Generalized Linear Mixed Models (GLMM)

NRES 710

Fall 2020

Download the R code for this lecture!

To follow along with the R-based lessons and demos, right (or command) click on this link and save the script to your working directory

Overview: Generalized Linear Models

Generalized Linear Models (GLM) are not generally covered in intro stats classes, but they are so flexible and so common in ecology and environmental science that you really need to at least know that these models exist!

The real data sets that we deal with as ecologists and environmental scientists tend to violate some key assumptions of classical linear regression and ANOVA. In particular, residuals are often decidedly non-normal and variance is decidedly non-equal (heteroskedastic) across the range of predictions.

Generalized linear models allow us to model response variables that are not amenable to classical linear regression – but allows us to use a model structure that closely resembles linear regression. Pretty much everything about running a GLM is like linear regression. The primary function for running GLM models ('glm') even looks very similar to the regression function 'lm'.

GLMs are parametric analyses – it's just that (1) we don't need to assume our response variable is normally distributed and (2) we don't need to assume the relationship between the response variable and the predictor variable(s) is linear on the scale of the untransformed response variable. Let's look into each of these in more detail:

Alternative distributions

So we don't need to assume the response variable is normally distributed – but we do need to assume it is distributed according to some known probability distribution – and we need to specify what distribution we ARE assuming. We can assume that the response process is Poisson distributed, or gamma distributed, or beta distributed, or any of a host of other distributions. But again, we have to specify which distribution!

Link functions

So we don't need to assume the relationship between the mean of the response variable and the predictor variable(s) is linear on the scale of the untransformed response variable – but the hypothesized relationship must be linear on some transformation of the response variable – and we need to specify what transformed version of the response process we wish to assume linearity for. This is called the 'link function'.

In the general case, a GLM can be described by the following pseudo-equations:

$f(\bar{y}) = \beta_0 + \beta_1 \cdot x_1 + \beta_2 \cdot x_2 \ldots$

The left side of this equation is the mean of the response variable, transformed according to the specified link function. The right side of this equation is called the **linear predictor** and describes how the (transformed) mean response varies as a function of the predictor variable(s).

To complete the picture, we need to represent the error/variability. To do this we represent the raw response as being distributed according to a distribution we specify:

$y \sim My.Dist(\ldots)$

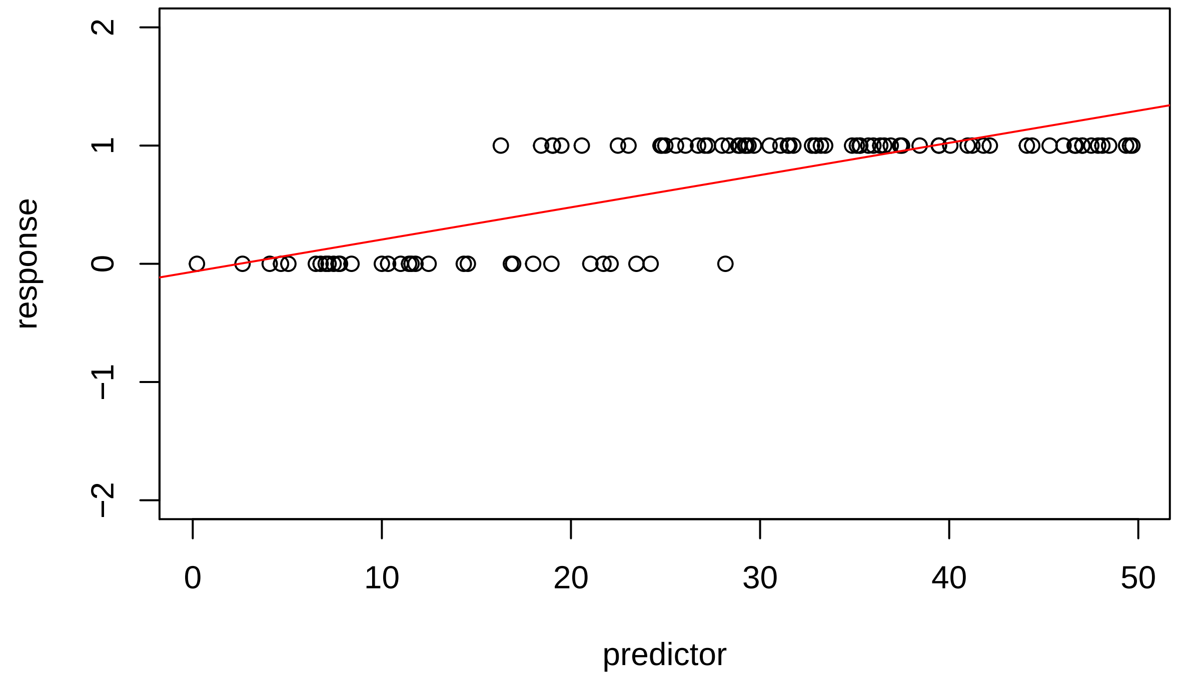
My.Dist can be the binomial distribution, Poisson distribution, or a number of other possibilities. The mean of the distribution is described by the covariates (preditor variables) as described in the top equation.

A simple example (logistic regression)

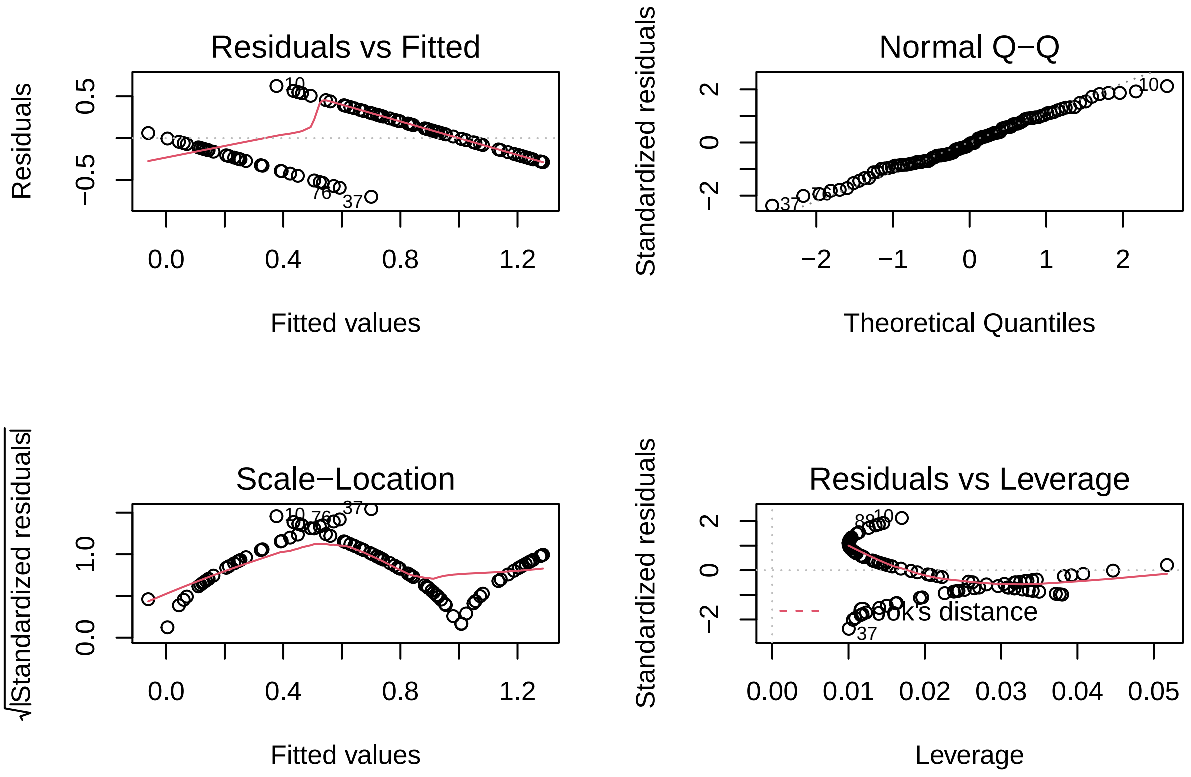
For example, we might have a binary response variable and a continuous predictor variable. Making the assumption of linearity would not necessarily make sense in this case.

Let's first make up an example:

```r
## made up data for glm #1 (logistic regression)

predictor <- runif(100,0,50)
response <- rbinom(100,1, plogis(-5 + 0.26*predictor) )

plot(response~predictor,ylim=c(-2,2))
abline(lm(response~predictor),col="red")    # overlay regression line
```



```r
layout(matrix(1:4,nrow=2,byrow=2))
plot(lm(response~predictor))
```



Note that the predicted mean response at high values of the predictor exceed 1- which is impossible for a binary response. Clearly we can't make the assumption of linearity on the untransformed binary response. Furthermore, the other diagnostic plots also do not look great…

Another way of saying this is that a binary response cannot go below zero or above 1, whereas a linear function of predictor variables (linear predictor) can do both!

One transformation that makes sense for a regression with a binary response is the **logit transformation**. The logit transformation is commonly used to take probabilities (which are constrained between 0 and 1) and transform them to values that can vary between -Inf and Inf.

That way, the linear predictor (which is not constrained and can therefore vary between -Inf and Inf) will always be interpretable as a probability!

For example, take the following probabilities:

```r
probs <- runif(10)
probs
```

```
##  [1] 0.02890619 0.90703641 0.36273363 0.36606820 0.29724323 0.53495948 0.81031538 0.61109969
##  [9] 0.60652891 0.72685561
```

Here's what happens if we apply the logit transformation:

$logit(p) = log(\frac{p}{(1-p)})$

```r
data.frame(
  p = probs,
  logit.p=log(probs/(1-probs))
)
```

If our response variable is binary and we want to assume that the mean response (on some transformed scale) is linearly dependent on our predictor variable, the logit transformation is a good candidate for our link function, because this way the mean response will never go below zero or above one.

So instead of:

$\bar{y} = \beta_0 + \beta_1 \cdot x$

We can use the **logit link** and assume instead that:

$logit(\bar{y}) = \beta_0 + \beta_1 \cdot x$

If we solve for y, this equation becomes:

$\bar{y} = \frac{e^{\beta_0+\beta_1 \cdot x}}{1+e^{\beta_0+\beta_1 \cdot x}}$

This is what we do when we conduct a *logistic regression*! Specifically, in a logistic regression we assume the following:

Response distribution: binomial with size=1 (can only be zero or 1; also known as a Bernoulli distribution)

Link function: logit (mean of the response is a linear function of the predictor variable(s), on the logit scale)

We use a binomial response distribution because our response variable is discrete, just like a binomial distribution. If you flip one coin you can get only a zero or a one, just like the response variable. The binomial distribution matches the response variable, so it is an appropriate distribution to assume!

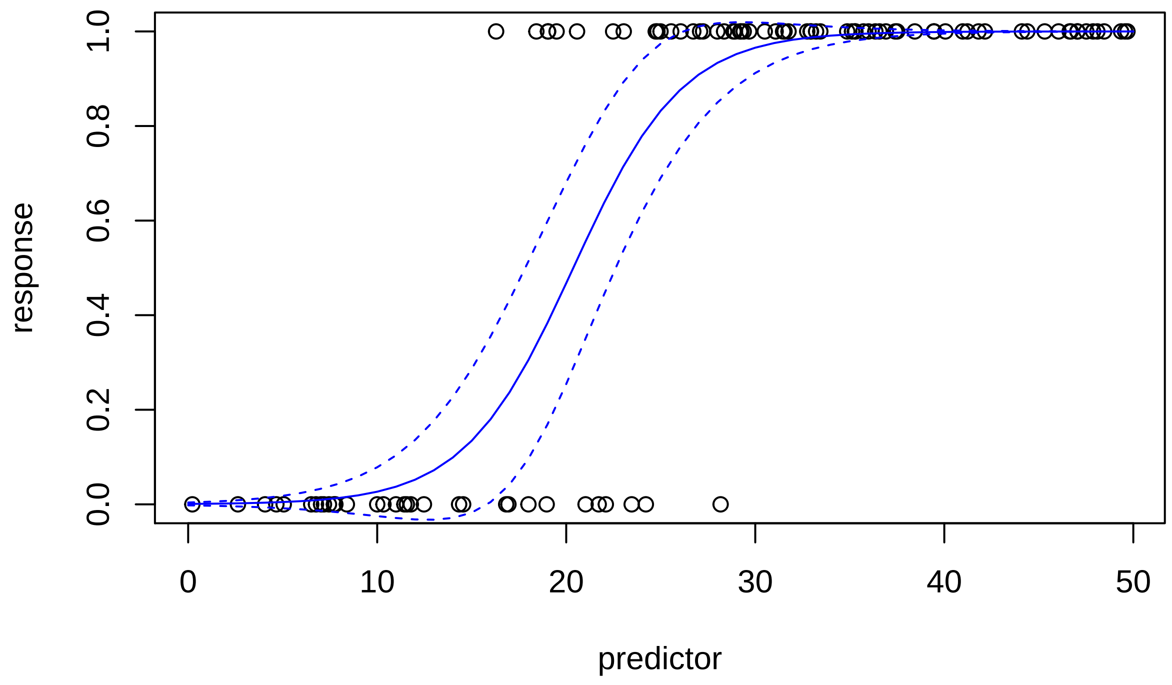
NOTE: it may not seem obvious, but the mean of a binary random variable is the probability of success (probability of getting a one). That is, if you repeatedly sampled from a random binary process many many times, the expected mean of those numbers (fraction of the total number of samples that are equal to one) is the same thing as the probability of success.

```r
  ## conduct logistic regression:

model <- glm(response~predictor,family=binomial(link="logit"))    # logistic regression in R
summary(model)   # summary looks similar to ordinary linear regression!
```

```
##
## Call:
## glm(formula = response ~ predictor, family = binomial(link = "logit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.35193  -0.13495   0.03897   0.22568   1.80712
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -7.06119    1.74949  -4.036 5.43e-05 ***
## predictor    0.34657    0.07972   4.347 1.38e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 125.374  on 99  degrees of freedom
## Residual deviance:  38.322  on 98  degrees of freedom
## AIC: 42.322
##
## Number of Fisher Scoring iterations: 7
```

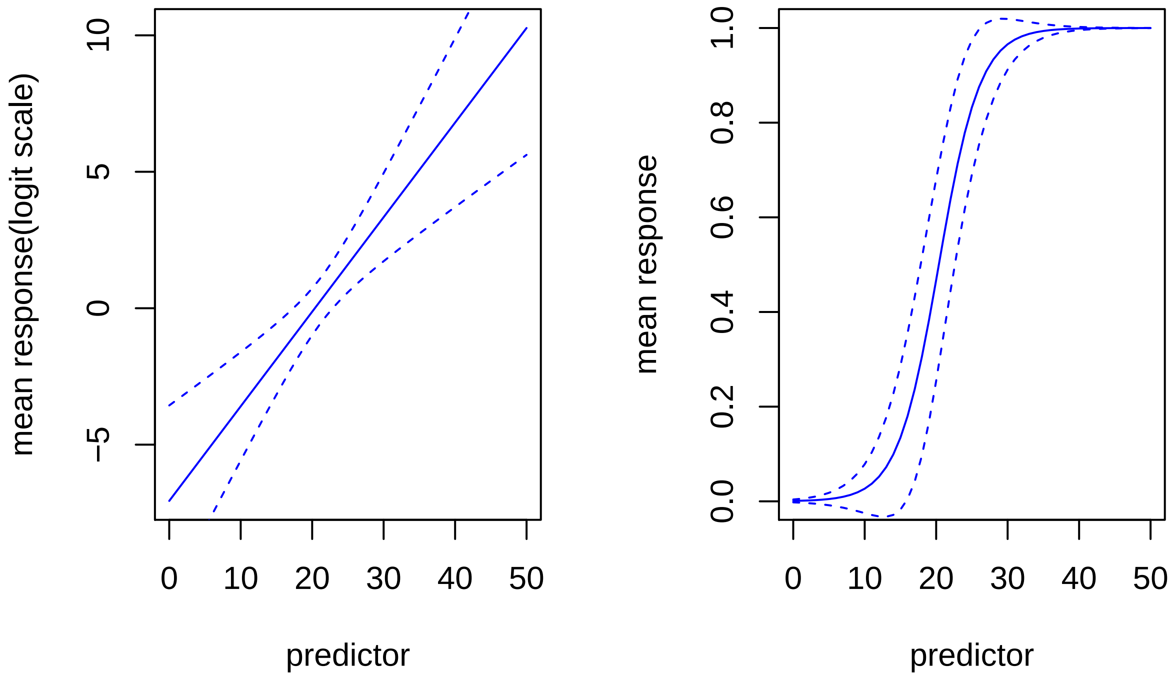
```
newdat <- data.frame(        # make predictions for plotting regression line and approx conf bounds
  predictor = seq(0,50,1)
)

mypred <- predict(model,type="response",se.fit=T,newdata = newdat)

plot(response~predictor)
lines(newdat$predictor,mypred$fit,col="blue")
lines(newdat$predictor,mypred$fit+2*mypred$se.fit,col="blue",lty=2)
lines(newdat$predictor,mypred$fit-2*mypred$se.fit,col="blue",lty=2)
```



Note that the relationship between the response and predictor looks non-linear. But this is not the same thing as non-linear regression. GLM is a type of linear model for a reason. It's just that the relationship is assumed to be linear on the logit scale. Here is another visualization of the same exact model:

```
par(mfcol=c(1,2))

mypred <- predict(model,type="link",se.fit=T,newdata = newdat)
```

```r
plot(newdat$predictor,mypred$fit,col="blue",type="l",ylab="mean response(logit scale)",xlab="predictor"
lines(newdat$predictor,mypred$fit+2*mypred$se.fit,col="blue",lty=2)
lines(newdat$predictor,mypred$fit-2*mypred$se.fit,col="blue",lty=2)


mypred <- predict(model,type="response",se.fit=T,newdata = newdat)

plot(newdat$predictor,mypred$fit,col="blue",type="l",ylab="mean response",xlab="predictor")
lines(newdat$predictor,mypred$fit+2*mypred$se.fit,col="blue",lty=2)
lines(newdat$predictor,mypred$fit-2*mypred$se.fit,col="blue",lty=2)
```



Another simple example (Poisson count regression)

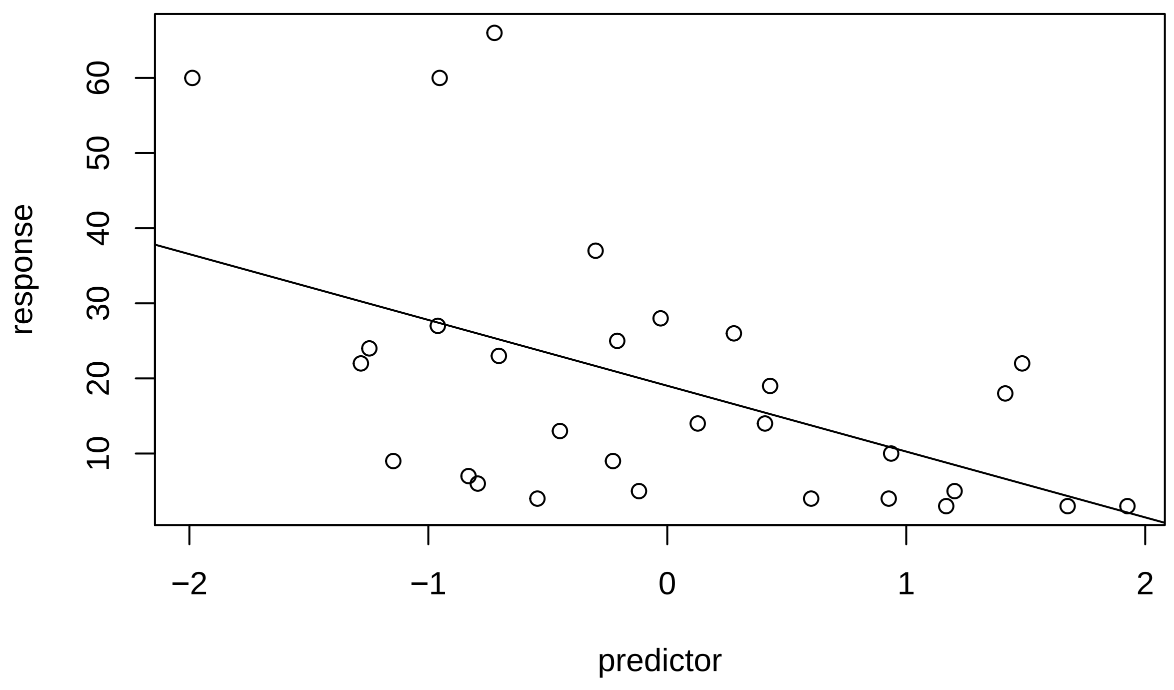
Sometimes our measured response is a *count* of something (e.g., number of stems in a plot). In such cases, our response variable cannot go below zero- and the response variable should ideally come from a discrete distribution that only allows integers. The simplest way to model this is:

Response distribution: Poisson
Link function: Natural logarithm

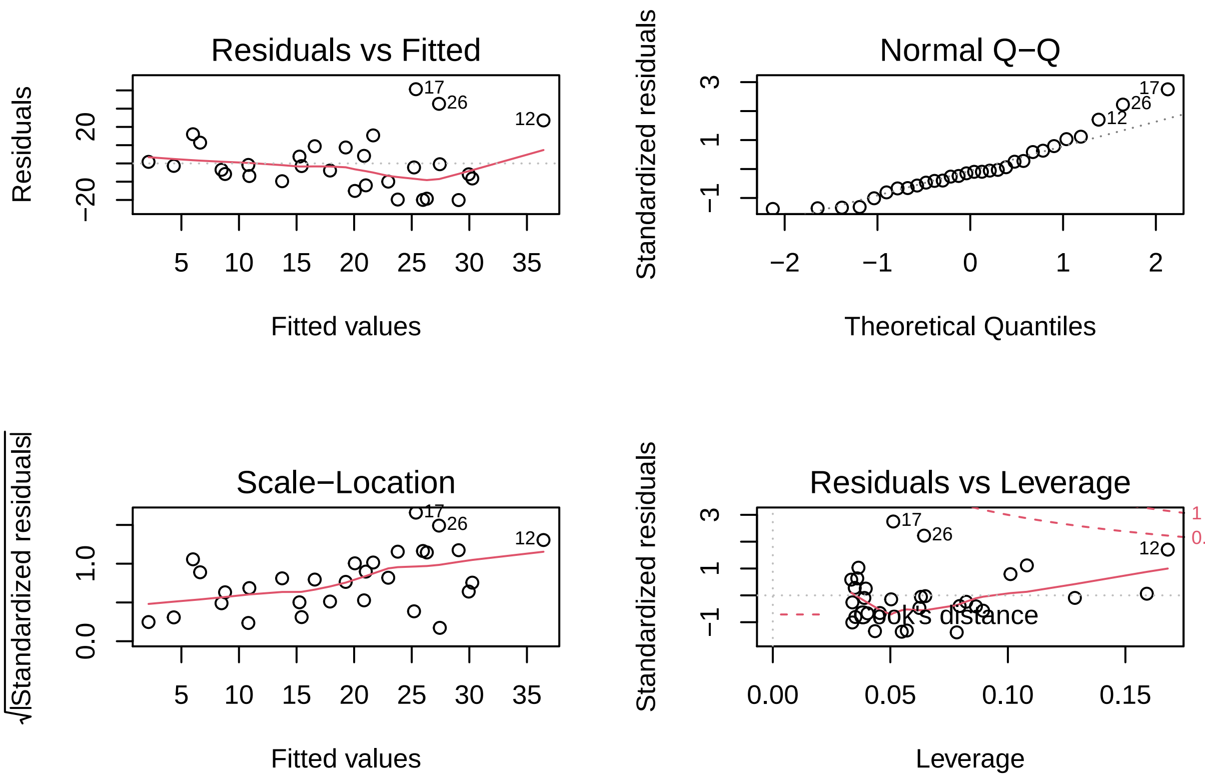
The Poisson distribution (with only one parameter) is the simplest discrete probability distribution, and the (natural) log is the simplest link function that maps a quantity with a lower bound of zero to a quantity with a lower bound of -Inf. After all, a count cannot go below zero, whereas a linear function of predictor variables can!

Let's make up some count data:

```r
# Count regression example

predictor = runif(30,-2,2)
response = rnbinom(30,mu=exp(3-0.5*predictor),size=2)      # make up data!

plot(response~predictor)
abline(lm(response~predictor))
```



```r
par(mfrow=c(2,2))
plot(lm(response~predictor))
```

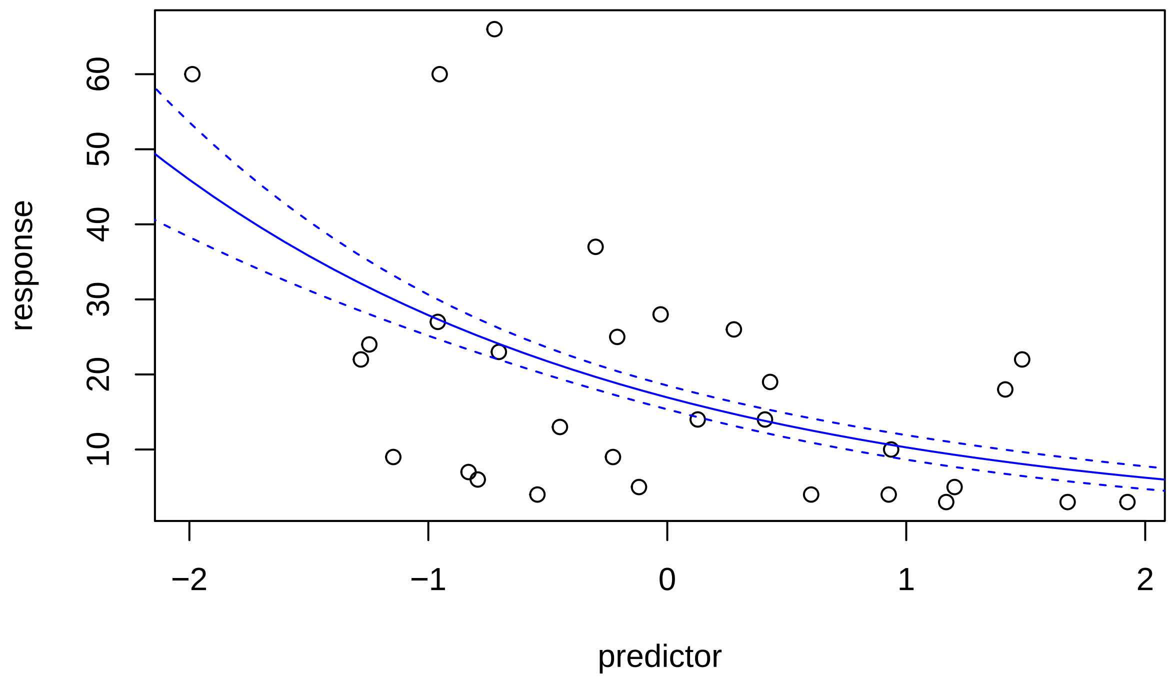


Here we see some potential issues with ordinary linear regression and we might consider Poisson count regression instead!

```r
## try Poisson count regression model!

model <- glm(response~predictor,family=poisson(link="log"))
summary(model)
```

```
##
## Call:
## glm(formula = response ~ predictor, family = poisson(link = "log"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -4.7653  -2.4269  -0.9443   1.8585   6.9647
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  2.82941    0.04680   60.46   <2e-16 ***
## predictor   -0.49898    0.04642  -10.75   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 399.51  on 29  degrees of freedom
## Residual deviance: 274.94  on 28  degrees of freedom
## AIC: 411.04
##
## Number of Fisher Scoring iterations: 5
```

```
plot(response~predictor)

newdat <- data.frame(
  predictor = seq(-3,3,0.1)
)

mypred <- predict(model,type="response",se.fit = T,newdata=newdat)

lines(newdat$predictor,mypred$fit,col="blue")
lines(newdat$predictor,mypred$fit+2*mypred$se.fit,col="blue",lty=2)
lines(newdat$predictor,mypred$fit-2*mypred$se.fit,col="blue",lty=2)
```



Diagnostic testing with GLM

Obviously, the standard diagnostic plots don't make much sense for GLM– after all, they are testing assumptions that we are no longer making! We don't need to test for normality of residuals if we are assuming our response variable is binomially distributed! We don't need to test for homogeneity of variance if our assumed probability distribution is heteroskedastic!

The Poisson distribution, for example, does not have homogeneous variance- in fact, the variance of the Poisson distribution is equal to the mean. So the larger the expected value, the larger the variance!

However, we need some way of testing whether the distribution we selected is a reasonable fit to our data. We could use so-called deviance residuals here (which is the default in r) but I prefer to use the DHARMa package in R.

NOTE: the DHARMa package also works for GLMM models (generalized linear mixed-effects models- see below!)

```
residuals(model)  # compute the deviance residuals for the poisson regression model
```

```
##           1           2           3           4           5           6           7           8
##  2.64536939 -1.52897913 -0.18955358  3.47768489 -4.76528872  2.88531836 -2.81787897  2.39214123
##           9          10          11          12          13          14          15          16
## -0.48450288 -2.45513986  4.03263178  2.02387510 -2.34202823 -4.37205205 -0.06638749 -4.59506624
##          17          18          19          20          21          22          23          24
##  6.96469076  0.05096160 -3.62516381 -2.55307666 -1.40408482 -0.22050549 -1.54574566  1.36019278
##          25          26          27          28          29          30
## -1.91780792  5.40722699 -4.51049432 -1.81991153  1.36245709 -1.89297208
```

```
summary(residuals(model))   # median should be near zero
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -4.7653 -2.4269 -0.9443 -0.3501  1.8585  6.9647
```

```
paste0(c("Null deviance: ", "Residual deviance: "),     # null deviance should be much higher than resi
       round(c(model$null.deviance, deviance(model)), 2))
```

```
## [1] "Null deviance: 399.51"     "Residual deviance: 274.94"
```

```
paste0(c("model df: ", "Residual deviance: "),     # resid deviance should be close to residual df
       round(c(model$df.residual, deviance(model)), 2))
```

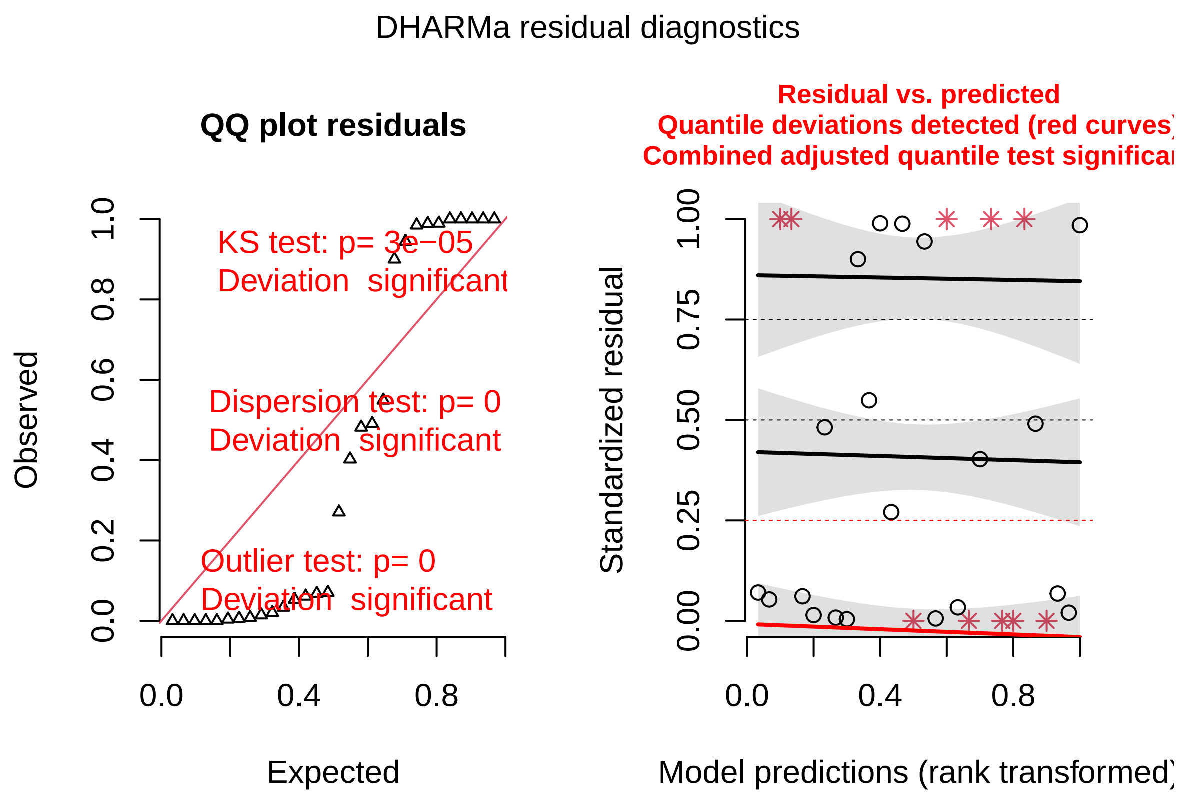
```
## [1] "model df: 28"              "Residual deviance: 274.94"
```

So using the deviance residuals we're starting to get a picture that the Poisson distribution may not be a great fit. Let's use the DHARMa package now…

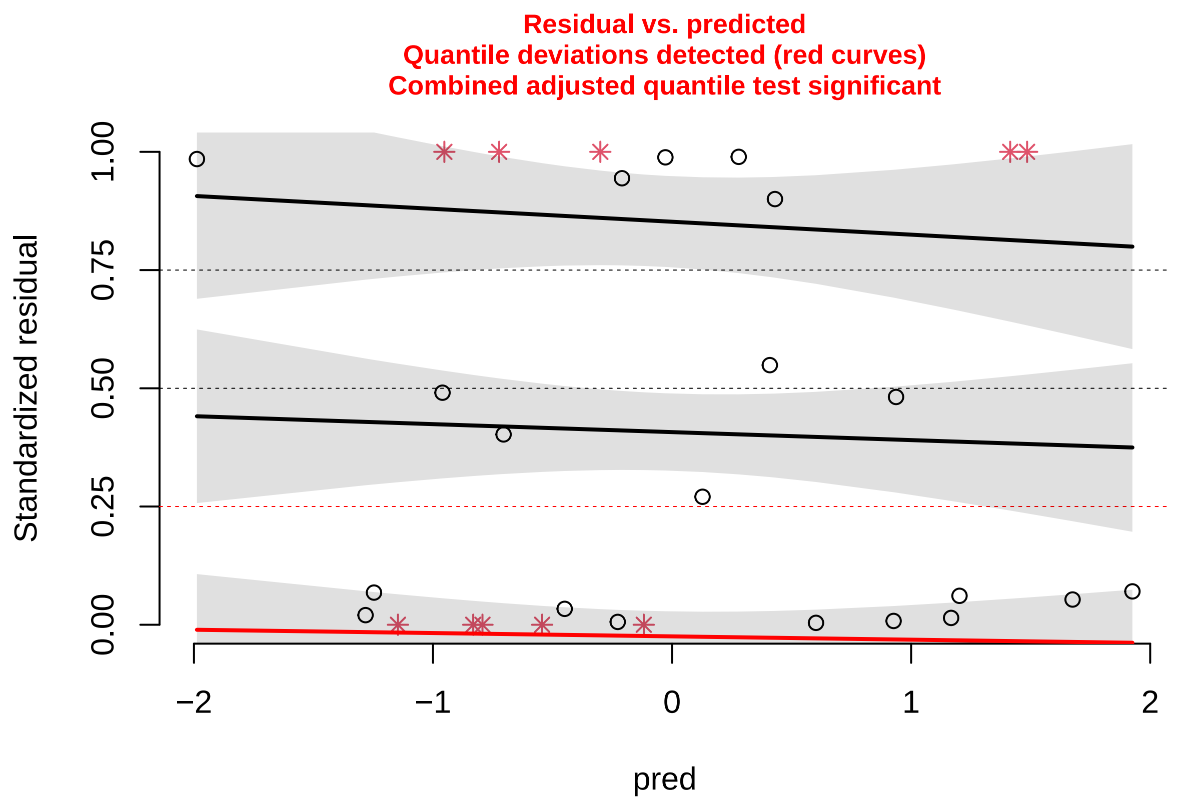
```
library(DHARMa)
```

```
## This is DHARMa 0.3.3.0. For overview type '?DHARMa'. For recent changes, type news(package = 'DHARMa
```

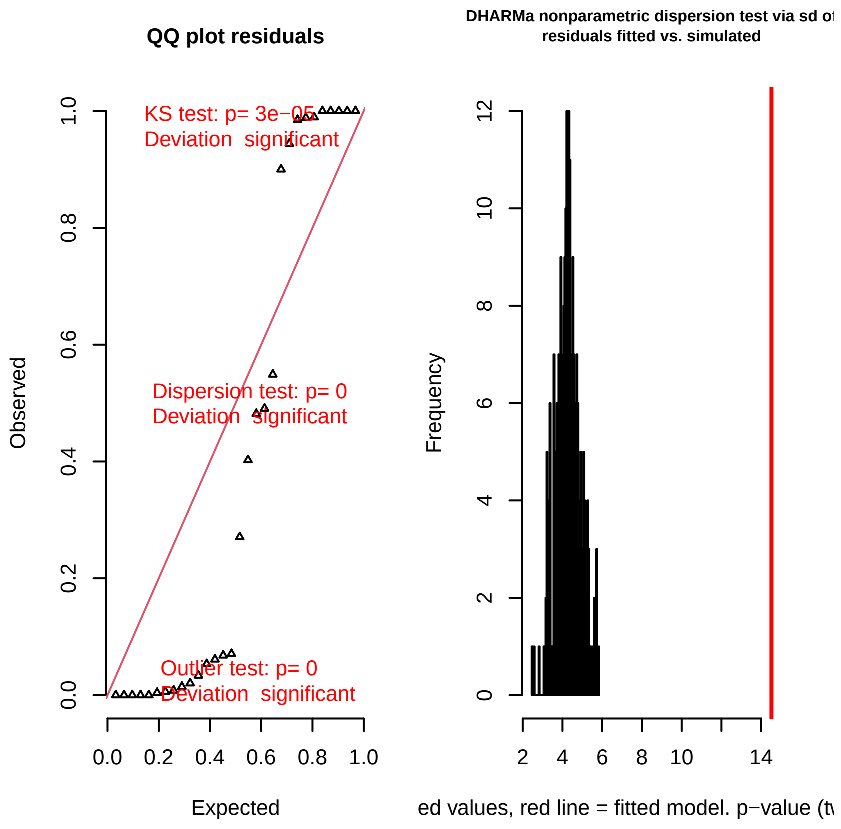
```
simresids <- simulateResiduals(model,n=250,plot=T)   # clearly this is a bad fit!
```

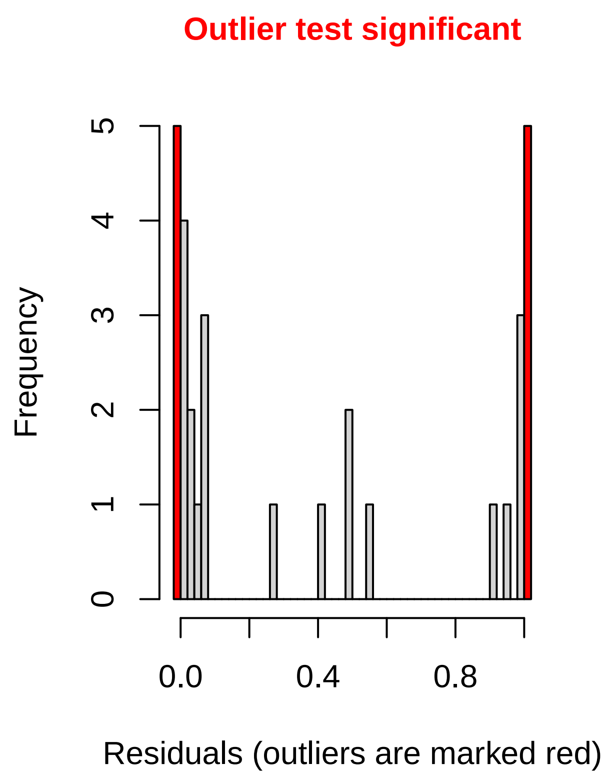


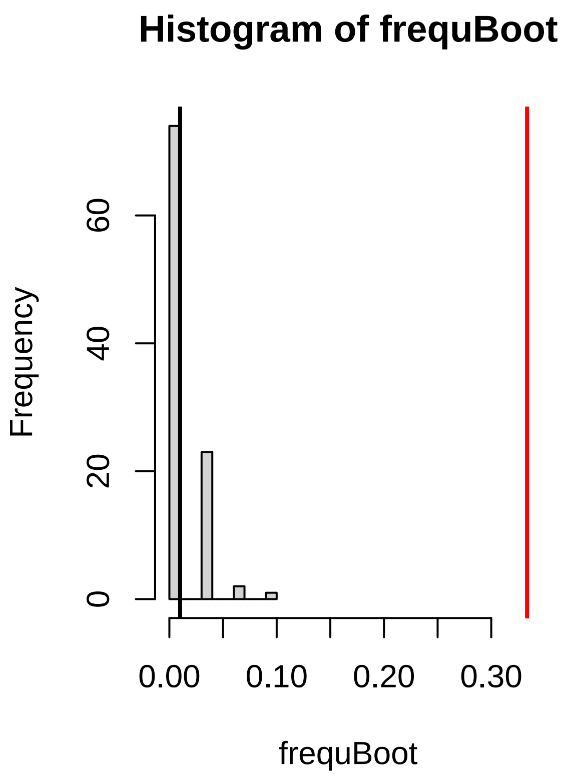
```
plotResiduals(simresids,predictor)    # look for patterns across a predictor variable
```



```
testResiduals(simresids)   # run tests on the residuals!
```


QQ plot residuals
Observed
Expected
KS test: p= 3e−05
Deviation significant
Dispersion test: p= 0
Deviation significant
Outlier test: p= 0
Deviation significant
0.0
0.2
0.4
0.6
0.8
1.0
DHARMa nonparametric dispersion test via sd of
residuals fitted vs. simulated
Frequency
0
2
4
6
8
10
12
14
ed values, red line = fitted model. p−value (tw





```
## $uniformity
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.42946, p-value = 3.126e-05
## alternative hypothesis: two-sided
##
##
## $dispersion
##
##  DHARMa nonparametric dispersion test via sd of residuals fitted vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 3.397, p-value < 2.2e-16
## alternative hypothesis: two.sided
##
##
## $outliers
##
##  DHARMa bootstrapped outlier test
##
## data:  simulationOutput
## outliers at both margin(s) = 10, observations = 30, p-value < 2.2e-16
## alternative hypothesis: two.sided
##  percent confidence interval:
```

```
##  0.00000000 0.05083333
## sample estimates:
## outlier frequency (expected: 0.01 )
##                          0.3333333
```

```
## $uniformity
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.42946, p-value = 3.126e-05
## alternative hypothesis: two-sided
##
##
## $dispersion
##
##  DHARMa nonparametric dispersion test via sd of residuals fitted vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 3.397, p-value < 2.2e-16
## alternative hypothesis: two.sided
##
##
## $outliers
##
##  DHARMa bootstrapped outlier test
##
## data:  simulationOutput
## outliers at both margin(s) = 10, observations = 30, p-value < 2.2e-16
## alternative hypothesis: two.sided
##  percent confidence interval:
##  0.00000000 0.05083333
## sample estimates:
## outlier frequency (expected: 0.01 )
##                          0.3333333
```

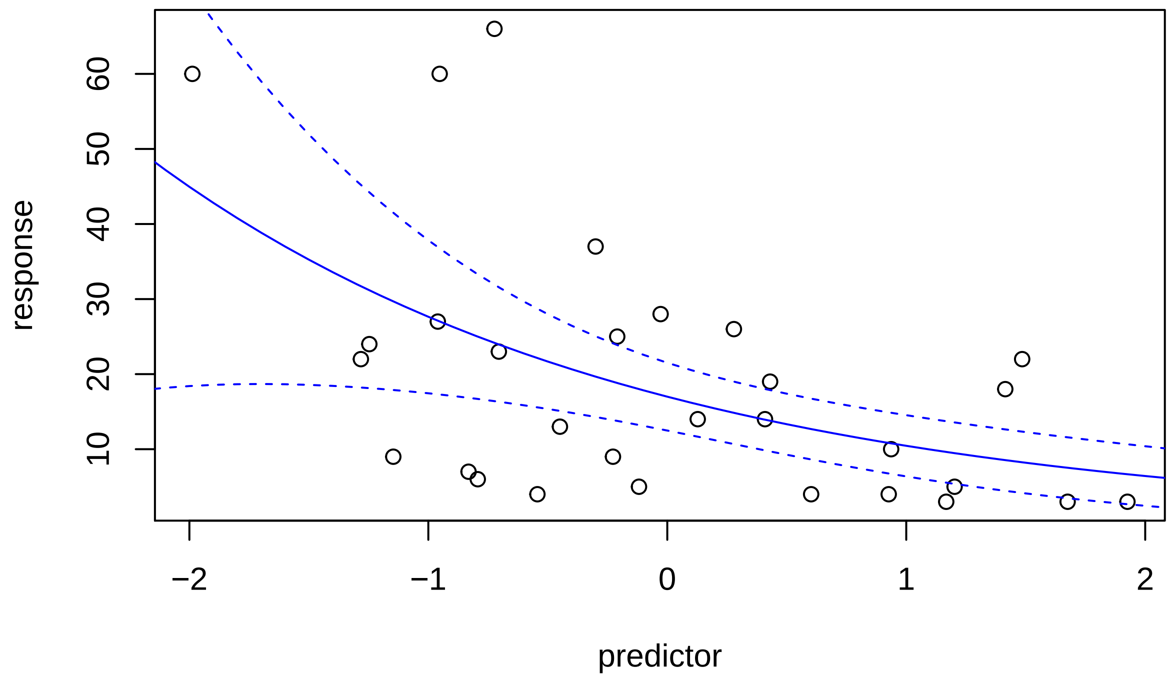
Okay so the DHARMa package diagnostics seem to indicate that the Poisson regression was a poor fit to the data (you will find this is usually true with Poisson regression). Let's try running a negative binomial regression instead!

```r
## try NegBinom count regression model!

library(MASS)

## NOTE: in reality you should use glm.nb because you don't know the additional parameter theta!
model <- glm(response~predictor,family=negative.binomial(link="log",theta = 2))
summary(model)
```

```
##
## Call:
## glm(formula = response ~ predictor, family = negative.binomial(link = "log",
##     theta = 2))
##
```

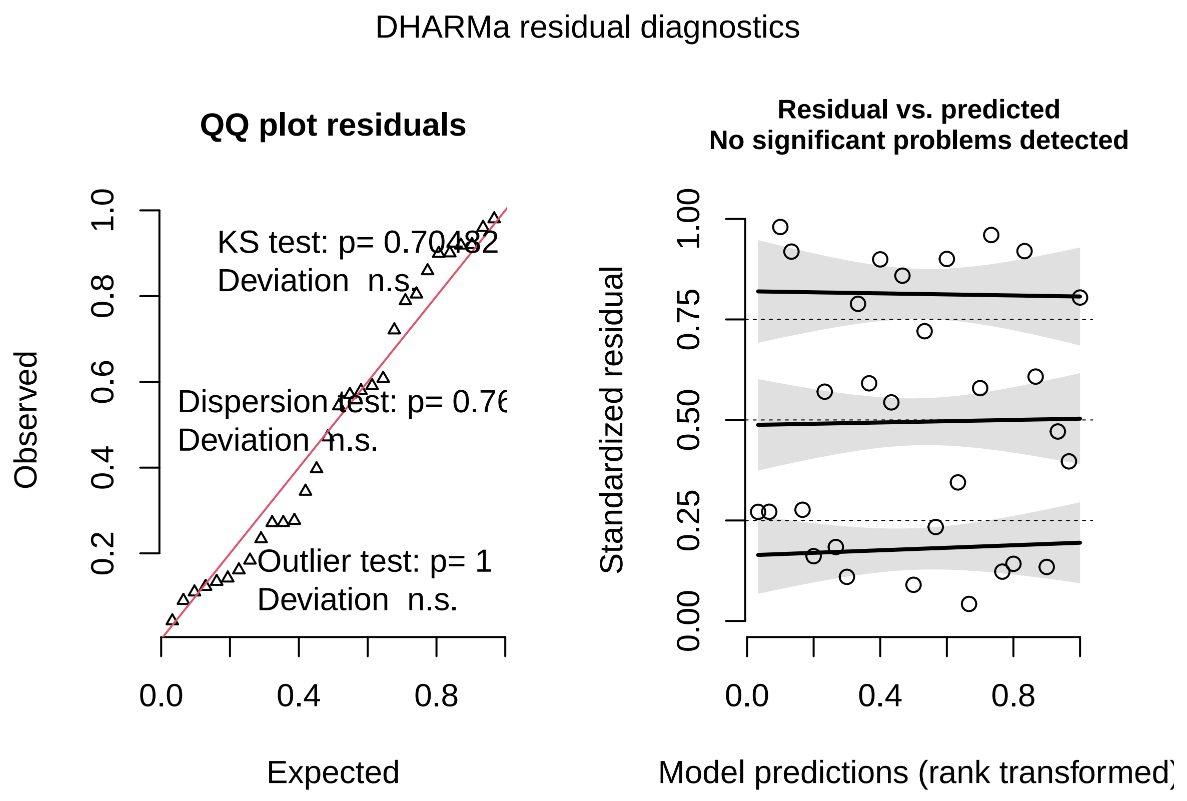
```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.7368  -1.0458  -0.2571   0.4452   1.6541
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.8330     0.1384  20.472  < 2e-16 ***
## predictor    -0.4863     0.1413  -3.441  0.00184 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(2) family taken to be 1.015104)
##
##     Null deviance: 41.429  on 29  degrees of freedom
## Residual deviance: 29.358  on 28  degrees of freedom
## AIC: 229.03
##
## Number of Fisher Scoring iterations: 5
```

```r
model <- glm.nb(response~predictor)

plot(response~predictor)

newdat <- data.frame(
  predictor = seq(-3,3,0.1)
)

mypred <- predict(model,type="response",se.fit = T,newdata=newdat)

lines(newdat$predictor,mypred$fit,col="blue")
lines(newdat$predictor,mypred$fit+2*mypred$se.fit,col="blue",lty=2)
lines(newdat$predictor,mypred$fit-2*mypred$se.fit,col="blue",lty=2)
```

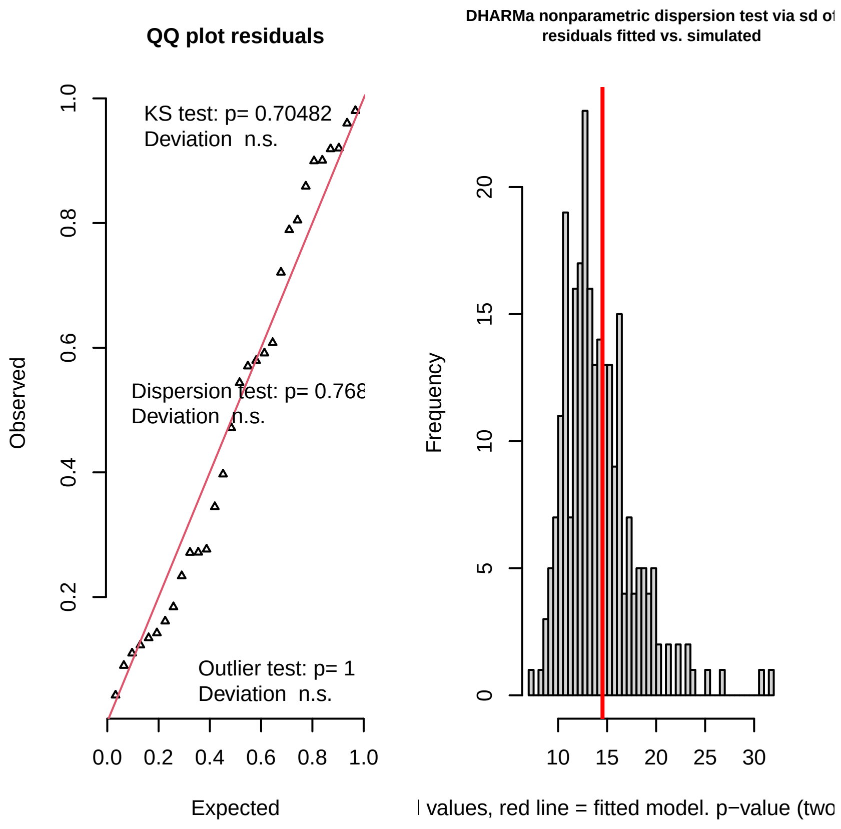


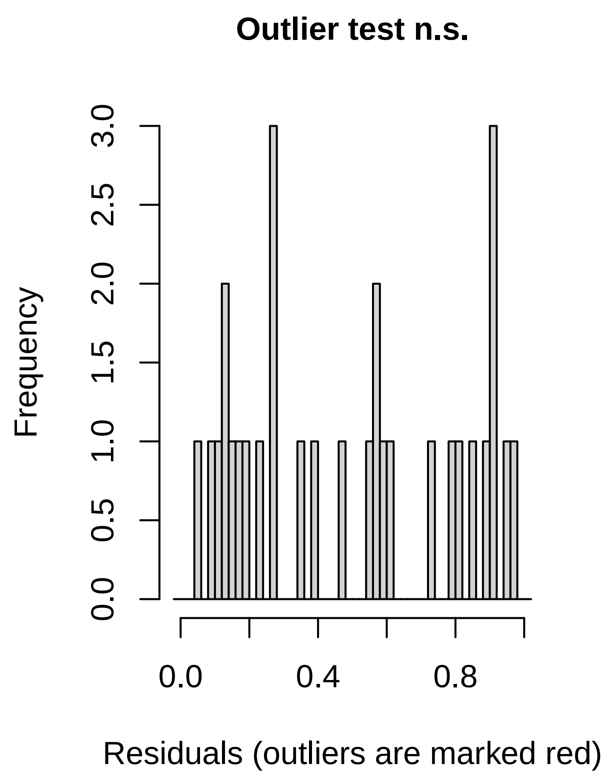
Now let's check the model fit!

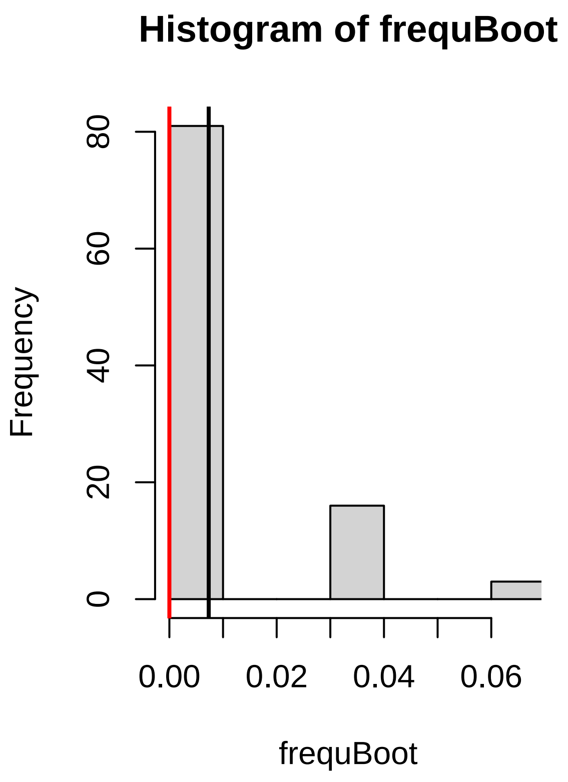
```
simresids <- simulateResiduals(model,n=250,plot=T)    # looks a lot better!
```



```r
testResiduals(simresids)   # run tests on the residuals!
```


QQ plot residuals
KS test: p= 0.70482
Deviation n.s.
Dispersion test: p= 0.768
Deviation n.s.
Outlier test: p= 1
Deviation n.s.
Observed
Expected
0.0
0.2
0.4
0.6
0.8
1.0
DHARMa nonparametric dispersion test via sd of
residuals fitted vs. simulated
Frequency
0
5
10
15
20
25
30
l values, red line = fitted model. p−value (two





```
## $uniformity
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.12344, p-value = 0.7048
## alternative hypothesis: two-sided
##
##
## $dispersion
##
##  DHARMa nonparametric dispersion test via sd of residuals fitted vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 1.027, p-value = 0.768
## alternative hypothesis: two.sided
##
##
## $outliers
##
##  DHARMa bootstrapped outlier test
##
## data:  simulationOutput
## outliers at both margin(s) = 0, observations = 30, p-value = 1
## alternative hypothesis: two.sided
##  percent confidence interval:
```

```
##  0.00000000 0.05083333
## sample estimates:
## outlier frequency (expected: 0.00733333333333333 )
##                                                   0
```

```
## $uniformity
##
##  One-sample Kolmogorov-Smirnov test
##
## data:  simulationOutput$scaledResiduals
## D = 0.12344, p-value = 0.7048
## alternative hypothesis: two-sided
##
##
## $dispersion
##
##  DHARMa nonparametric dispersion test via sd of residuals fitted vs. simulated
##
## data:  simulationOutput
## ratioObsSim = 1.027, p-value = 0.768
## alternative hypothesis: two.sided
##
##
## $outliers
##
##  DHARMa bootstrapped outlier test
##
## data:  simulationOutput
## outliers at both margin(s) = 0, observations = 30, p-value = 1
## alternative hypothesis: two.sided
##  percent confidence interval:
##  0.00000000 0.05083333
## sample estimates:
## outlier frequency (expected: 0.00733333333333333 )
##                                                   0
```

And this time our count regression model fits the data well, and we can report the results of this model with confidence!

NOTE: you should always run goodness-of-fit tests for any model you fit to data. But it is especially important in the case of Poisson regression, because in general most count data has much more variance that a Poisson distribution allows. This can lead to inflated type 1 error and can be a huge problem. You should always be wary of reported Poisson regression results with no goodness of fit checks!

Model selection with AIC

Often we have multiple **candidate models** for describing how our response variable relates to one or more of our predictor variables. This is true for multiple linear regression and GLM models, and mixed-effects models (see below) and non-linear regression models.

Information-theoretic criteria like **Akaike's Information Criterion** provide a common currency that allows us to compare and rank multiple models.

In general, the models with the lowest AIC are better than models with higher AIC.

AIC is defined as:

$AIC = -2 \cdot ln(Likelihood) + 2 \cdot k$

Where k is the number of fitted parameters in the model and *Likelihood* is the maximum likelihood of the data (probability of the observed data set under the fitted model).

There is a commonly used correction for small sample size called $AICc$:

$AICc = AIC + \frac{2k^2+2k}{n-k-1}$

Comparing multiple GLM models using AIC is relatively simple!

```
######
# Make up data!

predictor1 = runif(30,-2,2)
predictor2 <- runif(30,-100,100)
predictor3 <- rnorm(30)   # useless predictor
response = rnbinom(30,mu=exp(3-0.5*predictor1+0.01*predictor2),size=2)



###
# fit a bunch of candidate models

model.pois.all <- glm(response~predictor1+predictor2+predictor3,family="poisson")
model.nb.all <- glm.nb(response~predictor1+predictor2+predictor3)
model.nb.1 <- glm.nb(response~predictor1)
model.nb.12 <- glm.nb(response~predictor1+predictor2)
model.nb.2 <- glm.nb(response~predictor2)

cand.set <- list(
  Poisson=model.pois.all,
  NegBin_allvars = model.nb.all,
  NegBin_pred1 = model.nb.2,
  NegBin_preds1and2 = model.nb.12,
  NegBin_pred2 = model.nb.2
)

### Make AIC table

AICtab <- data.frame(
  ModelName = names(cand.set),
  LogLikelihood = sapply(cand.set,logLik),
  AIC = sapply(cand.set,AIC)
)

AICtab$DeltaAIC <- abs(AICtab$AIC-min(AICtab$AIC))

AICtab[order(AICtab$DeltaAIC,decreasing = F),]
```

NOTE: for AIC model comparison to make sense, the response variable must be exactly the same (note: you can't compare different transformations of the response variable) and the number of observations must be exactly the same. Careful with missing data here!

Overview: Mixed-effects models

Remember the assumption of independent observations? Every single one of the models we have considered so far makes that assumption. If that assumption is violated, we are committing pseudoreplication. If pseudoreplication is relatively minor we may be able to ignore it for modeling (but still reporting the potential issue). But in many cases the issue is too large to ignore.

Mixed models allow us to build more realistic models that incorporate some known potential sources of non-independence in our data.

The term **mixed-effects models** refers to the fact that these models have two kinds of predictor variables: *fixed effects* and *random effects*. The random effects are what allow us to incorporate potential inter-dependencies among our observations.

Is it a fixed effect or a random effect?

One point of confusion that quickly bubbles up around mixed-effects models is the question of whether you should include a predictor variable as a random or a fixed effect.

It might help to consider some typical random effects and some typical fixed effects:

Fixed effects Typical fixed effects include:

- Temperature (continuous fixed effect- e.g., linear regression)
- Treatment (categorical fixed effect- e.g., ANOVA)
- Interaction between temperature and treatment (interaction fixed effect)

Fixed effects are what we have been calling 'predictor variables' in this class. They usually represent the variables that we want to relate to our response variable. Ideally, our observations span the full range of our fixed effects (predictor variables) such that we can use our data to make inference about the relationship between our response and predictor variables across the range of values that we might encounter in our entire population of interest.

There is no assumption that our predictor variables are normally distributed- in fact, the only assumption we make about the predictor variables (usually) are that (1) they are measured with certainty and (2) we ideally want them to be distributed evenly across the range of values across which we want to make inference about our response process.

For example, if we want to make inference about the relationship between tree diameter and volume, we would like to take measurements of trees that vary from the smallest-diameter trees we wish to make inference about to the largest-diameter trees that we'd like to make inference about. To ensure that our sample includes the full range of diameters, we might specifically select trees that cover the entire range of interest. That is, we could **fix** the set of trees in our samples to include the full range of our predictor variable. There is no rule stating that we have to select our predictor variable levels from a random process. In fact, experimental design is all about pre-determining our predictor variable levels!

The term **fixed effect** comes from experimental design, where we literally arrange (fix) our observations into specific treatment and control groups. For example, we might take 100 otherwise interchangeable sapling trees and subject them randomly to different treatments of **fixed** levels of some factor that might influence growth (say nitrogen concentrations)- and then we can make inference about the effect of nitrogen across the range of concentrations that we determined.

SIDE NOTE: we can (and often do) make inferences about our response variable for levels of our predictor variables that are outside the range of values in our data set. This is called **extrapolation** and can be a dangerous practice because we lack empirical support for this type of inference, especially when the predictions are far outside the bounds of our data set.

Random effects Typical Random effects include:

- Block ID (blocks are a small subset of the units about which you want to make inference, multiple observations per block)
- Site ID (sites are a small subset of the units about which you want to make inference, multiple observations per site)
- Year (study years are a small subset of the years about which you want to make inference, multiple observations per year)
- Individual (individuals studied are randomly selected from the population of interest, each individual subjected to repeated measurements)

First of all, most random effects will be categorical.

Second, random effects are a random sample or otherwise a small subset of the units about which you want to make inference.

Third, random effects variables must have multiple (usually >3) observations per level. If you only have one observation per random effect level, it is not a random effect- it's just a replicate observation in an ordinary linear regression!

Mixed effects models in regression notation

Before we run examples in R, let's look at linear mixed-effects regression models in regression notation.

The equation should look familiar- the only difference is that there is more than one error term. Each random effect is now associated with a new error term…

Let's say we are fitting the following mixed-effects model:

Response variable: Tortoise clutch size
Predictor variables: fixed effects: Annual winter precipitation, Annual spring temperature
Predictor variables: random effects: Site, year

The simplest way to model random effects is to include them as **random intercepts**– that is, the intercept term changes randomly with each factor level.

In many cases, simply adding a random intercept term for each random effect is appropriate for accounting for sources of non-independence in your data set- but in many cases it is not sufficient. That is because the slope (the relationship between your response and predictor– the thing you are usually most interested in making inference about) can itself vary depending on your random effect levels. This is called a **random slopes** model.

Here is an equation to represent the simpler random-intercept analysis:

$Clutchsize = \beta_0 + \beta_1 \cdot precip_t + \beta_2 \cdot temp_t + \gamma_{site} + \gamma_{year} + \epsilon_{obs}$

Here we now have three sources of 'error': a random (normally distributed) term for each site (γ_{site}), a random (normally distributed) term for each year, and a normally distributed residual error term (ϵ_{obs}).

If we want to include random slope and random intercept terms (which is often the most appropriate model), the equation gets a bit more complicated:

$Clutchsize = \beta_0 + \beta_1 \cdot precip_t + \beta_2 \cdot temp_t + \gamma_{site} + \gamma_{year} + \gamma_{site,\beta_1} \cdot precip_t + \gamma_{site,\beta_2} \cdot temp_t + \epsilon_{obs}$

Here we have two addition "error" terms that allow the slope terms (beta1 and beta2) to vary randomly with each site and year.

Nested random effects

You will often read descriptions of mixed-effects models saying things like “Individual and Site were included as random effects, with Individual nested within Site”.

Let’s imagine we randomly selected 10 sites and within each site we capture a random sample of tortoises and we measure tortoise clutch size for 4 consecutive years for each tortoise.

In this case, we have two potential random effects: site (random subset of a much larger set of potential sites) and individual (random subset of a much larger set of individuals within each site). Individual #1 from Site #1 is obviously a different individual than Individual #1 from Site #2. So we can’t simply include a simple random effect term for “Individual #1”. Since there are many “Individual #1”s, each “Individual #1” must get its own random effect! In this case, Individual is nested within Site!

Let’s contrast this with a non-nested random effect- let’s say site and year.

In this case, Year “2014” in site #1 is the same as Year “2014” in site #2. Therefore the random effect associated with year “2014” does not differ depending on site. In this case, site and year are independent, non-nested random effects!

Assumptions of mixed-effects regression

The assumptions of mixed-effects regression are the same as in classical linear regression (for mixed-effects regression models) or generalized linear models (for generalized linear mixed models; GLMM). The only additional assumption is this:

- All random effects are normally distributed!

In some more complex models you might encounter models that assume random effects take other distributions- but this is still rare to see!

Example: mixed-effects regression in R

The ‘workhorse’ package in R for fitting mixed-effects regression models (and GLMM) is the ‘lme4’ package. However, there are some other packages you should be aware of that can make your life easier. One such package is “glmmTMB”- I have found this package has more flexibility and tends to have less trouble fitting complex mixed-effects models.

We will use this dataset as our example - this is a data set on carbon balance in the tundra. This example is taken from this website.

```r
#########
# TUNDRA EXAMPLE
#########

##### Read in the data

mc1 <- read.csv("tundra2.csv",sep=",",na.strings=c("-","NA"))

summary(mc1)
```

```
##        X               Year          Site                GS.NEE              n
##  Min.   : 1.00   Min.   :1966   Length:82          Min.   :-153.000   Min.   :1.000
##  1st Qu.:21.25   1st Qu.:1993   Class :character   1st Qu.: -55.350   1st Qu.:1.000
##  Median :41.50   Median :1998   Mode  :character   Median : -20.867   Median :1.000
##  Mean   :41.50   Mean   :1998                      Mean   : -10.012   Mean   :1.829
##  3rd Qu.:61.75   3rd Qu.:2005                      3rd Qu.:   3.275   3rd Qu.:2.750
##  Max.   :82.00   Max.   :2010                      Max.   : 390.000   Max.   :6.000
##      cYear
##  Min.   :-31.8698
##  1st Qu.: -4.8698
##  Median :  0.6302
##  Mean   : -0.1015
##  3rd Qu.:  7.1302
##  Max.   : 12.1302
```

```
table(mc1$Year)  # some years have many observations
```

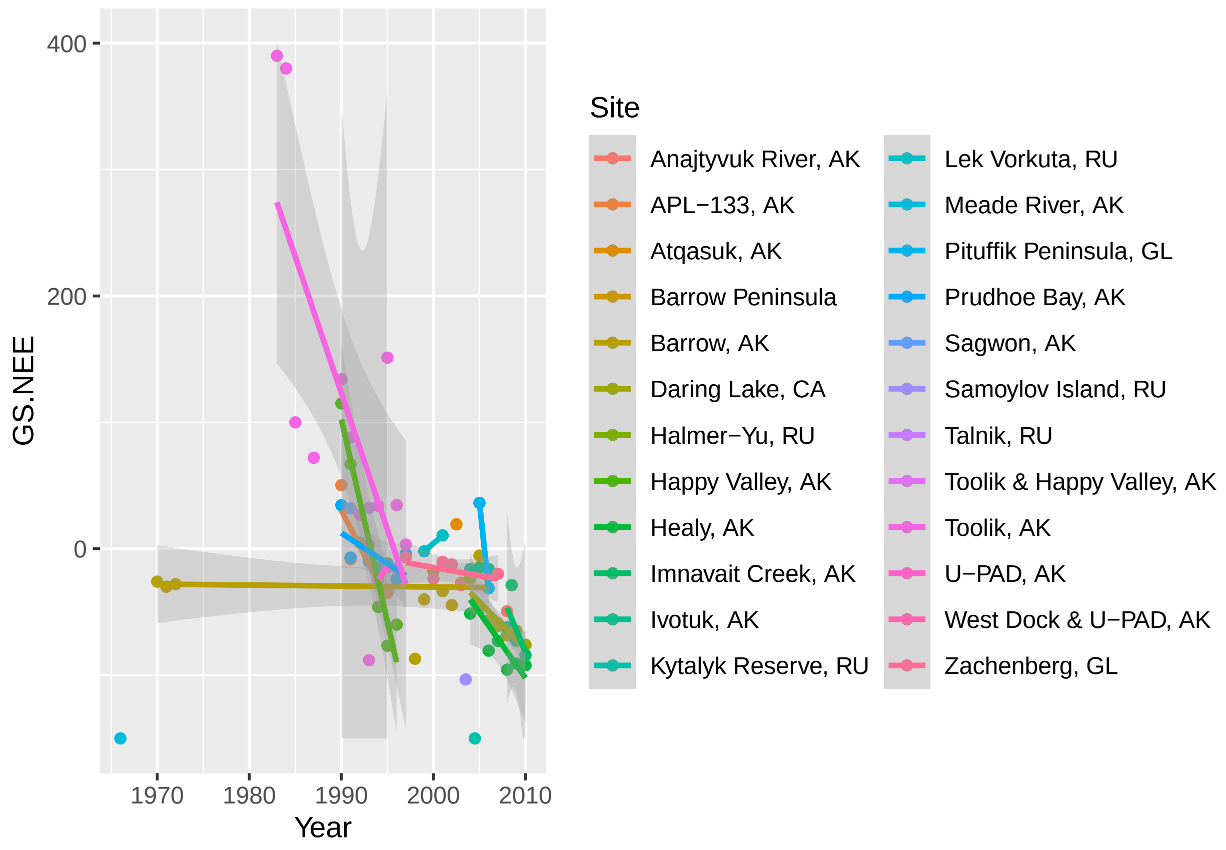
```
##
##   1966   1970   1971   1972   1983   1984   1985   1987   1990   1991   1992   1993   1994   1995
##      1      1      1      1      1      1      1      1      4      5      2      4      4      6
##   1996   1997   1998   1999   2000   2001   2002 2002.5   2003 2003.5   2004 2004.5   2005   2006
##      4      3      1      2      2      3      2      1      2      1      3      1      4      5
##   2007   2008 2008.5   2009   2010
##      4      4      1      3      3
```

```
table(mc1$Site)  # some sites have many observations
```

```
##
##       Anajtyvuk River, AK              APL-133, AK             Atqasuk, AK
##                         1                        3                       1
##          Barrow Peninsula               Barrow, AK         Daring Lake, CA
##                         1                       12                       7
##             Halmer-Yu, RU        Happy Valley, AK              Healy, AK
##                         1                        5                       7
##        Imnavait Creek, AK               Ivotuk, AK    Kytalyk Reserve, RU
##                         4                        2                       1
##            Lek Vorkuta, RU         Meade River, AK  Pituffik Peninsula, GL
##                         2                        1                       2
##            Prudhoe Bay, AK              Sagwon, AK     Samoylov Island, RU
##                         7                        1                       1
##                Talnik, RU Toolik & Happy Valley, AK              Toolik, AK
##                         1                        1                      12
##                U-PAD, AK      West Dock & U-PAD, AK          Zachenberg, GL
##                         2                        1                       6
```

```
library(ggplot2)

## visualize net ecosystem exchange by year- varying by site

ggplot(mc1,aes(x=Year,y=GS.NEE,colour=Site))+geom_point()+
    geom_smooth(method="lm",alpha=0.3)+
    scale_y_continuous(limits=c(-150,400),oob=scales::squish)
```

```
## `geom_smooth()` using formula 'y ~ x'
## Warning in qt((1 - level)/2, df): NaNs produced

## Warning in qt((1 - level)/2, df): NaNs produced

## Warning in qt((1 - level)/2, df): NaNs produced

## Warning in qt((1 - level)/2, df): NaNs produced

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -Inf

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -Inf

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -Inf

## Warning in max(ids, na.rm = TRUE): no non-missing arguments to max; returning -Inf
```



Okay now let's fit a mixed-effects regression model with Net Ecosystem Exchange (NEE) as the response variable and Year as the fixed-effect (covariate). For our random effect we will have site- and the intercept and trend (slope term) can vary with site. That is, we have random intercept terms and random slope terms for each site.

Fitting a mixed model in 'lme4' (using the 'lmer' function) looks a lot like fitting a linear model in 'lm'.

```r
library(lme4)
```

```
## Loading required package: Matrix
```

```
cmod_lmer <- lmer(GS.NEE ~ cYear + (1+cYear|Site),
                data=mc1, weights=n)
```

```
## boundary (singular) fit: see ?isSingular
```

A couple notes: first of all, the 'weights' term is there because some observations are aggregated- that is, there are multiple observations for a given site/year combination- those observations that aggregate 3 observations get triple the weight of a site/year combination with only one observation. (Technically, we are using an inverse-variance weighting scheme!)

Secondly, note that year is treated as a fixed effect in this model. That is- we are looking for a trend in our response variable over time!

Third, note the warning of a 'singular fit'. This is fairly common to see, and doesn't necessarily mean you can't use the model. But it means that lme4 struggled to fit the model and some of the parameters may have very wide confidence bounds!

Let's look at the model results using 'summary':

```
summary(cmod_lmer)
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: GS.NEE ~ cYear + (1 + cYear | Site)
##    Data: mc1
## Weights: n
##
## REML criterion at convergence: 874.2
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -2.90221 -0.35038 -0.07972  0.30155  2.93141
##
## Random effects:
##  Groups   Name        Variance Std.Dev. Corr
##  Site     (Intercept)  116.05  10.773
##           cYear         19.95   4.467   -1.00
##  Residual             3355.16  57.924
## Number of obs: 82, groups:  Site, 24
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  -16.296      7.338  -2.221
## cYear         -3.745      1.341  -2.792
##
## Correlation of Fixed Effects:
##       (Intr)
## cYear -0.417
## convergence code: 0
## boundary (singular) fit: see ?isSingular
```

The results from the 'summary' function should look somewhat familiar! But note that there are no p-values associated with your fixed effects (regression coefficients)- just t-statistics! This is because it is really hard to know what the degrees of freedom are for the test! Is it the number of observations? Is it the number of sites?

Also note that the summary now includes a summary of the random effects. One red flag here is that the random intercept term and random slope terms are perfectly (negagively) correlated (Corr=-1). This is the reason for the 'singular' warning in this case.

From the random effects summary we can see that site explains a relatively small percent of the total variance among observations.

To fit the model using glmmTMB we can use the following code:

```r
library(glmmTMB)
cmod_glmmTMB <- glmmTMB(GS.NEE ~ cYear + (1+cYear|Site),
                data=mc1,
                weights=n)
```

```
## Warning in fitTMB(TMBStruc): Model convergence problem; non-positive-definite Hessian matrix. See
## vignette('troubleshooting')
```

```r
summary(cmod_glmmTMB)
```

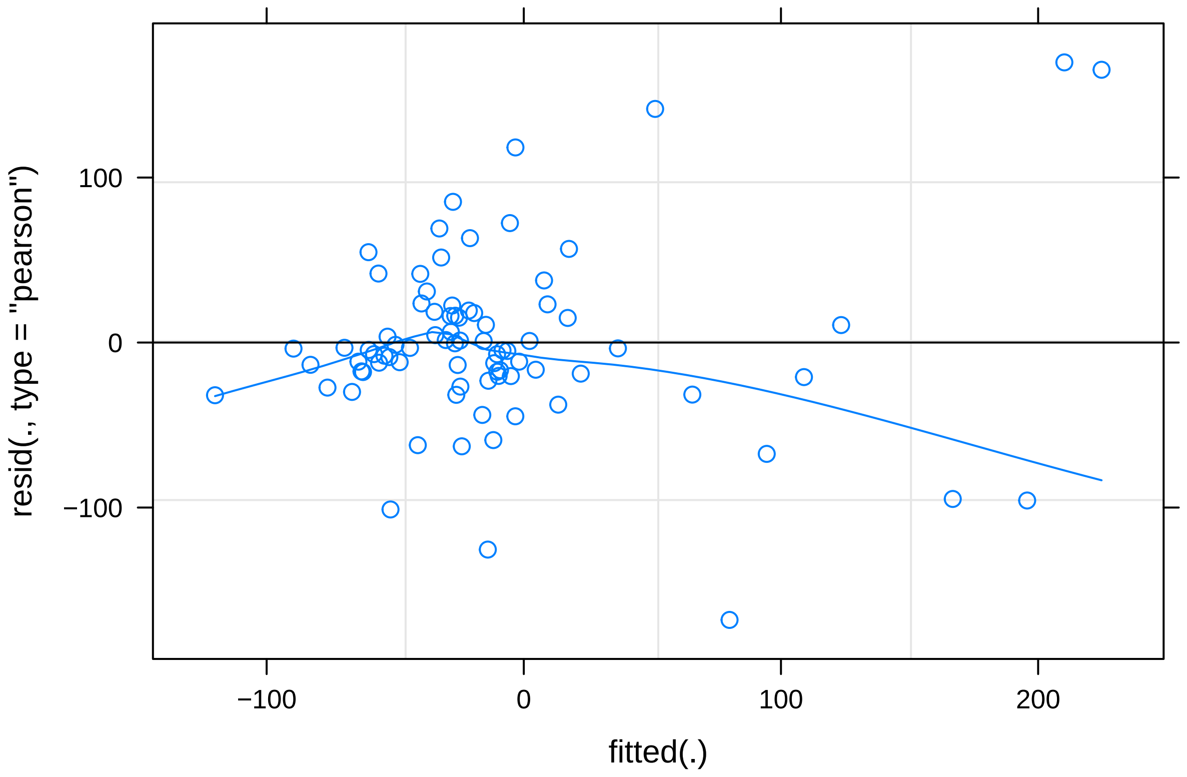
```
##  Family: gaussian  ( identity )
## Formula:          GS.NEE ~ cYear + (1 + cYear | Site)
## Data: mc1
## Weights: n
##
##      AIC      BIC   logLik deviance df.resid
##       NA       NA       NA       NA       76
##
## Random effects:
##
## Conditional model:
##  Groups   Name        Variance  Std.Dev. Corr
##  Site     (Intercept) 3.841e-02  0.196
##           cYear       3.148e+01  5.611   -0.79
##  Residual             1.543e+03 39.280
## Number of obs: 82, groups:  Site, 24
##
## Dispersion estimate for gaussian family (sigma^2): 1.54e+03
##
## Conditional model:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -22.347      5.579  -4.005 6.19e-05 ***
## cYear         -3.706      1.416  -2.617  0.00887 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The summary looks similar- but one difference you see is that there are p-values on the regression coefficients now.

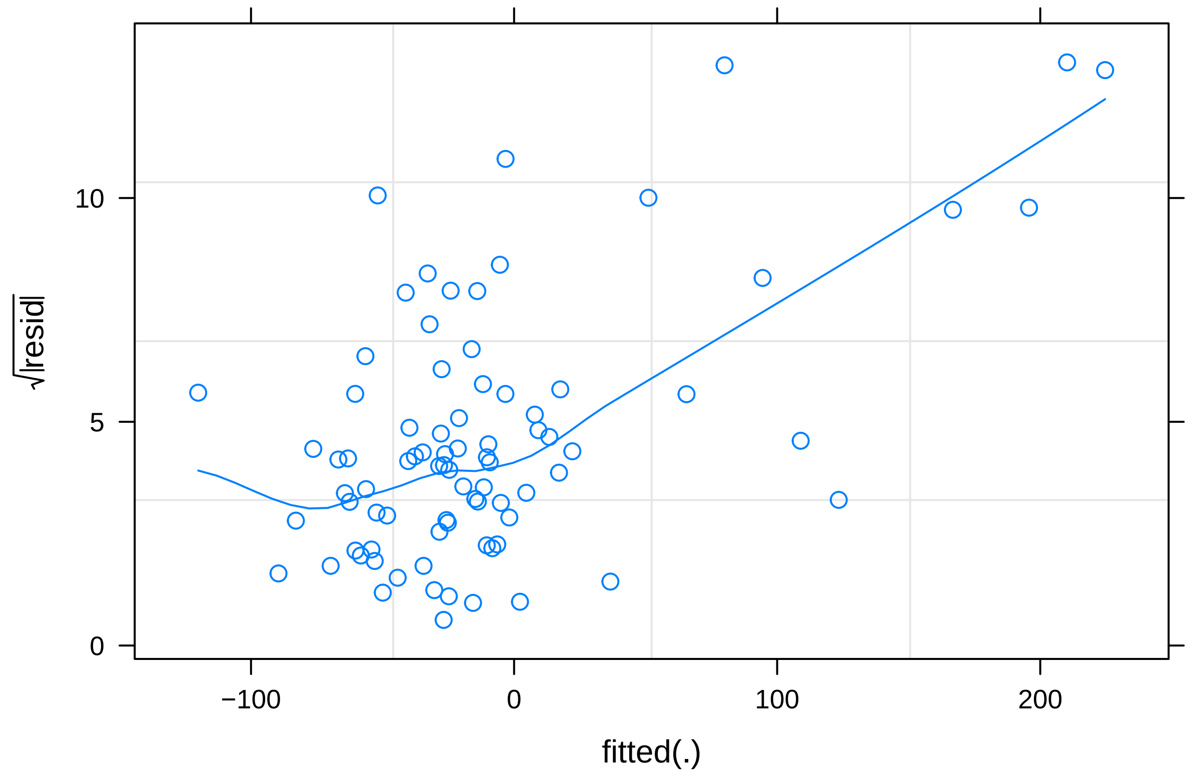
Let's perform some model diagnostics to test goodness-of-fit!

Since the standard linear regression assumptions apply, we could look at our standard diagnostic plots:

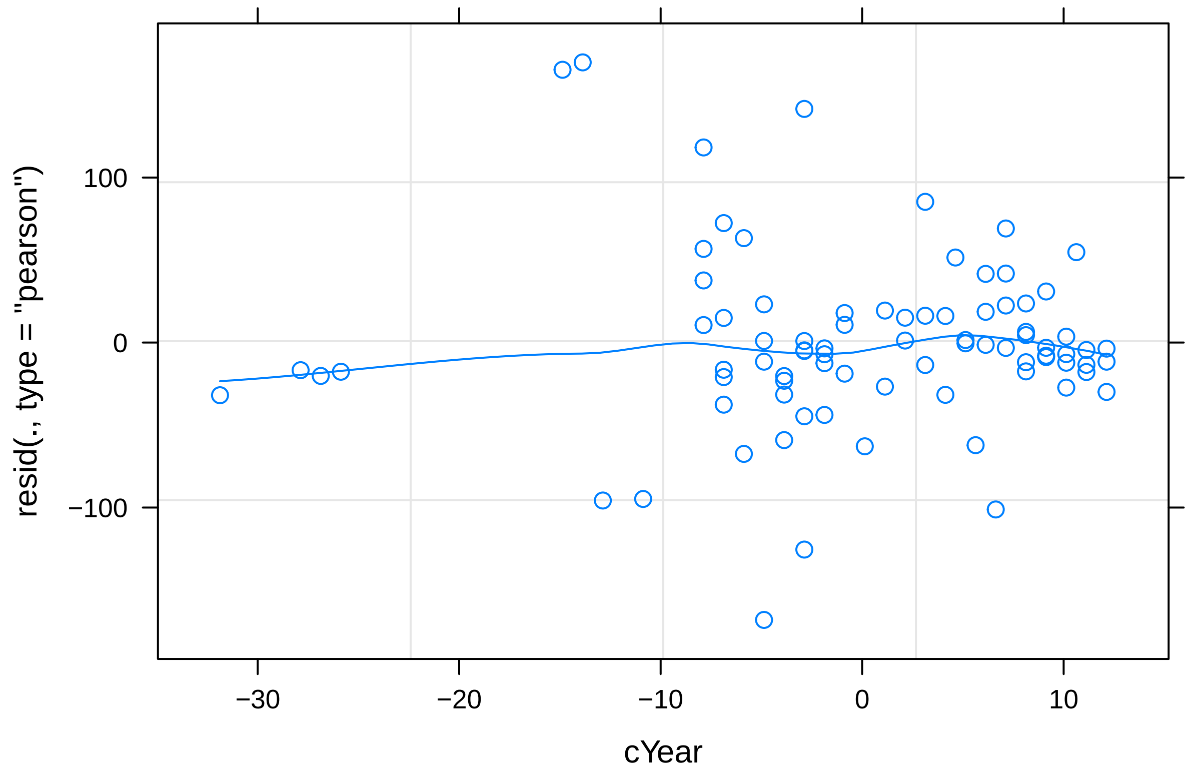
```r
plot(cmod_lmer,type=c("p","smooth"))
```



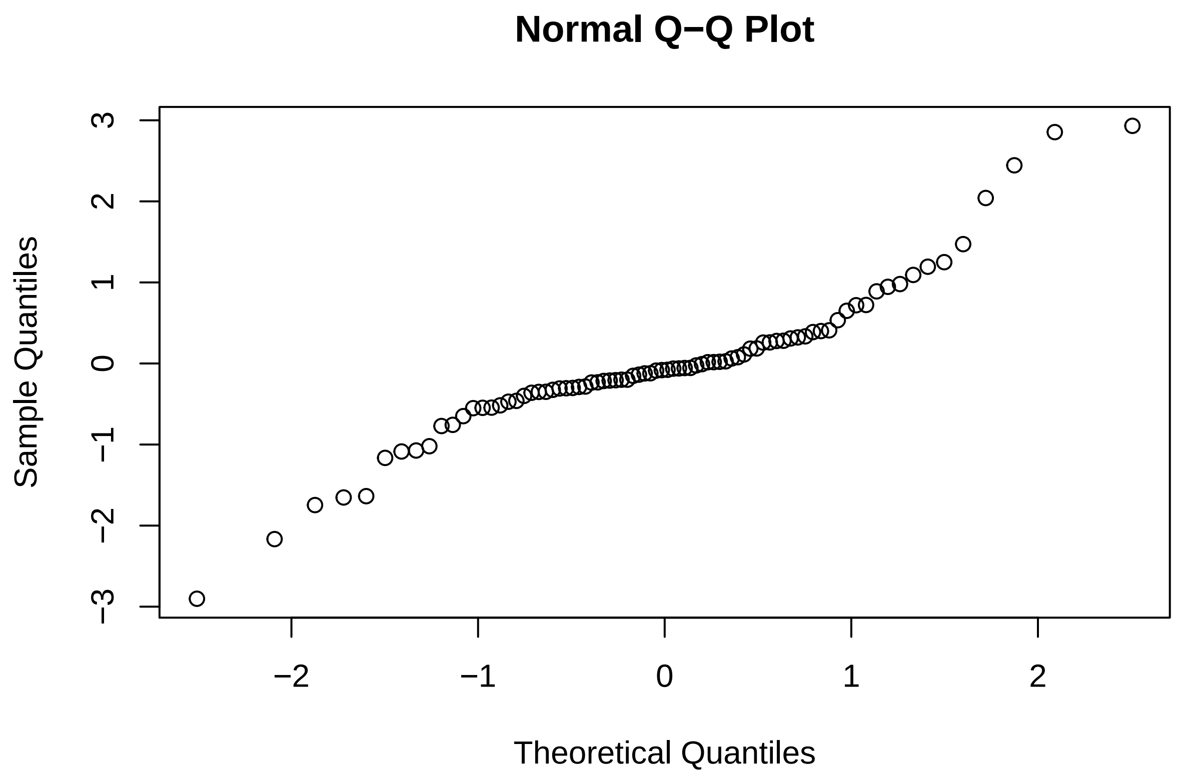
```
plot(cmod_lmer,sqrt(abs(resid(.)))~fitted(.),
                    type=c("p","smooth"),ylab=expression(sqrt(abs(resid))))
```



```
plot(cmod_lmer,resid(.,type="pearson")~cYear,
                   type=c("p","smooth"))
```



```
qqnorm(residuals(cmod_lmer,type="pearson",scaled=T))
```

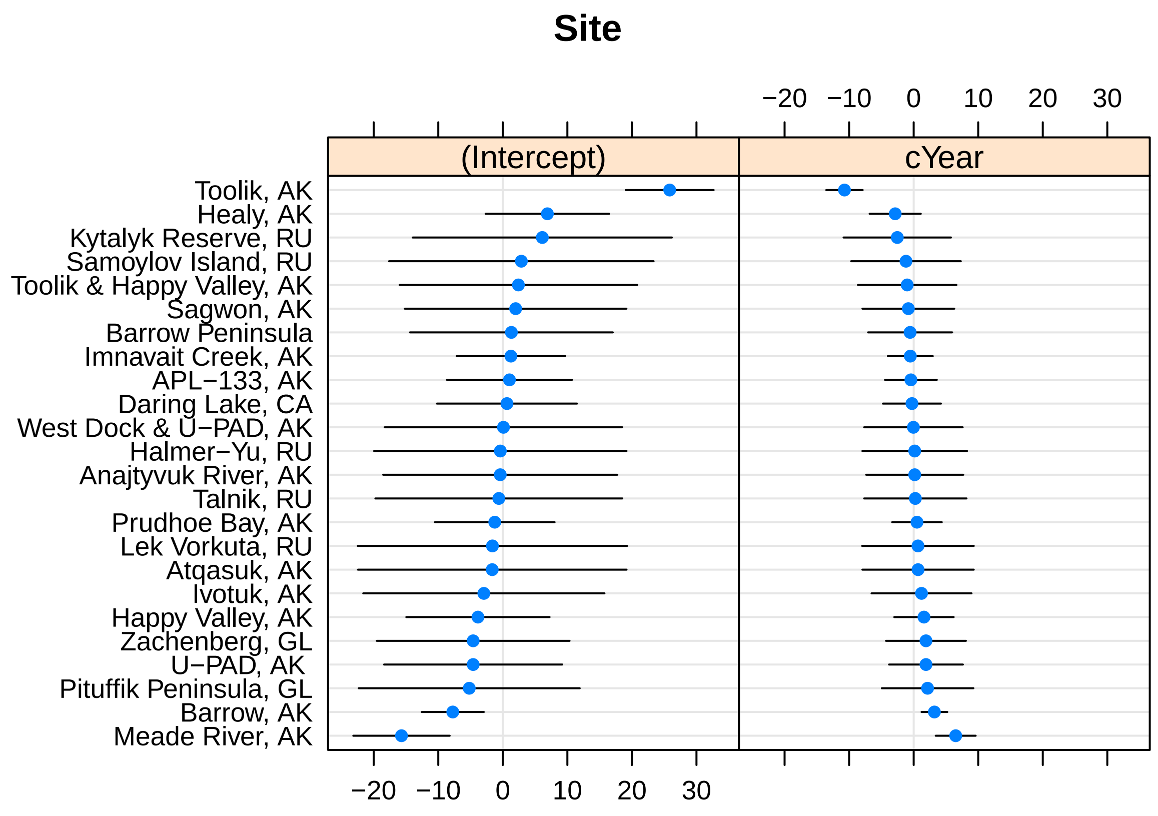


The diagnostics look, well, okay!. If you look into it more deeply, you see that that 'Toolik' site is the one causing most of the issues.

Okay let's visualize the random effects (both intercept and random slope terms):

```r
library(lattice)
dotplot(ranef(cmod_lmer,condVar=TRUE),
          lattice.options=list(layout=c(1,2)))
```

```
## $Site
```



If we want to test to see if year is explaining any of the variance, we can run an F-test (ANOVA):

```r
library(car)
Anova(cmod_lmer)
```

```
## Analysis of Deviance Table (Type II Wald chisquare tests)
##
## Response: GS.NEE
##        Chisq Df Pr(>Chisq)
## cYear 7.7956  1   0.005237 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Note that if we had a categorical variable and wanted to run pairwise comparisons, we could- using the 'emmeans' package (just like with standard ANOVA).

If we want confidence intervals on our fixed effects, we can use the 'confint' function, just like in ordinary linear regression. Here we use the Wald method (simplified version) and only extract confidence intervals for the fixed effects.

I had to use 'suppressWarnings' to avoid lots of warning messages here!

```r
confint(cmod_lmer,parm="beta_",method="Wald")
```

```
##                  2.5 %    97.5 %
## (Intercept) -30.677776 -1.914152
## cYear        -6.374309 -1.116174
```

–go to next lecture–