Advanced Bayesian Multilevel Modeling with the R Package brms

Paul-Christian Bürkner

Abstract

The **brms** package allows R users to easily specify a wide range of Bayesian single-level and multilevel models, which are fitted with the probabilistic programming language Stan behind the scenes. Several response distributions are supported, of which all parameters (e.g., location, scale, and shape) can be predicted at the same time thus allowing for distributional regression. Non-linear relationships may be specified using non-linear predictor terms or semi-parametric approaches such as splines or Gaussian processes. Multivariate models, in which each response variable can be predicted using the above mentioned options, can be fitted as well. To make all of these modeling options possible in a multilevel framework, **brms** provides an intuitive and powerful formula syntax, which extends the well known formula syntax of **lme4**. The purpose of the present paper is to introduce this syntax in detail and to demonstrate its usefulness with four examples, each showing other relevant aspects of the syntax. If you use **brms**, please cite this article as published in the R Journal (Bürkner 2018).

Keywords: Bayesian inference, multilevel models, distributional regression, MCMC, Stan, R.

1. Introduction

Multilevel models (MLMs) offer great flexibility for researchers across sciences (Brown and Prescott 2015; Demidenko 2013; Gelman and Hill 2006; Pinheiro and Bates 2006). They allow modeling of data measured on different levels at the same time – for instance data of students nested within classes and schools – thus taking complex dependency structures into account. It is not surprising that many packages for R have been developed to fit MLMs. Usually, however, the functionality of these implementations is limited insofar as it is only possible to predict the mean of the response distribution. Other parameters of the response distribution, such as the residual standard deviation in linear models, are assumed constant across observations, which may be violated in many applications. Accordingly, it is desirable to allow for prediction of *all* response parameters at the same time. Models doing exactly that are often referred to as *distributional models* or more verbosely *models for location, scale and shape* (Rigby and Stasinopoulos 2005). Another limitation of basic MLMs is that they only allow for linear predictor terms. While linear predictor terms already offer good flexibility, they are of limited use when relationships are inherently non-linear. Such non-linearity can be handled in at least two ways: (1) by fully specifying a non-linear predictor term with corresponding parameters each of which can be predicted using MLMs (Lindstrom and Bates 1990), or (2) estimating the form of the non-linear relationship on the fly using splines (Wood 2004) or Gaussian processes (Rasmussen and Williams 2006). The former are often simply

called *non-linear models*, while models applying splines are referred to as *generalized additive models* (GAMs; Hastie and Tibshirani, 1990).

Combining all of these modeling options into one framework is a complex task, both conceptually and with regard to model fitting. Maximum likelihood methods, which are typically applied in classical 'frequentist' statistics, can reach their limits at some point and fully Bayesian methods become the go-to solutions to fit such complex models (Gelman, Carlin, Stern, and Rubin 2014). In addition to being more flexible, the Bayesian framework comes with other advantages, for instance, the ability to derive probability statements for every quantity of interest or explicitly incorporating prior knowledge about parameters into the model. The former is particularly relevant in non-linear models, for which classical approaches struggle more often than not in propagating all the uncertainty in the parameter estimates to non-linear functions such as out-of-sample predictions.

Possibly the most powerful program for performing full Bayesian inference available to date is Stan (Stan Development Team 2017c; Carpenter, Gelman, Hoffman, Lee, Goodrich, Betancourt, Brubaker, Guo, Li, and Ridell 2017). It implements Hamiltonian Monte Carlo (Duane, Kennedy, Pendleton, and Roweth 1987; Neal 2011; Betancourt, Byrne, Livingstone, and Girolami 2014) and its extension, the No-U-Turn (NUTS) Sampler (Hoffman and Gelman 2014). These algorithms converge much more quickly than other Markov-Chain Monte-Carlo (MCMC) algorithms especially for high-dimensional models (Hoffman and Gelman 2014; Betancourt *et al.* 2014; Betancourt 2017). An excellent non-mathematical introduction to Hamiltonian Monte Carlo can be found in Betancourt (2017).

Stan comes with its own programming language, allowing for great modeling flexibility Stan Development Team (2017c); Carpenter *et al.* (2017)). Many researchers may still be hesitent to use Stan directly, as every model has to be written, debugged and possibly also optimized. This may be a time-consuming and error-prone process even for researchers familiar with Bayesian inference. The **brms** package (Bürkner 2017), presented in this paper, aims to remove these hurdles for a wide range of regression models by allowing the user to benefit from the merits of Stan by using extended **lme4**-like (Bates, Mächler, Bolker, and Walker 2015) formula syntax, with which many R users are familiar with. It offers much more than writing efficient and human-readable Stan code: **brms** comes with many post-processing and visualization functions, for instance to perform posterior predictive checks, leave-one-out cross-validation, visualization of estimated effects, and prediction of new data. The overarching aim is to have one general framework for regression modeling, which offers everything required to successfully apply regression models to complex data. To date, it already replaces and extends the functionality of dozens of other R packages, each of which is restricted to specific regression models[1].

The purpose of the present article is to provide an introduction of the advanced multilevel formula syntax implemented in **brms**, which allows to fit a wide and growing range of non-linear distributional multilevel models. A general overview of the package is already given in Bürkner (2017). Accordingly, the present article focuses on more recent developments. We begin by explaining the underlying structure of distributional models. Next, the formula syntax of **lme4** and its extensions implemented in **brms** are explained. Four examples that demonstrate the use of the new syntax are discussed in detail. Afterwards, the functionality of

[1]Unfortunately, due to the implementation via Stan, it is not easily possible for users to define their own response distributions and run them via **brms**. If you feel that a response distribution is missing in **brms**, please open an issue on GitHub (`https://github.com/paul-buerkner/brms`).

brms is compared with that of **rstanarm** (Stan Development Team 2017a) and **MCMCglmm** (Hadfield 2010). We end by describing future plans for extending the package.

2. Model description

The core of models implemented in **brms** is the prediction of the response y through predicting all parameters θ_p of the response distribution D, which is also called the model `family` in many R packages. We write

$$y_i \sim D(\theta_{1i}, \theta_{2i}, ...)$$

to stress the dependency on the i^{th} observation. Every parameter θ_p may be regressed on its own predictor term η_p transformed by the inverse link function f_p that is $\theta_{pi} = f_p(\eta_{pi})$[2]. Such models are typically refered to as *distributional models*[3]. Details about the parameterization of each `family` are given in `vignette("brms_families")`.

Suppressing the index p for simplicity, a predictor term η can generally be written as

$$\eta = \mathbf{X}\beta + \mathbf{Z}u + \sum_{k=1}^{K} s_k(x_k)$$

In this equation, β and u are the coefficients at population-level and group-level respectively and $\mathbf{X}, \mathbf{Z}$ are the corresponding design matrices. The terms $s_k(x_k)$ symbolize optional smooth functions of unspecified form based on covariates x_k fitted via splines (see Wood (2011) for the underlying implementation in the **mgcv** package) or Gaussian processes (Williams and Rasmussen 1996). The response y as well as $\mathbf{X}$, $\mathbf{Z}$, and x_k make up the data, whereas β, u, and the smooth functions s_k are the model parameters being estimated. The coefficients β and u may be more commonly known as fixed and random effects, but I avoid theses terms following the recommendations of Gelman and Hill (2006). Details about prior distributions of β and u can be found in Bürkner (2017) and under `help("set_prior")`.

As an alternative to the strictly additive formulation described above, predictor terms may also have any form specifiable in Stan. We call it a *non-linear* predictor and write

$$\eta = f(c_1, c_2, ..., \phi_1, \phi_2, ...)$$

The structure of the function f is given by the user, c_r are known or observed covariates, and ϕ_s are non-linear parameters each having its own linear predictor term η_{ϕ_s} of the form specified above. In fact, we should think of non-linear parameters as placeholders for linear predictor terms rather than as parameters themselves. A frequentist implementation of such models, which inspired the non-linear syntax in **brms**, can be found in the **nlme** package (Pinheiro, Bates, DebRoy, Sarkar, and R Core Team 2016).

3. Extended multilevel formula syntax

The formula syntax applied in **brms** builds upon the syntax of the R package **lme4** (Bates *et al.* 2015). First, we will briefly explain the **lme4** syntax used to specify multilevel models

[2] A parameter can also be assumed constant across observations so that a linear predictor is not required.
[3] The models described in Bürkner (2017) are a sub-class of the here described models.

and then introduce certain extensions that allow to specify much more complicated models in **brms**. An **lme4** formula has the general form

```
response ~ pterms + (gterms | group)
```

The `pterms` part contains the population-level effects that are assumed to be the same across obervations. The `gterms` part contains so called group-level effects that are assumed to vary accross grouping variables specified in `group`. Multiple grouping factors each with multiple group-level effects are possible. Usually, `group` contains only a single variable name pointing to a factor, but you may also use `g1:g2` or `g1/g2`, if both `g1` and `g2` are suitable grouping factors. The `:` operator creates a new grouping factor that consists of the combined levels of `g1` and `g2` (you could think of this as pasting the levels of both factors together). The `/` operator indicates nested grouping structures and expands one grouping factor into two or more when using multiple `/` within one term. If, for instance, you write `(1 | g1/g2)`, it will be expanded to `(1 | g1) + (1 | g1:g2)`. Instead of `|` you may use `||` in grouping terms to prevent group-level correlations from being modeled. This may be useful in particular when modeling so many group-level effects that convergence of the fitting algorithms becomes an issue due to model complexity. One limitation of the `||` operator in **lme4** is that it only splits up terms so that columns of the design matrix originating from the same term are still modeled as correlated (e.g., when coding a categorical predictor; see the `mixed` function of the **afex** package by Singmann, Bolker, and Westfall (2015) for a way to avoid this behavior).

While intuitive and visually appealing, the classic **lme4** syntax is not flexible enough to allow for specifying the more complex models supported by **brms**. In non-linear or distributional models, for instance, multiple parameters are predicted, each having their own population and group-level effects. Hence, multiple formulas are necessary to specify such models[4]. Then, however, specifying group-level effects of the same grouping factor to be correlated *across* formulas becomes complicated. The solution implemented in **brms** (and currently unique to it) is to expand the `|` operator into `|<ID>|`, where `<ID>` can be any value. Group-level terms with the same `ID` will then be modeled as correlated if they share same grouping factor(s)[5]. For instance, if the terms `(x1|ID|g1)` and `(x2|ID|g1)` appear somewhere in the same or different formulas passed to **brms**, they will be modeled as correlated.

Further extensions of the classical **lme4** syntax refer to the `group` part. It is rather limited in its flexibility since only variable names combined by `:` or `/` are supported. We propose two extensions of this syntax: Firstly, `group` can generally be split up in its terms so that, say, `(1 | g1 + g2)` is expanded to `(1 | g1) + (1 | g2)`. This is fully consistent with the way `/` is handled so it provides a natural generalization to the existing syntax. Secondly, there are some special grouping structures that cannot be expressed by simply combining grouping variables. For instance, multi-membership models cannot be expressed this way. To overcome this limitation, we propose wrapping terms in `group` within special functions that allow specifying alternative grouping structures: `(gterms | fun(group))`. In **brms**, there

[4] Actually, it is possible to specify multiple model parts within one formula using interactions terms for instance as implemented in **MCMCglmm** (Hadfield 2010). However, this syntax is limited in flexibility and requires a rather deep understanding of the way R parses formulas, thus often being confusing to users.

[5] It might even be further extended to `|fun(<ID>)|`, where `fun` defines the type of correlation structure, defaulting to unstructured that is estimating the full correlation matrix. The `fun` argument is not yet supported by **brms** but could be supported in the future if other correlation structures, such as compound symmetry or Toeplitz, turn out to have reasonable practical applications and effective implementations in Stan.

are currently two such functions implemented, namely `gr` for the default behavior and `mm` for multi-membership terms. To be compatible with the original syntax and to keep formulas short, `gr` is automatically added internally if none of these functions is specified.

While some non-linear relationships, such as quadratic relationships, can be expressed within the basic R formula syntax, other more complicated ones cannot. For this reason, it is possible in **brms** to fully specify non-linear predictor terms similar to how it is done in **nlme**, but fully compatible with the extended multilevel syntax described above. Suppose, for instance, we want to model the non-linear growth curve

$$y = b_1(1 - \exp(-(x/b_2)^{b_3})$$

between y and x with parameters b_1, b_2, and b_3 (see Example 3 in this paper for an implementation of this model with real data). Furthermore, we want all three parameters to vary by a grouping variable g and model those group-level effects as correlated. Additionally b_1 should be predicted by a covariate z. We can express this in **brms** using multiple formulas, one for the non-linear model itself and one per non-linear parameter:

```
y ~ b1 * (1 - exp(-(x / b2) ^ b3)
b1 ~ z + (1|ID|g)
b2 ~ (1|ID|g)
b3 ~ (1|ID|g)
```

The first formula will not be evaluated using standard R formula parsing, but instead taken literally. In contrast, the formulas for the non-linear parameters will be evaluated in the usual way and are compatible with all terms supported by **brms**. Note that we have used the above described ID-syntax to model group-level effects as correlated across formulas.

There are other syntax extensions implemented in **brms** that do not directly target grouping terms. Firstly, there are terms formally included in the `pterms` part that are handled separately. The most prominent examples are smooth terms specified through the `s` and `t2` functions of the **mgcv** package (Wood 2011). Other examples are category specific effects `cs`, monotonic effects `mo`, noise-free effects `me`, or Gaussian process terms `gp`. The former is explained in Bürkner (2017), while the latter three are documented in `help(brmsformula)`. Internally, these terms are extracted from `pterms` and not included in the construction of the population-level design matrix. Secondly, making use of the fact that `|` is unused on the left-hand side of $\sim$ in formula, additional information on the response variable may be specified via

```
response | aterms ~ <predictor terms>
```

The `aterms` part may contain multiple terms of the form `fun(<variable>)` separated by `+` each providing special information on the response variable. This allows among others to weight observations, provide known standard errors for meta-analysis, or model censored or truncated data. As it is not the main topic of the present paper, we refer to `help("brmsformula")` and `help("addition-terms")` for more details.

To set up the model formulas and combine them into one object, **brms** defines the `brmsformula` (or short `bf`) function. Its output can then be passed to the `parse_bf` function, which splits up the formulas in separate parts and prepares them for the generation of design matrices

and related data. Other packages may re-use these functions in their own routines making it easier to offer support for the above described multilevel syntax.

4. Examples

The idea of **brms** is to provide one unified framework for multilevel regression models in R. As such, the above described formula syntax in all of its variations can be applied in combination with all response distributions supported by **brms** (currently about 35 response distributions are supported; see `help("brmsfamily")` and `vignette("brms_families")` for an overview).

In this section, we will discuss four examples in detail, each focusing on certain aspects of the syntax. They are chosen to provide a broad overview of the modeling options. The first is about the number of fish caught be different groups of people. It does not actually contain any multilevel structure, but helps in understanding how to set up formulas for different model parts. The second example is about housing rents in Munich. We model the data using splines and a distributional regression approach. The third example is about cumulative insurance loss payments across several years, which is fitted using a rather complex non-linear multilevel model. Finally, the fourth example is about the performance of school children, who change school during the year, thus requiring a multi-membership model.

Despite not being covered in the four examples, there are a few more modeling options that we want to briefly describe. First, **brms** allows fitting so called phylogenetic models. These models are relevant in evolutionary biology when data of many species are analyzed at the same time. Species are not independent as they come from the same phylogenetic tree, implying that different levels of the same grouping-factor (i.e., species) are likely correlated. There is a whole vignette dedicated to this topic, which can be found via `vignette("brms_phylogenetics")`. Second, there is a canonical way to handle ordinal predictors, without falsely assuming they are either categorical or continuous. We call them monotonic effects and discuss them in `vignette("brms_monotonic")`. Last but not least, it is possible to account for measurement error in both response and predictor variables. This is often ignored in applied regression modeling (Westfall and Yarkoni 2016), although measurement error is very common in all scientific fields making use of observational data. There is no vignette yet covering this topic, but one will be added in the future. In the meantime, `help("brmsformula")` is the best place to start learning about such models as well as about other details of the **brms** formula syntax.

4.1. Example 1: Catching fish

An important application of the distributional regression framework of **brms** are so called zero-inflated and hurdle models. These models are helpful whenever there are more zeros in the response variable than one would naturally expect. Here, we consider an example dealing with the number of fish caught by various groups of people. On the UCLA website (https://stats.idre.ucla.edu/stata/dae/zero-inflated-poisson-regression), the data are described as follows: "The state wildlife biologists want to model how many fish are being caught by fishermen at a state park. Visitors are asked how long they stayed, how many people were in the group, were there children in the group and how many fish were caught. Some visitors do not fish, but there is no data on whether a person fished or not. Some visitors who did fish did not catch any fish so there are excess zeros in the data because of

the people that did not fish."

```
zinb <- read.csv("http://stats.idre.ucla.edu/stat/data/fish.csv")
zinb$camper <- factor(zinb$camper, labels = c("no", "yes"))
head(zinb)
```

```
  nofish livebait camper persons child         xb         zg count
1      1        0     no       1     0 -0.8963146  3.0504048     0
2      0        1    yes       1     0 -0.5583450  1.7461489     0
3      0        1     no       1     0 -0.4017310  0.2799389     0
4      0        1    yes       2     1 -0.9562981 -0.6015257     0
5      0        1     no       1     0  0.4368910  0.5277091     1
6      0        1    yes       4     2  1.3944855 -0.7075348     0
```

As predictors we choose the number of people per group, the number of children, as well as whether or not the group consists of campers. Many groups may not catch any fish just because they do not try and so we fit a zero-inflated Poisson model. For now, we assume a constant zero-inflation probability across observations.

```
fit_zinb1 <- brm(count ~ persons + child + camper, data = zinb,
                 family = zero_inflated_poisson("log"))
```

The model is readily summarized via

```
summary(fit_zinb1)
```

```
 Family: zero_inflated_poisson (log)
Formula: count ~ persons + child + camper
   Data: zinb (Number of observations: 250)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000
   WAIC: Not computed

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept    -1.01      0.17    -1.34    -0.67       2171    1
persons       0.87      0.04     0.79     0.96       2188    1
child        -1.36      0.09    -1.55    -1.18       1790    1
camper        0.80      0.09     0.62     0.98       2950    1

Family Specific Parameters:
   Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
zi     0.41      0.04     0.32     0.49       2409    1

Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
is a crude measure of effective sample size, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

A graphical summary is available through

```
conditional_effects(fit_zinb1)
```

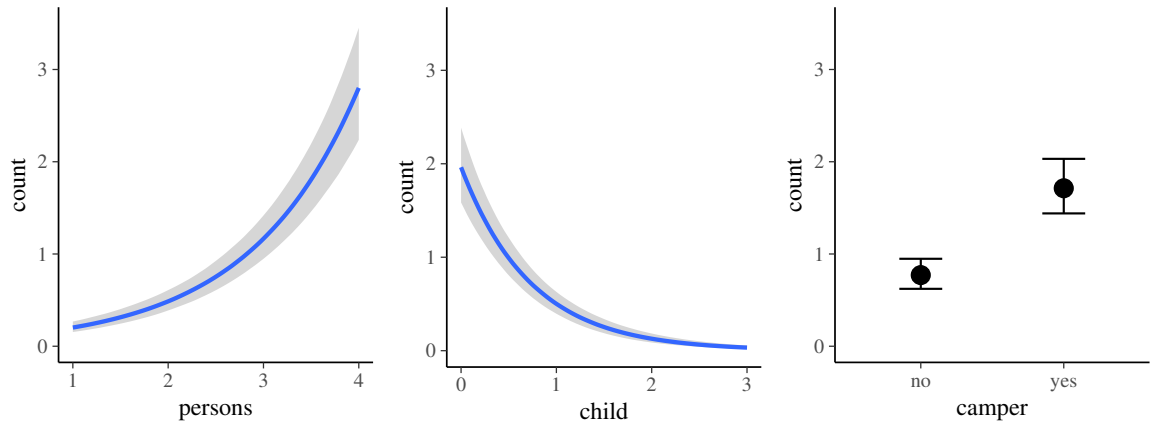

Figure 1: Conditional effects plots of the `fit_zinb1` model.

(see Figure 1). In fact, the `conditional_effects` method turned out to be so powerful in visualizing effects of predictors that I am using it almost as frequently as `summary`. According to the parameter estimates, larger groups catch more fish, campers catch more fish than non-campers, and groups with more children catch less fish. The zero-inflation probability `zi` is pretty large with a mean of 41%. Please note that the probability of catching no fish is actually higher than 41%, but parts of this probability are already modeled by the Poisson distribution itself (hence the name zero-*inflation*). If you want to treat all zeros as originating from a separate process, you can use hurdle models instead (not shown here).

Now, we try to additionally predict the zero-inflation probability by the number of children. The underlying reasoning is that we expect groups with more children to not even try catching fish, since children often lack the patience required for fishing. From a purely statistical perspective, zero-inflated (and hurdle) distributions are a mixture of two processes and predicting both parts of the model is natural and often very reasonable to make full use of the data.

```
fit_zinb2 <- brm(bf(count ~ persons + child + camper, zi ~ child),
                 data = zinb, family = zero_inflated_poisson())
```

To transform the linear predictor of `zi` into a probability, **brms** applies the logit-link, which takes values within $[0, 1]$ and returns values on the real line. Thus, it allows the transition between probabilities and linear predictors.

```
summary(fit_zinb2)
```

```
 Family: zero_inflated_poisson (log)
Formula: count ~ persons + child + camper
         zi ~ child
```

```
   Data: zinb (Number of observations: 250)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000
   WAIC: Not computed

Population-Level Effects:
             Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept       -1.07      0.18    -1.43    -0.73       2322    1
persons          0.89      0.05     0.80     0.98       2481    1
child           -1.17      0.10    -1.37    -1.00       2615    1
camper           0.78      0.10     0.60     0.96       3270    1
zi_Intercept    -0.95      0.27    -1.52    -0.48       2341    1
zi_child         1.21      0.28     0.69     1.79       2492    1

Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
is a crude measure of effective sample size, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

According to the model, trying to fish with children not only decreases the overall number fish caught (as implied by the Poisson part of the model) but also drastically increases your chance of catching no fish at all (as implied by the zero-inflation part), possibly because groups with more children spend less time or no time at all fishing. Comparing model fit via leave-one-out cross validation as implemented in the **loo** package (Vehtari, Gelman, and Gabry 2016a,b).

```
LOO(fit_zinb1, fit_zinb2)

                       LOOIC     SE
fit_zinb1             1639.52 363.30
fit_zinb2             1621.35 362.39
fit_zinb1 - fit_zinb2   18.16  15.71
```

reveals that the second model using the number of children to predict both model parts has better fit. However, when considering the standard error of the `LOOIC` difference, improvement in model fit is apparently modest and not substantial. More examples of distributional model can be found in `vignette("brms_distreg")`.

4.2. Example 2: Housing rents

In their book about regression modeling, Fahrmeir, Kneib, Lang, and Marx (2013) use an example about the housing rents in Munich from 1999. The data contains information about roughly 3000 apartments including among others the absolute rent (`rent`), rent per square meter (`rentsqm`), size of the apartment (`area`), construction year (`yearc`), and the district in Munich (`district`), where the apartment is located. The data can be found in the **gamlss.data** package (Stasinopoulos and Rigby 2016):

```
data("rent99", package = "gamlss.data")
head(rent99)
```

```
      rent   rentsqm area yearc location bath kitchen cheating district
1 109.9487  4.228797   26  1918        2    0       0        0      916
2 243.2820  8.688646   28  1918        2    0       0        1      813
3 261.6410  8.721369   30  1918        1    0       0        1      611
4 106.4103  3.547009   30  1918        2    0       0        0     2025
5 133.3846  4.446154   30  1918        2    0       0        1      561
6 339.0256 11.300851   30  1918        2    0       0        1      541
```

Here, we aim at predicting the rent per square meter with the size of the apartment as well as the construction year, while taking the district of Munich into account. As the effect of both predictors on the rent is of unknown non-linear form, we model these variables using a bivariate tensor spline (Wood, Scheipl, and Faraway 2013). The district is accounted for via a varying intercept.

```
fit_rent1 <- brm(rentsqm ~ t2(area, yearc) + (1|district), data = rent99,
                 chains = 2, cores = 2)
```

We fit the model using just two chains (instead of the default of four chains) on two processor cores to reduce the model fitting time for the purpose of the present paper. In general, using the default option of four chains (or more) is recommended.

```
summary(fit_rent1)
```

```
 Family: gaussian(identity)
Formula: rentsqm ~ t2(area, yearc) + (1 | district)
   Data: rent99 (Number of observations: 3082)
Samples: 2 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 2000
    ICs: LOO = NA; WAIC = NA; R2 = NA

Smooth Terms:
                   Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sds(t2areayearc_1)     4.93      2.32     1.61    10.77       1546 1.00
sds(t2areayearc_2)     5.78      2.87     1.58    13.15       1175 1.00
sds(t2areayearc_3)     8.09      3.19     3.66    16.22       1418 1.00

Group-Level Effects:
~district (Number of levels: 336)
              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(Intercept)     0.60      0.06     0.48     0.73        494 1.01

Population-Level Effects:
              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept         7.80      0.11     7.59     8.02       2000 1.00
t2areayearc_1    -1.00      0.09    -1.15    -0.83       2000 1.00
t2areayearc_2     0.75      0.17     0.43     1.09       2000 1.00
```

```
t2areayearc_3    -0.07      0.16    -0.40     0.24       1579 1.00

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sigma     1.95      0.03     1.90     2.01       2000 1.00

Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
is a crude measure of effective sample size, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

For models including splines, the output of `summary` is not tremendously helpful, but we get at least some information. Firstly, the credible intervals of the standard deviations of the coefficients forming the splines (under `'Smooth Terms'`) are sufficiently far away from zero to indicate non-linearity in the (combined) effect of `area` and `yearc`. Secondly, even after controlling for these predictors, districts still vary with respect to rent per square meter by a sizable amount as visible under `'Group-Level Effects'` in the output. To further understand the effect of the predictor, we apply graphical methods:

```
conditional_effects(fit_rent1, surface = TRUE)
```

In Figure 2, the conditional effects of both predictors are displayed, while the respective other predictor is fixed at its mean. In Figure 3, the combined effect is shown, clearly demonstrating an interaction between the variables. In particular, housing rents appear to be highest for small and relatively new apartments.

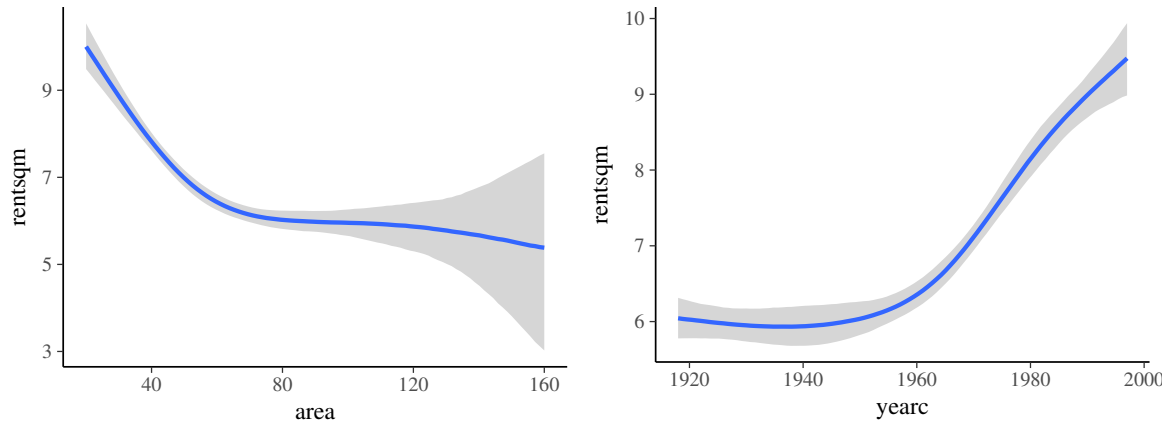

Figure 2: Conditional effects plots of the `fit_rent1` model for single predictors.

In the above example, we only considered the mean of the response distribution to vary by `area` and `yearc`, but this my not necessarily reasonable assumption, as the variation of the response might vary with these variables as well. Accordingly, we fit splines and effects of district for both the location and the scale parameter, which is called `sigma` in Gaussian models.

```
bform <- bf(rentsqm ~ t2(area, yearc) + (1|ID1|district),
```

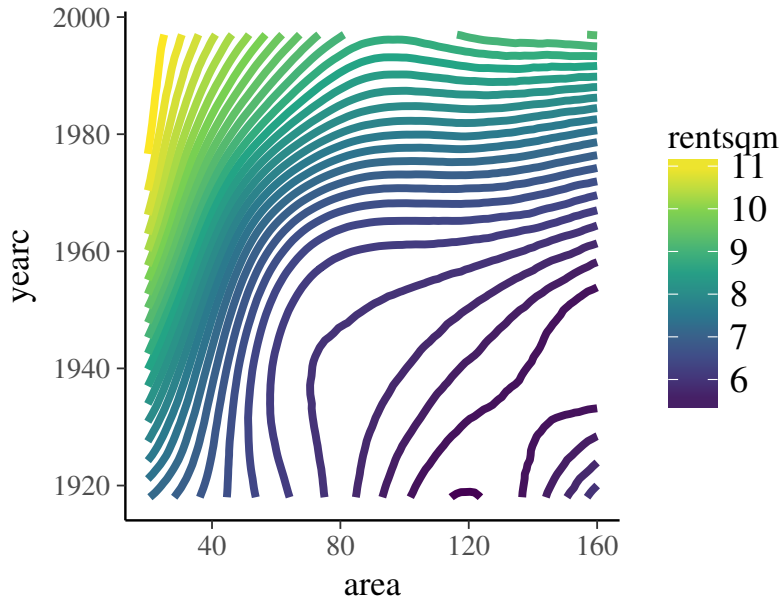

Figure 3: Surface plot of the `fit_rent1` model for the combined effect of `area` and `yearc`.

```
            sigma ~ t2(area, yearc) + (1|ID1|district))
fit_rent2 <- brm(bform, data = rent99, chains = 2, cores = 2)
```

If not otherwise specified, `sigma` is predicted on the log-scale to ensure it is positive no matter how the predictor term looks like. Instead of `(1|district)` as in the previous model, we now use `(1|ID1|district)` in both formulas. This results in modeling the varying intercepts of both model parts as correlated (see the description of the ID-syntax above). The group-level part of the `summary` output looks as follows:

```
Group-Level Effects:
~district (Number of levels: 336)
                              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(Intercept)                     0.60      0.06     0.49     0.73        744 1.00
sd(sigma_Intercept)               0.11      0.02     0.06     0.15        751 1.00
cor(Intercept,sigma_Intercept)    0.72      0.17     0.35     0.98        648 1.00
```

As visible from the positive correlation of the intercepts, districts with overall higher rent per square meter have higher variation at the same time. Lastly, we want to turn our attention to the splines. While `conditional_effects` is used to visualize effects of predictors on the expected response, `conditional_smooths` is used to show just the spline parts of the model:

```
conditional_smooths(fit_rent2)
```

The plot on the left-hand side of Figure 4 resembles the one in Figure 3, but the scale is different since only the spline is plotted. The right-hand side of 4 shows the spline for `sigma`. Since we apply the log-link on `sigma` by default the spline is on the log-scale as well. As

visible in the plot, the variation in the rent per square meter is highest for relatively small and old apartments, while the variation is smallest for medium to large apartments build around the 1960s.

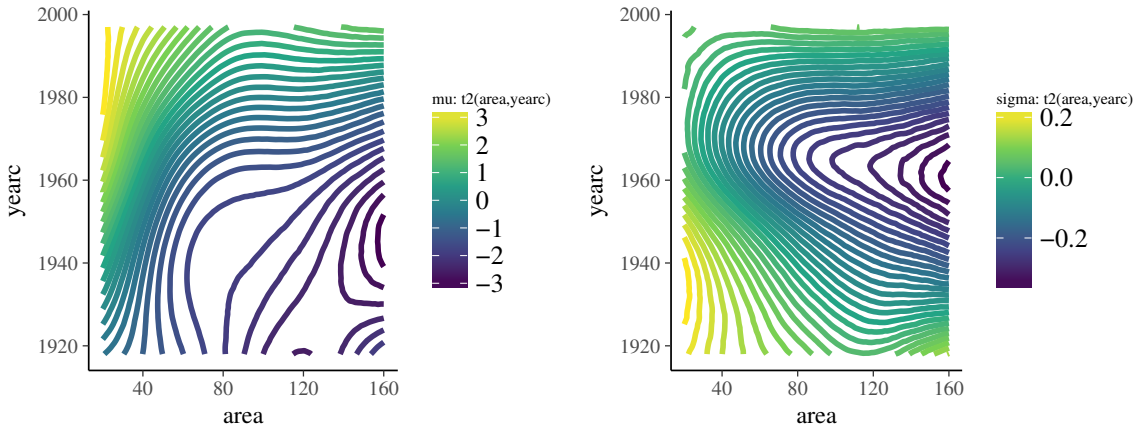

Figure 4: Plots showing the smooth terms of the `fit_rent2` model.

4.3. Example 3: Insurance loss payments

On his blog, Markus Gesmann predicts the growth of cumulative insurance loss payments over time, originated from different origin years (see http://www.magesblog.com/2015/11/loss-developments-via-growth-curves-and.html). We will use a slightly simplified version of his model for demonstration purposes here. It looks as follows:

$$cum_{AY,dev} \sim N(\mu_{AY,dev}, \sigma)$$

$$\mu_{AY,dev} = ult_{AY}\left(1 - \exp\left(-\left(\frac{dev}{\theta}\right)^{\omega}\right)\right)$$

The cumulative insurance payments cum will grow over time, and we model this dependency using the variable dev. Further, ult_{AY} is the (to be estimated) ultimate loss of accident each year. It constitutes a non-linear parameter in our framework along with the parameters θ and ω, which are responsible for the growth of the cumulative loss and are for now assumed to be the same across years. We load the data

```
url <- paste0("https://raw.githubusercontent.com/mages/",
              "diesunddas/master/Data/ClarkTriangle.csv")
loss <- read.csv(url)
head(loss)
```

```
    AY dev      cum
1 1991   6  357.848
2 1991  18 1124.788
3 1991  30 1735.330
4 1991  42 2182.708
```

```
5 1991  54 2745.596
6 1991  66 3319.994
```

and translate the proposed model into a non-linear **brms** model.

```
nlform <- bf(cum ~ ult * (1 - exp(-(dev / theta)^omega)),
             ult ~ 1 + (1|AY), omega ~ 1, theta ~ 1, nl = TRUE)

nlprior <- c(prior(normal(5000, 1000), nlpar = "ult"),
             prior(normal(1, 2), nlpar = "omega"),
             prior(normal(45, 10), nlpar = "theta"))

fit_loss1 <- brm(formula = nlform, data = loss, family = gaussian(),
                 prior = nlprior, control = list(adapt_delta = 0.9))
```

In the above functions calls, quite a few things are going on. The formulas are wrapped in `bf` to combine them into one object. The first formula specifies the non-linear model. We set argument `nl = TRUE` so that **brms** takes this formula literally and instead of using standard R formula parsing. We specify one additional formula per non-linear parameter (a) to clarify what variables are covariates and what are parameters and (b) to specify the predictor term for the parameters. We estimate a group-level effect of accident year (variable `AY`) for the ultimate loss `ult`. This also shows nicely how a non-linear parameter is actually a placeholder for a linear predictor, which in the case of `ult`, contains only a varying intercept for year. Both `omega` and `theta` are assumed to be constant across observations so we just fit a population-level intercept.

Priors on population-level effects are required and, for the present model, are actually mandatory to ensure identifiability. Otherwise, we may observe that different Markov chains converge to different parameter regions as multiple posterior distribution are equally plausible. Setting prior distributions is a difficult task especially in non-linear models. It requires some experience and knowledge both about the model that is being fitted and about the data at hand. Additionally, there is more to keep in mind to optimize the sampler's performance: Firstly, using non- or weakly informative priors in non-linear models often leads to problems even if the model is generally identified. For instance, if a zero-centered and reasonably wide prior such as `normal(0, 10000)` it set on `ult`, there is little information about `theta` and `omega` for samples of `ult` being close to zero, which may lead to convergence problems. Secondly, Stan works best when parameters are roughly on the same order of magnitude (Stan Development Team 2017b). In the present example, `ult` is of three orders larger than `omega`. Still, the sampler seems to work quite well, but this may not be true for other models. One solution is to rescale parameters before model fitting. For instance, for the present example, one could have downscaled `ult` by replacing it with `ult * 1000` and correspondingly the `normal(5000, 1000)` prior with `normal(5, 1)`.

In the `control` argument we increase `adapt_delta` to get rid of a few divergent transitions (cf. Stan Development Team, 2017b; Bürkner, 2017). Again the model is summarized via

```
summary(fit_loss1)
```

```
 Family: gaussian (identity)
Formula: cum ~ ult * (1 - exp(-(dev / theta)^omega))
         ult ~ 1 + (1 | AY)
         omega ~ 1
         theta ~ 1
   Data: loss (Number of observations: 55)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000
   WAIC: Not computed

Group-Level Effects:
~AY (Number of levels: 10)
                  Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(ult_Intercept)   745.74    231.31   421.05  1306.04        916    1

Population-Level Effects:
               Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
ult_Intercept    5273.70    292.34  4707.11  5852.28        798    1
omega_Intercept     1.34      0.05     1.24     1.43       2167    1
theta_Intercept    46.07      2.09    42.38    50.57       1896    1

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sigma   139.93     15.52    113.6   175.33       2358    1

Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
is a crude measure of effective sample size, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

as well as

```
conditional_effects(fit_loss1)
```

(see Figure 5). We can also visualize the cumulative insurance loss over time separately for each year.

```
conditions <- data.frame(AY = unique(loss$AY))
rownames(conditions) <- unique(loss$AY)
me_year <- conditional_effects(fit_loss1, conditions = conditions,
                               re_formula = NULL, method = "predict")
plot(me_year, ncol = 5, points = TRUE)
```

(see Figure 6). It is evident that there is some variation in cumulative loss across accident years, for instance due to natural disasters happening only in certain years. Further, we see that the uncertainty in the predicted cumulative loss is larger for later years with fewer available data points.

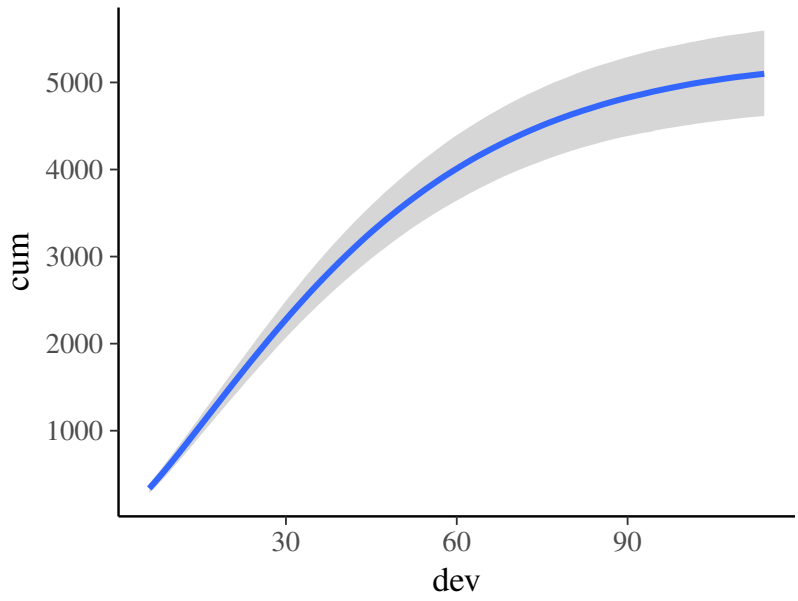

Figure 5: Conditional effects plots of the `fit_loss1` model.

In the above model, we considered `omega` and `delta` to be constant across years, which may not necessarily be true. We can easily investigate this by fitting varying intercepts for all three non-linear parameters also estimating group-level correlation using the above introduced `ID` syntax.

```
nlform2 <- bf(cum ~ ult * (1 - exp(-(dev / theta)^omega)),
              ult ~ 1 + (1|ID1|AY), omega ~ 1 + (1|ID1|AY),
              theta ~ 1 + (1|ID1|AY), nl = TRUE)

fit_loss2 <- update(fit_loss1, formula = nlform2,
                    control = list(adapt_delta = 0.90))
```

We could have also specified all predictor terms more conveniently within one formula as

```
ult + omega + theta ~ 1 + (1|ID1|AY)
```

because the structure of the predictor terms is identical. To compare model fit, we perform leave-one-out cross-validation.

```
LOO(fit_loss1, fit_loss2)

                      LOOIC    SE
fit_loss1             715.44 19.24
fit_loss2             720.60 19.85
fit_loss1 - fit_loss2  -5.15  5.34
```

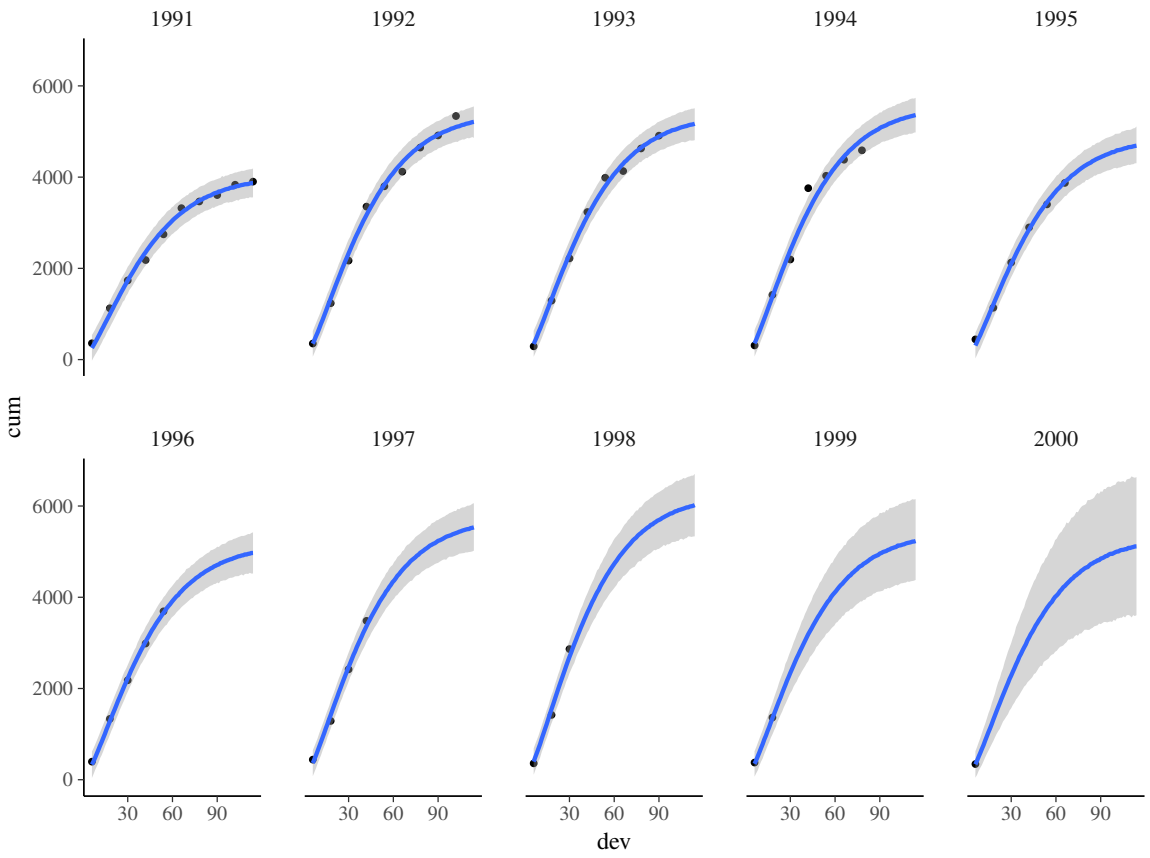

Figure 6: Conditional effects plots of the `fit_loss1` model separately for each accident year.

Since smaller values indicate better expected out-of-sample predictions and thus better model fit, the simpler model that only has a varying intercept over parameter `ult` is preferred. This may not be overly surprising, given that three varying intercepts as well as three group-level correlations are probably overkill for data containing only 55 observations. Nevertheless, it nicely demonstrates how to apply the `ID` syntax in practice. More examples of non-linear models can be found in `vignette("brms_nonlinear")`.

4.4. Example 4: Performance of school children

Suppose that we want to predict the performance of students in the final exams at the end of the year. There are many variables to consider, but one important factor will clearly be school membership. Schools might differ in the ratio of teachers and students, the general quality of teaching, in the cognitive ability of the students they draw, or other factors we are not aware of that induce dependency among students of the same school. Thus, it is advised to apply a multilevel modeling techniques including school membership as a group-level term. Of course, we should account for class membership and other levels of the educational hierarchy as well, but for the purpose of the present example, we will focus on schools only. Usually, accounting for school membership is pretty-straight forward by simply adding a varying intercept to the formula: `(1 | school)`. However, a non-negligible number of students might change schools

during the year. This would result in a situation where one student is a member of multiple schools and so we need a multi-membership model. Setting up such a model not only requires information on the different schools students attend during the year, but also the amount of time spend at each school. The latter can be used to weight the influence each school has on its students, since more time attending a school will likely result in greater influence. For now, let us assume that students change schools maximally once a year and spend equal time at each school. We will later see how to relax these assumptions.

Real educational data are usually relatively large and complex so that we simulate our own data for the purpose of this tutorial paper. We simulate 10 schools and 1000 students, with each school having the same expected number of 100 students. We model 10% of students as changing schools.

```
data_mm <- sim_multi_mem(nschools = 10, nstudents = 1000, change = 0.1)
head(data_mm)
```

```
  s1 s2  w1  w2        y
1  8  9 0.5 0.5 16.27422
2 10  9 0.5 0.5 18.71387
3  5  3 0.5 0.5 23.65319
4  3  5 0.5 0.5 22.35204
5  5  3 0.5 0.5 16.38019
6 10  6 0.5 0.5 17.63494
```

The code of function `sim_multi_mem` can be found in the online supplement of the present paper. For reasons of better illustration, students changing schools appear in the first rows of the data. Data of students being only at a single school looks as follows:

```
data_mm[101:106, ]
```

```
    s1 s2  w1  w2         y
101  2  2 0.5 0.5 27.247851
102  9  9 0.5 0.5 24.041427
103  4  4 0.5 0.5 12.575001
104  2  2 0.5 0.5 21.203644
105  4  4 0.5 0.5 12.856166
106  4  4 0.5 0.5  9.740174
```

Thus, school variables are identical, but we still have to specify both in order to pass the data appropriately. Incorporating no other predictors into the model for simplicity, a multi-membership model is specified as

```
fit_mm <- brm(y ~ 1 + (1 | mm(s1, s2)), data = data_mm)
```

The only new syntax element is that multiple grouping factors (`s1` and `s2`) are wrapped in `mm`. Everything else remains exactly the same. Note that we did not specify the relative weights of schools for each student and thus, by default, equal weights are assumed.

```
summary(fit_mm)

 Family: gaussian (identity)
Formula: y ~ 1 + (1 | mm(s1, s2))
   Data: data_mm (Number of observations: 1000)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000
   WAIC: Not computed

Group-Level Effects:
~mms1s2 (Number of levels: 10)
              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(Intercept)     2.76      0.82     1.69     4.74        682 1.01

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept       19      0.93    17.06     20.8        610    1

Family Specific Parameters:
      Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sigma     3.58      0.08     3.43     3.75       2117    1

Samples were drawn using sampling(NUTS). For each parameter, Eff.Sample
is a crude measure of effective sample size, and Rhat is the potential
scale reduction factor on split chains (at convergence, Rhat = 1).
```

With regard to the assumptions made in the above example, it is unlikely that all children who change schools stay in both schools equally long. To relax this assumption, we have to specify weights. First, we amend the simulated data to contain non-equal weights for students changing schools. For all other students, weighting does of course not matter as they stay in the same school anyway.

```
data_mm[1:100, "w1"] <- runif(100, 0, 1)
data_mm[1:100, "w2"] <- 1 - data_mm[1:100, "w1"]
head(data_mm)

  s1 s2        w1         w2        y
1  8  9 0.3403258 0.65967423 16.27422
2 10  9 0.1771435 0.82285652 18.71387
3  5  3 0.9059811 0.09401892 23.65319
4  3  5 0.4432007 0.55679930 22.35204
5  5  3 0.8052026 0.19479738 16.38019
6 10  6 0.5610243 0.43897567 17.63494
```

Incorporating these weights into the model is straight forward.

```
fit_mm2 <- brm(y ~ 1 + (1 | mm(s1, s2, weights = cbind(w1, w2))),
               data = data_mm)
```

The summary output is similar to the previous, so we do not show it here. The second assumption that students change schools only once a year, may also easily be relaxed by providing more than two grouping factors, say, `mm(s1, s2, s3)`.

5. Comparison between packages

Over the years, many R packages have been developed that implement MLMs, each being more or less general in their supported models. In Bürkner (2017), I compared **brms** with **lme4** (Bates *et al.* 2015), **MCMCglmm** (Hadfield 2010), **rstanarm** (Stan Development Team 2017a), and **rethinking** (McElreath 2017). Since then, quite a few new features have been added in particular to **brms** and **rstanarm**. Accordingly, in the present paper, I will update these comparisons, but focus on **brms**, **rstanarm**, and **MCMCglmm** as the possibly most important R packages implementing Bayesian MLMs. While **brms** and **rstanarm** are both based on the probabilistic programming language **Stan**, **MCMCglmm** implements its own custom MCMC algorithm. Modeling options and other important information of these packages are summarized in Table 1 and will be discussed in detail below.

Regarding general model classes, all three packages support the most common types such as linear, count data and certain survival models. Currently, **brms** and **MCMCglmm** provide more flexibility when modeling categorical and ordinal data. Additionally, they offer support for zero-inflated and hurdle models to account for access zeros in the data (see Example 1 above). For survival / time-to-event models, **rstanarm** offers great flexibility via the `stan_jm` function, which allows for complex association structures between time-to-event data and one or more models of longitudinal covariates (for details see https://cran.r-project.org/web/packages/rstanarm/vignettes/jm.html). Model classes currently specific to **brms** are robust linear models using Student's t-distribution (family `student`) as well as response times models via the exponentially modified Gaussian (family `exgaussian`) distribution or the Wiener diffusion model (family `wiener`). The latter allows to simultaneously model dichotomous decisions and their corresponding response times (for a detailed example see http://singmann.org/wiener-model-analysis-with-brms-part-i/).

All three packages offer many additional modeling options, with **brms** currently having the greatest flexibility (see Table 1 for a summary). Moreover, the packages differ in the general framework, in which they are implemented: **brms** and **MCMCglmm** each come with a single model fitting function (`brm` and `MCMCglmm` respectively), through which all of their models can be specified. Further, their framework allows to seamlessly combine most modeling options with each other in the same model. In contrast, the approach of **rstanarm** is to emulate existing functions of other packages. This has the advantage of an easier transition between classical and Bayesian models, since the syntax used to specify models stays the same. However, it comes with the main disadvantage that many modeling options cannot be used in combination within the same model.

Information criteria are available in all three packages. The advantages of WAIC and LOO implemented in **brms** and **rstanarm**, are their less restrictive assumptions and that their standard errors can be easily estimated to get a better sense of the uncertainty in the criteria. Comparing the prior options of the packages, **brms** offers a little more flexibility than **MCMCglmm** and **rstanarm**, as virtually any prior distribution can be applied on population-level effects as well as on the standard deviations of group-level effects. In addition, I believe

that the way priors are specified in **brms** is more intuitive as it is directly evident what prior is actually applied. In **brms**, Bayes factors are available both via Savage-Dickey ratios (Wagenmakers, Lodewyckx, Kuriyal, and Grasman 2010) and bridge-sampling (Gronau and Singmann 2017), while **rstanarm** allows for the latter option. For a detailed comparison with respect to sampling efficiency, see Bürkner (2017).

6. Conclusion

The present paper is meant to introduce R users and developers to the extended **lme4** formula syntax applied in **brms**. Only a subset of modeling options were discussed in detail, which ensured the paper was not too broad. For some of the more basic models that **brms** can fit, see Bürkner (2017). Many more examples can be found in the growing number of vignettes accompanying the package (see `vignette(package = "brms")` for an overview).

To date, **brms** is already one of the most flexible R packages when it comes to regression modeling. However, for the future, there are quite a few more features that I am planning to implement (see https://github.com/paul-buerkner/brms/issues for the current list of issues). In addition to smaller, incremental updates, I have two specific features in mind: (1) latent variables estimated via confirmatory factor analysis and (2) missing value imputation. I receive ideas and suggestions from users almost every day – for which I am always grateful – and so the list of features that will be implemented in the proceeding versions of **brms** will continue to grow.

Acknowledgments

First of all, I would like to thank the Stan Development Team for creating the probabilistic programming language Stan, which is an incredibly powerful and flexible tool for performing full Bayesian inference. Without it, **brms** could not fit a single model. Furthermore, I want to thank Heinz Holling, Donald Williams and Ruben Arslan for valuable comments on earlier versions of the paper. I also would like to thank the many users who reported bugs or had ideas for new features, thus helping to continuously improve **brms**.

References

Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48.

Betancourt M (2017). "A Conceptual Introduction to Hamiltonian Monte Carlo." *arXiv preprint*. URL https://arxiv.org/pdf/1701.02434.pdf.

Betancourt M, Byrne S, Livingstone S, Girolami M (2014). "The Geometric Foundations of Hamiltonian Monte Carlo." *arXiv preprint arXiv:1410.5110*.

Brown H, Prescott R (2015). *Applied Mixed Models in Medicine*. John Wiley & Sons.

Bürkner PC (2017). "**brms**: An R Package for Bayesian Multilevel Models using Stan." *Journal of Statistical Software*.

	brms	**rstanarm**	**MCMCglmm**
Model classes			
Linear models	yes	yes	yes
Robust linear models	yes	no	no
Count data models	yes	yes	yes
Survival models	yes	yes[1]	yes
Response times models	yes	no	no
Beta models	yes	yes	no
Categorical models	yes	yes[2]	yes
Multinomial models	no	no	yes
Ordinal models	various	cumulative[2]	cumulative
Zero-inflated and hurdle models	yes	no	yes
Modeling options			
Variable link functions	various	various	no
Multilevel structures	yes	yes	yes
Multi-membership	yes	no	yes
Multivariate responses	yes	yes[3]	yes
Non-linear predictors	yes	limited[4]	no
Distributional regression	yes	no	no
Finite mixtures	yes	no	no
Splines (additive models)	yes	yes	yes
Gaussian Processes	yes	no	no
Temporal autocorrelation	yes	yes[2,5]	no
Spatial autocorrelation	yes	yes[2,5]	no
Monotonic effects	yes	no	no
Category specific effects	yes	no	no
Measurement error	yes	no	no
Weights	yes	yes	no
Offset	yes	yes	using priors
Censored data	yes	yes[1]	yes
Truncated data	yes	no	no
Customized covariances	yes	no	yes
Missing value imputation	no	no	no
Bayesian specifics			
Population-level priors	flexible	flexible	normal
Group-level priors	normal	normal	normal
Covariance priors	flexible	restricted[6]	restricted[7]
Bayes factors	yes	yes[8]	no
Parallelization	yes	yes	no
Other			
Estimator	HMC, NUTS	HMC, NUTS	MH, Gibbs[9]
Information criterion	WAIC, LOO	WAIC, LOO	DIC
C++ compiler required	yes	no	no

Table 1: Comparison of the capabilities of the **brms**, **rstanarm** and **MCMCglmm** packages. Notes: (1) Advanced functionality available via `stan_jm`. (2) No group-level terms and related functionality allowed. (3) Cannot be combined with other modeling options such as splines. (4) Functionality limited to linear Gaussian models and certein pre-specified non-linear functions. (5) Functionality available only on GitHub branches (https://github.com/stan-dev/rstanarm). (6) For details see Hadfield (2010). (7) For details see https://github.com/stan-dev/rstanarm/wiki/Prior-distributions. (8) Available via

Bürkner PC (2018). "Advanced Bayesian Multilevel Modeling with the R Package brms." *The R Journal*, **10**(1), 395–411. doi:10.32614/RJ-2018-017.

Carpenter B, Gelman A, Hoffman M, Lee D, Goodrich B, Betancourt M, Brubaker MA, Guo J, Li P, Ridell A (2017). "Stan: A Probabilistic Programming Language." *Journal of Statistical Software*.

Demidenko E (2013). *Mixed Models: Theory and Applications with R*. John Wiley & Sons.

Duane S, Kennedy AD, Pendleton BJ, Roweth D (1987). "Hybrid Monte Carlo." *Physics Letters B*, **195**(2), 216–222.

Fahrmeir L, Kneib T, Lang S, Marx B (2013). *Regression: models, methods and applications*. Springer Science & Business Media.

Gelman A, Carlin JB, Stern HS, Rubin DB (2014). *Bayesian Data Analysis*, volume 2. Taylor & Francis.

Gelman A, Hill J (2006). *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press.

Gronau QF, Singmann H (2017). *bridgesampling: Bridge Sampling for Marginal Likelihoods and Bayes Factors*. R package version 0.4-0, URL https://CRAN.R-project.org/package=bridgesampling.

Hadfield JD (2010). "MCMC Methods for Multi-Response Generalized Linear Mixed Models: the **MCMCglmm** R Package." *Journal of Statistical Software*, **33**(2), 1–22.

Hastie TJ, Tibshirani RJ (1990). *Generalized Additive Models*, volume 43. CRC Press.

Hoffman MD, Gelman A (2014). "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." *The Journal of Machine Learning Research*, **15**(1), 1593–1623.

Lindstrom MJ, Bates DM (1990). "Nonlinear Mixed Effects Models for Repeated Measures Data." *Biometrics*, pp. 673–687.

McElreath R (2017). *rethinking: Statistical Rethinking Course and Book Package*. R package version 1.59, URL https://github.com/rmcelreath/rethinking.

Neal RM (2011). *Handbook of Markov Chain Monte Carlo*, volume 2, chapter MCMC Using Hamiltonian Dynamics. CRC Press.

Pinheiro J, Bates D (2006). *Mixed-Effects Models in S and S-PLUS*. Springer-Verlage Science & Business Media.

Pinheiro J, Bates D, DebRoy S, Sarkar D, R Core Team (2016). ***nlme**: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-124, URL http://CRAN.R-project.org/package=nlme.

Rasmussen CE, Williams CKI (2006). "Gaussian processes for machine learning."

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape." *Journal of the Royal Statistical Society C (Applied Statistics)*, **54**(3), 507–554.

Singmann H, Bolker B, Westfall J (2015). ***afex**: Analysis of Factorial Experiments*. R package version 0.15-2, URL https://CRAN.R-project.org/package=afex.

Stan Development Team (2017a). *rstanarm: Bayesian applied regression modeling via Stan*. R package version 2.17.2, URL http://mc-stan.org/.

Stan Development Team (2017b). *Stan: A C++ Library for Probability and Sampling, Version 2.17.0*. URL http://mc-stan.org/.

Stan Development Team (2017c). *Stan Modeling Language: User's Guide and Reference Manual*. URL http://mc-stan.org/manual.html.

Stasinopoulos M, Rigby B (2016). *gamlss.data: GAMLSS Data*. R package version 5.0-0, URL https://CRAN.R-project.org/package=gamlss.data.

Vehtari A, Gelman A, Gabry J (2016a). ***loo**: Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models*. R package version 1.0.0, URL https://github.com/stan-dev/loo.

Vehtari A, Gelman A, Gabry J (2016b). "Practical Bayesian Model Evaluation Using Leave-One-Out Cross-Validation and WAIC." *Statistics and Computing*, pp. 1–20.

Wagenmakers EJ, Lodewyckx T, Kuriyal H, Grasman R (2010). "Bayesian hypothesis testing for psychologists: A tutorial on the Savage–Dickey method." *Cognitive psychology*, **60**(3), 158–189.

Westfall J, Yarkoni T (2016). "Statistically Controlling for Confounding Constructs is Harder than You Think." *PloS one*, **11**(3), e0152719.

Williams CK, Rasmussen CE (1996). "Gaussian processes for regression." In *Advances in neural information processing systems*, pp. 514–520.

Wood SN (2004). "Stable and Efficient Multiple Smoothing Parameter Estimation for Generalized Additive Models." *Journal of the American Statistical Association*, **99**(467), 673–686.

Wood SN (2011). "Fast Stable Restricted Maximum Likelihood and Marginal Likelihood Estimation of Semiparametric Generalized Linear Models." *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, **73**(1), 3–36.

Wood SN, Scheipl F, Faraway JJ (2013). "Straightforward intermediate rank tensor product smoothing in mixed models." *Statistics and Computing*, pp. 1–20.

Affiliation:

Paul-Christian Bürkner
E-mail: paul.buerkner@gmail.com
URL: https://paul-buerkner.github.io