

Advanced Bayesian Multilevel Modeling with the R Package *brms*

by 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 fit 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. Non-linear relationships may be specified using non-linear predictor terms or semi-parametric approaches such as splines or Gaussian processes. Multivariate models can be fit 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 relevant aspects of the syntax.

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 students nested within classes and schools – thus taking complex dependency structures into account. It is not surprising that many packages have been developed to fit MLMs in R. Usually, the functionality of these implementations is limited insofar as they only 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, an assumption which may be violated in many applications. Accordingly, it is desirable to allow prediction of *all* response parameters at the same time. Models that perform simultaneous estimation are often referred to as *distributional models* or, more verbosely, as *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, see 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, typically applied in classical ‘frequentist’ statistics, can reach their limit at some point such that fully Bayesian methods become the go-to solutions to fit such complex models (Gelman et al., 2014). In addition to being more flexible, the Bayesian framework comes with other advantages, such as the ability to derive probability statements for every quantity of interest or explicitly incorporate 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 et al., 2017), which implements Hamiltonian Monte Carlo (Duane et al., 1987; Neal, 2011; Betancourt et al., 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 hesitant to use Stan directly, as every model has to be written, debugged, and possibly also optimized, which 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 formula syntax (Bates et al., 2015), with which many R users are familiar. The *brms* package offers much more than writing efficient and human-readable Stan code. It comes with many post-processing

and visualization functions, including functions for 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 apply regression models to complex data. To date, it replaces and extends the functionality of dozens of other R packages, each of which is restricted to specific regression models. Users may even define their own response distributions and run them via `brms` (for details, see `vignette("brms_customfamilies")`).

The purpose of the present article is to provide an introduction to the advanced multilevel formula syntax implemented in `brms`, which fits a wide and growing range of non-linear distributional multilevel models. A general overview of the package is 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 the extensions implemented in `brms` are explained. Four examples that demonstrate the use of the new syntax are discussed in detail. Afterwards, the functionality of `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.

Model description

The core model 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

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

to stress the dependency on the i^{th} observation. If desired, every parameter θ_p may be regressed on its own predictor term η_p , transformed by the inverse link function f_p that is $\theta_{pi} = f_p(\eta_{pi})$. Such models are typically referred to as *distributional models*. The models described in Bürkner (2017) are a sub-class of the models described here. 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

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

where β and u are the respective coefficients at the population-level and group-level, 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 we avoid these terms following the recommendations of Gelman and Hill (2006). Details about prior distributions for 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

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

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 et al., 2016).

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 and then introduce certain extensions that allow specifying much more complicated models in `brms`.

An `lme4` formula has the general form

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

The `pterm`s contain population-level effects, assumed to be the same across observations. The `gterm`s contain so called group-level effects, assumed to vary across the 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 users 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 $g2$. The $/$ operator indicates nested grouping structures and expands one grouping factor into two or more when using multiple $/$ within one term. For instance, if the user were to write $(1 | g1/g2)$, **brms** will expand this to $(1 | g1) + (1 | g1:g2)$. Instead of $|$, users may use $||$ in grouping terms to prevent group-level correlations from being modeled. This may be used to reduce model complexity if convergence of the sampler is an issue. 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 et al., 2015, for a way to avoid this behavior).

While intuitive and visually appealing, the classic **lme4** syntax is not flexible enough to specify the more complex models supported by **brms**. In non-linear or distributional models, multiple parameters are predicted, each having their own population and group-level effects. Hence, multiple formulas are necessary to specify such models.¹ 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) expands 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). 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 component. In the **lme4** version, the group component is rather limited in its flexibility since only variable names combined by $:$ or $/$ are supported. We propose two extensions of this syntax.

The first extension concerns the group element. We propose that group can generally be split up in its terms so that, for example, $(1 | g1 + g2)$ is expanded to $(1 | g1) + (1 | g2)$. This is fully consistent with the way $/$ is handled and provides a natural generalization to the existing syntax.

The second extension concerns special grouping structures that cannot currently 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 are currently two such functions implemented: `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 no function is specified.

While some non-linear relationships, such as quadratic relationships, can be expressed within the basic R formula syntax, more complicated ones cannot. Thus, we have made it possible in **brms** to specify non-linear predictor terms. Similar to the implementation in **nlme**, our extension is 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 wish to model the 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 is 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 correlated across formulas.

There are other syntax extensions implemented in **brms** that do not directly target grouping terms. First, there are terms formally included in the `pterm`s part that are handled separately. The most prominent examples are smooth terms specified using the `s` and `t2` functions of the **mgcv** package (Wood, 2011). Other examples are category specific effects `cs`, monotonic effects `m0`, noise-free effects

¹Actually, it is possible to specify multiple model parts within one formula using interactions terms, 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.

me, or Gaussian process terms gp. The first is explained in Bürkner (2017), while the latter three are documented in `help(brmsformula)`. Internally, these terms are extracted from `pterm`s and not included in the construction of the population-level design matrix.

Second, since the `|` operator is unused on the left-hand side of `~` in the 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>)`, each separated by `+` and providing special information on the response variable, including weighting observations, providing known standard errors for meta-analysis, or modeling censored or truncated data. As it is not the main topic of the present paper, we refer users 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` function, or `bf` for short. Its output can then be passed to the `parse_bf` function, which splits up the formulas into 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.

Examples

The idea of **brms** is to provide one unified framework for multilevel regression models in R. The above described formula syntax and 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. These examples are chosen to provide a broad overview of the modeling options. The first is about the number of fish caught by different groups of people. It does not actually contain any multilevel structure, but helps to understand 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.

In addition to the four examples, we wish to discuss briefly a few more modeling options. First, **brms** allows fitting of 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 if they come from the same phylogenetic tree, implying that different levels of the same grouping-factor (i.e., species) are likely correlated. See `vignette("brms_phylogenetics")` for an example. 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. Although often ignored in applied regression modeling (Westfall and Yarkoni, 2016), measurement error is common in all scientific fields that employ observational data. There is no vignette yet for 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.

Example 1: Catching fish

An important application of the distributional regression framework of **brms** are 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."

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

	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

For predictors, we choose the number of people per group, the number of children, and whether or not the group consists of campers. Many groups may not catch any fish 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).

Figure 1 shows the graphical summary output by

```
marginal_effects(fit_zinb1)
```

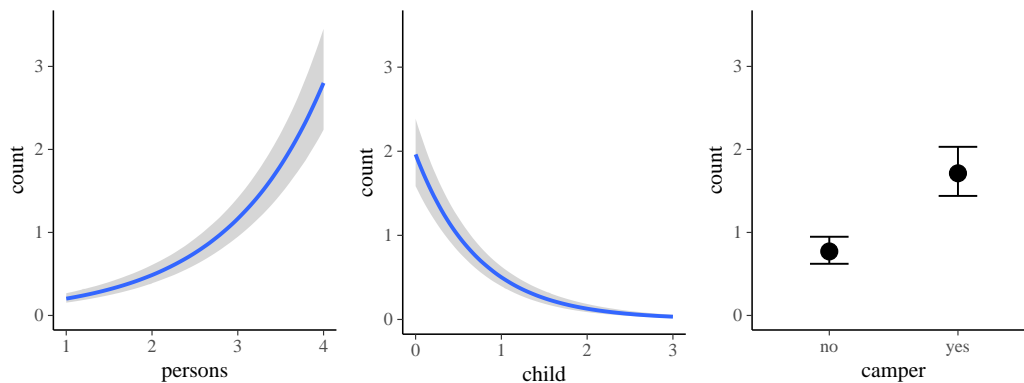


Figure 1: Marginal effects plots of the fit_zinb1 model.

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%. Note that the probability of catching no fish is actually higher than 41%, since parts of this probability are already modeled by the Poisson distribution itself (hence the name

zero-inflation). If the user wants to treat all zeros as originating from a separate process, hurdle models are also supported by **brms** (not shown here).

Since we expect groups with more children to avoid fishing, we next try to predict the zero-inflation probability using the number of children. From a purely statistical perspective, zero-inflated and hurdle distributions are a mixture of two processes. 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 `zi` into a probability, **brms** applies the logit-link, which takes values in $[0, 1]$ 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 decreases the overall number fish caught (as implied by the Poisson part of the model) and decreases the chance of catching any fish (as implied by the zero-inflation part).

Next we demonstrate how to comparing model fit via leave-one-out cross validation as implemented in the **loo** package (Vehtari et al., 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
```

The example above shows that the second model (using the number of children as a predictor) 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 models can be found in vignette("brms_distreg").

Example 2: Housing rents

Fahrmeir et al. (2013) use an example about housing rents in Munich. The data contains information on roughly 3000 apartments, and includes variables such as the absolute rent (`rent`), rent per square meter (`rentsqm`), size of the apartment (`area`), construction year (`yearc`), and the district in Munich where the apartment is located (`district`). 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 wish to predict the rent per square meter using the following explanatory variables: size of the apartment, the construction year, and the district in which the apartment is located. As the effect of apartment size and construction year predictors is of unknown non-linear form, we model these variables using a bivariate tensor spline (Wood et al., 2013). The district is accounted for using 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 gives limited information. First, 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. Second, even after controlling for these predictors, rent per square meter will still vary considerably by district, as visible under 'Group-Level Effects' in the output. To further understand the effect of the predictor, we apply graphical methods:

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

In Figure 2, the marginal effect of each predictor is displayed with the other predictor fixed at its mean. In Figure 3, the combined effect demonstrates the interaction between the variables. In particular, housing rents appear to be highest for small and relatively new apartments.

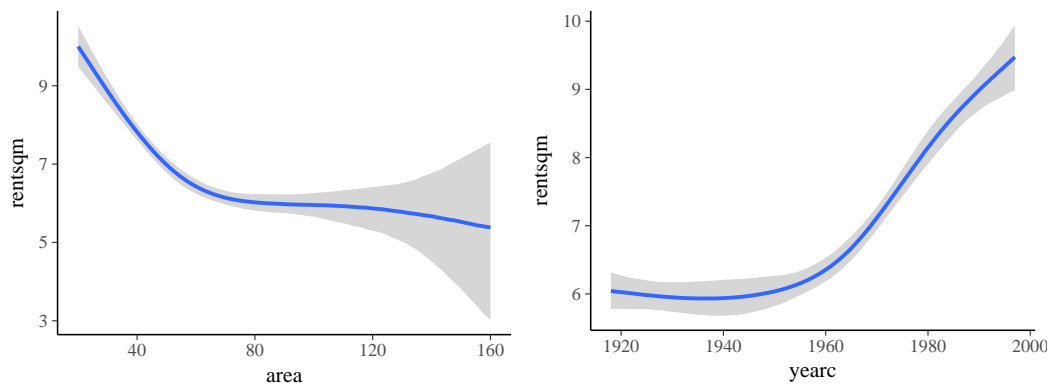


Figure 2: Marginal effects plots of the `fit_rent1` model for single predictors.

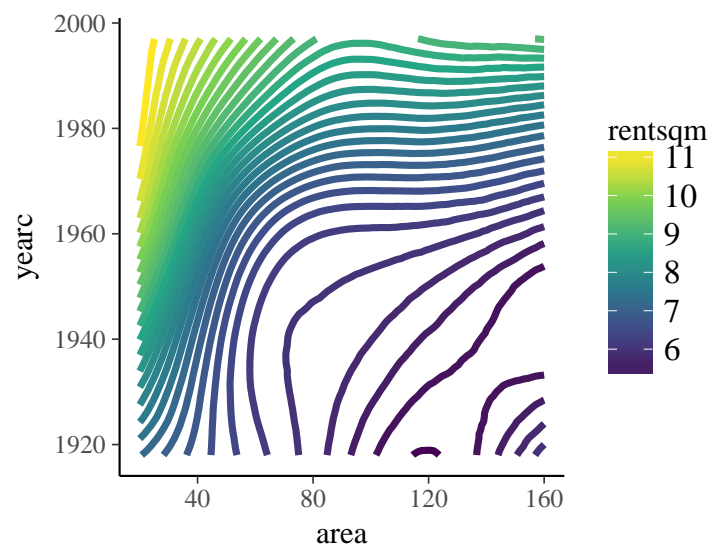


Figure 3: Surface plot of the `fit_rent1` model for the combined effect of area and yearc.

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

```
bform <- bf(rentsqsm ~ t2(area, yearc) + (1|ID1|district),
           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. Instead of `(1|district)` as in the previous model, we now use `(1|ID1|district)` in both formulas, which 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 like the following:

```
Group-Level Effects:
~district (Number of levels: 336)
              Estimate Est.Error 1-95% 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 also have higher variance in the outcome.

Lastly, we want to turn our attention to the splines. While `marginal_effects` is used to visualize effects of predictors on the expected response, `marginal_smooths` is used to show just the spline parts of the model:


```
marginal_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 σ . Since we apply the log-link on σ 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.

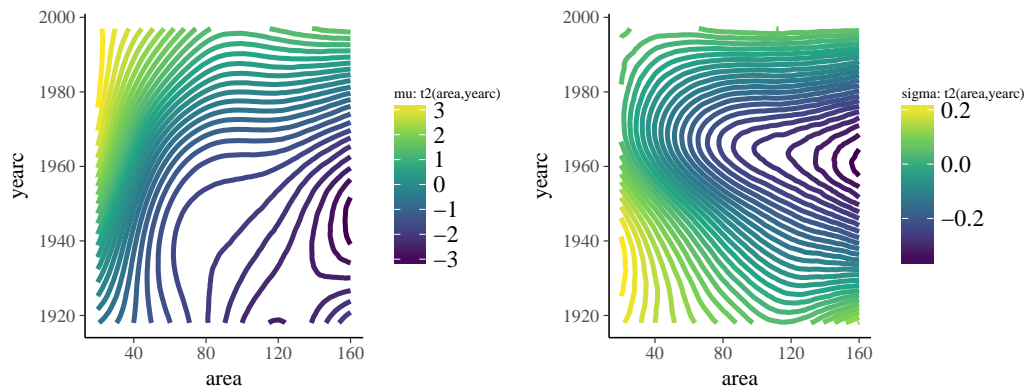


Figure 4: Plots showing the smooth terms of the `fit_rent2` model.

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:

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

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

The cumulative insurance payments cum will grow over time, and we model this dependency using the variable dev . Furthermore, 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 happen. The formulas are wrapped in `bf` to combine them into one object. The first formula specifies the non-linear model and we set argument `n1 = TRUE` so that `brms` takes this formula literally, instead of using standard R formula parsing. We specify one additional formula per non-linear parameter to: (a) clarify which variables are covariates and which are parameters; and (b) specify the predictor term for the parameters. We estimate a group-level effect for accident year (variable `AY`) and ultimate loss `ult`. This specification nicely shows 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 are mandatory to ensure identifiability for the present model. Otherwise, we may observe that different Markov chains converge to different parameter regions as multiple posterior distributions are equally plausible. Setting prior distributions is a difficult task especially in non-linear models because 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: First, 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)` is set for `ult`, there is little information about `theta` and `omega` for samples of `ult` being close to zero, which may lead to convergence problems. Second, Stan works best when parameters are roughly on the same order of magnitude (Stan Development Team, 2017b). In the present example, `ult` is three orders larger than `omega`. Although the sampler seems to work quite well in this case, it may not work well for other models. One solution is to rescale parameters before model fitting. For the present example, one could downscale `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

```
marginal_effects(fit_loss1)
```

which yields Figure 5.

We can also visualize the cumulative insurance loss over time separately for each year, as in Figure 6.

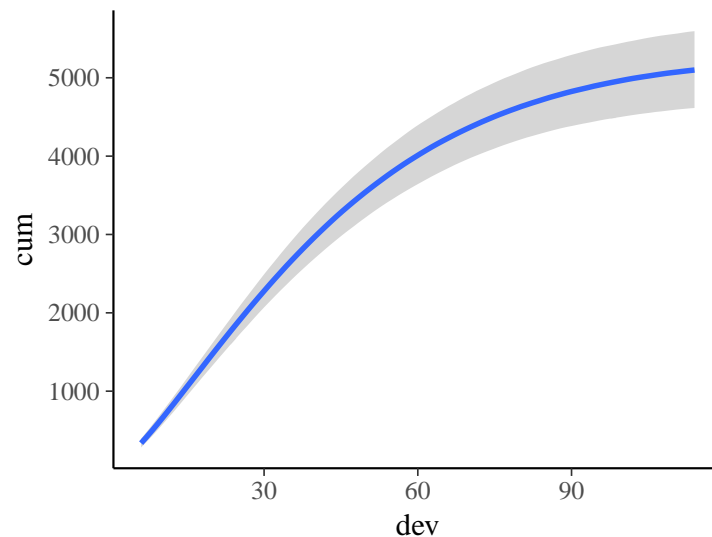


Figure 5: Marginal effects plots of the `fit_loss1` model.

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

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 in which there are fewer available data points.

In the above model, we considered ω and δ 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 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 with 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
<code>fit_loss1</code>	715.44	19.24
<code>fit_loss2</code>	720.60	19.85
<code>fit_loss1 - fit_loss2</code>	-5.15	5.34

Since smaller values indicate better expected out-of-sample predictions and thus a better model fit, the simpler model that has only a varying intercept over parameter `ult` is preferred. This result may not be overly surprising, given that modeling three varying intercepts and three group-level correlations is 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")`.

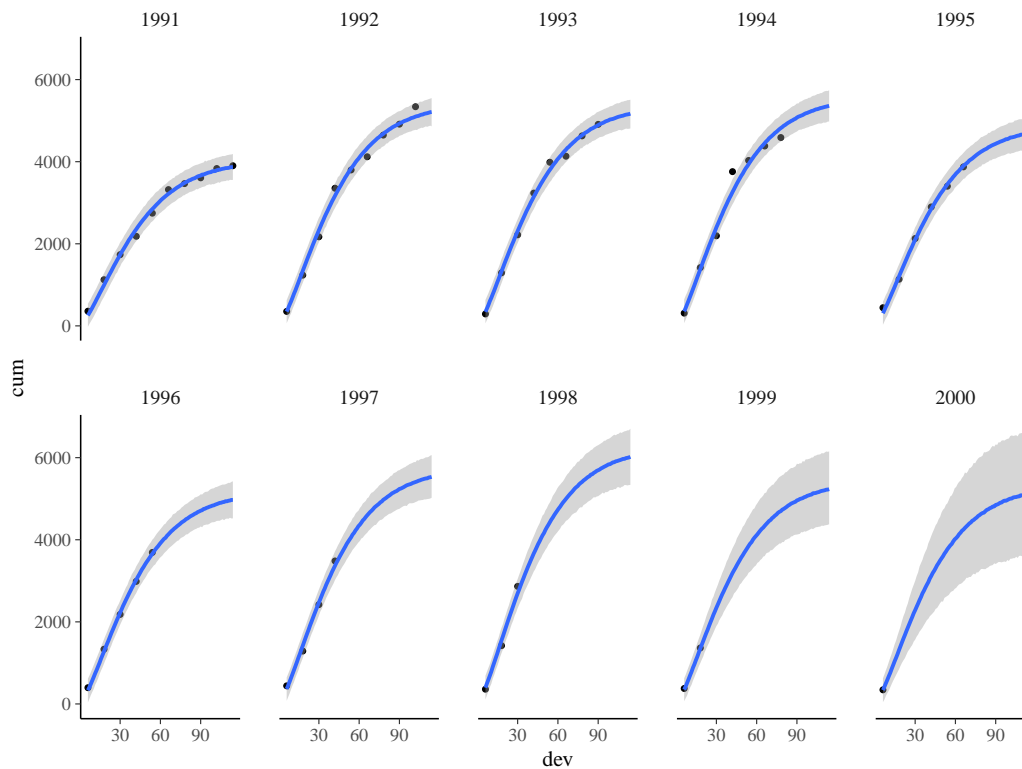


Figure 6: Marginal effects plots of the `fit_loss1` model separately for each accident year.

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 unobserved factors that may induce dependency among students in the same school. Thus, it is advisable to apply multilevel modeling techniques by including school membership as a group-level term. We should account for class membership and other levels of the educational hierarchy as well, but for the purposes of the present example, we will focus on schools only.

Usually, accounting for school membership is straight-forward; we simply add a varying intercept to the formula for each school with $(1 \mid \text{school})$. However, a non-negligible number of students might change schools during the year, which would result in a situation where one student is a member of multiple schools, necessitating a multi-membership model. Specifying such a model not only requires information on the different schools students attend during the year, but also the amount of time spent at each school. The time spent 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 example. We simulate 1000 students partitioned into 10 schools of 100 students each. 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 for this paper. To better illustrate the multimembership condition, 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
```

Although school variables `s1` and `s2` are identical, we still have to specify both in order to pass the data appropriately. Incorporating no other predictors 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 equal weights are assumed by default.

```
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 `w1` and `w2` for students changing schools:

```
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 straightforward:

```
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 omit it for parsimony. The second assumption that students change schools only once a year, may also easily be relaxed by providing more than two grouping factors in `mm` (i.e., `mm(s1, s2, s3)`).

Comparison between packages

Over the years, many R packages have offered implementations for 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 to **brms** and **rstanarm**. Accordingly, in the present paper, I will update these comparisons, focusing on **brms**, **rstanarm**, and **MCMCglmm**. While **brms** and **rstanarm** are both based on the Stan probabilistic programming language, **MCMCglmm** implements its own custom MCMC algorithm. Modeling options and other important information for these packages are summarized in Table 1 and will be discussed in detail below.

Regarding general model classes, all three packages support common types, including linear, count, 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 zeros in the data (see Example 1 above). For survival or 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`), response time models via the exponentially modified Gaussian distribution (family `exgaussian`), and the Wiener diffusion model (family `wiener`). The latter simultaneously models dichotomous outcomes 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 seamless combination of most modeling options in the same model. In contrast, the approach of **rstanarm** is to emulate existing functions of other packages, which has the advantage of an easier transition between classical and Bayesian models. Although the syntax used to specify models is most similar to that of a classical model, **rstanarm** comes with the main disadvantage that many modeling options cannot be used in combination in 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 options for prior distributions, **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, we believe that the way priors are specified in **brms** is more intuitive as it is directly evident which prior has been applied. In **brms**, Bayes factors are available via both Savage-Dickey ratios (Wagenmakers et al., 2010) and bridge-sampling (Gronau and Singmann, 2017), while **rstanarm** only allows for the latter option. For a detailed comparison with respect to sampling efficiency, see Bürkner (2017).

Conclusion

The present paper is meant to introduce R users and developers to the extended **lme4** formula syntax applied in **brms**. Four examples are presented to illustrate various features in the **brms** framework. 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 and there are more features in the works (see <https://github.com/paul-buerkner/brms/issues> for the current list of issues). In addition to smaller, incremental updates, major planned updates include: (1) latent variables estimated via confirmatory factor analysis; and (2) extended multilevel and residual

	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	yes	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 certain 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 the **bridgesampling** package (Gronau and Singmann, 2017). (9) Estimator consists of a combination of both algorithms.

correlation structures. The **brms** package is under continuous development thanks to constructive user input. Suggestions for future features may be submitted to <https://github.com/paul-buerkner/brms>.

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 would also like to thank the many users who reported bugs or had ideas for new features.

Bibliography

- D. Bates, M. Mächler, B. Bolker, and S. Walker. Fitting linear mixed-effects models using **lme4**. *Journal of Statistical Software*, 67(1):1–48, 2015. [p395, 396, 408]
- M. Betancourt. A conceptual introduction to hamiltonian monte carlo. *arXiv preprint*, 2017. URL <https://arxiv.org/pdf/1701.02434.pdf>. [p395]
- M. Betancourt, S. Byrne, S. Livingstone, and M. Girolami. The geometric foundations of hamiltonian monte carlo. *arXiv preprint arXiv:1410.5110*, 2014. [p395]
- H. Brown and R. Prescott. *Applied Mixed Models in Medicine*. John Wiley & Sons, 2015. [p395]
- P.-C. Bürkner. **brms**: An R package for bayesian multilevel models using stan. *Journal of Statistical Software*, 2017. [p395, 396, 398, 404, 408]
- B. Carpenter, A. Gelman, M. Hoffman, D. Lee, B. Goodrich, M. Betancourt, M. A. Brubaker, J. Guo, P. Li, and A. Ridell. Stan: A probabilistic programming language. *Journal of Statistical Software*, 2017. [p395]
- E. Demidenko. *Mixed Models: Theory and Applications with R*. John Wiley & Sons, 2013. [p395]
- S. Duane, A. D. Kennedy, B. J. Pendleton, and D. Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2): 216–222, 1987. [p395]
- L. Fahrmeir, T. Kneib, S. Lang, and B. Marx. *Regression: Models, Methods and Applications*. Springer-Verlag, 2013. [p400]
- A. Gelman and J. Hill. *Data Analysis Using Regression and Multilevel/Hierarchical Models*. Cambridge University Press, 2006. [p395, 396]
- A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin. *Bayesian Data Analysis*, volume 2. Taylor & Francis, 2014. [p395]
- Q. F. Gronau and H. Singmann. *Bridgesampling: Bridge Sampling for Marginal Likelihoods and Bayes Factors*, 2017. URL <https://CRAN.R-project.org/package=bridgesampling>. R package version 0.4-0. [p408, 409]
- J. D. Hadfield. Mcmc methods for multi-response generalized linear mixed models: The **MCMCglmm** R package. *Journal of Statistical Software*, 33(2):1–22, 2010. [p396, 397, 408, 409]
- T. J. Hastie and R. J. Tibshirani. *Generalized Additive Models*, volume 43. CRC Press, 1990. [p395]
- M. D. Hoffman and A. Gelman. The no-u-turn sampler: Adaptively setting path lengths in hamiltonian monte carlo. *The Journal of Machine Learning Research*, 15(1):1593–1623, 2014. [p395]
- M. J. Lindstrom and D. M. Bates. Nonlinear mixed effects models for repeated measures data. *Biometrics*, pages 673–687, 1990. [p395]
- R. McElreath. *Rethinking: Statistical Rethinking Course and Book Package*, 2017. URL <https://github.com/rmcelreath/rethinking>. R package version 1.59. [p408]
- R. M. Neal. *Handbook of Markov Chain Monte Carlo*, volume 2, chapter MCMC Using Hamiltonian Dynamics. CRC Press, 2011. [p395]

- J. Pinheiro and D. Bates. *Mixed-Effects Models in S and S-PLUS*. Springer-Verlag, 2006. [p395]
- J. Pinheiro, D. Bates, S. DebRoy, D. Sarkar, and R Core Team. **nlme**: *Linear and Nonlinear Mixed Effects Models*, 2016. URL <http://CRAN.R-project.org/package=nlme>. R package version 3.1-124. [p396]
- C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. Massachusetts Institute of Technology, 2006. [p395]
- R. A. Rigby and D. M. Stasinopoulos. Generalized additive models for location, scale and shape. *Journal of the Royal Statistical Society C*, 54(3):507–554, 2005. [p395]
- H. Singmann, B. Bolker, and J. Westfall. **afex**: *Analysis of Factorial Experiments*, 2015. URL <https://CRAN.R-project.org/package=afex>. R package version 0.15-2. [p397]
- Stan Development Team. *Rstanarm: Bayesian Applied Regression Modeling via Stan.*, 2017a. URL <http://mc-stan.org/>. R package version 2.17.2. [p396, 408]
- Stan Development Team. *Stan: A C++ Library for Probability and Sampling, Version 2.17.0*, 2017b. URL <http://mc-stan.org/>. [p404]
- Stan Development Team. *Stan Modeling Language: User’s Guide and Reference Manual*, 2017c. URL <http://mc-stan.org/manual.html>. [p395]
- M. Stasinopoulos and B. Rigby. *Gamlss.data: GAMLSS Data*, 2016. URL <https://CRAN.R-project.org/package=gamlss.data>. R package version 5.0-0. [p400]
- A. Vehtari, A. Gelman, and J. Gabry. **loo**: *Efficient Leave-One-Out Cross-Validation and WAIC for Bayesian Models.*, 2016a. URL <https://github.com/stan-dev/loo>. R package version 1.0.0. [p400]
- A. Vehtari, A. Gelman, and J. Gabry. Practical bayesian model evaluation using leave-one-out cross-validation and waic. *Statistics and Computing*, pages 1–20, 2016b. [p400]
- E.-J. Wagenmakers, T. Lodewyckx, H. Kuriyal, and R. Grasman. Bayesian hypothesis testing for psychologists: A tutorial on the savage–dickey method. *Cognitive psychology*, 60(3):158–189, 2010. [p408]
- J. Westfall and T. Yarkoni. Statistically controlling for confounding constructs is harder than you think. *PloS one*, 11(3):e0152719, 2016. [p398]
- C. K. Williams and C. E. Rasmussen. Gaussian processes for regression. In *Advances in Neural Information Processing Systems*, pages 514–520, 1996. [p396]
- S. N. Wood. Stable and efficient multiple smoothing parameter estimation for generalized additive models. *Journal of the American Statistical Association*, 99(467):673–686, 2004. [p395]
- S. N. Wood. Fast stable restricted maximum likelihood and marginal likelihood estimation of semi-parametric generalized linear models. *Journal of the Royal Statistical Society B*, 73(1):3–36, 2011. [p396, 397]
- S. N. Wood, F. Scheipl, and J. J. Faraway. Straightforward intermediate rank tensor product smoothing in mixed models. *Statistics and Computing*, pages 1–20, 2013. [p401]

Paul-Christian Bürkner
Faculty of Psychology, University of Münster
Fliegerstr. 21, 48151 Münster
Germany
paul.buerkner@gmail.com