GENERALISED ADDITIVE MIXED MODELS FOR DYNAMIC ANALYSIS IN LINGUISTICS: A PRACTICAL INTRODUCTION

Márton Sóskuthy
University of York

1 Introduction

This is a hands-on introduction to Generalised Additive Mixed Models (GAMMs; Wood 2006) in the context of linguistics with a particular focus on dynamic speech analysis. Dynamic speech analysis is a term used to refer to analyses that look at measureable quantities of speech that vary in space and/or time. Temporal variation can be short-term (e.g. formant contours and pitch tracks) or long-term (e.g. diachronic change or change over the life-span). Similarly, spatial variation can sometimes be measured in millimetres (e.g. tongue contours), and sometimes in kilometres (e.g. the acoustic realisation of a sound category as a function of location on a dialect map). The focus of this introduction is mostly on short-term temporal variation in phonetics, and more specifically on formant trajectories (though one of the examples also involves diachronic change). This choice is mostly practical: I chose to illustrate GAMMs through formant trajectories simply because I'm comfortable talking about them. However, most of the concepts and techniques discussed here are more general and are also applicable to other types of short-term and long-term temporal trajectories as well as spatial data.

The main goal of this introduction is to explain some of the main ideas underlying GAMMs, and to provide a practical guide to frequentist significance testing using these models. Some of the suggestions below are based on theoretical work presented in Sóskuthy (2016) and Sóskuthy (in prep), but this introduction is meant as a standalone guide.

The following discussion is divided into two parts, which can be read in slightly different ways. The first part (section 2) looks at what GAMMs actually are, how they work and why/when we should use them. Although the reader can replicate some of the example analyses in this section, this is not essential – reading the section should be enough. The second part (section 3) is a tutorial introduction that illustrates the process of fitting and evaluating GAMMs in the R statistical software environment (R Core Team, 2013), and the reader is strongly encouraged to work through the examples on their own machine.

Since a lot of the research in GAMM theory is closely intertwined with software development in R (e.g. the author of one of the main textbooks on GAMM, Wood 2006 is also the maintainer of one of the main GAMM software packages), it is difficult to talk about GAMMs without using some of the terminology and conventions of R. Therefore, although the discussion in the first part is mostly conceptual, I do rely on R code to illustrate the structure of different models. However, I've tried not to use too much code, and it should be possible to follow the discussion without a strong background in R. The second part relies much more heavily on R and will be mostly of interest to readers who want to fit their own models using R. Readers who want to learn more about R before reading this introduction may want to consult Baayen (2008) and Johnson (2008), who both provide thorough introductions to R for beginners using examples from linguistics.

GAMMs are a type of regression model and they are closely related to mixed effects regression. This tutorial assumes some background in regression modelling and it will help to be familiar with mixed effects models as well. Winter (2013) is a short but excellent introduction to mixed effects modelling, while Gelman & Hill (2007) and Baayen (2008) provide more in-depth treatments of the topic.

The examples in this introduction rely on three R packages: `mgcv` (Wood, 2006), `gamm4` (Wood & Scheipl, 2014) and `itsadug` (van Rij et al., 2016). These should be installed and loaded before trying to run the analyses that follow. I've also put together a script with a few 'GAMM hacks' (`gamm_hacks.r`), which help to keep the code in the tutorial tidier and may also be useful for the reader's own analyses. This should be sourced after loading `itsadug`, as it overrides some of the functions in that package. Finally, the data sets for the tutorial are in two separate files called `words_50.csv` and `glasgow_r.csv`, which should be imported into R as `words.50` and `gl.r`.

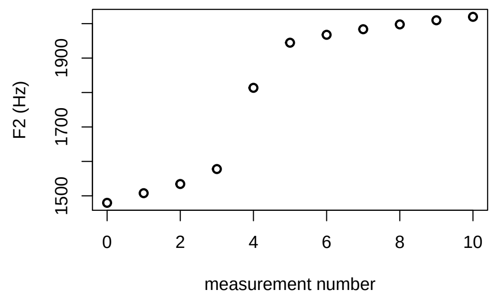
The data sets, the GAMM hacks script, the slides for Sóskuthy (2016) and the source code for the PDF are all available from my GitHub page: https://github.com/soskuthy/gamm_intro.

I relied on a number of sources for this introduction, but the main body of the text is light on references to make the discussion easier to follow. There is a more detailed list in the final section that also includes links and brief summaries of each source.

2 A gentle introduction to GAMM theory

2.1 GAMs

Before discussing GAMMs, let's start with a slightly simpler type of model called Generalised Additive Models (that's GAMM without the 'mixed' part). The easiest way to understand GAMs is through a comparison with linear regression models. We will work through a specific example, where our goal will be simply to fit a regression line/curve to an F2 trajectory. The formant measurements are in Hz and the trajectory is represented by 11 measurement points taken at equal intervals (going from the very beginning to the very end). The figure below shows the trajectory:



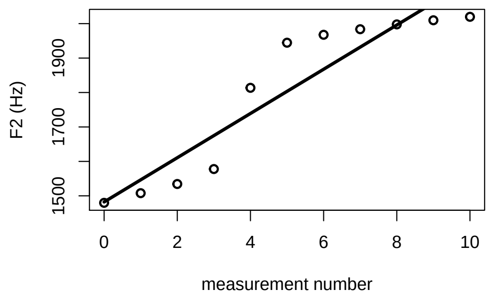
Here is the most straightforward way to specify a linear model that fits a line to the trajectory in R:

```r
demo.lm <- lm(f2 ~ measurement.no, data = dat)
```

Though I'll try to use mathematical notation sparingly in this tutorial, it is useful to write out the model as a formula, as it will help with the transition to GAMs:[1]

$$Y_{\mathrm{f2}} = \alpha + \beta_1 X_{\mathrm{measurement.no}} \tag{1}$$

This is simply an equation for a straight line: $X_{\mathrm{measurement.no}}$ stands for the values along the x-axis (measurement number), while Y_{f2} stands for corresponding values along the y-axis (F2 in Hz). α and β_1 specify the *intercept* (i.e. the height) and the *slope* of the regression line that represents the relationship between F2 and `measurement.no`. When we fit a regression line to the actual trajectory, the result looks like this:



Though this isn't necessarily a terrible fit, the regression line is unable to capture the non-linear nature of the trajectory. Depending on what one wants to do with the model, this may or may not be an issue. If, however, we do want to account for non-linear relationships in our model, we'll need to take a different approach.

GAMs provide one way of getting around this problem. Let us refer to the slopes and intercepts of linear regression models as *parametric terms*. GAMs differ from traditional linear regression models by allowing so-called *smooth terms* alongside parametric terms. Smooth terms are extremely flexible, so much so that their mathematical representation in model specifications is simply 'some function of x':

$$Y_{\mathrm{f2}} = \alpha + f_1(X_{\mathrm{measurement.no}}) \tag{2}$$

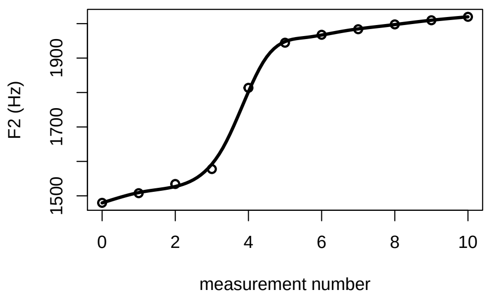
This model specification does not say anything about the shape of the function linking $X_{\mathrm{measurement.no}}$ and Y_{f2}. The only requirement on the smooth term $f_1(X_{\mathrm{measurement.no}})$ is that it should be a smooth function of one or more predictor variables. Since the shape of this smooth function is not constrained in the same way as it is for regression lines, GAMs can deal with non-linearity. The following R function can be used to fit a GAM with the structure above to the F2 trajectory:

```r
demo.gam <- gam(f2 ~ s(measurement.no, bs = "cr"), data = dat)
```

The `s()` notation is used to distinguish smooth terms from parametric ones. The `bs="cr"` parameter tells R to use a so called 'cubic regression spline' as the smooth term

[1] I've left out the so-called error term to keep things simple, though this should technically be part of the model specification. Note that the error term is exactly the same in linear models and GAMs.

(bs actually stands for 'basis', a concept that will be discussed in more detail below). The model fit is shown below.



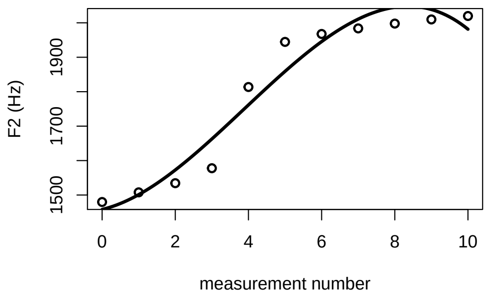
So how are such wiggly smooth terms created in practice? We are not going to discuss the mathematical underpinnings of GAMs, but there are two fundamental and relatively straightforward concepts that need to be understood if one wants to work with these models: *basis functions* and the *smoothing parameter*. Basis functions are simple functions that add up to a (potentially) more complex curve. For instance, consider the following model:

```
demo.poly <- lm(f2 ~ measurement.no + I(measurement.no^2) +
                     I(measurement.no^3), data=dat)
```

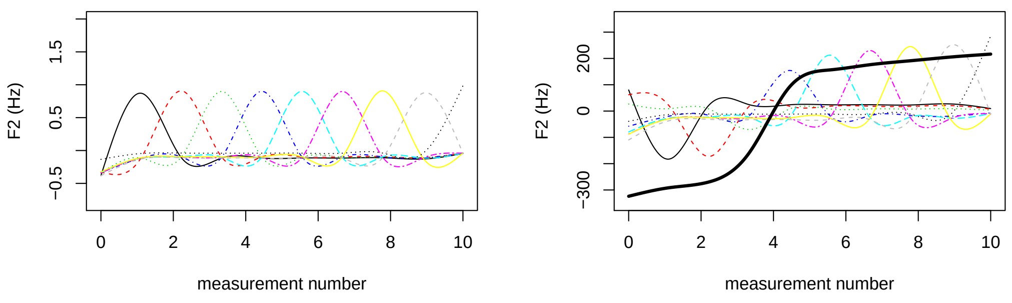
Or, in mathematical notation:

$$Y_{\text{f2}} = \alpha + \beta_1 X_{\text{measurement.no}} + \beta_2 X^2_{\text{measurement.no}} + \beta_3 X^3_{\text{measurement.no}} \quad (3)$$

And the model fit:

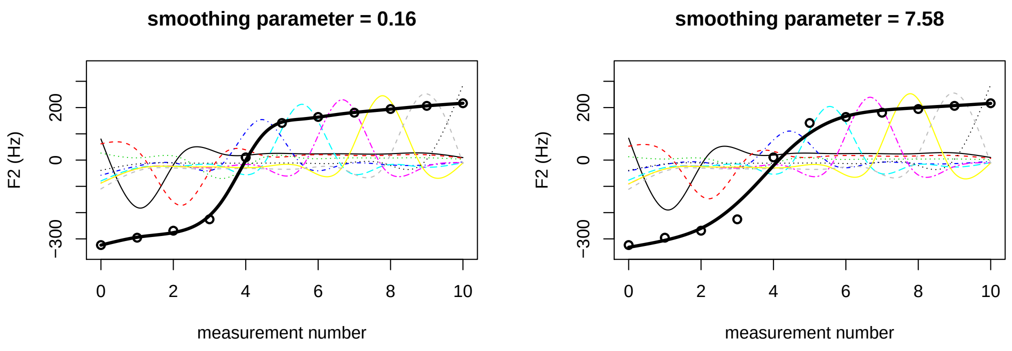


This is an example of polynomial regression, where both a variable x and some of its powers (x^2, x^3, . . .) are included as predictors. The terms x, x^2, x^3 are referred to as basis functions. The fitted curve is obtained by multiplying each of the basis functions by the corresponding coefficient and then adding them up. The plot on the left below shows the basis functions of the GAM smooth for the formant trajectory before multiplication by the coefficients. The plot on the right shows the same basis functions after multiplication, and their sum (i.e. the predicted formant trajectory minus the intercept term).

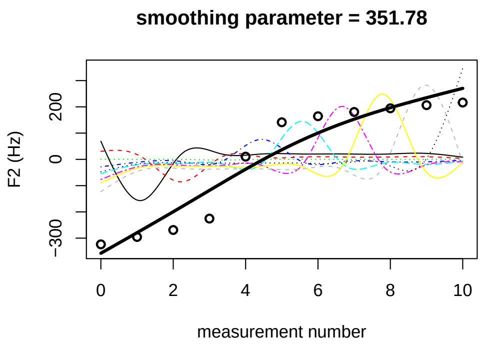


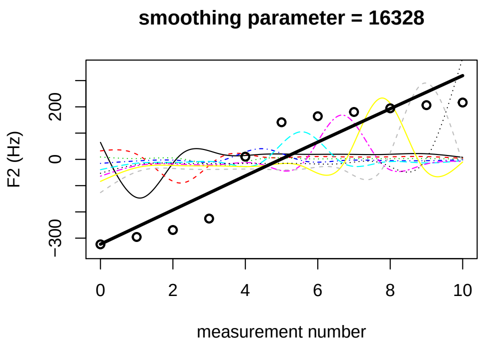
The above smooth is made up of 9 basis functions, which is a default setting for certain types of smooths in R. Note that the basis functions are placed at regular intervals, and that they converge on each other at certain points (this is clearer in the plot on the left-hand side). These 'convergence points' are called *knots*, and there are 10 of them (the number of basis functions + 1): 2 at the edges of the plot and 8 in the middle. There is a simple intuition linked to the number of knots / basis functions: increasing this number allows for more wiggliness in the smooth, while decreasing it makes the smooth. . . well, smoother.

In linear regression models where the basis functions are included manually (e.g. polynomial regression), the number of basis functions (or knots) has to be chosen by the modeller. This can be tricky, as using too few basis functions can lead to oversmoothing (i.e. missing some of the non-linearity in the data), while using too many can lead to overfitting (i.e. missing the real trend in the data by fitting a curve to random noise). This is where GAMs come into their own. GAMs rely on a value called the smoothing parameter. The coefficients for the individual basis functions contained in a GAM smooth are estimated in such a way that the resulting curve cannot exceed a certain degree of wiggliness – and the degree of wiggliness is determined by the smoothing parameter. The higher the smoothing parameter, the smoother (= less wiggly) the estimated curve. This is illustrated below, where the number of basis functions is always the same, but the value of the smoothing parameter is varied.[2]



[2]As before, the curve is shown without the intercept, which means that it is centred around 0 along the y-axis. This is why some of the values are negative. The graphs also include shifted versions of the actual data points to make the degree of smoothing clearer.



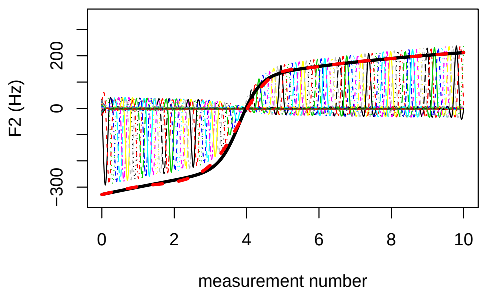


In other words, the degree of smoothness / wiggliness in GAMs is mostly determined by the smoothing parameter. The role of the number of basis functions / knots is reduced to setting an upper limit on the degree of wiggliness: having 3 knots (= 2 basis functions) obviously allows for much less wiggliness than 10 or 100 knots, even if we set the smoothing parameter extremely low.

GAMs use a nifty trick to spare you the trouble of having to decide on a value for the smoothing parameter: they use a method called *cross-validation* (and sometimes other estimation methods) to estimate it directly from the data. Since the number of basis functions has little bearing on the shape of the smoother provided that there are enough of them to represent the degree of wiggliness in the data, a smooth with a high number of basis functions will often look very similar to one with a lower number. The code below fits a smooth with 99 basis functions (= 100 knots) to the same data set (the parameter k controls the number of knots).[3]

```
demo.gam.100 <- gam(f2 ~ s(measurement.no, bs = "cr", k = 100),
                    data = dat)
```

The model predictions are shown in the following plot. The original smooth (red dashed line) is also included for comparison.



Theoretically speaking, one could always just set the number of basis functions really high and then let R determine the degree of smoothness that is appropriate for the data. So why should we worry about basis functions and the smoothing parameter at all?

[3]Actually, the number of measurements had to be increased to 101 from 11, as a curve with only 11 samples is simply not sufficient to determine the coefficients for each of the 99 basis functions. This is the same issue that arises when we try to fit a model with $p > n$ predictors to n data points. Otherwise, the underlying curve is exactly the same.

Couldn't we just add `k = 1000` to our smooth specifications and forget about the whole thing? Or just trust R to do the right thing for us?

The answer is: no. As is often the case with statistical methods, knowing only a little about GAMs and setting up models without understanding what they do is probably worse than using less advanced methods in an informed way. First, there are several situations where the number of basis functions needs to be changed, and these are almost impossible to avoid while working with dynamic speech data:

- *too few measurements to support* `k` *knots*: If our trajectories only have 11 measurements, the maximum number of knots is also 11 (that is, the maximum number of basis functions is 10; cf. footnote 3). Since the default value for `k` is 10 (although this number may be different depending on the type of smooth), it needs to be lowered if there are less than 10 unique values for a given variable.
- *not enough wiggliness allowed*: The default value of `k` can only support a certain amount of wiggliness in the data. If the actual trajectories show a greater degree of non-linearity, `k` needs to be increased.
- *computational inefficiency due to high* `k`: The higher the value of `k`, the longer it will take to fit the model. Therefore, it is a good idea not to increase `k` any further than necessary. In realistic scenarios, the modeller may be forced to choose a `k` that is actually lower than would be ideal.

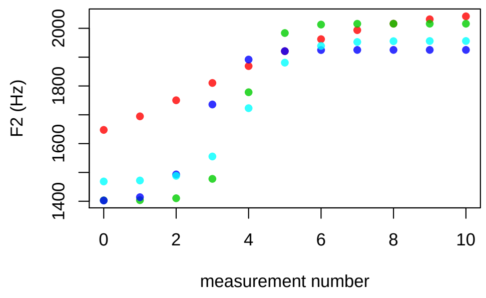
Moreover, the concepts of basis function and smoothing parameter are closely related to the so-called *estimated degrees of freedom* (or EDF). When the coefficients for basis functions are estimated without any constraints, as in the case of polynomial regression, a smooth with p basis functions uses up exactly p degrees of freedom. However, when the coefficients are constrained by a smoothing parameter, the effective degrees of freedom taken up by the smooth go down. For instance, at extremely high values of the smoothing parameter, the smooth becomes a straight line, which only uses up a single degree of freedom, even if it is represented by more than one basis function. Conversely, at extremely low values of the smoothing parameter, the smooth will use all the potential wiggliness provided by the $\mathtt{k} - 1$ basis functions, which corresponds to $\mathtt{k} - 1$ degrees of freedom. At intermediate values, the smooth uses an intermediate number of degrees of freedom. The EDF is an estimate of the degrees of freedom that are actually used by a smooth with a given number of basis functions and a given smoothing parameter. Significance tests of smooth terms in GAMs rely on the EDF and not the number of basis functions – and, as a result, model summaries for GAMs always include the EDF. One of the advantages of using GAMs is that there is no need to worry about overfitting even if the number of basis functions is very high: the EDF will typically be substantially lower than `k`, and it is only the former that matters from the perspective of overfitting.

2.2 GAMMs

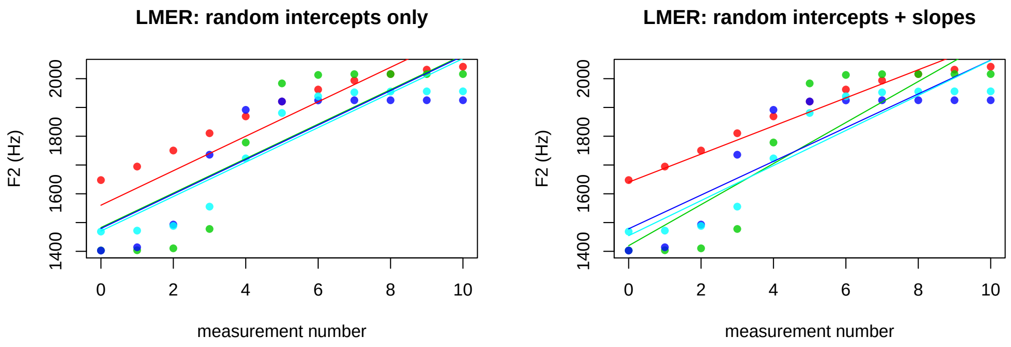
We are now in a position to move on to GAMMs, that is, generalised additive *mixed* models. This tutorial assumes that the reader is already familiar with random intercepts and slopes from linear mixed effects models and knows how to implement them in R. Just as a reminder: random intercepts in linear mixed models capture by-group random variation in the outcome variable (e.g. between-speaker differences in average

F2 values); random slopes capture by-group random variation in the effect of a predictor variable on the outcome variable (e.g. between-speaker differences in the effect of style on F2 – certain speakers exhibit more stylistic adaptation than others). GAMMs are to GAMs as linear mixed effects models are to linear models. That is, GAMMs incorporate random effects alongside parametric and smooth terms. Similar to linear mixed effects models, these random effects can be random intercepts and random slopes. However, GAMMs also offer a third option: random smooths.

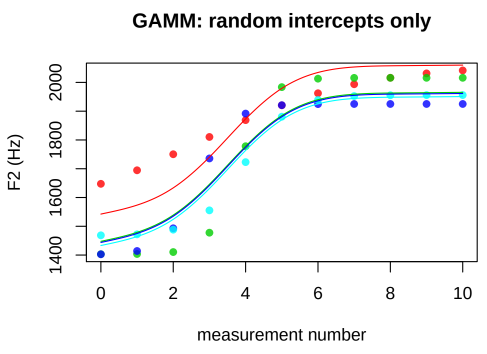
Random smooths are similar to random slopes, but they are more flexible than the latter: while random slopes can only capture by-group variation in linear effects, random smooths can also deal with by-group variation in non-linear effects. An example will help to make this point clearer. Let's assume that the trajectory that we've looked at earlier is an example of a specific vowel, and that we have four further tokens for this vowel. The four trajectories are shown below:

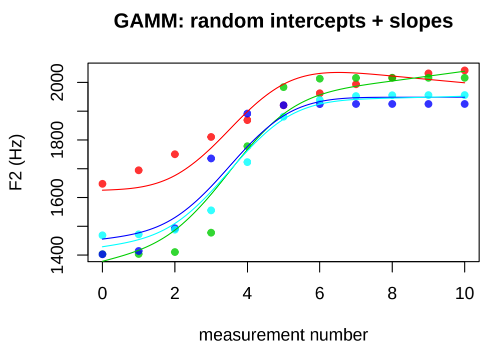


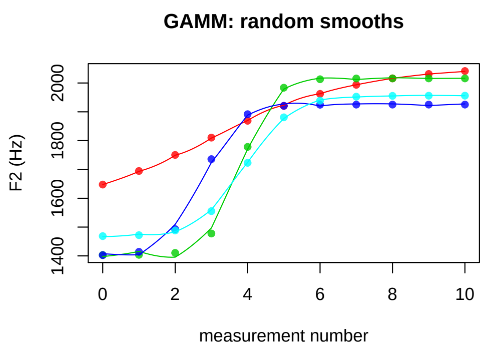
First, let's use linear mixed effects models to fit straight line approximations to the trajectories. We'll fit two versions of the same model: one with random intercepts only, and a second one with random intercepts and random slopes. The random intercepts model is shown on the left below, while the random intercepts + slopes model is shown on the right.



Unsurprisingly, the model fit is not great. In the random intercepts only model, the fitted lines are parallel to each other, as the slopes are not allowed to vary. In the random intercepts + slopes model, both the height and the slope of the lines vary. Now let's fit three GAMMs to the same trajectories: one with random intercepts only, a second one with random intercepts + slopes and a third one with random smooths.







The model with random intercepts simply varies the height of the lines, but does not yield a particularly good fit. The one with slopes does slightly better: in this case, the same curve is essentially rotated and stretched to match the actual trajectories. Random smooths clearly provide the best fit by fitting individual curves to each trajectory. Note, however, that random smooths are also extremely resource intensive: fitting four separate random smooths to the data requires $4 \times \mathtt{k}$ basis functions, and the same number of coefficients need to be estimated.[4]

2.3 Why do we need random smooths?

But why do we need to add random effects to our models, and why is it important to make sure that they fit the data as accurately as possible? This question is partly answered in standard texts dealing with mixed effects regression models (Baayen, 2008; Gelman & Hill, 2007; Zuur et al., 2009). More detailed treatments are given in Barr et al. (2013) and Bates et al. (2015), while Winter (2013) provides a particularly clear explanation that should be fairly easy to follow for readers with no background in statistics. The focus in many of these texts is on how models without random effects may violate the underlying assumptions of regression models, in particular the assumptions of *linearity* and *independence of errors*. What I'd like to do here is offer a slightly different approach to the question, and discuss the importance of random effects in more intuitive terms.

Regression models attempt to make educated guesses about 'hidden' underlying

[4]The separate random smooths each use up `k` rather than $\mathtt{k}-1$ basis functions (as opposed to the smooths that we've looked at before). The additional basis function plays the role of a random intercept. Since there are four smooths (one for each trajectory), the overall number of basis functions is $4 \times \mathtt{k}$.

parameters (which correspond to properties of interest in the real world) based on observable data. Each of the data points in a data set provides some information about these parameters, increasing our confidence in the guesses the model makes about them (we will refer to these guesses as estimates). However, it turns out that individual data points don't always provide the same amount of information about the underlying parameters. In other words, two data sets with the same number of observations may provide different amounts of information about the same parameters. These differences are often dependent on the presence or absence of grouping structure in the data.

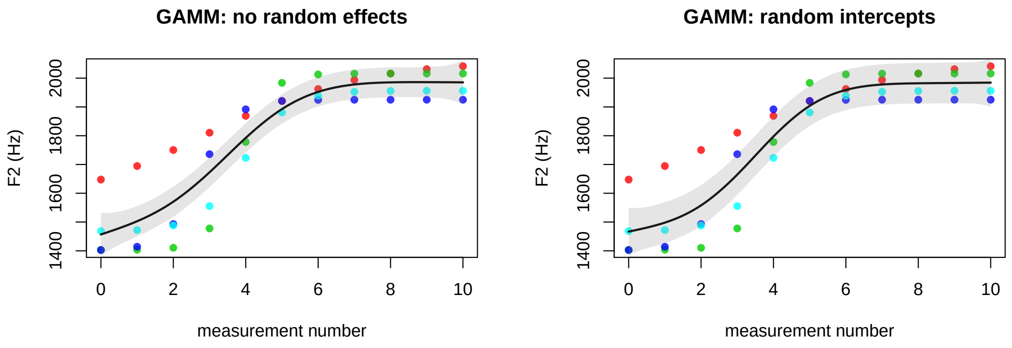
In terms of the amount of information per data point, the best case scenario is a data set where there is no group structure beyond the variables whose effects we want to estimate. In such data sets, a regression model can assume that individual data points are determined solely by the underlying parameters plus a bit of random noise, so all the information in a given data point can be used towards determining the underlying parameters. To stay with the formant trajectory example, our data set would have this structure if each of the measurements (i.e. each of the dots in the graphs above) came from a separate vowel. This would mean that we would have to sample 44 different vowel tokens, and only take a single measurement at a random time point for each of them (this would be admittedly a rather weird data set). If we were to, say, double the number of sampled vowels, our confidence in our estimates would also increase – each new data point would contribute additional information about the underlying parameters

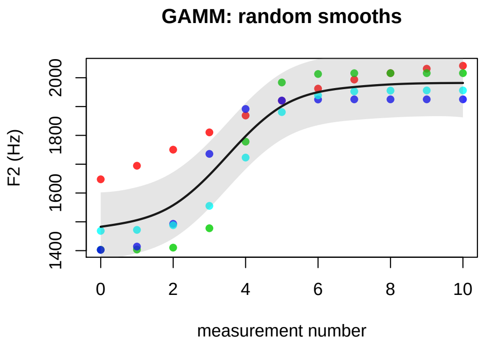
However, that is not the way our data set is structured. The 44 measurements actually come from only 4 vowels, and measurements taken from a single vowel are not independent of each other. Each of the data points is determined by a combination of factors: the underlying parameters, which trajectory it is in and what values the neighbouring points take on. Therefore, only part of the information in a given data point can be used towards estimating the underlying parameters – the rest of the information is actually about other stuff that we might not be interested in. An increase in the size of this data set wouldn't necessarily increase our confidence in our estimates of the underlying parameters. For instance, if we doubled the size of the data set by taking another 11 measurements along the *same* 4 trajectories at different time points, we wouldn't gain much additional information about the parameters that define the underlying curve – instead, we would gain more information about the individual trajectories themselves. If, on the other hand, we added measurements from 4 additional vowels, that would contribute more information towards the underlying parameters.

When we fit a regression model to our data set without random effects, we essentially ignore the grouping structure in the data: we pretend that the data set consists of independent measurements, each of them taken from a separate vowel. As a result, we will also erroneously assume that all the information in the individual data points can be used towards estimating the underlying parameters, even though some of the information is actually about other things like the separate trajectories. Since we think that we have more information that we actually do, we become overconfident about our estimates, which leads to anti-conservative results: p-values that are biased downwards and overly narrow confidence intervals. When the random effects are correctly specified, the model uses the right amount of information towards estimating the underlying parameters, and the overconfidence issue disappears.

We run into the same problem when we use random intercepts and slopes to fit straight lines to non-linear trajectories. The straight lines correctly 'soak up' some of the trajectory-specific information, but not all of it: they can't deal with non-linear dependencies, so those remain in the data. As a result, some of the information that we use towards estimating the underlying parameters will still be about the individual trajectories, so our model remains overconfident.

The three graphs below illustrate this point by plotting 95% confidence intervals for three models: one without random effects, one with random intercepts only and one with random smooths.





The confidence interval becomes slightly wider when we move from the no random effects model to the random intercepts one, and substantially wider for the random smooths model. Although the confidence interval for the final model may seem too wide, it is probably more accurate than the other two. After all, we need to bear in mind that this estimate is really only based on 4 vowels. For comparison, how confident would you be about a group mean based on 4 measurements?

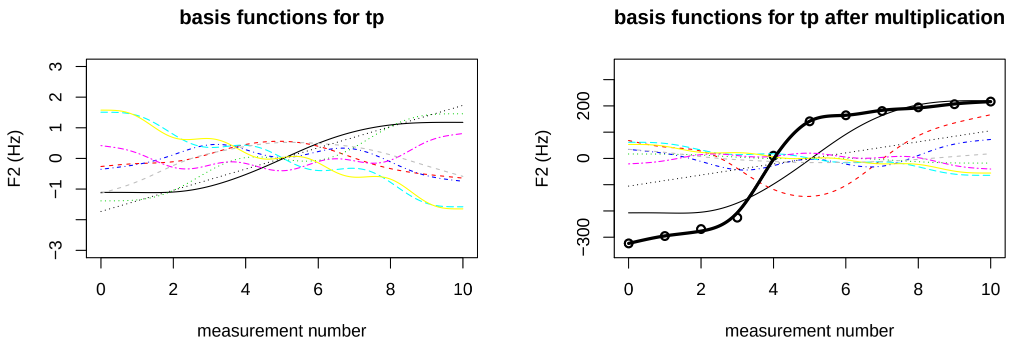
2.4 Different smooth classes

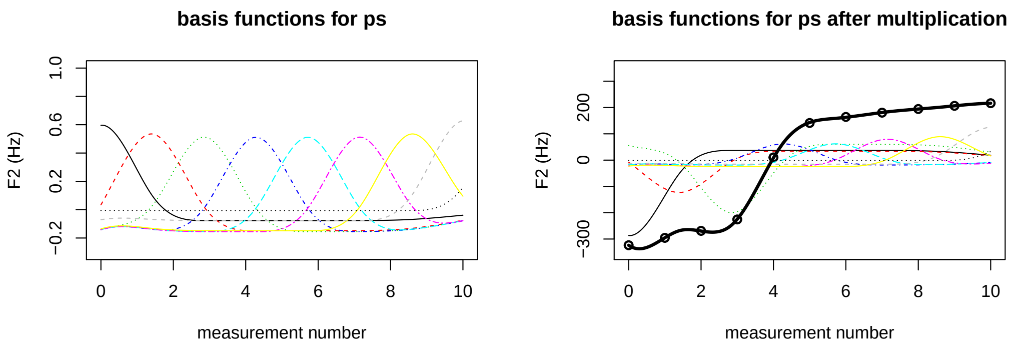
Now that we know what GAMMs are, how they are different from GAMs and why random smooths are necessary, it is time to introduce the concept of different smooth classes. We've actually already seen a range of different smooth classes in action, but only one of these has been mentioned explicitly: cubic regression splines. In what follows, we'll look at what distinguishes smooth classes from each other and consider a few examples.

Smooth classes are mainly defined by the basis functions used to generate the smoothers (and also by the type of *smoothing penalty* applied to them, but we won't discuss this). The graphs on page 5 show the basis functions for a smooth that belongs to the class of cubic regression splines. The models below exemplify two further smooth classes: thin plate regression splines (bs="tp") and P-splines (bs="ps").

```
demo.gam.tp <- gam(f2 ~ s(measurement.no, bs = "tp"), data = dat)
demo.gam.ps <- gam(f2 ~ s(measurement.no, bs = "ps"), data = dat)
```

While the exact differences among smooth classes are not so important for us, it is instructive to compare their basis functions and the fitted curves. The figures below show the basis functions for thin plate regression splines and P-splines before and after multiplication by the model coefficients, and should be compared with the cubic regression splines on page 5.





The basis functions can be very different (compare e.g. tp vs. ps), but the overall smooths are quite similar across the three models. P-splines seem to provide a slightly worse fit in this case, while cubic and thin plate regression splines do equally well. The default smooth in the GAMM packages that we'll be using is the thin plate regression spline.

The smooths that we've looked at so far are all univariate smooths: they fit a smooth line to the outcome variable as a function of a single predictor. However, it is possible to specify multivariate smooths as well, and most smooth classes are capable of representing such smooths without any issues. For instance, one might want to look at how vowel duration affects the shape of F2 trajectories for a given vowel. One way to do this is to specify a bivariate smooth, where one of the predictor variables is measurement.no (where the measurement was taken along the trajectory), and the other one duration (the overall duration of the vowel in seconds). The fitted bivariate smooth will be a two-dimensional surface, which essentially consists of trajectory shapes that vary smoothly as a function of overall duration. We'll see examples of multivariate smooths in section 3.

One final note about smooth classes. GAMMs can use smooths to represent not only fitted curves and surfaces, but also random intercepts and slopes. For instance, the linear mixed model with intercepts and slopes on page 8 can be specified in two different ways: using the traditional mixed model specification from the lme4 package (Bates et al., 2011), or using a GAMM model specification.

```
demo.slope.lmer <- lmer(f2 ~ measurement.no +
                              (1 + measurement.no | vowel),
                        data=dat.random)

demo.slope.gamm <- gam(f2 ~ measurement.no +
                            s(vowel, bs="re") +
                            s(vowel, measurement.no, bs="re"),
                       data=dat.random)
```

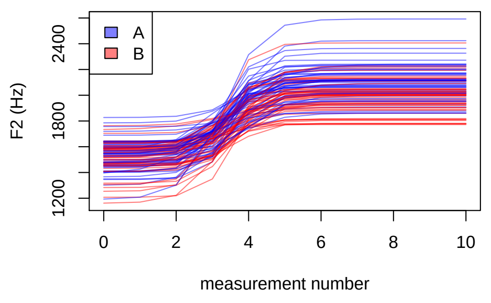
These formulae actually specify the same model, though the model fits may differ slightly across the two models due to differences in the way the parameters are estimated. Random intercepts and slopes are represented by the random effects (bs="re") smooth class in GAMMs, and they behave in much the same way as random effects in linear mixed effects models. Random smooths are a different type of construct, and they do not have a straightforward equivalent in linear models. The smooth class for random smooths is called *factor smooth interactions* (bs="fs"). Its use will be illustrated later in section 3.

2.5 Significance testing using GAMMs

2.5.1 Methods for significance testing with GAMMs

The models and data discussed in the previous section are useful in that they illustrate some of the basic concepts of GAMMs, but they are also a bit weird. There is only a single group of trajectories, and the shape of these trajectories does not vary as a function of any other predictors (although the individual F2 measurements do, of course, vary as a function of time). This type of situation does not typically arise in real examples: dynamic analyses of linguistic data are usually conducted with the goal of testing whether a given set of predictors has a significant effect on the trajectories under investigation. For instance, one might look at whether the shape of F2 trajectories is affected by vowel duration, whether pitch contours are different across questions and statements, or whether a diachronic change in spectral centre of gravity for a given fricative follows different patterns in different communities. While significance testing is relatively straightforward for linear models (though less so for linear mixed models), GAMMs offer a number of different ways of testing for significance. In this section, we briefly review these methods, and I outline a few recommendations from Sóskuthy (2016, prep) about how these should be used.

We start with a simple scenario. Let's say we have two words sharing the same diphthong, and we suspect that the realisation of the diphthong differs between the two words. We'll refer to the words as *A* and *B*. We collect dynamic F2 measurements for 50 tokens of each word. The two sets of 50 trajectories are shown below:



We use a GAMM of the following structure to test for significant differences between the two words (note that the main function is not `gam()` but `bam()` here; the difference between the two functions is explained in section 3.1 – but both of them are used to fit GAMMs):

```
demo.w.gamm <- bam(f2 ~ word +
                        s(measurement.no) +
                        s(measurement.no, by=word) +
                        s(measurement.no, traj, bs="fs", m=1),
                   data=dat.words, method="ML")
```

Let's go through this model specification quickly. The first predictor is a parametric term that captures overall differences in the height of the trajectories as a function of the word they come from. The second predictor, `s(measurement.no)`, corresponds to a single smooth fit at the reference value of the categorical predictor `word` (i.e. `A`). The third predictor is a so-called *difference smooth* that captures the difference between the trajectories for `A` and `B`. We will discuss difference smooths in more detail later. The last predictor corresponds to the random smooths by trajectory. Again, we'll say more about these later.

Confusingly, there are at least *six* different ways of testing whether the difference between *A* and *B* is significant (and probably more). Four of these are available as part of standard model summaries, and the other two can be performed by plotting confidence intervals.

Let's start with the model summary-based methods. Here is the model summary for `demo.words.gamm`:

```
summary(demo.w.gamm)

##
## Family: gaussian
## Link function: identity
##
## Formula:
## f2 ~ word + s(measurement.no) + s(measurement.no, by = word) +
##     s(measurement.no, traj, bs = "fs", m = 1)
##
## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1858.438      8.726 212.980  < 2e-16 ***
## word.L       -57.198     12.105  -4.725 3.58e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##                             edf  Ref.df       F p-value
## s(measurement.no)         8.936   8.953 275.315 < 2e-16 ***
## s(measurement.no):wordB   3.378   3.618   4.566 0.00272 **
## s(measurement.no,traj)  793.816 898.000  81.655 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.997   Deviance explained = 99.9%
## -ML = 5908.6  Scale est. = 240.07    n = 1100
```

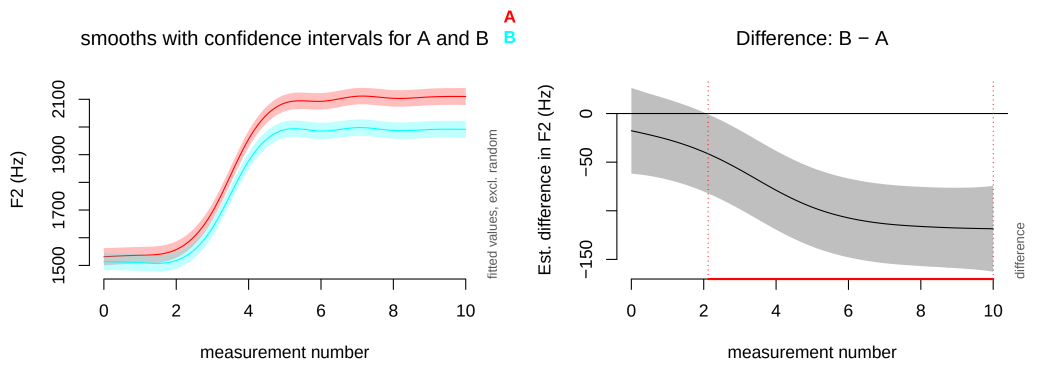
The initial section up to the first table should be familiar from other types of regression models. The `Parametric coefficients` table shows all non-smooth terms, that is, the intercept and the categorical predictor `word`. The next table (`Approximate significance of smooth terms`) summarises the smooth terms: the reference smooth, the difference smooth and the random smooths. Both the parametric and the smooth tables show p-values for the terms, although they are based on different tests: t-tests for parametric terms and ‘approximate’ F-tests for smooth terms. Wood (2006, 191) warns that the approximate p-values values for smooth terms should be taken with a pinch of salt: they can be anticonservative in certain cases.

So which p-values should we be looking at? First, we can look at the p-value for the parametric term `word` (method 1). Assuming an alpha-level of 0.05, this p-value is $< \alpha$, which means that there is a significant difference in the overall height of the two trajectories. We can also look at the p-value of the difference smooth, which, again, is < 0.05 – that is, there is a significant difference between the shapes of the two trajectories (method 2). Another option is to claim a significant difference between the two trajectories if either the parametric term or the difference smooth (or both) are significant (method 3). The last non-visual option is to set up another model which excludes the parametric term and the difference smooth, and compare this to the original model using the `compareML()` command, which is actually a form of `anova` (method 4):

```
demo.w.gamm2 <- bam(f2 ~ s(measurement.no) +
                      s(measurement.no, traj, bs="fs", m=1),
                    data=dat.words, method="ML")
compareML(demo.w.gamm, demo.w.gamm2, print.output=F)$table

##          Model    Score Edf  Chisq    Df   p.value Sig.
## 1 demo.w.gamm2 5923.394   5
## 2  demo.w.gamm 5908.614   8 14.780 3.000 1.707e-06  ***
```

According to the model comparison, the inclusion of the parametric term and the smooth difference term significantly improves the model fit. Importantly, these methods tell us little about the exact nature of the difference between the two words.

There are two visual methods for significance testing, both of which rely on confidence intervals. First, we can plot the predicted trajectories for both words with corresponding pointwise confidence intervals and check for overlap / lack of overlap at different points (method 5). Second, we can plot the difference smooth itself along with a confidence interval and check whether the confidence interval includes 0 at different points (method 6). One of the advantages of these methods is that they allow us to see where and in what way the trajectories words differ. These methods are illustrated below:



Both methods suggest that there is a significant difference between the two sets of trajectories. Moreover, they also reveal that the trajectories only diverge after measurement number 2 (that's the 3rd measurement out of 11). In other words, the trajectories differ significantly, but only between measurement numbers 2 and 10.

As a final note, significance testing with GAMMs can be substantially more complicated when the predictor of interest is a continuous variable. Although the general principles discussed in this section apply to continuous variables as well, their implementation can be a lot trickier due to the potential complexity of smooth interactions and constraints on the software packages used to fit GAMMs. We will go through a few worked examples later in the tutorial.

2.5.2 Recommendations

In Sóskuthy (2016) and Sóskuthy (prep), I present simulation-based results that reveal a wide range of variation in the rate of false positives and false negatives across the different methods discussed above. A point-by-point summary of these results is presented below:

- Significance testing based on the t/F-values for the parametric / smooth terms in the model summary is only justified when our predictions are directly about these individual terms. For example, if our prediction is that the average value of a trajectory will be higher in one condition than in another, but we have no predictions about trajectory shapes, we can safely rely on the p-value for the parametric term (method 1). Conversely, if our prediction relates to the shapes of the trajectories but does not concern their average value, we can use the p-value for the smooth term (method 2).

- Although predictions about the parametric / smooth terms do arise occasionally, most of the time researchers are interested in overall differences between trajectories regardless of whether those differences are in their average value or their shape. In such cases, it is tempting to declare significance if either the parametric term or the smooth term is significant (method 3). However, this leads to higher-than-nominal false positive rates (just below 0.10 at $\alpha = 0.05$). Model comparison where both the parametric *and* the smooth terms are excluded in the nested model (method 4) yields close-to-nominal false positive rates.

- In the simulations, the model comparisons never yielded significant results when neither the parametric, nor the smooth terms were significant on their own. Therefore, it may be safe to report a non-significant difference when neither of the terms in the model summary are significant (this can save a bit of time, as fitting nested models can take a while).

- Both visual methods (5 and 6) suffer from an anti-conservativity issue. If we report significant differences whenever there is *any* point where confidence intervals are non-overlapping (method 5) / the confidence interval for the estimated difference excludes 0 (method 6), the rate of false positives is too high (around 0.12 in the simulations at $\alpha = 0.05$). The rate of false positives decreases if we require significant differences at more than one point to report a significant overall difference, but this requires an arbitrary decision about the number of points, and can easily lead to results that are too conservative.

- Comparison based on whether there is overlap between two confidence intervals (method 5) is inadequate for significance testing, and graphs with separate trajectories (rather than a difference smooth) should only be used for graphical illustration. People tend to misinterpret the meaning of overlapping confidence intervals in such graphs. When the confidence intervals do not overlap, there is indeed a significant difference between the estimated quantities. However, we cannot make any conclusions about the significance of the difference when the confidence intervals do overlap: it may be significant but it may also be non-significant. Difference smooths (method 6) do not suffer from this problem, and should be the preferred option. Note that this issue is independent of the anti-conservativity problem described above.

- The simulations also reveal that failing to include random smooths-per-trajectory can lead to catastrophically high false positive rates (up to 0.6 at $\alpha = 0.05$) regardless of what method is used for significance testing. However, when there is a nested group structure, (e.g. 10 different words with 10 different trajectories each, where trajectories are nested within words), it is not absolutely necessary to include the lower-level smooths if the hypothesis we are testing relates to variation *between* levels of the higher-level grouping factor. For instance, let's say we are trying to see if there is a difference between the F2 trajectories for two vowels. The two vowels are each represented by 10 words, and each word is represented by 10 trajectories. In this case, it is necessary to include by-word smooths in the model, but not necessary to include by-trajectory smooths (though these are still recommended, as including them does lead to more accurate estimates).

Based on these results, the most reliable (i.e. least anti-conservative) option for significance testing is to first use an ANOVA (method 4) to see if there is an overall difference between groups of trajectories, and then look at difference smooths (method 6) to identify where the difference lies along the trajectory. Additionally, it is also useful to plot individual smooths (with or without confidence intervals) to provide a visual summary of the main trends in the data set, but these should not be used for significance testing.

3 A GAMM tutorial

In this tutorial, we will first discuss the different GAMM toolkits that are currently available, and then we will work through two detailed examples. The first of these is based on simulated data, while the second one uses a real data set taken from Stuart-Smith et al. (2015).

3.1 R packages and functions for fitting GAMMs

Two different packages are used for fitting GAMMs in this tutorial: `mgcv` (Wood, 2006) and `gamm4` (Wood & Scheipl, 2014). In addition, the package `itsadug` (van Rij et al., 2016) is used for plotting smooth trajectories. `mgcv` and `gamm4` actually provide four different functions for fitting GAMMs: `gam()`, `bam()`, `gamm()` (all three from `mgcv`) and `gamm4()` (perhaps not so surprisingly from the package `gamm4`). These functions have slightly different strength and weaknesses, so it's worth discussing them in a bit more detail before we start working with them. Some of the discussion below is fairly technical, and probably makes more sense to readers who already have a bit of experience with GAMMs. I have included it here as I think it may be useful for reference, but readers should feel free to skip to the actual tutorial.

`gam()`/`bam()` are the best documented and most reliable tools for fitting GAMMs, and most existing tutorials focus on these functions. Although these are two separate functions, they are pretty much identical in terms of their use, and produce very similar results. However, `bam()` can be much faster than `gam()` and it uses less memory, so it is the preferred option for large or complex data sets. `gam()`/`bam()` are also more versatile than `gamm()` and `gamm4()`, though this versatility comes at a price: they can be much slower and more memory intensive than the latter two functions when the models contain random smooths with many levels (which is often the case when analysing dynamic speech data). The package `itsadug` has been developed primarily with `gam()` in mind, and many of its functions do not work properly with models fitted using `gamm()` or `gamm4()`. Here is a brief list of the main features of `gam()` that are relevant for us (don't worry if some of these features are unclear at this point; many of them will be discussed soon):

- can fit all smooth types, including traditional random effects and random smooths
- can fit crossed random smooths (e.g. random smooths by words and by speakers at the same time)
- fully compatible with `itsadug`
- can fit smooth interactions (`te()` and `ti()`), where the main effects and interaction terms are separable
- possible to compare models using `anova()` / `compareML()`
- possible to model simple autocorrelation in the data when using `bam()` (but not `gam()`)

`gamm()` is superficially similar to `gam()/bam()`, but it relies on an external package (`nlme`) for estimating models. In practical terms, this means that `gamm()` can be faster than `gam()/bam()` when the model includes randoms smooths with many levels. In addition, `gamm()` differs from `gam()/bam()` in the following ways:

- cannot fit crossed random smooths
- only partly compatible with `itsadug`
- not possible to compare models using `anova()`
- can include complex models of autocorrelation
- can deal with heteroscedascity in the data by using variance components

We won't use this specific function in this tutorial, as model comparison through `anova()` and potentially crossed random smooths are essential for us, while autocorrelation and heteroscedascity are issues that are somewhat less relevant for the type of data that we will be looking at.

`gamm4()` is in many ways similar to `gamm()`: it mostly uses the same syntax as `gam()/bam()`, but performs model fitting with the help of the `lme4` package. Like `gamm()`, `gamm4()` is good at fitting models with random smooths, and possibly even faster than `gamm()`. It also comes with its specific set of pros and cons compared to `gam()/bam()` and `gamm()`:

- can fit crossed random smooths
- only partly compatible with `itsadug`
- possible to compare models using `anova()`
- cannot deal with autocorrelation or heteroscedascity
- can only fit smooth interactions where the main effects and interaction terms are inseparable (so their significance can't be evaluated separately)

For large data sets with complex random structures, `gamm4()` might be the only option, though it is still in development, and can be quite unstable. We will use this function as an alternative to `gam()/bam()` in the tutorial, and briefly explore its advantages and limitations.

3.2 Analysing a simple simulated data set

The first data set contains simulated F2 trajectories, and is very similar to the example data set introduced in section 2.5.1. It contains 50 F2 trajectories, each of them represented by 11 measurements taken at equal intervals (at 0%, 10%, 20%, . . . , 100%). The observations in the data set are the individual measurements, which means that there are 550 data points altogether. The variable `measurement.no` codes the location of individual data points along the trajectory. Each of the trajectories has an ID (a number between 1–50), which is encoded in the column `traj`. The trajectories represent two different words: 25 of them come from word *A* and 25 of them from word *B*. This is encoded by the `word` variable. The underlying curves that served as the basis of the simulated trajectories overlap at the beginning, but are different by about 100 Hz near the end. There is an additional variable termed `duration`, which stands for overall vowel duration measured in seconds. The simulation was set up so that long vowels have slightly wider trajectories than short vowels. The data set is called `words.50` and can be accessed at the following address:

`http://www-users.york.ac.uk/~ms1341/gam_intro_data.zip`

Here's a small sample of the data:

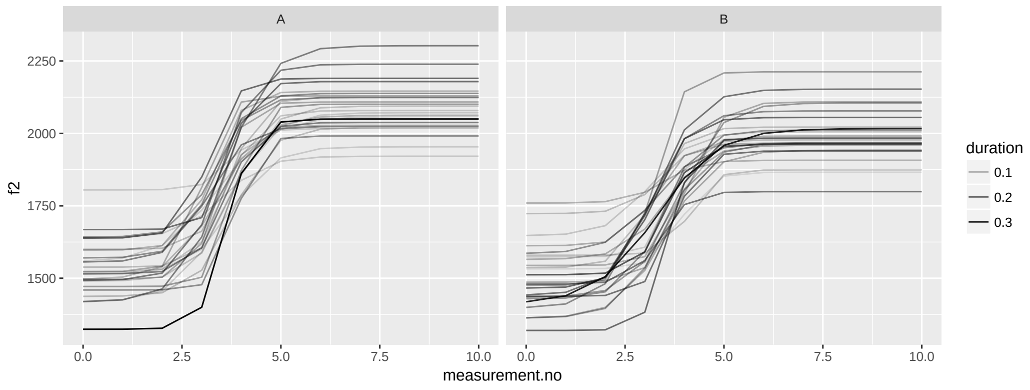
```
head(words.50)

##   traj word measurement.no       f2  duration
## 1    1    A              0 1642.761 0.1378182
## 2    1    A              1 1644.162 0.1378182
## 3    1    A              2 1659.948 0.1378182
## 4    1    A              3 1788.793 0.1378182
## 5    1    A              4 2044.977 0.1378182
## 6    1    A              5 2115.984 0.1378182
```

Let's import the libraries that we'll be using.

```
library(ggplot2)
library(mgcv)
library(itsadug)
library(gamm4)
```

First of all, let's create a simple plot to see what the raw data look like. This is good practice regardless of the type of model one is planning to fit, but it is especially useful for GAMMs, where the shape of the trajectories has implications for the model fitting procedure (e.g. choosing the maximum number of basis functions). We'll use the package `ggplot2` to create the plot, as it makes it really easy to show the structure of the data set through colours and other devices. The trajectories representing the two words are shown in separate panels and the overall duration of each trajectory is indicated by shading: longer trajectories are darker (the trajectories are time-normalised, so they all have the same length along the x-axis). Since this tutorial is not about `ggplot2`, we won't discuss the code below.

```
ggplot(words.50, aes(x=measurement.no, y=f2, group=traj,
                     alpha=duration)) +
  facet_grid(~word) + geom_line()
```



So what do we want to capture in our model? First, we want to fit separate smooths to the two trajectories, and we'll want to use model comparison and difference smooths to see whether they are different. We also want to include some type of interaction between duration and the shape of the trajectories. Finally, we want to include random smooths by trajectory in order to avoid false positives and obtain more accurate estimates of the underlying curves.

Let's start with a very simple model that fits separate smooths to the two words. The best way to do this is to fit one smooth to word *A* and then another smooth that represents the difference between *A* and *B*. In terms of the final model fit, this is identical to simply fitting two separate smooths to the two trajectories. However, it is easier to interpret the model output when a difference smooth is included, as it tells us directly whether there is a significant difference between the shapes of the two trajectories. Both types of models are shown below along with relevant bits of the model summary. The models use cubic regression splines (`bs="cr"`) with the default number of knots (`k=10`). Using different basis functions does not substantially alter the results.

```
# model with separate smooths

words.50$word <- as.factor(words.50$word)
words.50.gam.sep <- gam(f2 ~ word + s(measurement.no, by=word, bs="cr"),
                        data=words.50, method="ML")
summary.coefs(words.50.gam.sep)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1885.149      5.755 327.592  < 2e-16 ***
## wordB        -67.372      8.138  -8.279 1.01e-15 ***
##
## Approximate significance of smooth terms:
##                            edf Ref.df     F p-value
## s(measurement.no):wordA 7.319  8.263 212.9  <2e-16 ***
## s(measurement.no):wordB 6.840  7.878 169.3  <2e-16 ***

# model with smooth for A & difference smooth
words.50$word <- as.ordered(words.50$word)
words.50.gam.diff <- gam(f2 ~ word + s(measurement.no, bs="cr") +
                              s(measurement.no, by=word, bs="cr"),
                         data=words.50, method="ML")
summary.coefs(words.50.gam.diff)

## Parametric coefficients:
```

```
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1851.463      4.055 456.576  < 2e-16 ***
## word.L       -47.639      5.735  -8.307 7.98e-16 ***
##
## Approximate significance of smooth terms:
##                            edf Ref.df      F p-value
## s(measurement.no)        7.811  8.597 233.63 < 2e-16 ***
## s(measurement.no):wordB 1.067  1.131  10.47 0.00114 **
```

`summary.coefs()` (a function from `gamm_hacks.r`) is used to save space by excluding parts of the model summary. Readers are encouraged to use the standard `summary()` function with GAMMs, as it includes some additional useful information about the model fit. Models with difference smooths require two things. First, the categorical grouping variable for the words needs to be converted to a so-called 'ordered factor' using `as.ordered()`. Second, the model formula needs to include (i) a parametric term for `word`,[5] (ii) a smooth over `measurement.no` without any grouping specification and (iii) a smooth over `measurement.no` with the grouping specification `by=word`. In the first model summary, the separate smooths simply represent the two different words. The significance values in the model summary refer to the individual terms: they suggest that both curves are significantly different from 0 (i.e. not simply flat lines). However, they do not tell us anything about the difference between the two terms. They could both be significant even if the two underlying curves were exactly the same. In the second model summary, the term `s(measurement.no)` represents the *reference smooth*, that is, a curve fit to trajectories at the reference level of the factor `word` (i.e. *A*). The term `s(measurement.no):wordB` represents the difference smooth, that is, the difference between the trajectories for *A* and *B*. The fact that this term and the parametric term for `word` are both significant suggests that the trajectories for the two words are indeed different. In order to confirm this, we can compare a model including the difference smooth and a model without the difference smooth using ANOVA.[6]

```
# fitting a nested model without the difference smooth
words.50.gam.diff.0 <- gam(f2 ~ s(measurement.no, bs="cr"),
                           data=words.50, method="ML")

# model comparison using the compareML() function from itsadug
# this is very similar to the anova() function, but better suited
# to models fitted using gam(); some parts of the output are suppressed
compareML(words.50.gam.diff, words.50.gam.diff.0, print.output=F)$table

##                 Model    Score Edf  Chisq    Df   p.value Sig.
## 1 words.50.gam.diff.0 3334.192   3
## 2   words.50.gam.diff 3296.427   6 37.765 3.000 2.789e-16  ***
```

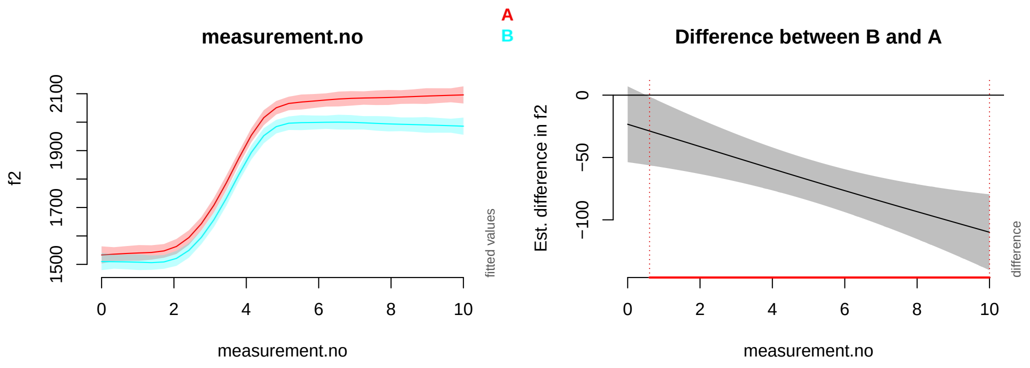
The model comparison suggests that the inclusion of the difference smooth improves the model fit significantly. Note that both the smooth and the parametric terms representing `word` were excluded at the same time.

[5]This parametric term needs to be included in both models. Otherwise, the models could not capture overall differences in F2 and would (incorrectly) force both fitted smooths to have the same average value over the trajectory.

[6]The `method="ML"` option is strongly recommended for performing model comparisons between models with different fixed effects, and usually results in more stable fits anyway (see e.g. Wood 2006; Zuur et al. 2009). See below for more detail.

Let's create a plot of the model predictions and the difference smooth. These can be generated using the plot_smooth and the plot_diff functions from itsadug.

```r
plot_smooth(words.50.gam.diff, view="measurement.no", plot_all="word", rug=F)
plot_diff(words.50.gam.diff, view="measurement.no",
          comp=list(word=c("B","A")))
```



The view option determines what variable to show along the x-axis; the plot_all = "word" option tells R to plot predictions separately for each value of the factor word; and the comp option specifies the levels of word that the difference smooth is based on. In this case, the difference smooth shows $B - A$. Note that the confidence interval for the difference smooth seems a bit too narrow: the underlying curves were specified in such a way that there is no actual difference between the curves until about measurement.no 2, but the difference smooth shows a significant difference along almost the entire trajectory.

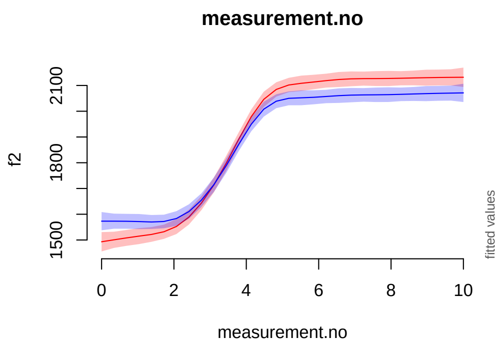
As a second step, let's try to account for the influence of duration on the trajectories. Simply including it in the model as a parametric term or even as a smooth on its own won't work: its effect is not on average F2 values, but on the shapes of the trajectories. So what we really want is a non-linear interaction between duration and the smooths for measurement.no. Moreover, it would be useful if we could separate this interaction term from the main effects of duration and measurement.no. The solution is to use so-called 'tensor product interactions'. Although the name is quite intimidating, these work very similarly to interactions in linear models. All we need to do is include three terms: s(measurement.no) (a smooth for the main effect of measurement.no), s(duration) (main effect of duration) and ti(measurement.no, duration) (the interaction between the two variables).

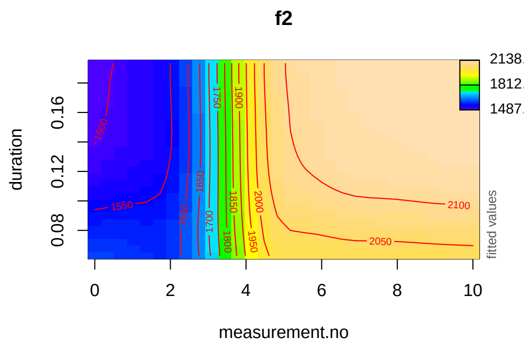
```
words.50.gam.dur <- gam(f2 ~ word + s(measurement.no, bs="cr") +
                              s(duration, bs="cr") +
                              ti(measurement.no, duration) +
                              s(measurement.no, by=word, bs="cr"),
                         data=words.50, method="ML")
summary.coefs(words.50.gam.dur)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 1851.463      3.718 498.006   <2e-16 ***
## word.L       -51.380      5.379  -9.552   <2e-16 ***
##
## Approximate significance of smooth terms:
##                               edf Ref.df       F  p-value
## s(measurement.no)           7.964  8.684 267.314  < 2e-16 ***
## s(duration)                 3.455  4.150   7.650 4.06e-06 ***
## ti(measurement.no,duration) 6.414  8.557   8.222 5.99e-11 ***
## s(measurement.no):wordB     1.681  2.087  10.171 4.10e-05 ***
```

The interaction term is significant according to the model summary. If we wanted to run a more rigorous test, we could fit a nested model where both smooths that include `duration` are dropped, and then compare it to the full model using `compareML()`.

Let's plot this interaction. There are two ways to do this. First, we can plot smooths at a few different values of `duration`. Or, alternatively, we can plot the surface that represents the interaction between `duration` and `measurement.no` using colours (with warmer colours representing higher values). Both options are shown below:

```
plot_smooth(words.50.gam.dur, view="measurement.no", cond=list(duration=0.16),
            rug=F, col="red")
plot_smooth(words.50.gam.dur, view="measurement.no", cond=list(duration=0.08),
            rug=F, col="blue", add=T)
fvisgam(words.50.gam.dur, view=c("measurement.no","duration"),
        ylim=quantile(words.50$duration, c(0.1, 0.9)))
```





The function `plot_smooth` needs to be run twice to generate the separate smooths: once to create the plotting area and a smooth for longer trajectories (with a duration of 0.16 s) and once to add another smooth for shorter trajectories (with a duration of 0.08 s). Adding a smooth to an existing plot is done by including the `add=T` option. In this case, the plot on the right is somewhat less informative, as the difference between trajectories with different durations is relatively small. Nonetheless, it does show that for shorter trajectories, the beginning of the trajectory is higher and the end is lower. It is often worth inspecting both types of graphs to get a better sense of what the model fit actually looks like.

As a final step, we'll add random structures to the model and perform model comparison to see what type of random structure is best suited to our data set. We'll try three different options: (i) random intercepts only, (ii) random intercepts plus slopes and (iii) random smooths. The code for fitting these models is shown below (the first model is shown in full, but only relevant lines are shown for the second and third models to save space). The last model may take a while to fit, so be prepared to wait for a few minutes.

```
# random intercepts only
words.50.gam.int <- gam(f2 ~ word + s(measurement.no, bs="cr") +
                             s(duration, bs="cr") +
                             ti(measurement.no, duration) +
                             s(measurement.no, by=word, bs="cr") +
                             s(traj, bs="re"),
                        data=words.50, method="REML")

# random intercepts + slopes
words.50.gam.slope <- gam(f2 ~ ...
                              s(traj, bs="re") +
                              s(traj, measurement.no, bs="re"),
                          data=words.50, method="REML")

# random smooths
words.50.gam.smooth <- gam(f2 ~ ...
                           s(measurement.no, by=word, bs="cr") +
                           s(measurement.no, traj, bs="fs", m=1, k=5, xt="cr"),
                           data=words.50, method="REML")
```

Random intercepts are coded by including a smooth over the grouping variable with the smoothing class specified as `bs="re"`. This is really a technicality: `gam` is able to use the mathematics of smooths to estimate various random structures, and this is reflected in its syntax as well. Random slopes are coded by adding the slope variable (`duration`) after the grouping variable (`traj`) inside the smooth, and keeping the smoothing class as `bs="re"`. This is a bit confusing, since random smooths have the opposite syntax: here, the continuous variable comes first, followed by the grouping variable. For random smooths, we also have to change the smoothing class to `bs="fs"`, which is short for 'factor smooth interactions'. Although the rest of the options above are optional, they can be quite useful. The `m=1` specification is recommended in several papers (e.g. Baayen et al. 2016) for random smooths; what it does is slightly change the way the smoothing penalty is estimated (the default value is 2). The `xt="cr"` option sets the smooth class for the individual random smooths to cubic regression splines – this may not be necessary, but my impression was that cubic regression splines did a better job at capturing the current vowel trajectories. Finally, `k=5` sets a relatively low upper limit on the wiggliness of the individual random smooths. Although a higher number might result in a marginally better fit, this model already takes a long time to fit, and increasing `k` for random smooths can drastically increase the amount of resources (memory and time) needed to fit GAMMs.

The estimation method for the three models above is set to `method="REML"`; this is usually not necessary, and my general recommendation is to use `method="ML"`, as before. REML is needed in this case since we want to compare models with the same fixed effects, but different random effects. If the comparison was between models with different fixed effects but the same random effects (the typical case), the models would have to be estimated using `method="ML"` (see also Zuur et al. 2009).

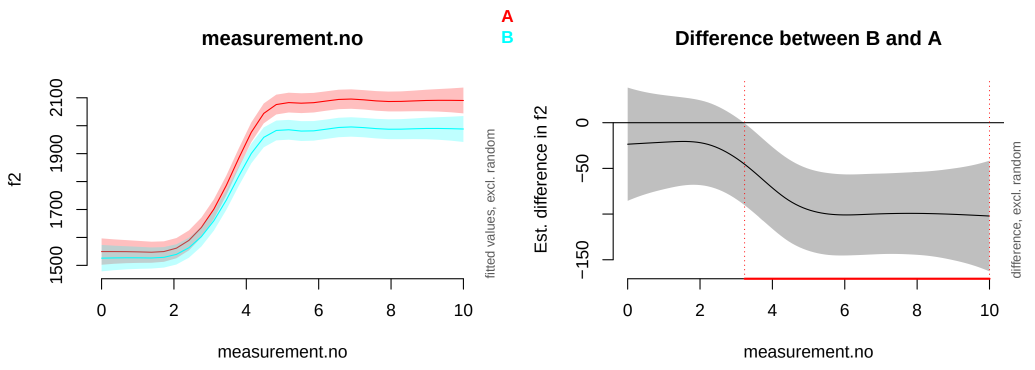
In order to find out which model fits the data best, we'll use a statistic called AIC (Akaike Information Criterion). AIC is a combination of two quantities: how surprising the data are given our fitted model (the lower this number, the better the fit) and how many parameters are used in the model. That is, AIC penalizes both bad model fits and unnecessary model complexity. When comparing two models, the one with a lower AIC should be preferred. Here are the AIC values for the three models above:

```
AIC(words.50.gam.int, words.50.gam.slope, words.50.gam.smooth)

##                            df      AIC
## words.50.gam.int     71.53459 6176.776
## words.50.gam.slope  118.84113 5638.150
## words.50.gam.smooth 215.39176 5275.282
```

Based on these values, the model with random smooths is a clear winner: the added model complexity is more than compensated for by the improvement in model fit.

Let's recreate the model prediction plots that we previously generated for the simple model without random smooths. The plots below show predicted trajectories for words *A* and *B* (left), and a difference smooth for $B - A$.

```
plot_smooth(words.50.gam.smooth, view="measurement.no", plot_all="word", rug=F,
            rm.ranef=T)
plot_diff(words.50.gam.smooth, view="measurement.no",
          comp=list(word=c("B","A")), rm.ranef=T)
```



This is very similar to what we did for the simple model, but both functions include an additional option: `rm.ranef=T`. This option ensures that the predictions are shown for words *A* vs. *B* in general. If this option was left out, the predictions would relate to a given value of the grouping factor for the random effect, that is, `traj`. In other words, the plot would show predictions for a specific trajectory. But instead of just plotting that single trajectory, R would attempt to figure out what that trajectory would look like if it belonged to word *A* and what it would look like if it belonged to word *B*. Since a given trajectory can only belong to one word, this can lead to results that are difficult to interpret and unreliable confidence intervals.

As expected, the confidence intervals for the model with random smooths are much wider than they are for the simple model (cf. the earlier graphs). Moreover, the shape of the difference smooth changes considerably between the two models: it looks more non-linear for the current model, and the confidence interval includes 0 until about `measurement.no` 3. Since the initial sections of the underlying curves for the two words are identical, this is a clear improvement on the previous model.

Since the last model (`words.50.gam.smooth`) correctly captures all the main trends in the data, we do not have to add any additional components. The model fit can be improved slightly by raising the value of `k`, but this also makes the fitting procedure more resource intensive. Note that there is still a small issue with the fitted model: although the initial and final sections of the raw trajectories are completely straight, the fitted trajectories are actually a bit wavy. This likely results from the shape and number of basis functions used for the smooths. This unsupported waviness introduces weak correlations in the residuals of the model, which can be captured by using autocorrelation structures. These will not be discussed in the current tutorial, but many of the suggested readings at the end provide advice on using them.

How about fitting this model with `bam()`, `gamm()` or `gamm4()`? As explained in section 3.1, these function very similarly to `gam()`, but can be faster for complicated models with a lot of random smooths. On my computer, R took about 46 s to fit the model with random smooths using `gam()` (with the option `method="ML"`, which is what users will typically want). `bam()` (with the default `method`, i.e. fREML) took only 3 s to fit the same model using the same syntax. `gamm()` was even faster: it only took 1.4 s (using the same syntax, with `method="ML"`). The model fits for these three variants are nearly identical.

Since `ti()` type interactions do not work with `gamm4()`, we need to change our model slightly if we want to use `gamm4()`. An alternative to `ti()` is to use the simpler `s()` construct instead.[7] The main drawback of `s()` is that it has trouble dealing with interactions when the component terms are on different scales, so we need to standardize our predictors first.

```
# standardising predictors
words.50$measurement.no.s <- scale(words.50$measurement.no)
words.50$duration.s <- scale(words.50$duration)

# fitting model using gamm4
system.time(
words.50.gamm4 <- gamm4(f2 ~ word + s(measurement.no.s, bs="cr") +
                        s(duration.s, bs="cr") +
                        s(measurement.no.s, duration.s) +
                        s(measurement.no.s, by=word, bs="cr") +
                        s(measurement.no.s, traj, bs="fs", xt="cr", m=1, k=5),
                        data=words.50, REML=F)
)

##    user  system elapsed
##  17.206   0.068  17.314
```

Once again, the model fit is nearly identical to those produced by the three other functions. Since the options available in `gamm4` differ slightly from those in `gam()`/`bam()` and

[7]Theoretically, it is possible to use an alternative construct named `t2()` instead of `ti()`, but this doesn't seem to be supported in models that include random smooths.

gamm(), we need to use the option REML=F instead of method="ML". For this specific model, gamm4() is actually pretty slow: it took about 16 s to fit the model. However, gamm4() can be much faster than other methods for models with a complicated random smooths.

The output of gamm() and gamm4() is different from that of gam()/bam(), so we cannot simply use the same syntax for getting model summaries, running model comparisons and plotting. The code below shows how these tasks can be performed for gamm() and gamm4(). Since the plot_diff() function from itsadug cannot handle models fit using gamm() or gamm4(), we need to load a hacked version of this function from gamm_hacks.r. This script needs to be sourced after loading itsadug.[8]

```
# model summary for gamm / gamm4 (same for both functions)
summary(words.50.gamm4$gam)

# model comparison for gamm
# (remember to use ML for fixed eff comp, REML for random eff comp)
anova(words.50.gamm$lme, words.50.gamm.0$lme)

# model comparison for gamm4
anova(words.50.gamm4$mer, words.50.gamm4.0$mer)

# plotting (same for gamm)
# (rug=T doesn't work; ylab needs to be specified for plot_smooth)
plot_smooth(words.50.gamm4$gam, view="measurement.no.s", plot_all="word",
            rug=F, rm.ranef=T, ylab="f2")
source("gamm_hacks.r")
plot_diff(words.50.gamm4$gam, view="measurement.no.s",
          comp=list(word=c("B","A")), rm.ranef=T)
```

4 Analysing data from Stuart-Smith et al. (2015)

We've applied GAMMs to various simulated data sets, but we haven't yet looked at any real data. In this section, we will use the methods introduced above to analyse a subset of the data presented in Stuart-Smith et al. (2015). This data set contains dynamic F3 measurements of word-final /r/ in spontaneous recordings of Glaswegian. The speakers are older males recorded at four different time points between 1970 and 2000. The data set is a fairly typical example of dynamic speech data. It is thoroughly unbalanced with varying numbers of tokens across speakers, decades, words and environments. Moreover, it likely shows the influence of a large number of different variables, though only a few of these are of interest for our present purposes. It is also larger than the toy data sets that we've worked with so far: the subset that we will look at contains 420 individual trajectories.

GAMMs can get computationally very complex, and may be impossible to fit within a reasonable time frame using an ordinary computer. When dealing with complex data sets like the current one, we are inevitably forced to try out a range of different solutions and make certain compromises: it is almost guaranteed that we won't be able to fit the model that seems most appropriate. In this section, I will suggest some methods for

[8]Some of the models in this code chunk are not included in the tutorial, so the model comparisons won't work.

testing different modelling strategies in cases where individual models take a long time to fit. I will also lay out the different options and explain how I decided on a specific solution. The 'final' model that we will arrive at is not, by any means, the only possible solution and probably not even the best one – but at least it converges and provides sensible results.

The data set consists of F3 trajectories measured at 11 evenly spaced points. The trajectories include both /r/ and the preceding vowel. The data come from four sets of three speakers recorded in the 1970s, the 1980s, the 1990s and the 2000s (i.e. 12 speakers altogether). The following variables will be used in the analysis:

- `measurement.no`: see above
- `duration`: overall duration of the vowel + /r/ sequence
- `decade`: decade of recording coded as a continuous variable
- `stress`: whether the previous vowel is an unstressed schwa or a full vowel
- `traj`: a grouping variable by trajectory
- `speaker`: a grouping variable by speaker

The main question that we'll try to answer is whether the acoustic characteristics of final /r/ have changed over time. That is, we'll be looking for an effect of `decade` on the F3 trajectories. However, there are a few complicating factors. First, the trajectories vary quite substantially in terms of their duration, and this is likely to affect their shapes: we might expect to see flatter trajectories for shorter vowel + /r/ sequences. Second, the vowels in the vowel + /r/ sequences are not always the same, but their distribution is unbalanced: about 65% of all the vowels are schwas. Since the quality of the vowel may affect the F3 trajectory (though F3 is not expected to vary as much as F1 or F2 would), it is important to bring this factor into the analysis. To keep things simple, the quality of the vowel is encoded by the `stress` variable, which splits the data set according to whether the vowel is an unstressed schwa or a full vowel. We may also expect that stressed vs. unstressed vowels will change differently, though we have no clear prediction about what such a difference would look like.

Before we start analysing the data, there is one further important point that should be discussed. Although the analysis presented here will be shown in a relatively streamlined form, the model fitting procedure was actually preceded by a detailed exploration of the data through various plots. These plots won't be shown here, but many of the analytical decisions below are actually based on observations that emerged from this exploratory analysis. In my experience, this type of data exploration is absolutely crucial, and there is almost no point in starting to fit GAMMs until we get a sense of the range of variation in the trajectories that we are trying to model (unless the data and the models are exceedingly simple).

To avoid starting with a very complex model, let's first simply try to capture the effect of `decade` on the trajectories. Since `decade` only varies across speakers, it's essential to include `speaker` as a random smooth in the model. Otherwise the estimated effect of `decade` will be based on the assumption that there are 420 independent data points where, in reality, there are really only about 12. We'll start with the `gam()` function and use ML. We'll also record and display the amount of time it takes to fit the model using the `system.time()` function. This function displays three timing values. The last one of these shows how long it takes to fit the model overall; the other two are less relevant.

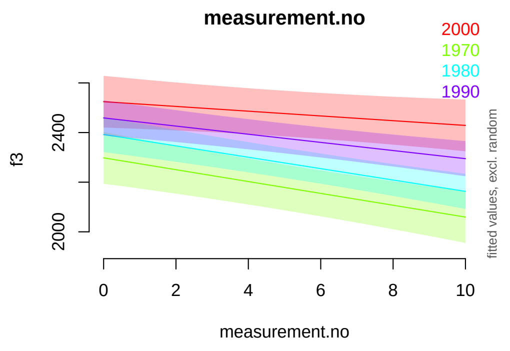
```
gl.r <- read.csv("glasgow_r.csv")
gl.r.gamm.simple.t <- system.time(
  gl.r.gamm.simple <- gam(f3 ~ s(measurement.no) +
                             s(decade, k=4) +
                             ti(measurement.no, decade, k=c(10,4)) +
                             s(measurement.no, speaker, bs="fs", m=1, k=4),
                        dat=gl.r, method="ML")
)
gl.r.gamm.simple.t

##    user  system elapsed
##  12.145   0.239  12.412

summary.coefs(gl.r.gamm.simple)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2327.38      28.05   82.97   <2e-16 ***
##
## Approximate significance of smooth terms:
##                               edf Ref.df      F  p-value
## s(measurement.no)           1.008  1.012 44.053 3.01e-11 ***
## s(decade)                   1.005  1.005 15.625 7.88e-05 ***
## ti(measurement.no,decade)   1.633  1.688  3.261   0.0884 .
## s(measurement.no,speaker) 31.250 46.000 23.284  < 2e-16 ***
```

There are a few things to note here. `k` has to be set to 4 (or even lower) for `s(decade)`, since it only has 4 values. This also holds for the `ti()` interaction term, where `k` needs to be set separately for the two main terms. `k` is also set to 4 for the by-speaker random smooths, partly because the raw data don't show that much wiggliness and partly because we will eventually want to include some further random smooths, so it's worth keeping things reasonably simple.

Here's a plot illustrating the effect of `decade` on the trajectories:[9]

```
source("gamm_hacks.r")
plot_smooth.cont(gl.r.gamm.simple, view="measurement.no", plot_all.c="decade",
                 rug=F, rm.ranef=T)
```

[9]A slightly modified version of `plot_smooth()` is used to keep things simple. This function can be loaded by sourcing the file `gamm_hacks.r`.



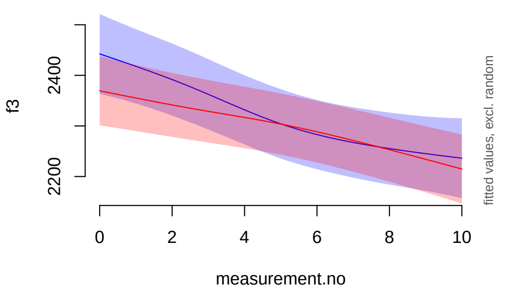
We won't run proper hypothesis testing on this model, as a few things are still missing, but `decade` does seem to have a significant effect on average F3 and possibly even the slope of the trajectories. Interestingly, the predicted trajectories come out as completely straight (this can also be read off the model summary, where the EDF for all three fixed smooth terms is close to 1).

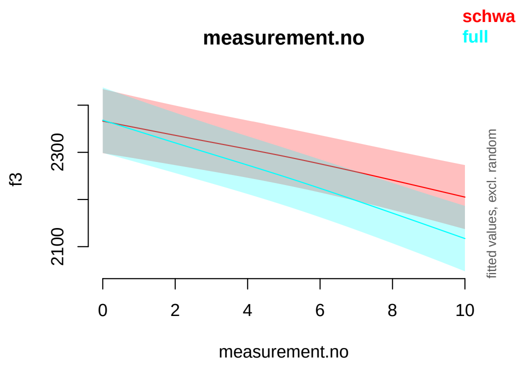
Next, we'll add two further variables: `duration` and `stress`. These two variables actually show a slight correlation, as unstressed schwas are shorter on average than full vowels, but there is sufficient overlap between the `duration` values for the two `stress` groups to estimate these effects separately. We'll add a main smooth term for `duration` as well as a `ti()` interaction with `measurement.no`. The predictor `stress` is included as a parametric term and a difference smooth (using `by=stress`).

```
gl.r$stress <- as.ordered(gl.r$stress)
gl.r.gamm.covs.t <- system.time(
  gl.r.gamm.covs <- gam(f3 ~ stress +
                             s(measurement.no) + s(measurement.no, by=stress) +
                             s(duration) + ti(measurement.no, duration) +
                             s(decade, k=4) +
                             ti(measurement.no, decade, k=c(10,4)) +
                             s(measurement.no, speaker, bs="fs", m=1, k=4),
                        dat=gl.r, method="ML")
)
gl.r.gamm.covs.t

##    user  system elapsed
##  17.193   0.264  17.468
```

This model already takes a few seconds to fit on a relatively fast computer. To save space, the model summary is not shown, but here's a graphical summary of the effects of `duration` and `stress`.

```
plot_smooth(gl.r.gamm.covs, view="measurement.no", cond=list(duration=0.3),
            rug=F, rm.ranef=T, col="blue")
plot_smooth(gl.r.gamm.covs, view="measurement.no", cond=list(duration=0.1),
            rug=F, rm.ranef=T, col="red", add=T)
plot_smooth(gl.r.gamm.covs, view="measurement.no", plot_all="stress",
                 rug=F, rm.ranef=T)
```





The F3 trajectories start lower for shorter /Vr/ sequences and are a bit flatter (the red line represents a short trajectory). This suggests that short vowels are more strongly coarticulated with the following /r/. Moreover, there is a more pronounced dip near the /r/ part of the sequence for sequences with full vowels. Again, the unstressed sequence shows a flatter trajectory. One possible interpretation is that /r/ is more likely to be weakened or deleted after unstressed vowels.

Finally, let's see whether `stress` interacts with the effect of `decade`. Two further `by` terms need to be added: one for the `decade` main term and another one for the `ti()` interaction between `measurement.no` and `decade`. Only the added terms are shown in the code chunk below.

```
gl.r.gamm.intr <- gam(f3 ~ stress +
                           ...
                           s(decade, k=4, by=stress) +
                           ti(measurement.no, decade, k=c(10,4), by=stress),
                        dat=gl.r, method="ML")

##   user  system elapsed
## 62.137   1.53   63.333
```

The summary for this model (not shown above) suggests that `stress` and `decade` might interact significantly, though the difference between stressed and unstressed vowels does not seem particularly pronounced. Unfortunately, this model cannot be used as it is: the effect of `stress` should really be estimated from differences across trajectories, not individual data points. Since the model does not include by-trajectory random smooths, it is not aware of the dependence between data points from the same trajectory. However, this model already takes about a minute to fit on my computer, and adding a random smooth with 420 levels will almost certainly result in a model that (i) takes forever to converge and (ii) needs more memory than I have.

One strategy than can be used in situations like this is to try out different methods on a small subset of the data. Of course, these subsets cannot be used to make robust conclusions about the full data set, but they can help us get a sense of which methods are feasible. Below, we'll create a subset with 30 trajectories overall, and try out the following methods: (i) `gam()` with ML, (ii) `bam()` with ML and `bam()` with fREML. The full code will be shown for the first model, but only the timing data are included for the other two.

```
set.seed(20)

# create subset: 30 trajectories
gl.r.subset <- gl.r[gl.r$traj %in% sample(unique(gl.r$traj), 30),]

# gam with method="ML"
gl.r.gam.ml.t <- system.time(
  gl.r.gam.ml.s <- gam(f3 ~ stress +
                          s(measurement.no) + s(measurement.no, by=stress) +
                          s(duration) + ti(measurement.no, duration) +
                          s(decade, k=4) +
                          s(decade, k=4, by=stress) +
                          ti(measurement.no, decade, k=c(10,4)) +
                          ti(measurement.no, decade, k=c(10,4), by=stress) +
                          s(measurement.no, speaker, bs="fs", m=1, k=4) +
                          s(measurement.no, traj, bs="fs", m=1, k=4),
                        dat=gl.r.subset, method="ML")
)

gl.r.gam.ml.t

##    user  system elapsed
##  40.492   0.973  42.158

# bam with method="ML"

gl.r.bam.ml.t

##    user  system elapsed
##  28.388   0.913  29.677

# bam with method="fREML"

gl.r.bam.fREML.t

##    user  system elapsed
##   3.431   0.436   3.951
```

Both `gam()` and `bam()` are painfully slow when ML is used, but `bam()` with fREML seems promising. Unfortunately, `anova` comparisons between models with different fixed effects only yield accurate results with ML, not REML or fREML. This leads to a dilemma: if we use ML, we can't use random smooths but the results of the model comparison are reliable; if we use fREML, we can use random smooths, but the results of the model comparison are unreliable. In this case, my recommendation is to go for the second option. Ignoring the grouping structure in the data introduces a much higher potential for anticonservativity than the use of fREML for model comparison (and, in fact, the fREML estimates for the subset are very close to the ML estimates). In situations where the model only has one set of random smooths (but with many levels), we can also try refitting the model with `gamm()`. Moreover, `gamm4()` can also be useful if the fixed effects structure of the model is a bit simpler (but I haven't been able to fit a `gamm4()` version of the current model). However, in the current case, `bam()` with fREML seems to be the best option.[10] Here is the final model, which took me about 7 minutes to fit:

[10]Note that `bam()` also allows the use of parallel computing on computers with more than one processor

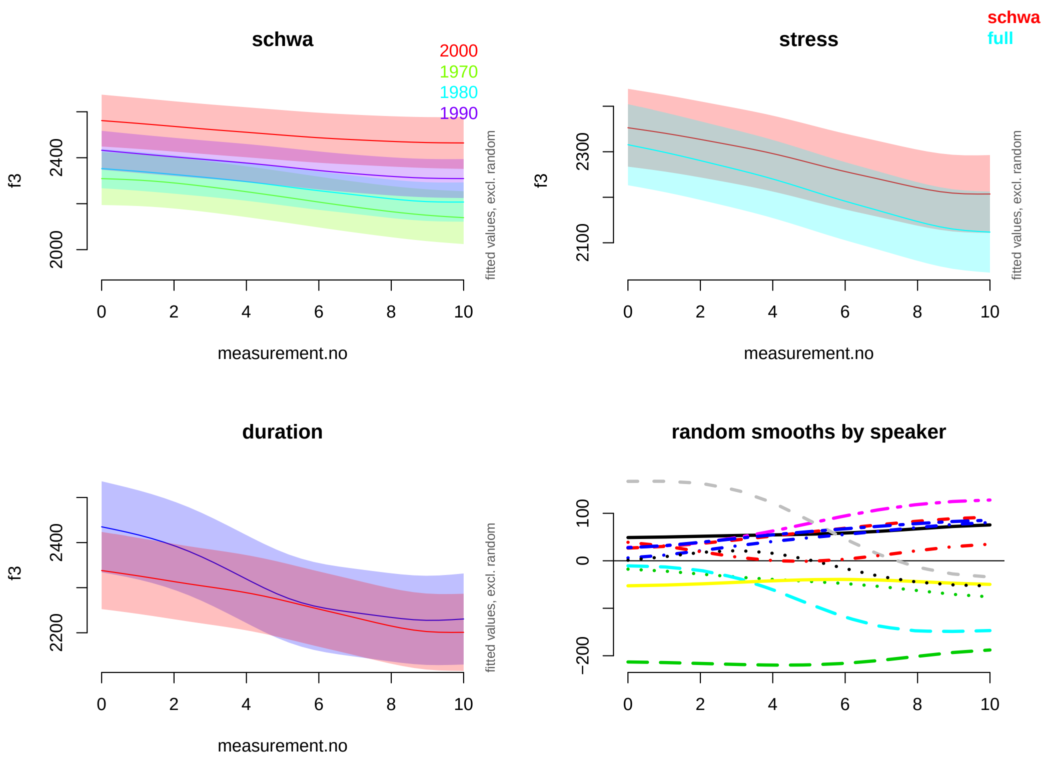
```
gl.r.bam.fREML <- bam(f3 ~ stress +
                          s(measurement.no) + s(measurement.no, by=stress) +
                          s(duration) + ti(measurement.no, duration) +
                          s(decade, k=4) +
                          s(decade, k=4, by=stress) +
                          ti(measurement.no, decade, k=c(10,4)) +
                          ti(measurement.no, decade, k=c(10,4), by=stress) +
                          s(measurement.no, speaker, bs="fs", m=1, k=4) +
                          s(measurement.no, traj, bs="fs", m=1, k=4),
                       dat=gl.r, method="fREML")
summary.coefs(gl.r.bam.fREML, digits=2)

## Parametric coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2318.87      29.34  79.042   <2e-16 ***
## stress.L       36.51      14.18   2.575   0.0101 *
##
## Approximate significance of smooth terms:
##                                             edf   Ref.df       F  p-value
## s(measurement.no)                         5.277    6.346   8.091  4.7e-09
## s(measurement.no):stressschwa             1.000    1.000   3.111 0.077873
## s(duration)                               2.062    2.066   1.944 0.150608
## ti(measurement.no,duration)               5.483    6.227   3.655 0.000965
## s(decade)                                 1.628    1.628   7.891 0.000700
## s(decade):stressschwa                     1.000    1.000   1.048 0.305979
## ti(measurement.no,decade)                 6.303    8.833   0.790 0.609148
## ti(measurement.no,decade):stressschwa     1.000    1.000   0.000 0.997451
## s(measurement.no,speaker)                32.943   46.000   3.911  < 2e-16
## s(measurement.no,traj)                 1554.455 1675.000 103.230  < 2e-16
##
## s(measurement.no)                      ***
## s(measurement.no):stressschwa          .
## s(duration)
## ti(measurement.no,duration)            ***
## s(decade)                              ***
## s(decade):stressschwa
## ti(measurement.no,decade)
## ti(measurement.no,decade):stressschwa
## s(measurement.no,speaker)              ***
## s(measurement.no,traj)                 ***
```

Neither of the difference terms representing the interaction between `decade` and `stress` are significant, which suggests that the change in rhoticity has progressed in the same way after schwa and full vowels. This should be confirmed by performing a model comparison, but, as noted in section 2.5.2, model comparisons based on multiple terms generally yield non-significant results when both of the terms are non-significant. Note also that in this model, the sole effect of `stress` seems to be an overall lowering of F3 (as indicated by the significant parametric term) in full vowels without any substantial variation in the shape of the trajectories (as indicated by the non-significant smooth term). The following plots show (i) the `decade` effect for schwa, (ii) the interaction between `stress` and `measurement.no`, (iii) the effect of duration and (iv) the by-subject

core. I was able to achieve a two-fold improvement in speed on a computer with four physical cores. Parallel computing can be enabled through the `discrete=T` option; the number of threads can be set using the `nthreads` option.

random smooths (just to provide a clearer sense of between-subject variation).

```
plot_smooth.cont(gl.r.bam.fREML, view="measurement.no", plot_all.c="decade",
                 cond=list(stress="schwa"), rug=F, rm.ranef=T, main="schwa",
                 ylim=c(1900,2700))
plot_smooth(gl.r.bam.fREML, view="measurement.no", plot_all="stress",
                 rug=F, rm.ranef=T, main="stress")
plot_smooth(gl.r.bam.fREML, view="measurement.no", cond=list(duration=0.3),
            rug=F, rm.ranef=T, col="blue", main="duration")
plot_smooth(gl.r.bam.fREML, view="measurement.no", cond=list(duration=0.1),
            rug=F, rm.ranef=T, col="red", add=T)
inspect_random(gl.r.bam.fREML, select=9, lwd=3, main="random smooths by speaker")
```



In order to check whether the effect of decade itself is significant, we can perform a model comparison between the current model and one that does not have any terms including decade at all. This comparison is shown below:

```
gl.r.bam.fREML.0 <- bam(f3 ~ stress +
                             s(measurement.no) + s(measurement.no, by=stress) +
                             s(duration) + ti(measurement.no, duration) +
                             s(measurement.no, speaker, bs="fs", m=1, k=4) +
                             s(measurement.no, traj, bs="fs", m=1, k=4),
                        dat=gl.r, method="fREML")
compareML(gl.r.bam.fREML, gl.r.bam.fREML.0, print.output=F)$table

##                Model    Score Edf  Chisq     Df   p.value Sig.
## 1 gl.r.bam.fREML.0 25961.28  15
## 2   gl.r.bam.fREML 25935.98  25 25.300 10.000 2.070e-07  ***
```

The difference between the models is significant, indicating that there is indeed a change in F3 as a function of time. This concludes our analysis of the Glasgow /r/ data set.

5 Final comments

The goal of this paper was to provide an introduction to GAMMs in the context of dynamic speech analysis. We have discussed a range of theoretical concepts and gone through two example data sets, covering many of the questions and issues that come up when working with these models. However, there are many more issues that I simply had to leave out in order to keep things short:

- checking modelling assumptions, e.g. normally distributed residuals
- dealing with violations of modelling assumptions such as autocorrelation and heteroskedascity
- other smoother types such as adaptive smoothers and cyclic smoothers (might be appropriate for pitch contours)
- GAMMs for two-dimensional spatial data

5.1 Useful references

Below is a short (and incomplete) list of useful GAMM references. I consulted many of these while putting this introduction together.

- Wood (2006): The standard reference for GAMMs. It is an excellent book with a lot of detail, and can be very useful for finding out more about the methods and assumptions underlying GAMMs. Though many of the sections assume a strong background in mathematics, most of the example analyses and parts of the conceptual discussion are possible to follow without mathematical training.
- Baayen et al. (2016): A paper that focuses specifically on modelling autocorrelation using GAMMs, and presents analyses for three separate linguistic examples. It also includes some complicated GAMMs (even some with three-way interactions).
- Kelly (2014): An online tutorial. This one covers not just GAMMs, but also a range of other methods for dealing with non-linearity, and makes various comparisons across these methods. Note that this tutorial relies on the `gam` library, not `mgcv`.
- Simpson (2014): An interesting blog post that shows how to model seasonal data using GAMMs. It also explains basis functions in a clear and concise way.
- Simpson (2011): Another great blog post with a lot of useful detail on autocorrelation components. It also has an interesting section on generating confidence intervals for the derivative of a smooth function.
- van Rij (2015): A great online tutorial that covers many of the technical aspects of fitting GAMMs. It also includes autocorrelation and model comparisons.

- van Rij (2016): A brief R vignette that describes three different methods for significance testing using GAMMs. Note that some of the model comparisons are based on GAMMs fitted with fREML, which may lead to unreliable results (this is noted in the text as well).
- Winter & Wieling (2016): Discusses GAMMs in the context of language change and evolution and provides a clear and concise introduction to GAMMs and mixed models in general. It also covers a number of topics that are not discussed here, such as autocorrelation and logistic / Poisson GAMMs. It comes with thoroughly commented example code that is also available at `https://github.com/bodowinter/change_tutorial_materials`.

References

Baayen, R. H. (2008). *Analyzing linguistic data: A practical introduction to statistics using R*. Cambridge: Cambridge University Press.

Baayen, R. H., van Rij, J., de Cat, C., & Wood, S. N. (2016). Autocorrelated errors in experimental data in the language sciences: Some solutions offered by generalized additive mixed models. arXiv preprint arXiv:1601.02043.

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of memory and language*, 68(3), 255–278.

Bates, D., Kliegl, R., Vasishth, S., & Baayen, H. (2015). Parsimonious mixed models. *arXiv preprint*, arXiv:1506.04967.

Bates, D., Maechler, M., & Bolker, B. (2011). *lme4: Linear mixed-effects models using S4 classes*. R package version 0.999375-42.

Gelman, A. & Hill, J. (2007). *Data analysis using regression and multilevel/hierarchical models*. Cambridge: Cambridge University Press.

Johnson, K. (2008). *Quantitative methods in linguistics*. Malden, MA & Oxford: Blackwell.

Kelly, R. (2014). Extending linear models: Non-linearity. `https://rstudio-pubs-static.s3.amazonaws.com/24589_7552e489485b4c2790ea6634e1afd68d.html`. Accessed on 23/01/2017.

R Core Team (2013). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Simpson, G. (2011). Additive modelling and the hadcrut3v global mean temperature series. `http://www.fromthebottomoftheheap.net/2011/06/12/additive-modelling-and-the-hadcrut3v-global-mean-temperature-series/`. Accessed on 23/01/2017.

Simpson, G. (2014). Modelling seasonal data with GAMs. `http://www.fromthebottomoftheheap.net/2014/05/09/modelling-seasonal-data-with-gam/`. Accessed on 23/01/2017.

Sóskuthy, M. (2016). Generalised additive mixed modelling for dynamic formant analysis. Talk given at LabPhon 2015, Cornell University, Ithaca, NY.

Sóskuthy, M. (in prep). Significance testing in dynamic speech analysis using generalised additive mixed models. Unpublished manuscript.

Stuart-Smith, J., Lennon, R., Macdonald, R., Robertson, D., Sóskuthy, M., José, B., & Evers, L. (2015). A dynamic acoustic view of real-time change in word-final liquids in spontaneous glaswegian. *Proceedings of the 18th International Congress of Phonetic Sciences*, 10-14 August 2015, Glasgow.

van Rij, J. (2015). Overview GAMM analysis of time series data. `www.sfs.uni-tuebingen.de/~jvanrij/Tutorial/GAMM.html`. Accessed on 23/01/2017.

van Rij, J. (2016). Testing for significance. `https://cran.r-project.org/web/packages/itsadug/vignettes/test.html`. Accessed on 23/01/2017.

van Rij, J., Wieling, M., Baayen, R. H., & van Rijn, H. (2016). itsadug: Interpreting Time Series and Autocorrelated Data using GAMMs. R package version 2.2.

Winter, B. (2013). Linear models and linear mixed effects models in r with linguistic applications. *arXiv preprint*, arXiv:1308.5499.

Winter, B. & Wieling, M. (2016). How to analyze linguistic change using mixed models, Growth Curve Analysis and Generalized Additive Modeling. *Journal of Language Evolution*, 1(1), 7–18.

Wood, S. (2006). *Generalized additive models: an introduction with R*. Boca Raton: CRC Press.

Wood, S. & Scheipl, F. (2014). *gamm4: Generalized additive mixed models using mgcv and lme4*. R package version 0.2-3.

Zuur, A., Ieno, E. N., Walker, N., Saveliev, A. A., & Smith, G. M. (2009). *Mixed effects models and extensions in ecology with R*. New York: Springer.