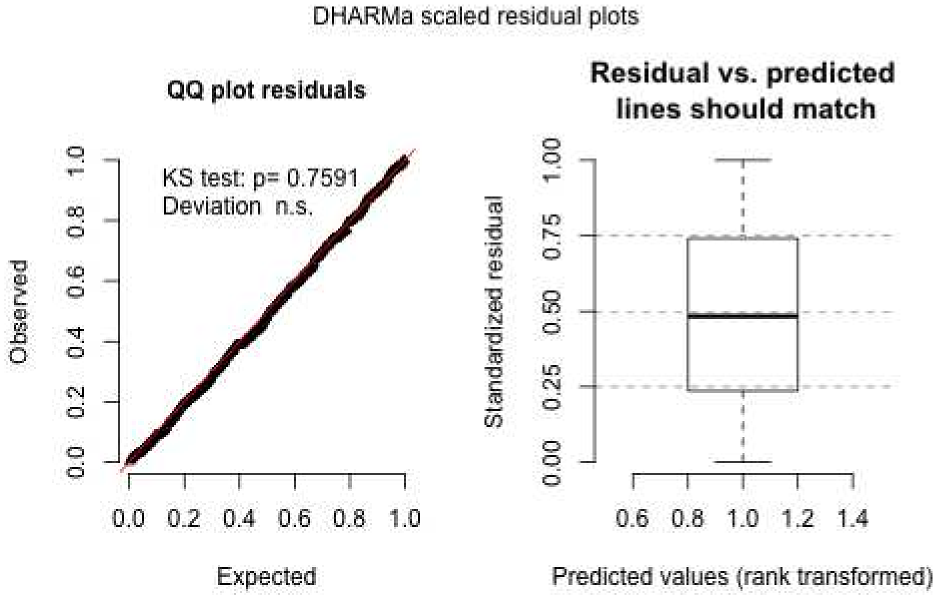
Binomial 0/1 data

There are a lot of rumors about that can and cannot be checked with binomial 0/1 data. Let's consider a clearly misspecified binomial model

```
testData = createData(sampleSize = 500, overdispersion = 0, fi
fittedModel <- glm(observedResponse ~ 1, family = "binomial",

simulationOutput <- simulateResiduals(fittedModel = fittedMode
```

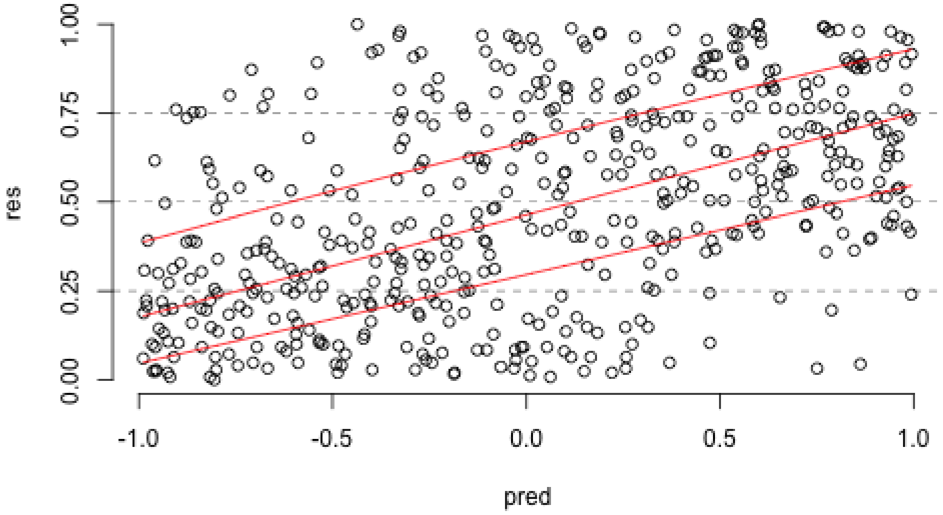
A rumor that is true is that, unlike in k/n or count data, such a misspecification will not produce overdispersion.

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



However, you can clearly see the misfit if you plot

```
plotResiduals(testData$Environment1, simulationOutput$scaledRe
```

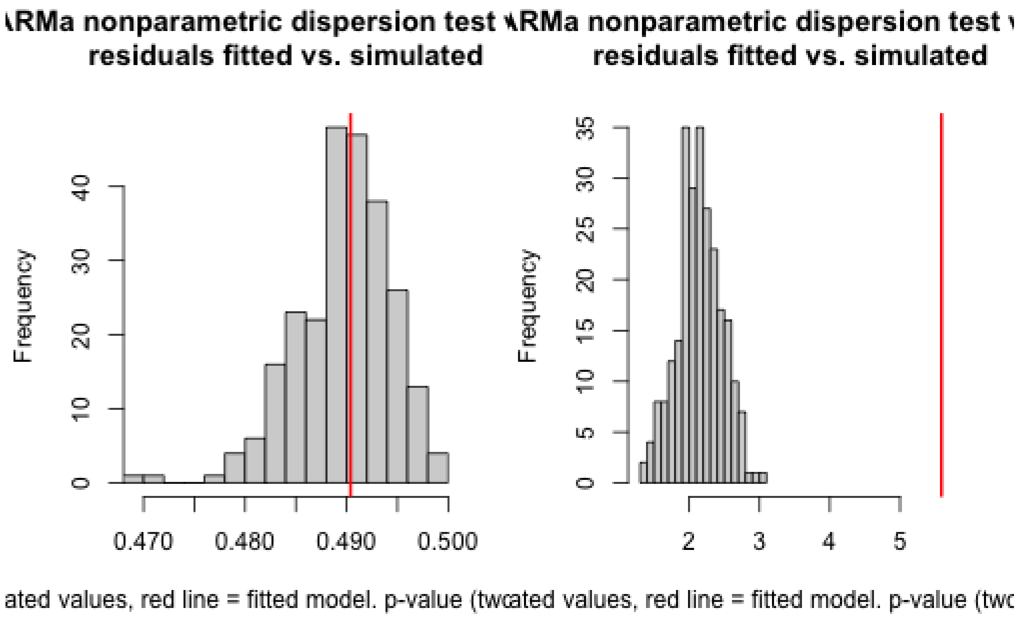


Moreover, if you perform the DHARMa procedure per group, you can see overdispersion created by the random effect variance (left without grouping, right with grouping)

```
par(mfrow = c(1,2))
testDispersion(simulationOutput)
```

```
##
##  DHARMa nonparametric dispersion test via sd of residuals f
##  vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 1.0012, p-value = 1
## alternative hypothesis: two.sided
```

```
simulationOutput = recalculateResiduals(simulationOutput , gro
testDispersion(simulationOutput)
```



```
##
##  DHARMa nonparametric dispersion test via sd of residuals f
##  vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 2.6099, p-value < 2.2e-16
## alternative hypothesis: two.sided
```