Define Custom Response Distributions with brms

Paul Bürkner

Introduction

The **brms** package (Bürkner, 2017a, 2017b) implements Bayesian regression models using the probabilistic programming language **Stan** (Carpenter et al., 2016) behind the scenes. It has grown to be one of the most flexible R packages when it comes to multilevel regression modelling. A wide range of response distributions are supported, allowing users to fit – among others – linear, robust linear, count data, survival, response times, ordinal, zero-inflated, hurdle, and even self-defined mixture models all in a multilevel context. Predictor terms can be specified with a simple yet powerful formula syntax. Thanks to **Stan**, even very complex models tend to converge well in a reasonable amount of time.

While **brms** comes with a lot of built-in response distributions (usually called *families* in R), there is still a long list of distributions which are not natively supported. The present notebook will explain how to specify such *custom families* in **brms**. By doing that, users can benefit from the modeling flexibility and post-processing options of **brms** even when applying self-defined response distributions.

Case Study 1: Beta-Binomial Models

For the first case study, we will use the `cbpp` data of the **lme4** package (Bates et al. 2015), which describes the development of Contagious Bovine Pleuropneumonia (CBPP; an infectiouns lung disease in cattle) in Africa. The data set contains four variables: `period` (the time period), `herd` (a factor identifying the cattle herd), `incidence` (number of new disease cases for a given herd and time period), as well as `size` (the herd size at the beginning of a given time period).

```
data("cbpp", package = "lme4")
head(cbpp)
```

```
  herd incidence size period
1    1         2   14      1
2    1         3   12      2
3    1         4    9      3
4    1         0    5      4
5    2         3   22      1
6    2         1   18      2
```

In a first step, we will be predicting `incidence` using a simple binomial model, which will serve as our baseline model. For observed number of events y (`incidence` in our case) and total number of trials T (`size`), the probability mass function of the binomial distribution is defined as

$$p(y|T,\pi) = \binom{T}{y}\pi^y(1-\pi)^{N-y}$$

where π is the event probability. In the classical binomial model, we will directly predict π on the logit-scale, which means that for each observation i we compute the success probability π_i as

$$\pi_i = \frac{\exp(\eta_i)}{1+\exp(\eta_i)}$$

where η_i is the linear predictor term of observation i. Predicting `incidence` by `period` and a varying intercept of `herd` is straight forward in **brms**:

```
fit1 <- brm(incidence | trials(size) ~ period + (1|herd),
            data = cbpp, family = binomial())
```

In the summary output, we see that the incidence probability varies substantially over herds but reduces over the course of the time as indicated by the negative coefficients of `period`.

```
summary(fit1)
```

```
 Family: binomial
  Links: mu = logit
Formula: incidence | trials(size) ~ period + (1 | herd)
   Data: cbpp (Number of observations: 56)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000

Group-Level Effects:
~herd (Number of levels: 15)
              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(Intercept)     0.77      0.24     0.40     1.29       1019 1.01

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept    -1.42      0.26    -1.93    -0.93       2137 1.00
period2      -1.00      0.31    -1.62    -0.41       4000 1.00
period3      -1.14      0.34    -1.81    -0.51       4000 1.00
period4      -1.63      0.44    -2.54    -0.83       4000 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).
```

A drawback of the binomial model is that – after taking into account the linear predictor – its variance is fixed to $\text{Var}(y_i) = T_i\pi_i(1-\pi_i)$. All variance exceeding this value cannot be not taken into account by the model. This phenomanon is called *overdispersion* and can be dealt with in multiple ways – one of which is going to serve as an illustrative example of how to define custom families in **brms**.

The Beta-Binomial Distribution

The so called *beta-binomial* model is a generalization of the binomial model with an additional parameter to account for overdispersion. In the beta-binomial model, we do not predict the binomial probability π directly but assume it to be beta distributed with hyperparameters $\alpha > 0$ and $\beta > 0$:

$$p(\pi, \alpha, \beta) = \frac{\pi^{\alpha-1}(1-\pi)^{\beta-1}}{B(\alpha, \beta)}$$

where B is the beta function. The α and β parameters are both hard to interpret and generally not recommended for use in regression models. Thus, we will apply a different parameterization with parameters $\mu \in [0, 1]$ and $\phi > 0$, which we will call Beta2:

$$\text{Beta2}(\mu, \phi) = \text{Beta}(\alpha = \mu\phi, \beta = (1-\mu)\phi)$$

The parameters μ and ϕ specify the mean and precision parameter, respectively. By defining

$$\mu_i = \frac{\exp(\eta_i)}{1 + \exp(\eta_i)}$$

we still predict the expected probability by means of our transformed linear predictor (as in the original binomial model), but account for potential overdispersion via the parameter ϕ.

Fitting Beta-Binomial Models

The beta-binomial distribution is not natively supported in **brms** and so we will have to define it ourselves using the `custom_family` function. This function requires the family's name, the names of its parameters (`mu` and `phi` in our case), corresponding link functions (only applied if parameters are prediced), their theoretical lower and upper bounds (only applied if parameters are not predicted), information on whether the distribuion is discrete or continuous, and finally, whether additional non-parameter variables need to be passed to the distribution. For our beta-binomial example, this results in the following custom family:

```
# requires brms 2.1.9 or higher
beta_binomial2 <- custom_family(
  "beta_binomial2", dpars = c("mu", "phi"),
  links = c("logit", "log"), lb = c(NA, 0),
  type = "int", vars = "trials[n]"
)
```

Next, we have to provide the relevant **Stan** functions if the distribution is not defined in **Stan** itself. For the `beta_binomial2` distribution, this is straight forward since the ordinal `beta_binomial` distribution is already implemented.

```
stan_beta_binomial2 <- "
  real beta_binomial2_lpmf(int y, real mu, real phi, int T) {
    return beta_binomial_lpmf(y | T, mu * phi, (1 - mu) * phi);
  }
  int beta_binomial2_rng(real mu, real phi, int T) {
    return beta_binomial_rng(T, mu * phi, (1 - mu) * phi);
  }
"
```

For the model fitting, we will only need `beta_binomial2_lpmf` but `beta_binomial2_rng` will come in handy when it comes to post-processing. If additional data needs to be passed to the distribution, we need to prepare those data as well. In the present example, this is the case for the `size` of the herd and so we go ahead and define:

```
stanvars2 <- stanvar(as.integer(cbpp$size), name = "trials") +
  stanvar(scode = stan_beta_binomial2, block = "functions")
```

In this particular case, we could more elegantly apply the addition argument `trials()` as in the basic binomial model. However, since the present notebook is meant to give a general overview of the topic, we will go with the more general method of specifying custom data via `stanvar`.

We now have all components together to fit our custom beta-binomial model:

```
fit2 <- brm(
  incidence ~ period + (1|herd), data = cbpp,
  family = beta_binomial2, stanvars = stanvars2
)
```

The summary output reveals that the uncertainty in the coefficients of `period` is somewhat larger than in the basic binomial model which is the result of including the overdispersion parameter `phi` in the model. Apart from that, the results look pretty similar.

```
summary(fit2)
```

```
 Family: beta_binomial2
  Links: mu = logit; phi = identity
Formula: incidence ~ period + (1 | herd)
   Data: cbpp (Number of observations: 56)
Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;
         total post-warmup samples = 4000

Group-Level Effects:
~herd (Number of levels: 15)
              Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
sd(Intercept)     0.40      0.27     0.02     1.01        968 1.01

Population-Level Effects:
          Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
Intercept    -1.36      0.27    -1.92    -0.85       4000 1.00
period2      -1.00      0.42    -1.85    -0.18       4000 1.00
period3      -1.26      0.45    -2.17    -0.41       4000 1.00
period4      -1.55      0.51    -2.61    -0.60       4000 1.00

Family Specific Parameters:
    Estimate Est.Error l-95% CI u-95% CI Eff.Sample Rhat
phi    17.63     14.93     5.59    54.95       1319 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).
```

Post-Processing Beta-Binomial Models

Some post-processing methods such as `summary` or `plot` work out of the box for custom family models. However, there are three particularly important methods which require additional input by the user. These are `fitted`, `predict` and `log_lik` computing predicted mean values, predicted response values, and log-likelihood values, respectively. They are not only relevant for their own sake but also provide the basis of many other post-processing methods. For instance, we may be interested in comparing the fit of the binomial model with that of the beta-binomial model by means of approximate leave-one-out cross-validation implemented in method `LOO`, which in turn requires `log_lik` to be working.

The `log_lik` function of a family should be named `log_lik_<family-name>` and have the two arguments `i` (indicating observations) and `draws`. You don't have to worry too much about how `draws` is created. Instead, all you need to know is that parameters are stored in slot `dpars` and data are stored in slot `data`. Generally, parameters take on the form of a $S \times N$ matrix (with S = number of posterior samples and N = number of observations) if they are predicted (as is `mu` in our example) and a vector of size N if they are not predicted (as is `phi`).

We could define the complete log-likelihood function in R directly, or we could expose the self-defined **Stan** functions and apply them. The latter approach is usually more convenient but the former is more stable and the only option when implementing custom families in other R packages building upon **brms**. For the purpose of this case study, we will go with the latter approach

```
expose_functions(fit2, vectorize = TRUE)
```

and define the required `log_lik` functions with a few lines of code.

```
log_lik_beta_binomial2 <- function(i, draws) {
  mu <- draws$dpars$mu[, i]
  phi <- draws$dpars$phi
  trials <- draws$data$trials[i]
  y <- draws$data$Y[i]
  beta_binomial2_lpmf(y, mu, phi, trials)
}
```

Having defined the log-likelihood function, all of the post-processing methods requiring `log_lik` will work as well. For instance, model comparison can simply be performed via

```
LOO(fit1, fit2)
```

```
             LOOIC    SE
fit1        200.53 20.49
fit2        189.62 16.54
fit1 - fit2  10.90  8.51
```

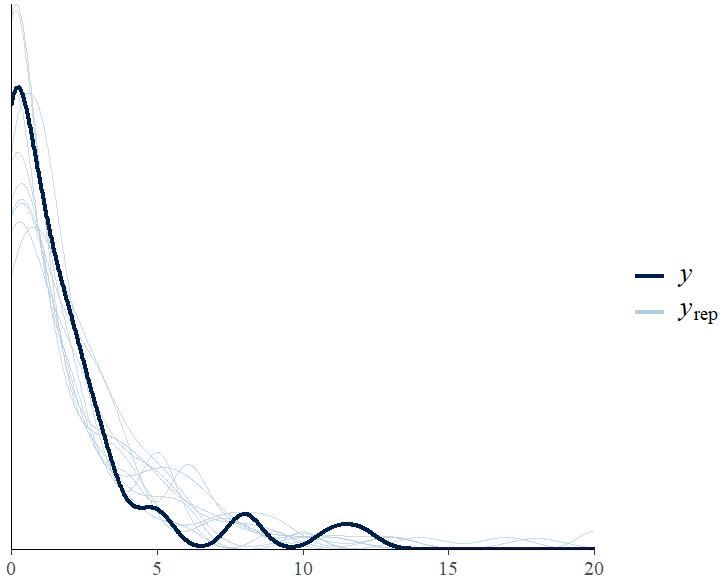
Since smaller `LOOIC` values indicate better predictions, we see that the beta-binomial model fits somewhat better, although the corresponding standard error reveals that the difference is not substantial.

Next, we will define the function necessary for the `predict` method:

```
predict_beta_binomial2 <- function(i, draws, ...) {
  mu <- draws$dpars$mu[, i]
  phi <- draws$dpars$phi
  trials <- draws$data$trials[i]
  beta_binomial2_rng(mu, phi, trials)
}
```

The `predict` function looks pretty similar to the corresponding `log_lik` function, except that we are now creating random samples of the response instead of log-liklihood values. Again, we are using an exposed **Stan** function for convenience. Make sure to add a `...` argument to your `predict` function even if you are not using it since some families require additional arguments. With `predict` working, we can engage for instance in posterior-predictive checking:

```
pp_check(fit2)
```

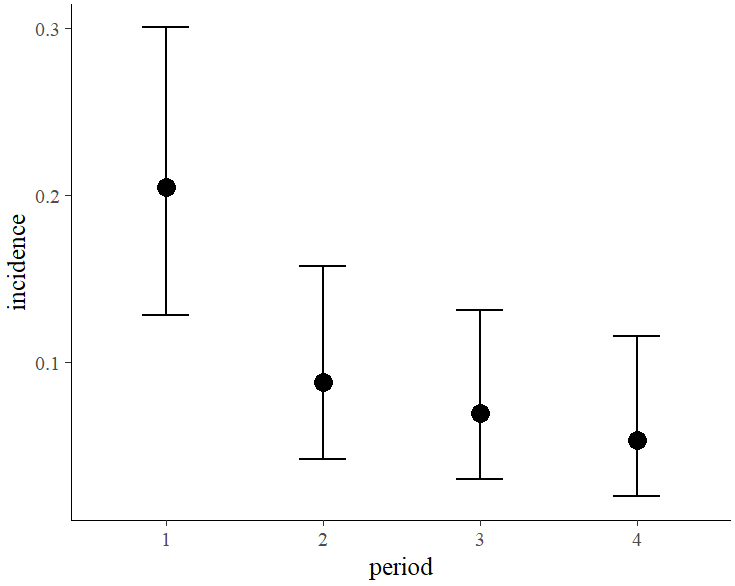


When defining the `fitted` function, you have to keep in mind that it has only a `draws` argument and should compute the mean response values for all observations at once. Since the mean of the beta-binomial distribution is $\mathrm{E}(y) = \mu T$ definition of the corresponding `fitted` function is not too complicated but we need to get the dimension of parameters and data in line.

```
fitted_beta_binomial2 <- function(draws) {
  mu <- draws$dpars$mu
  trials <- draws$data$trials
  trials <- matrix(trials, nrow = nrow(mu), ncol = ncol(mu), byrow = TRUE)
  mu * trials
}
```

A post-processing method relying directly on `fitted` is `marginal_effects`, which allows visualizing effects of predictors.

```
marginal_effects(fit2, new_objects = list(trials = 1))
```



For ease of interpretation, we have set `trials` to 1 so that the y-axis of the above plot indicates probabilities.

Case Study 2: Mean-Variance Gamma Models

For the second case study, we will use a data set of the housing rents in Munich from 1999 (Fahrmeier et al. 2013), which is available in the **gamlss.data** package (Rigby & Stasinopoulos, 2005). The data contain 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.

```
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
```

Our goal is to predict the rent per square meter by the size of the apartment and the construction year. The rent per square meter is naturally positive, and so we choose a gamma distribution as our response distribution. Typically the gamma distribution is parameterized in terms of shape α and rate β with density

$$p(y|\alpha, \beta) = \frac{\beta^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp(-\beta y)$$

where Γ is the gamma function. This distribution has mean $\mathrm{E}(y) = \frac{\alpha}{\beta}$ and variance $\mathrm{Var}(y) = \frac{\alpha}{\beta^2}$. For regression models, we usually want one of the parameters to represent the mean. Accordingly, in **brms** as well as in other model fitting packages, the native `Gamma` family is parameterized in terms of mean μ and shape α:

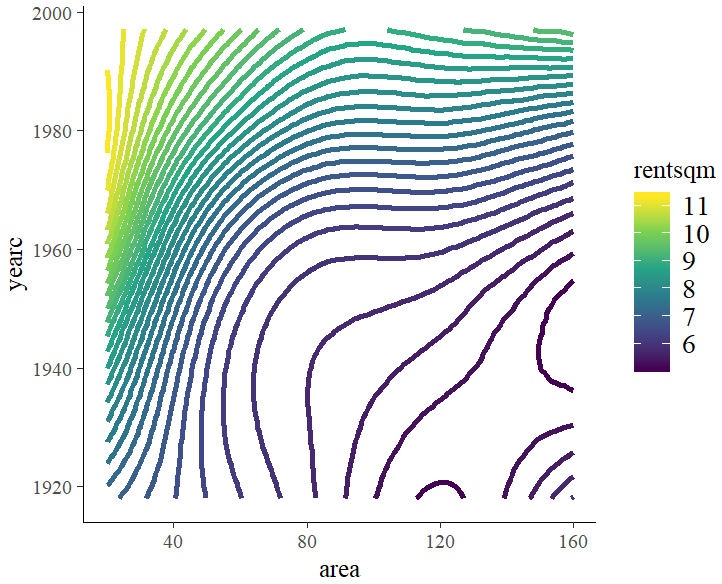
$$p(y|\mu, \alpha) = \frac{(\alpha/\mu)^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp\left(-\frac{\alpha y}{\mu}\right)$$

so that $\mathrm{E}(y) = \mu$ and $\mathrm{Var}(y) = \frac{\mu^2}{\alpha}$. Before we engage in a more complex model requiring a custom family, we are going to predict the (log) mean of `rentsqm` via a two dimensional tensor spline of `area` and `yearc` as the form of the relationship is unclear a-priori. Further, since observations belong to different districts of Munich, which may differ in their average rent per square meter, we will add a varying intercept of `district`. The `shape` parameter is assumed to be constant across observations.

```
fit3 <- brm(
  rentsqm ~ t2(area, yearc) + (1 | district),
  data = rent99, family = Gamma("log"),
  cores = 2, chains = 2
)
```

As usual when dealing with splines, the summary output won't tell us much and so we directly plot the model implied predictions after having made sure the model converged.

```
marginal_effects(fit3, "area:yearc", surface = TRUE)
```



In the plot, the combined effect of `area` and `yearc` is shown, clearly demonstrating an interaction between the variables. In particular, housing rents appear to be highest for small and relatively new apartments.

The Mean-Variance Gamma Distribution

It is desirable to not only predict the mean of the response but also its variance, since the variation in the rents per square meter may vary across years and apartment sizes. For this purpose, the parameterization applied above is suboptimal, because the variance depends on the mean parameter μ. As we may very reasonably wish to model the variance independtly of the mean, we can once again use the custom family framework of **brms** to define a response distribution tailored to our needs with mean μ and variance v:

$$p(y|\mu, v) = \frac{(\alpha/\mu)^\alpha}{\Gamma(\alpha)} y^{\alpha-1} \exp\left(-\frac{\alpha y}{\mu}\right)$$

Fitting Mean-Variance Gamma Models

First, we will define the custom family function. We choose a `log` link for both μ and v to ensure that the transformed linear predictors are always positive, thus matching the definition spaces of the parameters.

```
gamma2 <- custom_family(
  "gamma2", dpars = c("mu", "v"),
  links = c("log", "log"), lb = c(0, 0)
)
```

Next, we define the custom log-density **Stan** function via a reparameterzation of the built-in gamma distribution.

```
stan_gamma2 <- "
  real gamma2_lpdf(real y, real mu, real v) {
    return gamma_lpdf(y | mu * mu / v, mu / v);
  }
"
```

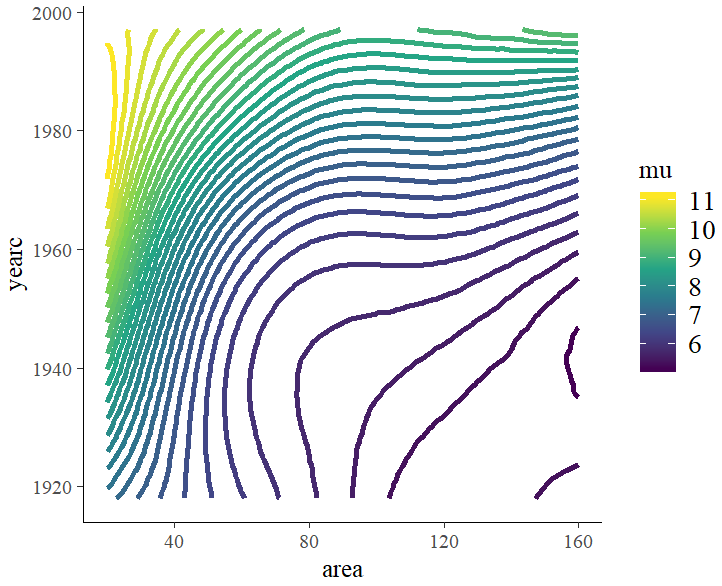
We are now ready to specify the model. This time, we predict both (log) μ and (log) v via two dimensional tensor splines of `area` and `yearc`. Also, we add varying intercepts of `district` for both response parameters and model them as correlated via the ID-syntax of **brms**. The `p` in `(1 | p | district)` is chosen arbitrarily and just ensures that the varying intercepts are modeled as correlated.

```
bform4 <- bf(
  rentsqm ~ t2(area, yearc) + (1 | p | district),
  v ~ t2(area, yearc) + (1 | p | district)
)
stanvars4 <- stanvar(scode = stan_gamma2, block = "functions")
fit4 <- brm(
  bform4, data = rent99, family = gamma2,
  stanvars = stanvars4, cores = 2, chains = 2
)
```

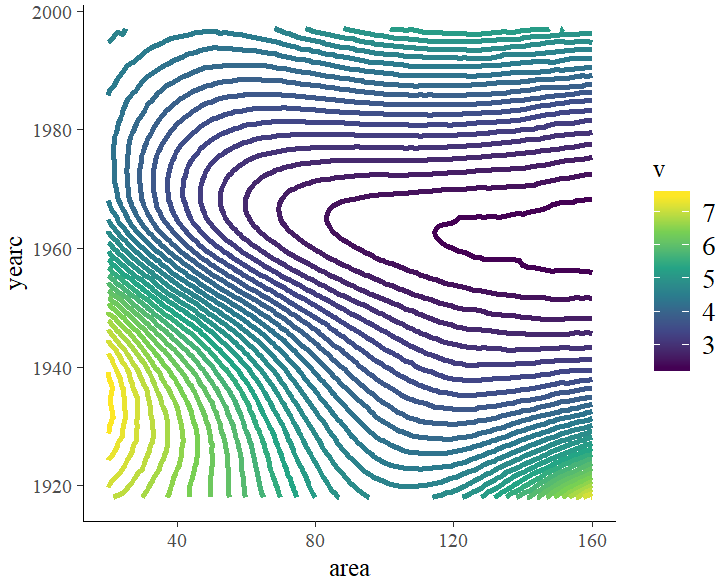
Post-Processing Mean-Variance Gamma Models

Again, we directly plot the model implied predictions for `mu` and `v` after having made sure the model has converged.

```
marginal_effects(fit4, "area:yearc", surface = TRUE, dpar = "mu")
```




```
marginal_effects(fit4, "area:yearc", surface = TRUE, dpar = "v")
```



In the first plot, the predictions of the mean `mu` are shown and look comparable to what we have seen for the simpler model above. However, as visible in the plot of v, the variation in the rent per square meter varies substantially with construction year and apartment size: It is highest for relatively small and old apartments and lowest for medium to large apartments built around the 1960s.

Similar to what we did in the first case study, we can go ahead and define `fitted`, `predict` and `log_lik` post-processing functions for our custom `gamma2` family.

```
fitted_gamma2 <- function(draws) {
  draws$dpars$mu
}
```

```
predict_gamma2 <- function(i, draws, ...) {
  mu <- draws$dpars$mu[, i]
  v <- draws$dpars$v[, i]
```

```
  alpha <- mu^2 / v
  beta <- mu / v
  rgamma(length(mu), alpha, beta)
}
```

```
log_lik_gamma2 <- function(i, draws) {
  mu <- draws$dpars$mu[, i]
  v <- draws$dpars$v[, i]
  alpha <- mu^2 / v
  beta <- mu / v
  y <- draws$data$Y[i]
  dgamma(y, alpha, beta, log = TRUE)
}
```

We can, for instance, make use of the `log_lik_gamma2` function to investigate via `LOO` if modeling the variance in addition to the mean improves model fit.

```
LOO(fit3, fit4)
```

```
               LOOIC    SE
fit3        13270.82 90.55
fit4        12944.12 87.15
fit3 - fit4   326.69 47.54
```

As smaller `LOOIC` values indicate better predictions, we see that modeling the variance in addition to the mean improves model fit considerably.

Conclusion

The possibility of using custom families with **brms** extends its modeling flexibility even further. Users can specify their models in the intuitive and powerful formula syntax of **brms** and combine them with their own response distributions. These models are fully compatible with nearly all modeling and post-processing options, thus extending the package in a natural way without the need to make any trade-offs.

References

Bates, D., Maechler, M., Bolker, B., & Walker S. (2015). Fitting Linear Mixed-Effects Models Using lme4. *Journal of Statistical Software*, 67(1), 1–48. doi:10.18637/jss.v067.i01.

Bürkner, P.C. (2017a). brms: An R Package for Bayesian Multilevel Models Using Stan. *Journal of Statistical Software*, 80(1), 1–28. doi:10.18637/jss.v080.i01

Bürkner, P.C. (2017b). Advanced Bayesian Multilevel Modeling with the R Package brms. *arXiv preprint*.

Carpenter, B., Gelman, A., Hoffman, M., Lee, D., Goodrich, B., Betancourt, M., ... & Riddell, A. (2016). Stan: A probabilistic programming language. *Journal of Statistical Software*, 20(2), 1–37.

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

Rigby, R. A. & Stasinopoulos D. M. (2005). Generalized additive models for location, scale and shape (with discussion). *Applied Statistics*, 54(3), 507–554.