Zero-inflation, and how to deal with it in R and JAGS

(requires R-packages AER, coda, lme4, R2jags, DHARMa/devtools)

Carsten F. Dormann

07 December, 2016

Contents

1 Introduction: what is zero-inflation?

Put simply, if you have more 0s in your data than you would expect, you are facing zero-inflation. One common cause of zero-inflation is overdispersion (dealt with in a separate example). If there is zero-inflation even *after* properly modelling overdispersion (e.g. through a different family or observation-level random effects), then we are talking *real* zero-inflation, in the strict sense.

We imagine the excess 0s to be the result of observing the outcome of two co-occurring processes, each contributing some of the 0s. Let's take an ecological example.

Imagine we count the number of frogs in 100 ponds at different distances from the river Elbe (as in the paper by Dick et al., about to be published in J. Herpetology). We find that some ponds have no frog, others hundreds. A histogram reveals a high number of 0s (not shown), and an excess even after using the negative binomial. The authors hypothesis that two processes determine the number of frogs in a pond: (1) the distance to the river determines whether a pond is colonised; (2) if colonised, the local conditions (pond area, hydroperiod, fish) determine survival of spawning frogs, and hence finally the number of individuals.

2 Mixture of distributions

Thus, our data are a *mixture* of two distributions: one that describes *whether* a frog has reached the pond, and one that describes *how many* eggs hatched *if* a frog reached the pond. In perfect analogy, we also have to model the data as a mixture of two distributions, one for each of these two processes:

$$Y \sim \begin{cases} Pois(\lambda = \text{mean abundance}) & \text{, frog arrived} \\ 0 & \text{, frog did not arrive, with probability } \pi \end{cases}$$

A mixture distribution is defined (according to Wikipedia and my understanding) as "a collection of random variables derived as follows: first, a random variable is selected by chance from the collection according to given probabilities of selection, and then the value of the selected random variable is realized". This may sound unnecessarily complicated, but essentially we use one distribution to pick another one, from which we then draw the actual realised observation Y_i. In our pond example, we draw from the Bernoulli distribution whether a pond has been colonised, and then draw 0, if it hasn't, or from a Poisson if it has.

We can also write down the actual probabilities of observing x frogs in a pond, remembering that the Poisson distribution looks like this:

$$P(k = x) = \frac{\lambda^x e^{-lambda}}{x!}$$

Then our new mixture of the Bernoulli (for the colonisation process) and Poisson (for the population dynamics) is:

$$P(k = 0) = \pi(1 - \pi)e^{-\lambda}$$

$$P(k = x) = (1 - \pi)\frac{\lambda^x e^{-\lambda}}{x!}$$

So we see that our observed 0s have two sources: those that are 0 because of the Bernoulli distribution (the proportion π), plus those from the Poisson distribution for the ponds that *have* been colonised, but failed to generate surviving frogs (the proportion $(1 - \pi)$ times the proportion of $0s$ in the Poisson distribution with a given λ, which we can get from the Poisson distribution equation).

3 Modelling mixture distributions in JAGS

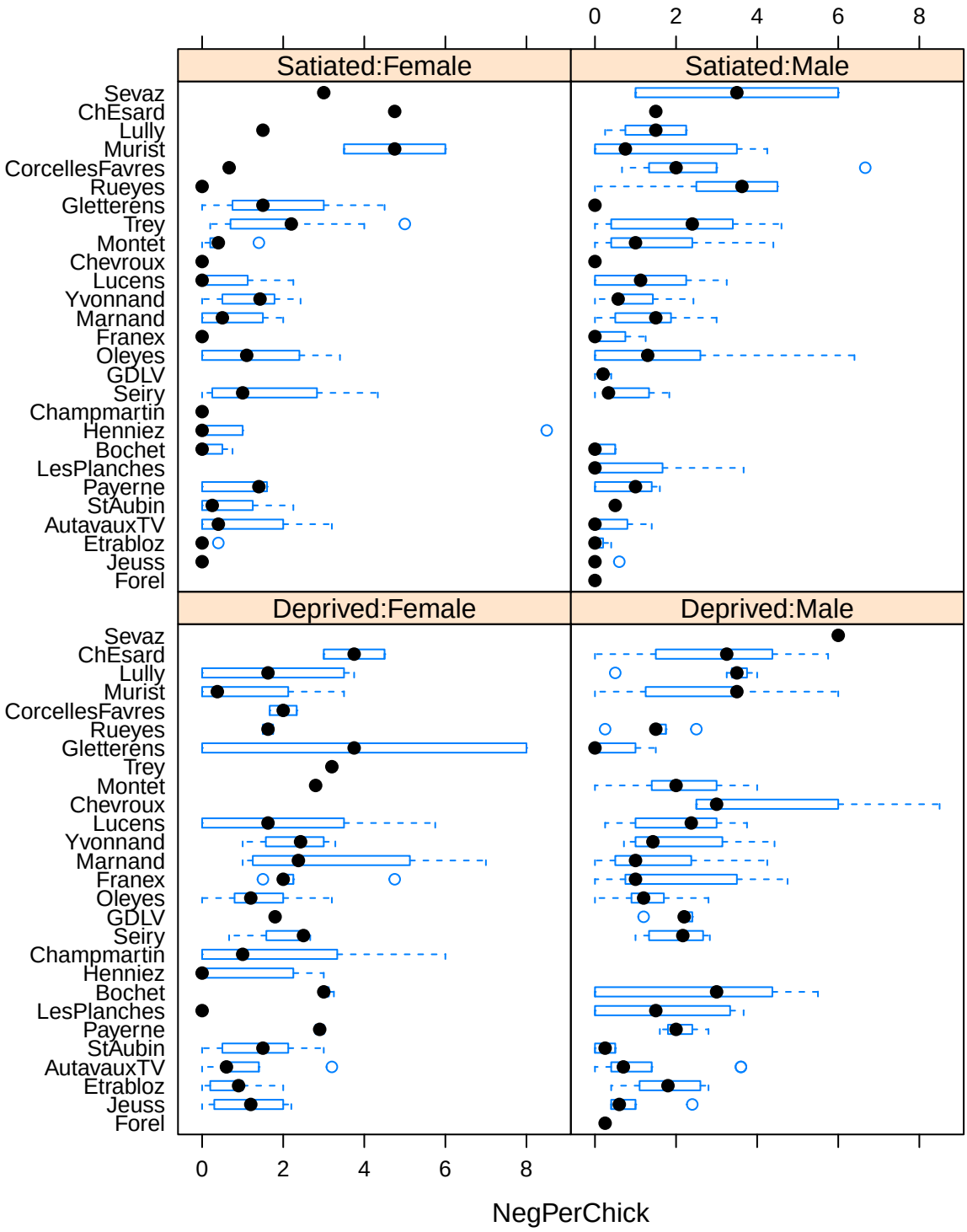
We use the owl begging data set of Roulin & Bersier (2007) from the **glmmADMB**-package. It describes the number of begging calls ("sibling negotiations") in a nest for females and males, being well-fed or food-deprived. The data look like this:

```
if ("glmmADMB" %in% rownames(installed.packages()) == FALSE){
    install.packages("R2admb")
    install.packages("glmmADMB", repos=c("http://glmmadmb.r-forge.r-project.org/repos",  getOption("repo
}

library(glmmADMB)
data(Owls)
summary(Owls)
```

```
          Nest      FoodTreatment  SexParent    ArrivalTime    SiblingNegotiation   BroodSize
 Oleyes     : 52   Deprived:320   Female:245   Min.   :21.71   Min.   : 0.00      Min.   :1.000
 Montet     : 41   Satiated:279   Male  :354   1st Qu.:23.11   1st Qu.: 0.00      1st Qu.:4.000
 Etrabloz   : 34                                Median :24.38   Median : 5.00      Median :4.000
 Yvonnand   : 34                                Mean   :24.76   Mean   : 6.72      Mean   :4.392
 Champmartin: 30                                3rd Qu.:26.25   3rd Qu.:11.00      3rd Qu.:5.000
 Lucens     : 29                                Max.   :29.25   Max.   :32.00      Max.   :7.000
 (Other)    :379
  NegPerChick     logBroodSize
 Min.   :0.000   Min.   :0.000
 1st Qu.:0.000   1st Qu.:1.386
 Median :1.200   Median :1.386
 Mean   :1.564   Mean   :1.439
```

```
 3rd Qu.:2.500   3rd Qu.:1.609
 Max.   :8.500   Max.   :1.946
```

```
library(lattice)
bwplot(reorder(Nest,NegPerChick)~NegPerChick|FoodTreatment:SexParent, data=Owls)
```



The model will become slightly complicated by the fact that "SiblingNegotiations" are measured *per nest*, rather than per chick. We hence would need to divide them by the number of chicks per nest, but that would

yield non-integer values! The solution is to use brood size as an offset (at the link-scale, i.e. using log(brood size) instead).

3.1 Data preparation for JAGS

Let's see how we can prepare the data for JAGS:

```
library(R2jags)
# prepare data as JAGS likes it:
attach(Owls)
head(Owls)
```

```
        Nest FoodTreatment SexParent ArrivalTime SiblingNegotiation BroodSize NegPerChick
1 AutavauxTV      Deprived      Male       22.25                 4         5         0.8
2 AutavauxTV      Satiated      Male       22.38                 0         5         0.0
3 AutavauxTV      Deprived      Male       22.53                 2         5         0.4
4 AutavauxTV      Deprived      Male       22.56                 2         5         0.4
5 AutavauxTV      Deprived      Male       22.61                 2         5         0.4
6 AutavauxTV      Deprived      Male       22.65                 2         5         0.4
  logBroodSize
1     1.609438
2     1.609438
3     1.609438
4     1.609438
5     1.609438
6     1.609438
```

Note that FoodTreatment and SexParent are factors. In a model, they need to be numerical values. The simplest way to convert them is like this:

```
head(as.numeric(FoodTreatment))
```

```
[1] 1 2 1 1 1 1
```

This leads to values of 1, 2, ... Since there are only two levels, I want them to be 0 and 1:

```
head(as.numeric(FoodTreatment) - 1)
```

```
[1] 0 1 0 0 0 0
```

```
# and
head(as.numeric(SexParent) - 1)
```

```
[1] 1 1 1 1 1 1
```

However, there is a more convenient function to do this for us, and include interactions, too!

```
Xterms <- model.matrix(~ FoodTreatment*SexParent, data=Owls)[,-1]
head(Xterms)
```

```
  FoodTreatmentSatiated SexParentMale FoodTreatmentSatiated:SexParentMale
1                     0            1                                    0
2                     1            1                                    1
3                     0            1                                    0
4                     0            1                                    0
5                     0            1                                    0
6                     0            1                                    0
```

Nice, ey? The "[,-1]" removes the intercept that would automatically be produced.

Which leads us to the JAGS-data:

```
OwlsData <- list(SibNeg = SiblingNegotiation, FoodTreatment=Xterms[,1], SexParent=Xterms[,2], FoodSex=X
detach(Owls)
```

Now comes the crucial bit!

3.2 The first (very primitive) zero-inflation model in JAGS

Note that it is costumary to model the proportion of 1s (now called $\psi = 1 - \pi$), rather than the proportion of 0s (π)!

```
ZIPR <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error
     z[i] ~ dbern(psi)

    log(lambda[i]) <-  lgBroodSize[i] + alpha
    # lgBroodSize is offset
    # alpha is overall intercept
  }

  # priors:
  alpha ~ dnorm(0, 0.01)      # overall model intercept
  psi ~ dunif(0, 1)           # proportion of non-zeros
}
```

Now we need to define which parameters to monitor, how to initialise them, and what the chain settings are:

```
parameters <- c("alpha", "psi") # which parameters are we interested in getting reported?

ni <- 1E3; nb <- ni/2 # number of iterations; number of burnins
nc <- 3; nt <- 5      # number of chains; thinning

inits <- function(){list(alpha=runif(1, 0, 2), psi = runif(1, 0, 1))}
```

And now we run it and look at the outcome.

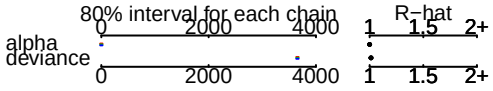
3.3 Run the model

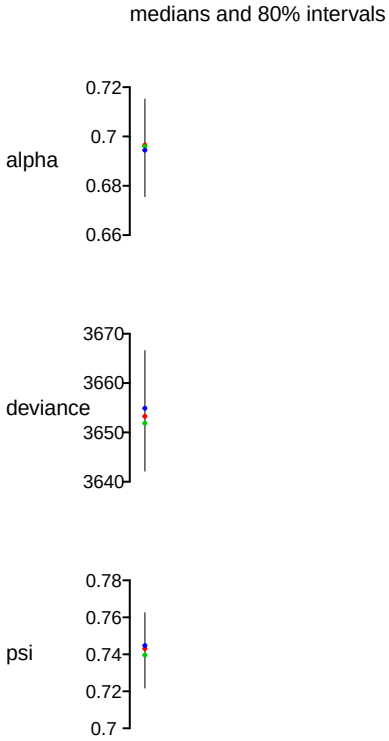
As usual, running the model takes a bit of time.

```
ZIPRjags <- jags(OwlsData, inits=inits, parameters, model.file = ZIPR, n.chains = nc, n.thin = nt, n.it
```

```
plot(ZIPRjags)
```

```
lders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf76cb9378.txt", fit using jags, 3 chains, each with 1000 iteration
```






```
ZIPRjags
```

```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf76cb
 3 chains, each with 1000 iterations (first 500 discarded), n.thin = 5
 n.sims = 300 iterations saved
          mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha       0.695   0.016    0.664    0.684    0.696    0.707    0.726 0.997   300
psi         0.742   0.017    0.709    0.730    0.743    0.754    0.771 1.018   220
deviance 3654.960  10.158 3641.827 3649.634 3653.851 3661.293 3679.038 1.013   300

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 51.9 and DIC = 3706.9
DIC is an estimate of expected predictive error (lower deviance is better).
```

So we get estimates for ψ (around 0.74) and α (around 0.69), indicating that there is quite a bit of zero-inflation! However, our model is currently really stupid and does not use any information on the predictors to explain begging. Maybe once we put these in we can explain more of the 0s by "Poisson-zeros", rather than "Bernoulli-zeros" (aka excess zeros).

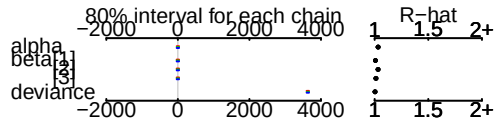
4 Model 2: add predictors for λ

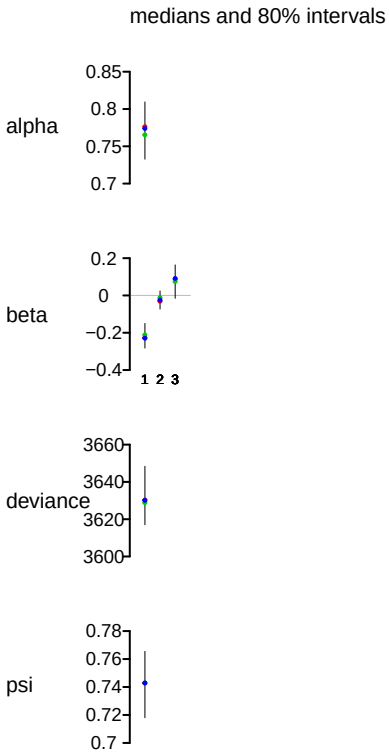
```
ZIPR2 <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error
     z[i] ~ dbern(psi)

    log(lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta[3
    # lgBroodSize is offset
    # alpha is overall intercept
  }

  # priors:
  alpha ~ dnorm(0, 0.01)     # overall model intercept
  for (m in 1:3){
      beta[m] ~ dnorm(0, 0.01)      # Linear effects
  }
  psi ~ dunif(0, 1)          # proportion of non-zeros
}
```

```
parameters <- c("alpha", "beta", "psi") # which parameters are we interested in getting reported?
ZIPR2jags <- jags(OwlsData, inits=inits, parameters, model.file = ZIPR2, n.chains = nc, n.thin = nt, n.i
```

```
plot(ZIPR2jags)
```

lders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf330df586.txt", fit using jags, 3 chains, each with 1000 iteration






```
ZIPR2jags
```

```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf330d
 3 chains, each with 1000 iterations (first 500 discarded), n.thin = 5
 n.sims = 300 iterations saved
```

```
          mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha       0.771   0.030    0.710    0.751    0.772    0.792    0.825 1.033    68
beta[1]    -0.220   0.051   -0.319   -0.257   -0.222   -0.188   -0.127 1.007   240
beta[2]    -0.024   0.038   -0.100   -0.049   -0.025    0.005    0.045 1.032    59
beta[3]     0.076   0.069   -0.072    0.037    0.082    0.120    0.200 1.012   130
psi         0.743   0.018    0.708    0.730    0.743    0.754    0.775 1.009   250
deviance 3631.462  11.647 3613.885 3622.779 3629.631 3638.248 3656.015 1.004   300

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 68.3 and DIC = 3699.7
DIC is an estimate of expected predictive error (lower deviance is better).
```

So while the model has improved (the DIC is lower by 10 units), the value for ψ hasn't changed much.

We can do better still.

Notice that so far the siblings within a nest are treated as independent, while they are in fact "nested" (pun intended). So we need to incorporate a random term for nest as well.

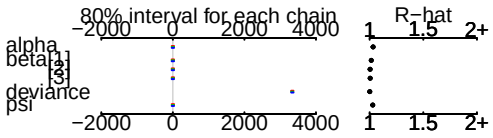
5 Model 3: add random effect for nest

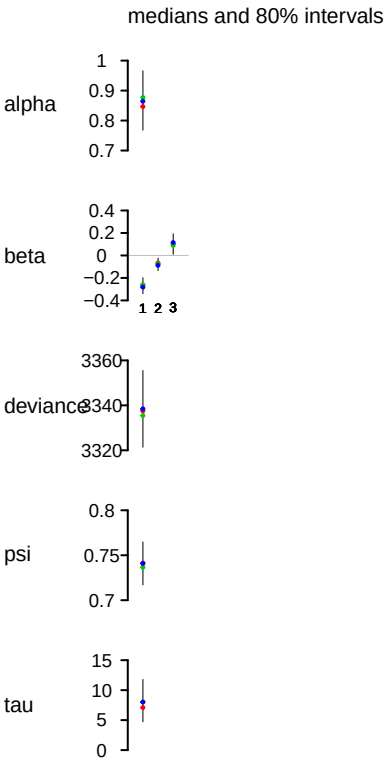
```
ZIPR3 <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error
     z[i] ~ dbern(psi)

    log(lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta[3
    # lgBroodSize is offset
    # alpha is overall intercept
    # "a" is random effect of nest; because alpha is overall intercept, a should be centred on 0.
  }

  # priors:
  alpha ~ dnorm(0, 0.01)     # overall model intercept
  for (m in 1:3){
      beta[m] ~ dnorm(0, 0.01)      # Linear effects
  }
  psi ~ dunif(0, 1)          # proportion of non-zeros
  for (j in 1:nnests){
    a[j] ~ dnorm(0, tau)     # random effect for each nest
  }
  tau ~ dgamma(0.001, 0.001) # prior for mixed effect variance

}
```

```
parameters <- c("alpha", "beta", "psi", "tau") # which parameters are we interested in getting reported
ZIPR3jags <- jags(OwlsData, inits=inits, parameters, model.file = ZIPR3, n.chains = nc, n.thin = nt, n.
```

```
plot(ZIPR3jags)
```

lders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf1972b705.txt", fit using jags, 3 chains, each with 1000 iteration






```
ZIPR3jags
```

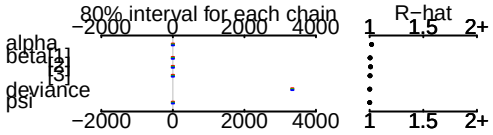
```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf1972
 3 chains, each with 1000 iterations (first 500 discarded), n.thin = 5
 n.sims = 300 iterations saved
          mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha       0.864   0.075    0.711    0.813    0.863    0.918    1.010 1.032    78
beta[1]    -0.269   0.055   -0.370   -0.305   -0.268   -0.234   -0.155 1.015   110
beta[2]    -0.080   0.043   -0.164   -0.109   -0.079   -0.053    0.002 1.005   240
beta[3]     0.102   0.071   -0.021    0.049    0.103    0.154    0.228 1.005   240
psi         0.741   0.019    0.699    0.730    0.740    0.754    0.774 1.029    88
tau         8.028   2.919    3.649    5.968    7.638    9.504   15.186 1.007   280
deviance 3337.625  13.781 3313.468 3328.301 3337.022 3344.906 3367.716 1.002   300

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 95.5 and DIC = 3433.1
DIC is an estimate of expected predictive error (lower deviance is better).
```

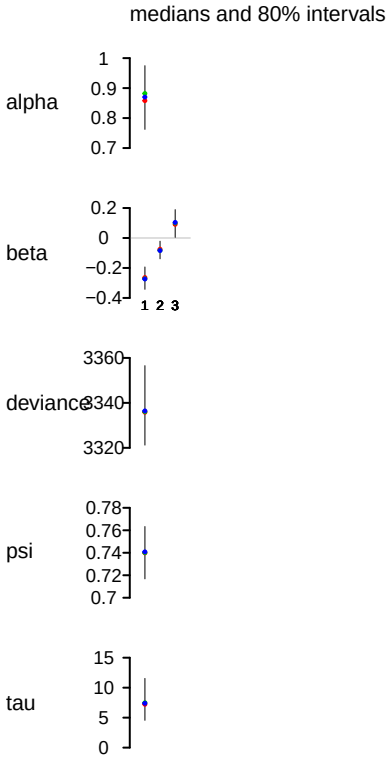
While we are now seeing a dramatic improvement in fit (DIC down by another 260 units or so!), we also notice that convergence has suffered, and the $\hat{R}$-values are higher than they should be for α. We re-adjust our settings and repeat the run (which will take around 10 times as long).

```
ni <- 1E4
parameters <- c("alpha", "beta", "psi", "tau") # which parameters are we interested in getting reported
ZIPR3jags <- jags(OwlsData, inits=inits, parameters, model.file = ZIPR3, n.chains = nc, n.thin = nt, n.
```

```
plot(ZIPR3jags)
```

ders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf219b3c62.txt", fit using jags, 3 chains, each with 10000 iteratior






```
ZIPR3jags
```

```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf219b3
 3 chains, each with 10000 iterations (first 500 discarded), n.thin = 5
 n.sims = 5700 iterations saved
          mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha       0.869   0.086    0.693    0.812    0.870    0.927    1.032 1.019   140
beta[1]    -0.269   0.059   -0.383   -0.310   -0.270   -0.229   -0.154 1.004   550
beta[2]    -0.080   0.046   -0.171   -0.111   -0.080   -0.049    0.012 1.008   260
beta[3]     0.097   0.072   -0.046    0.048    0.098    0.146    0.235 1.004   680
psi         0.740   0.018    0.704    0.728    0.740    0.752    0.775 1.001  5700
tau         7.759   2.797    3.372    5.774    7.377    9.303   14.367 1.001  5100
deviance 3337.721  13.820 3315.223 3327.676 3336.110 3345.888 3368.959 1.001  5700

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 95.5 and DIC = 3433.2
DIC is an estimate of expected predictive error (lower deviance is better).
```

Okay. Again, ψ is still high and seems to be a feature of the data, rather than due to our poor modelling of λ.

So, a first result interpretation is indicated: **What effects do you see, and what do they mean?**

6 Model 4: add effect of brood size on whether the chicks call at all

It could be that a clutch of chicks is more vocal when it is larger. A single chick may remain silent more often than it would when in a group of siblings (maybe I am extrapolating too much from football supporters on their way to the stadium). Statistically, we can make ψ a function of other predictors, too, in this case of brood size. Let's try.

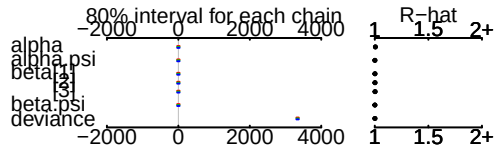
```
ZIPR4 <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error

    z[i] ~ dbern(psi[i])
    logit(psi[i]) <- alpha.psi + beta.psi*exp(lgBroodSize[i])

    log(lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta[
    # lgBroodSize is offset
    # alpha is overall intercept
    # "a" is random effect of nest; because alpha is overall intercept, a should be centred on 0.
  }

  # priors:
  alpha ~ dnorm(0, 0.01)     # overall model intercept
  for (m in 1:3){
      beta[m] ~ dnorm(0, 0.01)      # Linear effects
  }
  # remove this: psi ~ dunif(0, 1)          # proportion of non-zeros
  for (j in 1:nnests){
    a[j] ~ dnorm(0, tau)     # random effect for each nest
  }
  tau ~ dgamma(0.001, 0.001) # prior for mixed effect variance
  alpha.psi ~ dnorm(0, 0.01)
  beta.psi  ~ dnorm(0, 0.01)

}
```

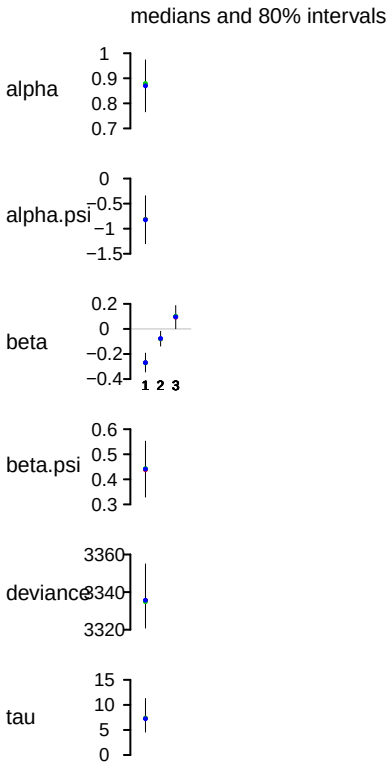
Since I am too lazy to re-code the inits-function, I simply set the inits-argument to auto-pilot.

```
parameters <- c("alpha", "beta", "tau", "alpha.psi", "beta.psi") # which parameters are we interested i
ZIPR4jags <- jags(OwlsData, inits=NULL, parameters, model.file = ZIPR4, n.chains = nc, n.thin = nt, n.i
```

```
plot(ZIPR4jags)
```

ders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf31a83d73.txt", fit using jags, 3 chains, each with 10000 iteratior






```
ZIPR4jags
```

```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf31a8
 3 chains, each with 10000 iterations (first 500 discarded), n.thin = 5
 n.sims = 5700 iterations saved
           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha        0.872   0.082    0.706    0.818    0.874    0.926    1.031 1.005   480
alpha.psi   -0.818   0.372   -1.564   -1.059   -0.818   -0.575   -0.091 1.001  5700
beta[1]     -0.269   0.059   -0.384   -0.308   -0.269   -0.230   -0.154 1.002  2500
beta[2]     -0.078   0.046   -0.167   -0.108   -0.078   -0.046    0.012 1.002  2100
beta[3]      0.096   0.073   -0.049    0.048    0.097    0.144    0.240 1.002  1200
beta.psi     0.441   0.087    0.271    0.383    0.440    0.497    0.614 1.001  5700
tau          7.683   2.735    3.526    5.700    7.311    9.247   14.171 1.001  5700
deviance  3336.881  13.374 3314.804 3327.225 3335.423 3344.837 3366.959 1.001  3600

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 89.4 and DIC = 3426.3
DIC is an estimate of expected predictive error (lower deviance is better).
```

And this model is better still (although “only” by 5 DIC-units).

So the one thing that we could still add is overdispersion as observation-level random effect. This is more to show that we can, and less because I think it is really necessary.

7 Model 5: add OLRE-overdispersion

We add OLRE in the form of an additive effect ξ at the level of the Poisson regression. All ξ are normally distributed with mean 0 (otherwise they'd compete with intercept α), and the precision of that normal distribution is taken to be γ-distributed (as is common for precision).

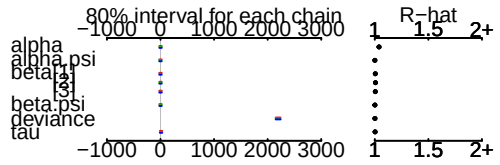
Note that we now have two competing random effects: one at the level of the nest (a) and one at the level of the individual observation (ξ).

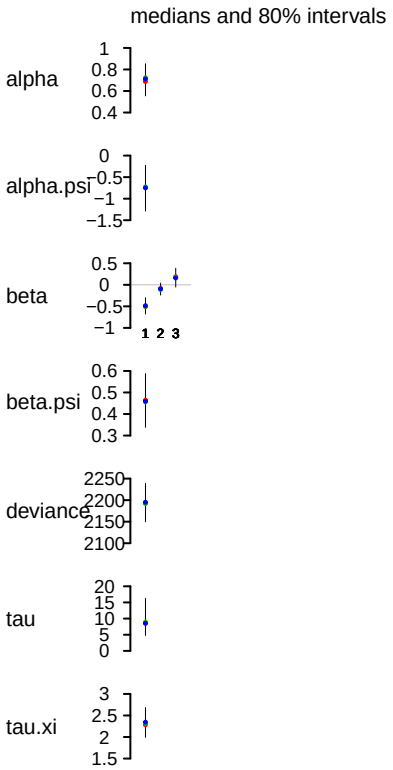
```
ZIPR5 <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error

    z[i] ~ dbern(psi[i])
    logit(psi[i]) <- alpha.psi + beta.psi*exp(lgBroodSize[i])

    log(lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta[
    # lgBroodSize is offset
    # alpha is overall intercept
    # "a" is random effect of nest; because alpha is overall intercept, a should be centred on 0.
  }

  # priors:
  alpha ~ dnorm(0, 0.01)     # overall model intercept
  for (m in 1:3){
      beta[m] ~ dnorm(0, 0.01)      # Linear effects
  }
  # remove this: psi ~ dunif(0, 1)          # proportion of non-zeros
  for (j in 1:nnests){
    a[j] ~ dnorm(0, tau)     # random effect for each nest
  }
  tau ~ dgamma(0.001, 0.001) # prior for mixed effect variance
  alpha.psi ~ dnorm(0, 0.01)
  beta.psi  ~ dnorm(0, 0.01)

  for (i in 1:N){
    xi[i] ~ dnorm(0, tau.xi) # on average, xi should be 0 otherwise it competes with the intercept alph
  }
  tau.xi ~ dgamma(0.001, 0.001) # prior for mixed effect variance

}
```

```
parameters <- c("alpha", "beta", "tau", "alpha.psi", "beta.psi", "tau.xi") # which parameters are we in
ZIPR5jags <- jags(OwlsData, inits=NULL, parameters, model.file = ZIPR5, n.chains = nc, n.thin = nt, n.i
```

```
plot(ZIPR5jags)
```

ders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf64d413a3.txt", fit using jags, 3 chains, each with 10000 iteratior



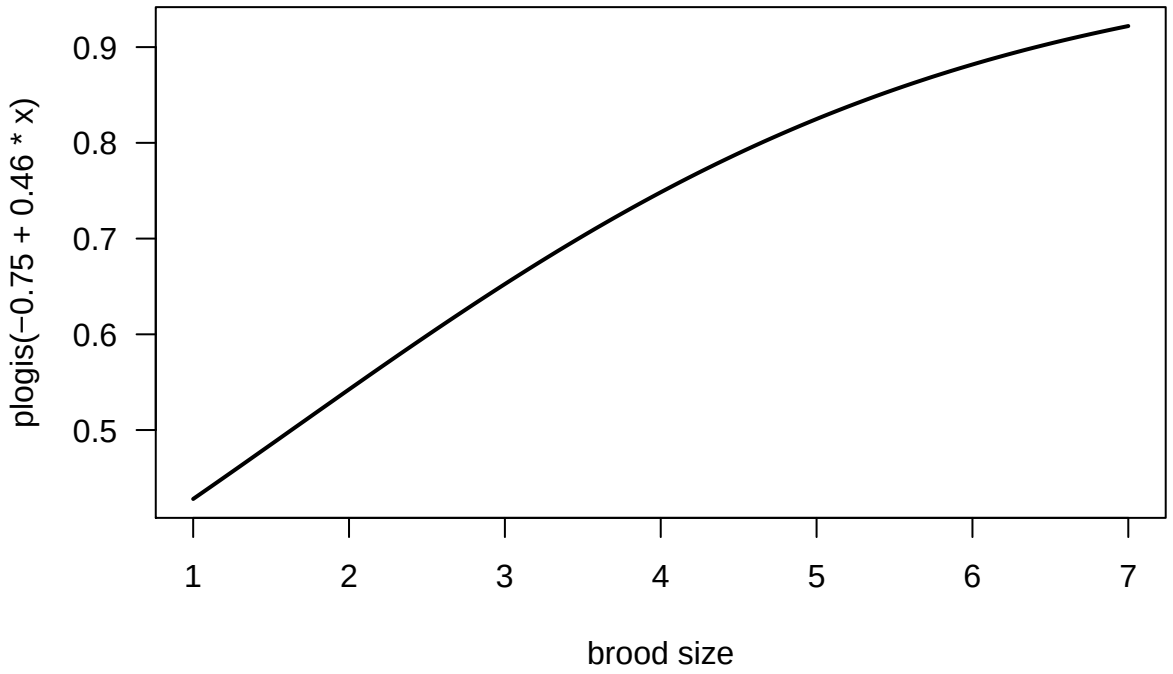



```
ZIPR5jags
```

```
Inference for Bugs model at "/var/folders/cc/3jfhfx190rb2ptxnqrqxj94m0000gp/T//RtmpeuCK00/model51bf64d4
 3 chains, each with 10000 iterations (first 500 discarded), n.thin = 5
 n.sims = 5700 iterations saved
           mu.vect sd.vect     2.5%      25%      50%      75%    97.5%  Rhat n.eff
alpha        0.707   0.120    0.474    0.629    0.709    0.786    0.936 1.040    85
alpha.psi   -0.746   0.414   -1.571   -1.023   -0.739   -0.469    0.070 1.001  5700
beta[1]     -0.494   0.147   -0.772   -0.597   -0.498   -0.391   -0.208 1.005   510
beta[2]     -0.096   0.108   -0.302   -0.169   -0.097   -0.022    0.114 1.006   370
beta[3]      0.170   0.175   -0.183    0.052    0.174    0.289    0.503 1.005   520
beta.psi     0.461   0.098    0.271    0.394    0.460    0.525    0.657 1.001  5600
tau          9.940   5.798    3.519    6.386    8.692   12.190   23.060 1.003  1000
tau.xi       2.330   0.269    1.844    2.137    2.318    2.510    2.890 1.007   330
deviance  2194.515  35.049 2128.138 2171.191 2193.477 2218.101 2265.241 1.001  2800

For each parameter, n.eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor (at convergence, Rhat=1).

DIC info (using the rule, pD = var(deviance)/2)
pD = 614.0 and DIC = 2808.5
DIC is an estimate of expected predictive error (lower deviance is better).
```

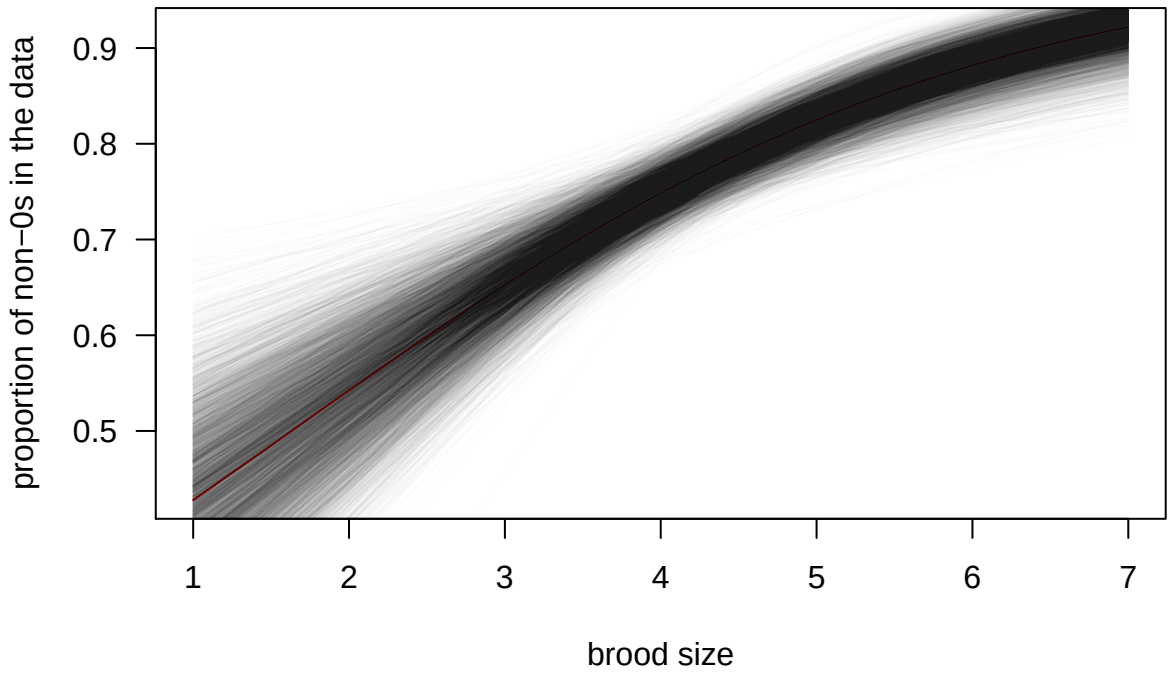
Oh, that's somewhat of a surprise! The DIC dropped precipitously to just under 2800, i.e. by over 500 units. So I guess the data were substantially overdispersed, not only zero-inflated. Zero-inflation is still prevalent, and we should plot the relationship for ψ to see which values it takes.

```r
curve(plogis(-0.75 + 0.46*x), from=min(Owls$BroodSize), to=max(Owls$BroodSize), lwd=2, las=1, xlab="bro
```



Obviously we should do this with all samples, not just the mean estimates:

```r
# str(ZIPR5jags$BUGSoutput$sims.list)
curve(plogis(-0.75 + 0.46*x), from=min(Owls$BroodSize), to=max(Owls$BroodSize), lwd=1, col="red", las=1
for (i in 1:length(ZIPR5jags$BUGSoutput$sims.list$alpha.psi)){
  thisA <- ZIPR5jags$BUGSoutput$sims.list$alpha.psi[i]
  thisB <- ZIPR5jags$BUGSoutput$sims.list$beta.psi[i]
  curve(plogis(thisA + thisB*x), from=min(Owls$BroodSize), to=max(Owls$BroodSize), add=T, col=rgb(0.1, 
}
```



So the proportion of 0s is between 0.4 and 0.9, and the trend is positive (i.e. more non-0s when there are more siblings). So owl chicks are very much like football supporters, it seems.

8 Model diagnostics: simulation from model (5b)

So far we have not spend any time on evaluating whether any of the models was really "good". That is not a trivial task, and we need to consider a new idea before being able to do so.

We call a model "good", if it is able to invent data that look like those we used to fit it to.

That is (I hope) logical. If a model fits poorly, then simulating (= inventing) data based on this model will lead to data that may look very different from the original data. A near-perfect fit, in contrast, will yield simulated data very similar to those observed.

In the following code, we simulate data from the model, not once, but several thousand times. We can then see, how our observed data are positioned within the several thousand simulations (e.g. on which quantile they lie; this is called the "Bayesian p-value").

To simulate, it is easiest to use JAGS itself, rather than its output. To do so, we "invent" our response again, within the model, with a new name (in this case S.SibNeg, with S. standing for "simulated"). Unsurprisingly, almost doubling the number of parameters will also lead to substantially longer computation time!

```
ZIPR5s <- function() {
  for(i in 1:N){ # loop through all data points
    SibNeg[i] ~ dpois(mu[i])
    mu[i] <- lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error

    z[i] ~ dbern(psi[i])
    logit(psi[i]) <- alpha.psi + beta.psi*exp(lgBroodSize[i])

    log(lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta[3
    # lgBroodSize is offset
    # alpha is overall intercept
    # "a" is random effect of nest; because alpha is overall intercept, a should be centred on 0.
  }

  # priors:
  alpha ~ dnorm(0, 0.01)     # overall model intercept
  for (m in 1:3){
      beta[m] ~ dnorm(0, 0.01)      # Linear effects
  }
  # remove this: psi ~ dunif(0, 1)          # proportion of non-zeros
  for (j in 1:nnests){
    a[j] ~ dnorm(0, tau)     # random effect for each nest
  }
  tau ~ dgamma(0.001, 0.001) # prior for mixed effect variance
  alpha.psi ~ dnorm(0, 0.01)
  beta.psi  ~ dnorm(0, 0.01)

  for (i in 1:N){
    xi[i] ~ dnorm(0, tau.xi) # on average, xi should be 0 otherwise it competes with the intercept alph
  }
  tau.xi ~ dgamma(0.001, 0.001) # prior for mixed effect variance

  # # # # # # # # # # # # # # #
```

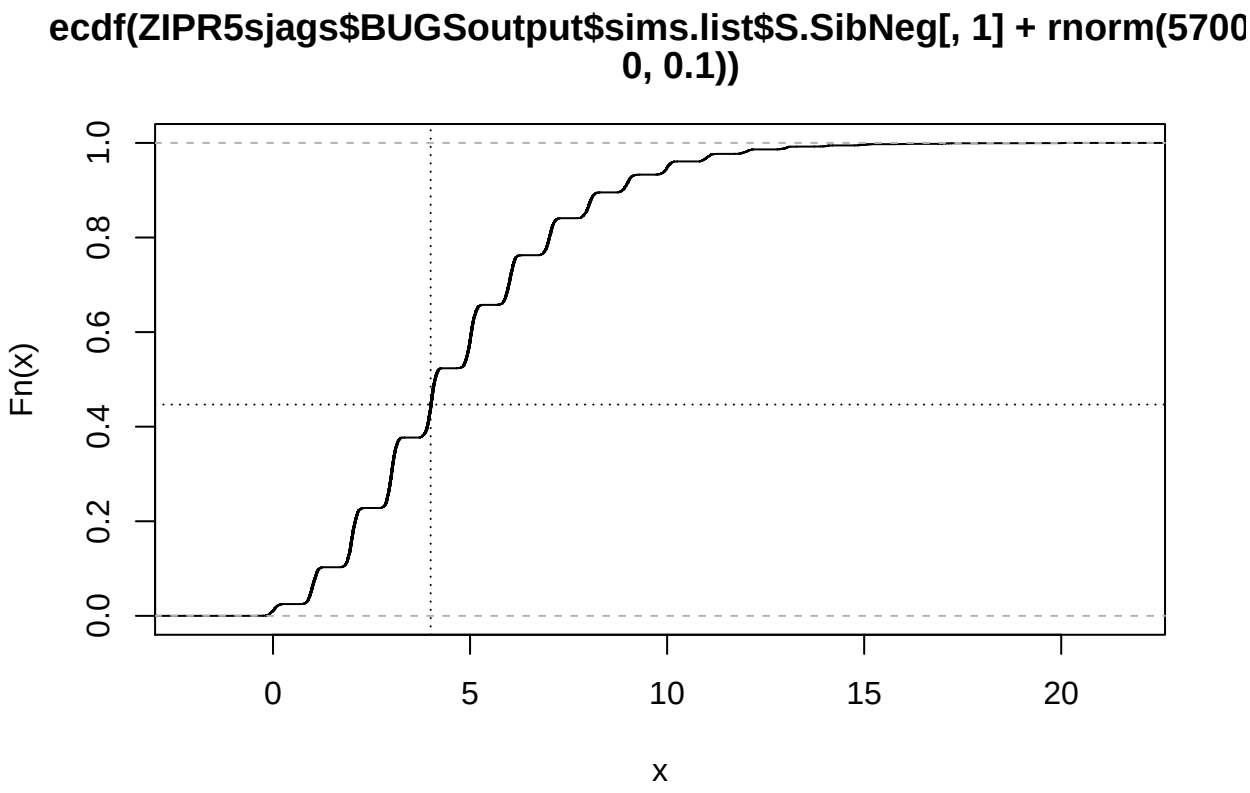
```
  # simulate data here:
  # replace all latent variables L with an S.L (mu, lambda, psi, z), as well as the response:
  for (i in 1:N){ # loop through all data points
    S.SibNeg[i] ~ dpois(S.mu[i])
    S.mu[i] <- S.lambda[i]*z[i] + 0.00001 ## hack required for Rjags -- otherwise 'incompatible'-error

    S.z[i] ~ dbern(S.psi[i])
    logit(S.psi[i]) <- alpha.psi + beta.psi*exp(lgBroodSize[i])

    log(S.lambda[i]) <-  lgBroodSize[i] + alpha + beta[1]*FoodTreatment[i] + beta[2]*SexParent[i] + beta
  }

}
```

```
parameters <- c("alpha", "beta", "tau", "alpha.psi", "beta.psi", "tau.xi", "S.SibNeg") # which paramete
ZIPR5sjags <- jags(OwlsData, inits=NULL, parameters, model.file = ZIPR5s, n.chains = nc, n.thin = nt, n
```

We rather not look at the plot, where there would now be 599 values for S.SibNeg in addition to all the model parameters we have looked at before. Same for the summary of the model, which also should be the same as in model 5.

Instead, we extract the simulated data for each original data point. First, as an example, for the first data point only:

```
plot(ecdf(ZIPR5sjags$BUGSoutput$sims.list$S.SibNeg[,1]+rnorm(5700, 0, 0.1)), verticals=T)
abline(v=Owls$SiblingNegotiation[1], lty=3)
abline(h=ecdf(ZIPR5sjags$BUGSoutput$sims.list$S.SibNeg[,1]+rnorm(5700, 0, 0.1))(Owls$SiblingNegotiation
```

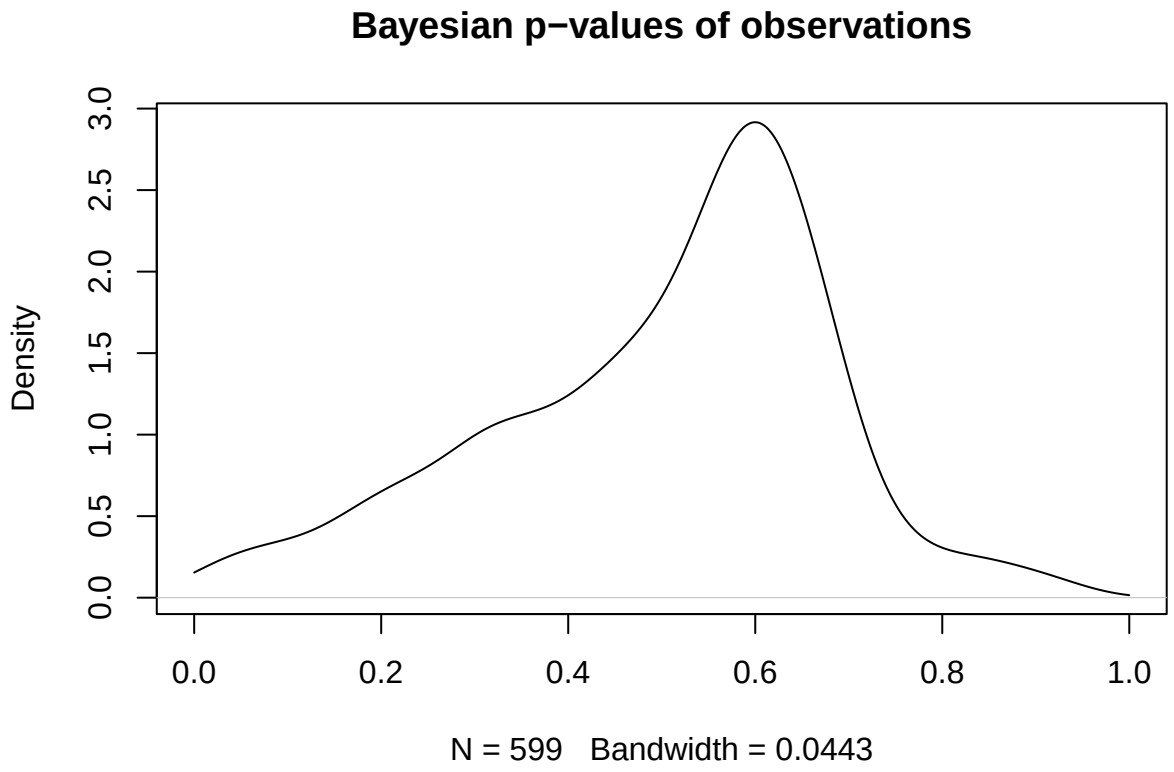


Notice that adding some noise smoothes out the ECDF-curve, as has been recommended (somewhere).

So we see that the first data point (4 calls) lies roughly at the 0.5 quantile of the simulated data. Let's do

this computation for all observations (and simulations), and plot these quantiles:

```
qq <- numeric(599)
for (i in 1:599){
    qq[i] <- ecdf(ZIPR5sjags$BUGSoutput$sims.list$S.SibNeg[,i]+rnorm(5700, 0, 0.1))(Owls$SiblingNegotia
}
plot(density(qq, from=0, to=1), main="Bayesian p-values of observations")
```

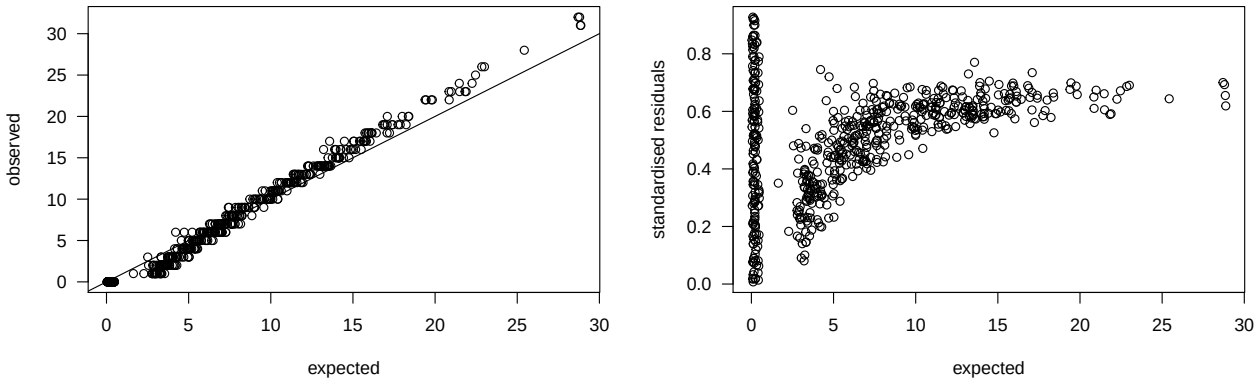



```
summary(qq)
```

```
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
0.007544 0.382100 0.545800 0.498700 0.619100 0.927400
```

Ideally, we want our observations to be more or less evenly distributed across the range from 0 to 1. That is clearly not the case. What we see is that most observations are around quantiles 0.45 and 0.6, in a very non-uniform fashion.

```
par(mfrow=c(1,2))
plot(apply(ZIPR5sjags$BUGSoutput$sims.list$S.SibNeg, 2, mean), Owls$SiblingNegotiation, las=1, xlab="ex
abline(0,1)
plot(qq ~ apply(ZIPR5sjags$BUGSoutput$sims.list$S.SibNeg, 2, mean), las=1, xlab="expected", ylab="standa
```



The quantile-quantile-plot looks almost fine, with some overestimation at high values. The standardised residuals (which are actually the quantiles) show should no pattern with expectation, however!

That means: we're not done yet. More model tuning is required to improve the model, so that the distribution of simulations is more in line with the distribution of observations. (Small note: there is an integer problem here, so some tricks such as adding noise was indicated. See DHARMa and its vignette for some comments on that.)