Lecture 23. Generalized linear mixed models and generalized additive mixed models

We've spent a lot of time on mixed models, assuming a normally distributed response variable. These are typically called 'linear mixed models' (LMMs). I'm going to spend a short amount of time on mixed models with a non-normal response, which area called 'generalized linear mixed models' (GLMMs). This is somewhat counterintuitive, because GLMMs are in some ways 'harder' than LMMs. However, the good news is that nearly everything we covered for LMMs applies directly to GLMMs, its just that GLMMs have a link function and a non-normal response. Because you're already quite familiar with these things from GLMs, combining the GLM stuff with the LMM stuff is mostly a matter of extrapolation.

Let's do a simple example using survey data of roundworm infection in red deer in Spain. The deer are coded as being either infected or not, so this is binary data, and we want to use a binomial distribution. We want to see how sex and size affect the infection status, which is simple enough, but the data were collected on a number of farms, and so we need to account for this grouping structure. Since this is binary data, rather than make exploratory plots I'll just make some tables:

```
table(deer$Farm)

## 
##   AL   AU   BA   BE   CB  CRC   HB   LN  MAN   MB   MO   NC   NV   PN   QM 
##   15   32   50   13   85    1   17   33   27   34  209   27   20   37   60 
##   RF   RN   RO  SAU   SE   TI   TN VISO   VY 
##   20   23   30    3   26   19   25   13    7
```

Some farms have many samples, and some have few. Definitely something better handled with random effects.

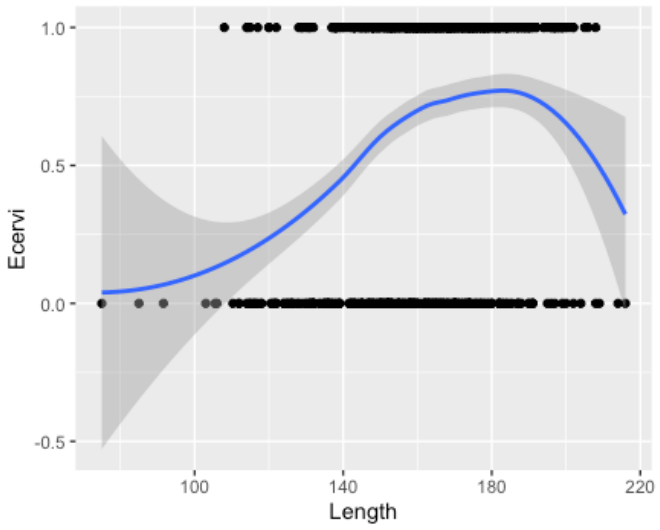
```
tapply(deer$Ecervi, deer$Farm, mean)

##     AL     AU     BA     BE     CB    CRC     HB     LN    MAN     MB 
## 0.4000 0.8750 0.8200 1.0000 0.6941 0.0000 0.1176 0.9091 0.4074 0.8824 
##     MO     NC     NV     PN     QM     RF     RN     RO    SAU     SE 
## 0.3923 0.4815 0.5500 0.9189 0.8000 0.9000 0.3043 0.9333 0.0000 0.7692 
##     TI     TN   VISO     VY 
## 0.9474 0.7200 0.8462 0.8571
```

Farms appear to vary a lot in the proportion of infections, so it's important to account for that.

```
tapply(deer$Ecervi, deer$Sex, mean)

## female   male 
## 0.6942 0.5899
```

Females are on average more likely to be infected than males, a difference of about 10%.

```
ggplot(deer, aes(Length, Ecervi)) + geom_point() + geom_smooth()
```



A loess smoother shows that the probability of infection tends to increase as deer size increases. Now let's make a model that incorporates these parts. I'm going to make one modification to the data: center the predictor. In this case, centering the predictor makes the interpretation of the coefficients easier. Without centering the intercept will be the probability of infection when Length = 0. Length = 0 does not make sense for a deer, so we will center around the mean Length:

```
deer$Length = deer$Length - mean(deer$Length)
```

Now the intercept will be the probability of infection for the average-sized deer.

```
mod = glmer(Ecervi ~ Sex*Length + (1|Farm), data = deer, family =
binomial)
```

The specification is the same as we used for LMMs, except now we use the function 'glmer', and we set family = binomial. Just like a GLM, this is essentially a linear model that is put inside a link function (the logit link for binomial data). That means that *the random effects are assumed to be normally distributed on the scale of the link function*, not on the scale of the raw data.

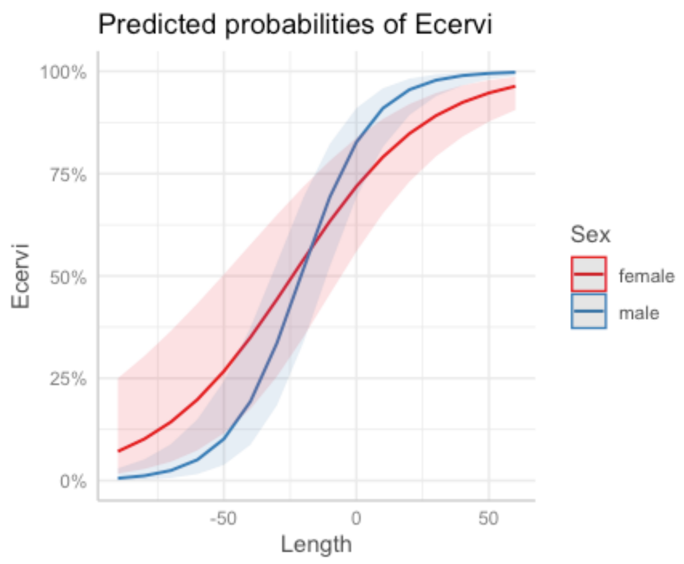
```
summary(mod)

## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: Ecervi ~ Sex * Length + (1 | Farm)
##    Data: deer
##
```

```
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Farm   (Intercept) 2.39     1.55
## Number of obs: 826, groups:  Farm, 24
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.93897    0.35600    2.64   0.0084 **
## Sexmale         0.62449    0.22294    2.80   0.0051 **
## Length          0.03896    0.00692    5.63  1.8e-08 ***
## Sexmale:Length  0.03586    0.01141    3.14   0.0017 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

We've got one random effect, and it has a Std. Dev. of 1.55. The fixed effect intercept is 0.94; in this model, that represents the probability of infection for an averaged-sized female deer, on the logit scale. On the original scale, it's exp(0.94)/(1 + exp(0.94)) = 0.72. Clearly most deer have roundworms. The random effect has a fairly large standard deviation relative to 0.94, so that means that probability of infection varies a lot by Farm. It also looks like the Sex:Length interaction is important, but in general the reported p-values are not that trustworthy. They are z-statistics, which means they assume a very large sample size. In this case we do have a large sample size (826 deer on 24 farms), but in general these aren't the preferred p-values to use.

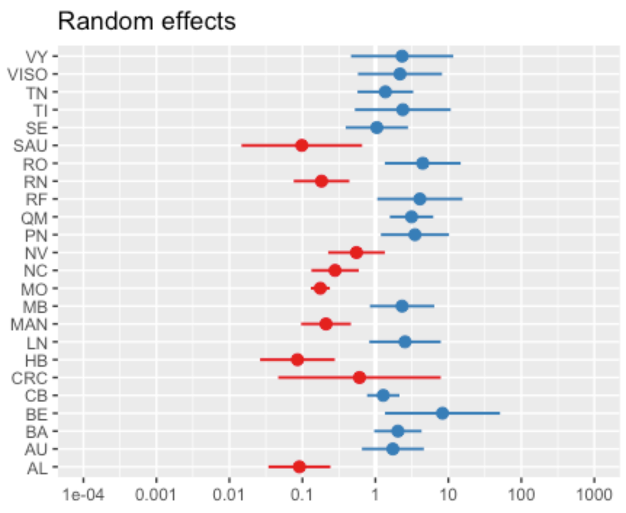
Let's visualize the fitted model.

```
plot(ggeffect(mod, terms = c('Length', 'Sex')))
```



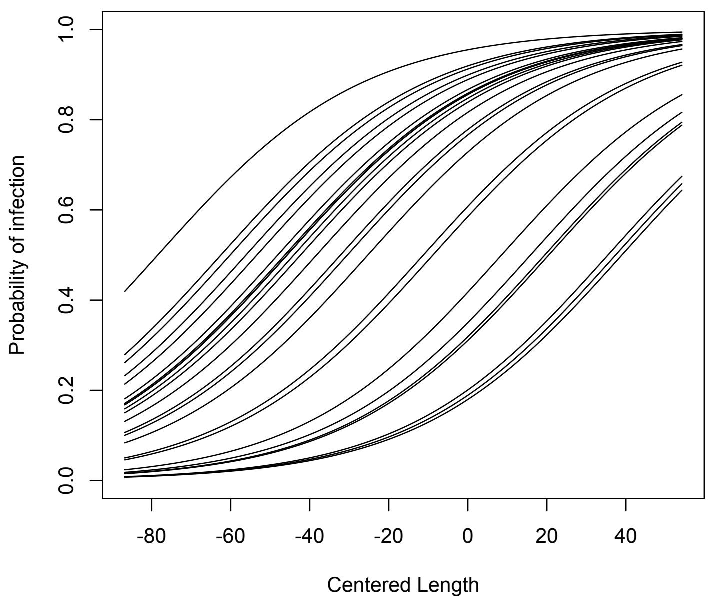
It looks like for both males and females, larger deer are much more likely to be infected; the predicted essentially goes from very little infection to very certain infection. At the same time, the slopes differ between female and males. Small females are more likely to be infected, but large females are less likely to be infected. However, the interaction effect is modest relative to the overall trend for both sexes.

```
plot_model(mod, type = 're')
```



As we saw from the random effects variance estimate, the variation among farms is large. I don't know enough about this data to know why, but clearly some ecological factors that vary among farms produce big differences in infection probability. To get a sense for the ramifications of these differences, we can extract the random effects, and add them to the fixed effects, to visualize variation among farms. I'll just use the fixed effects for females, since we want to focus on among-farm variation:

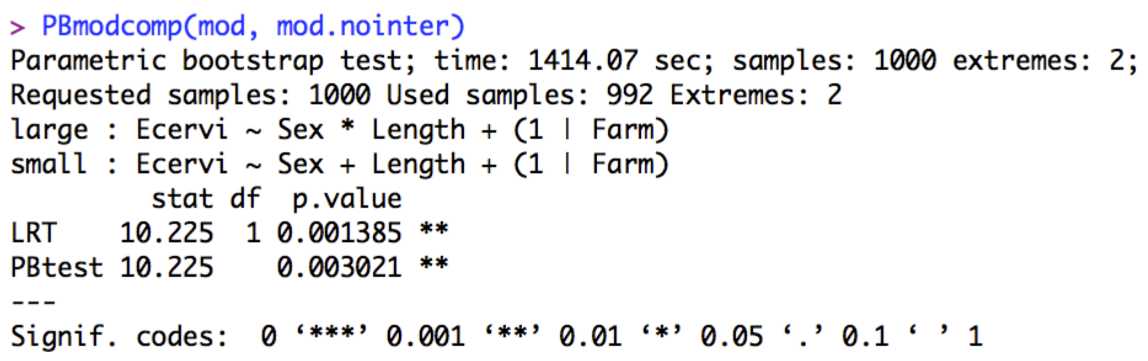
```
random.intercepts = ranef(mod)$Farm[["(Intercept)"]]
farm.intercepts = fixef(mod)[1] + random.intercepts
for (i in 1:length(random.intercepts)) {
  if (i == 1) curve(exp(farm.intercepts[i] + fixef(mod)["Length"]*x)/(1
 +  exp(farm.intercepts[i] + fixef(mod)["Length"]*x)), from = min(deer
$Length), to = max(deer$Length), ylim = c(0, 1))
  if (i > 1) curve(exp(farm.intercepts[i] + fixef(mod)["Length"]*x)/(1
+ exp(farm.intercepts[i] + fixef(mod)["Length"]*x)), add = T)
}
```



Now we have a predicted curve for each farm, for female deer. Some farms have a decent amount of infection even for the smallest deer, while others do not. Likewise, for the largest deer, some farms have only 60% infection, while other farms have essentially 100% infection.

Hypothesis tests with GLMMs

So far the difference between GLMMs and LMMs is minor, from the user's point of view. One of the important differences is that inference with GLMMs is even harder. Now we don't have the tool of approximate F-tests, which try to account for the amount of data we have when calculating p-values. We can still do LRTs or AIC, both of which tend to be anti-conservative. We can also still use the parametric bootstrap. The parametric bootstrap is probably the most legit option in this case, but it tends to be very slow. Fitting GLMMs is more difficult than LMMs, in terms of the numerics involved. When we use the parameteric bootstrap, we have to fit two models (full and restricted) for every simulation, and this can be really slow. For this dataset, I used PBmodcomp to compare a model with and without the interaction, and it took 23 minutes:



```
> PBmodcomp(mod, mod.nointer)
Parametric bootstrap test; time: 1414.07 sec; samples: 1000 extremes: 2;
Requested samples: 1000 Used samples: 992 Extremes: 2
large : Ecervi ~ Sex * Length + (1 | Farm)
small : Ecervi ~ Sex + Length + (1 | Farm)
         stat df  p.value
LRT    10.225  1 0.001385 **
PBtest 10.225    0.003021 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

In this case, the simulation results are not much different from the LRT results, although you can see that the p-value is a bit higher as expected. As a first pass, it can be easier to use an LRT, which can be done with anova():

```
anova(mod, mod.nointer)

## Data: deer
## Models:
## mod.nointer: Ecervi ~ Sex + Length + (1 | Farm)
## mod: Ecervi ~ Sex * Length + (1 | Farm)
##             Df AIC BIC logLik deviance Chisq Chi Df Pr(>Chisq)
## mod.nointer  4 841 860   -416      833
## mod          5 833 856   -411      823  10.2      1     0.0014 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The interaction is highly significant, and plus we have a large sample size, so this result is probably fine. But in cases with a small sample size, or a small effective sample size (depending on which scale the predictor varies), you should treat the LRT results with caution, and the parametric bootstrap is a good idea. We could also use AIC, and again treat the results with caution due to anti-conservative properties.

Overdispersion in GLMMs

I introduced GLMMs with an example using binary data. Binary data cannot be overdispersed by definition, but grouped binomial data (#trials > 1) and count data can be overdispersed. How should we deal with potential overdispersion in GLMMs? In the past we used either quasi-likelihood (for counts or proportional data) or the negative binomial (for count data). In principle either of these can be implemented with GLMMs. Here I will present an additional option (individual-level random effects), and the negative binomial.

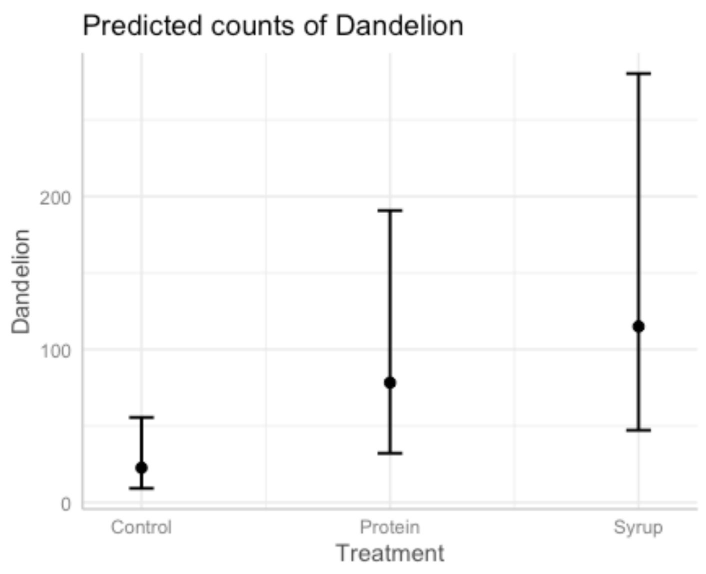
The data comes from an experiment on honeybee pollination. Commercial honeybees are often supplemented with sugar syrup or protein, so that they're well-fed and more likely to pollinate crops that do not provide a lot of nectar. In this experiment, the idea was to quantify how much supplementing with syrup or protein changed the amount of dandelion pollen collected. There were three treatments: control (no supplement), syrup, protein. Each treatment was applied to 5 hives, and each hive was sampled 4 times by counting the number of pollen grains in a pollen trap at the hive entrance.

So this is count data, with 3 treatments, and with 4 observations per hive. To properly model the structure of the data, we should use a random effect for hive, to account for differences between hives in mean pollen. We can use this model:

```
mod = glmer(Dandelion ~ Treatment + (1|Hive), data = pollen, family = poisson)
summary(mod)
```

```
plot(ggeffect(mod.id))
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: Dandelion ~ Treatment + (1 | Hive)
##    Data: pollen
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  Hive   (Intercept) 1.03     1.01
## Number of obs: 60, groups:  Hive, 15
##
## Fixed effects:
##                   Estimate Std. Error z value Pr(>|z|)
## (Intercept)          3.119      0.458    6.81    1e-11 ***
## TreatmentProtein     1.241      0.645    1.92    0.054 .
## TreatmentSyrup       1.626      0.645    2.52    0.012 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) TrtmnP
## TretmntPrtn -0.710
## TretmntSyrp -0.710  0.504
```



This is pretty simple model, and it looks like the amount of pollen collected is about 3-5 times higher when the bees are supplemented, though there is a lot of uncertainty around those treatment means. We can do some hypothesis tests or model selection to see how much support there is for the treatment effect, but first let's consider overdispersion.

When we looked at overdispersion previously, we calculated a 'dispersion parameter' that quantified how much extra variance there is in the data. This was equal to the sum of the squared pearson residuals, divided by the residual degrees of freedom (the number of observations minus the number of estimated parameters). This number was already a rough guide, and now it's going to get rougher, because of the challenge in defining #parameters and degrees of freedom in a mixed model. In the case of the honeybees, we have 60 observations from 15 hives, so the 'effective' degrees could be lower than (60 - #parameters), if observations from the same hive tend to be very similar. For this reason, the dispersion parameter we calculate will probably be too small, because the effective degrees of freedom can be smaller than what's used in the formula. Someone made a function to calculate this number using glmer:

```
library(RVAideMemoire)

overdisp.glmer(mod)

## Residual deviance: 971.236 on 56 degrees of freedom (ratio: 17.343)
```

Here 'ratio' is the dispersion parameter. 17 is super high, so the data are overdispersed.

How should we model overdispersion in a GLMM? One option is to use a negative binomial distribution instead of a Poisson distribution, and there is a special glmer.nb() function for doing this:

```
mod.nb = glmer.nb(Dandelion ~ Treatment + (1|Hive), data = pollen)

overdisp.glmer(mod.nb)

## Residual deviance: 54.634 on 56 degrees of freedom (ratio: 0.976)
```

Note that the negative binomial distribution has successfully accounted for overdispersion. How should we test the treatment effect? Chi-square likelihood ratio test and parametric bootstrap are options. The bootstrap might be preferred in this case because the number of experimental replicates is not large, and the chi-square LRT is correct only under large sample size.

```
mod.nb.notreat = glmer.nb(Dandelion ~ 1 + (1|Hive), data = pollen)

PBmodcomp(mod.nb, mod.nb.notreat)
```

```
## Bootstrap test; time: 60.75 sec; samples: 1000; extremes: 108;
## large : Dandelion ~ Treatment + (1 | Hive)
## Dandelion ~ 1 + (1 | Hive)
##          stat df p.value
## LRT    5.7163  2 0.05738 .
## PBtest 5.7163    0.10889
```

The results indicate a marginal treatment effect, and the p-value for the parameteric bootstrap is roughly double that from the chi-square test, which is consistent with the chi-square test being anti-conservative when sample size is small.

Another option for modeling overdispersion in mixed models, which is useful to know about, is a called an 'individual-level random effect' or 'observation-level random effect'. The idea is that we will create a new factor where each observation gets its own level, and use that as a random effect:

```
pollen$ID = factor(1:nrow(pollen))
mod.id = glmer(Dandelion ~ Treatment + (1|Hive) + (1|ID), data = pollen,
family = poisson)
```

What does this do? Recall that with a GLMM the model is linear, and the random effects are normally distributed, on the link scale. For a Poisson model, the expected value of an observation is λ_i, which is the mean of the Poisson distribution for observation *i*. When we put in an observation-level random effect, we are assuming that $\log(\lambda_i)$ is normally distributed (because the log is the link function), or equivalently that λ_i is lognormally distributed. So this is how we are modeling the overdispersion: in addition to whatever fixed and random effects are in the model, we assume that there is extra variation in the λ_i's that is lognormally distributed.

This will capture the extra variance in the counts that is beyond what the Poisson distribution predicts.

```
summary(mod.id)

## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: Dandelion ~ Treatment + (1 | Hive) + (1 | ID)
##    Data: pollen
##
##      AIC      BIC   logLik deviance df.resid
##    642.4    652.9   -316.2    632.4       55
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.4602 -0.1445  0.0223  0.1031  0.6259
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  ID     (Intercept) 0.389    0.624
##  Hive   (Intercept) 1.062    1.031
## Number of obs: 60, groups:  ID, 60; Hive, 15
##
## Fixed effects:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)         2.977      0.487    6.11    1e-09 ***
## TreatmentProtein    1.228      0.686    1.79    0.073 .
## TreatmentSyrup      1.664      0.686    2.43    0.015 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) TrtmnP
## TretmntPrtn -0.711
## TretmntSyrp -0.710  0.505
```

The standard deviation for ID is 0.62, which means that overdispersion causes the typical λ_i to vary by a factor of exp(0.62) = 1.9. Let's see what the 'dispersion parameter' is:

```
overdisp.glmer(mod.id)

## Residual deviance: 12.721 on 55 degrees of freedom (ratio: 0.231)
```

The ratio is 0.23. This is certainly not overdispersed, and actually is too low (should be around 1). Probably this is due to the fact that it is calculated using 55 degrees of freedom but the effective degrees of freedom is lower.

To finish the analysis, we can see if there's significant variation explained by Treatment:

```
mod.id.notreat = glmer(Dandelion ~ 1 + (1|Hive) + (1|ID), data = pollen, famil
y = poisson)
PBmodcomp(mod.id, mod.id.notreat)
```

```
## Bootstrap test; time: 89.07 sec; samples: 1000; extremes: 110;
## large : Dandelion ~ Treatment + (1 | Hive) + (1 | ID)
## Dandelion ~ 1 + (1 | Hive) + (1 | ID)
##          stat df p.value
## LRT    5.3079  2 0.07037 .
## PBtest 5.3079    0.11089
```

The results are similar to the negative binomial model: a marginal treatment effect, and a smaller p-value from the chi-square LRT compared to the parametric bootstrap.

Generalized additive mixed models

After we covered GLMs, we allowed for the predictors to have smooth nonlinear effects using GAMs. Now we've added random effects into the mix, to model various kinds of grouped structure in the data. Naturally you might want to allow for nonlinear predictors while also including random effects to account for spatiotemporal structure etc. This can be done with generalized additive mixed models (GAMMs).

GAMMs are kind of a new and developing technique, and there are various R packages that can fit them using slightly different algorithms. I won't spend a lot of time parsing out the subtle differences in how different functions work, and if you're interested one resource is a recent book on GAMMs in R by Zuur and colleagues. I'll go through one example to think about when and how you might use a GAMM, using the package gamm4, which combines functions from mgcv and lme4.

In a homework assignment you used GAMs to look at the spatial structure of coyote-wolf hybridization in Ontario. In the paper from which that data came, the authors actually used GAMMs to analyze the data, because multiple packs were sampled, with 1-4 individuals per pack. Because individuals from the same pack are likely to have similar ancestry, it's important to account for that structure in the model. The gamm4 function combines syntax from gam() and lmer():

```
library(gamm4)
mod.gamm = gamm4(CoyoteAncestry.Logit. ~ s(PrimaryRds) + s(sqrt(SecondaryRds))
+ s(TertiaryRds, by = StudyArea) + s(Deer) + s(Moose) + StudyArea, random = ~
(1|PackNumber), data = wolf)
```

Note that the random effects get their own argument now, where you specify the right-hand-side of a formula, but otherwise the syntax is the same. Interpreting the output from this function is a little tricky, because it gives you two versions, a 'gam' version and a 'mer' version. Let's look at these:

```
summary(mod.gamm$gam)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## CoyoteAncestry.Logit. ~ s(PrimaryRds) + s(sqrt(SecondaryRds)) +
##     s(TertiaryRds, by = StudyArea) + s(Deer) + s(Moose) + StudyArea
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -2.647      0.864   -3.06   0.0030 **
## StudyAreaOut   3.053      0.958    3.19   0.0021 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                              edf Ref.df     F p-value
## s(PrimaryRds)               1.00   1.00  0.61 0.43553
## s(sqrt(SecondaryRds))       1.00   1.00 13.38 0.00046 ***
## s(TertiaryRds):StudyAreaAPP 2.61   2.61  1.82 0.15537
## s(TertiaryRds):StudyAreaOut 1.00   1.00  5.10 0.02671 *
## s(Deer)                     1.00   1.00  1.24 0.26892
## s(Moose)                    1.00   1.00  6.37 0.01367 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.583
## lmer.REML = 344.75  Scale est. = 2.3626    n = 85
```

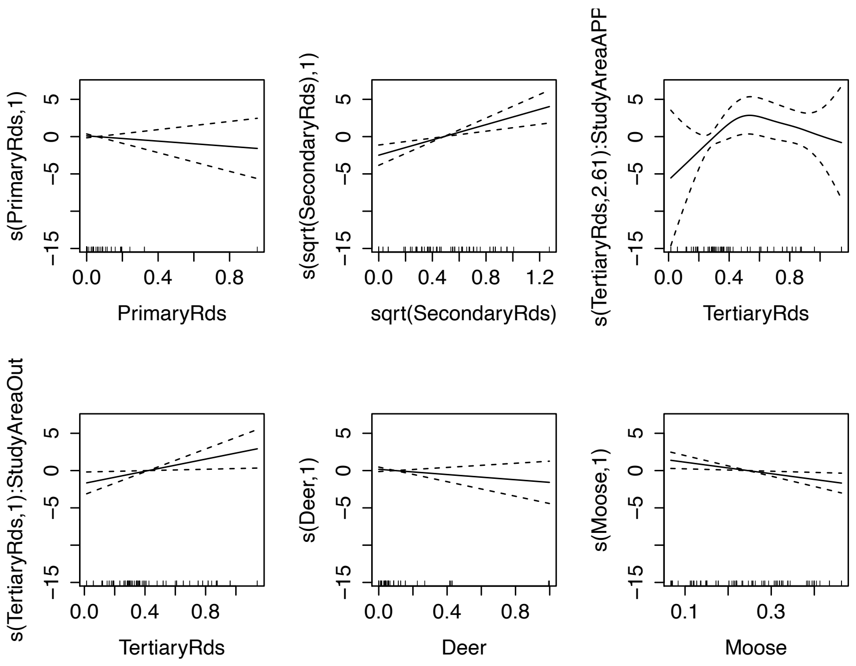
This looks pretty similar to the output from the gam() function. We get info on each smoother, but there is nothing about the random effect for PackNumber.

```
summary(mod.gamm$mer)


## Random effects:
##  Groups     Name                        Variance Std.Dev.
##  PackNumber (Intercept)                   1.38    1.17
##  Xr.4       s(Moose)                      0.00    0.00
##  Xr.3       s(Deer)                       0.00    0.00
##  Xr.2       s(TertiaryRds):StudyAreaOut   0.00    0.00
##  Xr.1       s(TertiaryRds):StudyAreaAPP 200.64   14.16
##  Xr.0       s(sqrt(SecondaryRds))         0.00    0.00
##  Xr         s(PrimaryRds)                 0.00    0.00
##  Residual                                 2.36    1.54
## Number of obs: 85, groups:
## PackNumber, 47; Xr.4, 8; Xr.3, 8; Xr.2, 8; Xr.1, 8; Xr.0, 8; Xr, 8
##
## Fixed effects:
##                                 Estimate Std. Error t value
## X(Intercept)                          -2.647      0.864   -3.06
## XStudyAreaOut                          3.053      0.958    3.19
## Xs(PrimaryRds)Fx1                     -0.216      0.276   -0.78
## Xs(sqrt(SecondaryRds))Fx1              1.511      0.413    3.66
## Xs(TertiaryRds):StudyAreaAPPFx1        1.681      3.968    0.42
## Xs(TertiaryRds):StudyAreaOutFx1        1.071      0.474    2.26
## Xs(Deer)Fx1                           -0.351      0.316   -1.11
## Xs(Moose)Fx1                          -0.840      0.333   -2.52
```

Now we get a lot more info: a random effect estimate for PackNumber, plus random effect variances for all the smoothers, and then fixed effect estimates for the smoothers as well. What's going on here? gamm4 uses a kind of computational trick that allows GAMs and mixed models to be combined. Mathematically, the smoother can be estimated in terms of a random effect variance that quantifies how nonlinear the smoother is. In general we don't need to think about this too hard, because the GAM fit is summarized more intuitively in the mod.gamm$gam slot, and we can just refer to that for the smoothers. But now you know why the smoothers are also in the mod.gamm$mer slot, which represents what the full model is going. We need to use the $mer slot to get info on 1) any random effects, such as PackNumber, 2) any fixed effects that are not smoothers (none in this model), 3) to extract the likelihood of the model for use in LRT or AIC.

Let's see what the smoothers look like:

```
par(mfrow = c(2,3))
plot(mod.gamm$gam)
```



Interestingly, 5 of the 6 smoothers are estimated to be linear functions, and the 6th smoother (TertiaryRds in StudyAreaAPP) is not significant according to the F-test in the summary. One thing we could do is just refit the whole thing as an LMM to simplify things, but I won't do that now.

One reason the gamm4 function is nice, compared to some other GAMM options, is that the model has a likelihood that we can use for LRTs and AIC. The usual mixed model caveats apply, about anti-conservative or conservative properties of the different tests. As an example, I'll test whether the effect of Moose on %coyote ancestry is significant. Since it appears to be a linear relationship, first I'll fit a model with Moose as a linear effect, and compare to a model with Moose removed:

```
mod.gamm.mooselinear = gamm4(CoyoteAncestry.Logit. ~ s(PrimaryRds) + s(sqrt(Se
condaryRds)) + s(TertiaryRds, by = StudyArea) + s(Deer) + Moose + StudyArea, r
andom = ~ (1|PackNumber), data = wolf, REML = FALSE)


mod.gamm.nomoose = gamm4(CoyoteAncestry.Logit. ~ s(PrimaryRds) + s(sqrt(Second
aryRds)) + s(TertiaryRds, by = StudyArea) + s(Deer) + StudyArea, random = ~ (1
|PackNumber), data = wolf, REML = FALSE)


anova(mod.gamm.nomoose$mer, mod.gamm.mooselinear$mer)

## Data:
## Models:
## mod.gamm.nomoose$mer: NULL
## mod.gamm.mooselinear$mer: NULL
##                          Df AIC BIC logLik deviance Chisq Chi Df
## mod.gamm.nomoose$mer     14 377 411   -174      349
## mod.gamm.mooselinear$mer 15 375 412   -173      345  3.85      1
##                          Pr(>Chisq)
## mod.gamm.nomoose$mer
## mod.gamm.mooselinear$mer       0.05 *
## ---
```

Looks like the effect has p = 0.05. Note that I fit each model with REML = FALSE, and use the $mer slot to compare the likelihoods. We could also compare with AIC:

```
library(MuMIn)
AICc(mod.gamm.mooselinear$mer)

## [1] 382.1

AICc(mod.gamm.nomoose$mer)

## [1] 383
```

Looks like the information theory approach is less sanguine about whether moose are important; removing them increases AICc, but by less than 1. So moose are in the best model, though it's not super convincing. The AICc comparison is probably a little different from the LRT because AICc attempts to account for sample size.