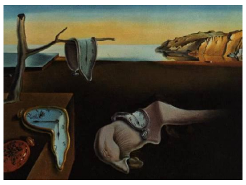

Journal of Statistical Software

November 2021, Volume 100, Issue 4. doi: 10.18637/jss.v100.i04

bamlss: A Lego Toolbox for Flexible Bayesian Regression (and Beyond)

Nikolaus Umlauf Universität Innsbruck

Nadja Klein Humboldt Universität zu Berlin

Thorsten Simon Universität Innsbruck

Achim Zeileis Universität Innsbruck

Abstract

Over the last decades, the challenges in applied regression and in predictive modeling have been changing considerably: (1) More flexible regression model specifications are needed as data sizes and available information are steadily increasing, consequently demanding for more powerful computing infrastructure. (2) Full probabilistic models by means of distributional regression – rather than predicting only some underlying individual quantities from the distributions such as means or expectations – is crucial in many applications. (3) Availability of Bayesian inference has gained in importance both as an appealing framework for regularizing or penalizing complex models and estimation therein as well as a natural alternative to classical frequentist inference. However, while there has been a lot of research on all three challenges and the development of corresponding software packages, a modular software implementation that allows to easily combine all three aspects has not yet been available for the general framework of distributional regression.

To fill this gap, the R package **bamlss** is introduced for Bayesian additive models for location, scale, and shape (and beyond) – with the name reflecting the most important distributional quantities (among others) that can be modeled with the software. At the core of the package are algorithms for highly-efficient Bayesian estimation and inference that can be applied to generalized additive models or generalized additive models for location, scale, and shape, or more general distributional regression models. However, its building blocks are designed as "Lego bricks" encompassing various distributions (exponential family, Cox, joint models, etc.), regression terms (linear, splines, random effects, tensor products, spatial fields, etc.), and estimators (MCMC, backfitting, gradient boosting, lasso, etc.). It is demonstrated how these can be easily combined to make classical models more flexible or to create new custom models for specific modeling challenges.

Keywords: backfitting, distributional regression, gradient boosting, MCMC, penalization, probabilistic forecasting, R.

1. Introduction

Many modern modeling tasks necessitate flexible regression tools that can deal with: (1) Large data sets that can be both long (many observations) and/or wide (many variables or complex effect types). (2) Probabilistic forecasts that capture the entire distribution and not only its mean or expectation. (3) Enhanced inference infrastructure, typically Bayesian, broadening classical frequentist methodology. A popular framework to combine flexible regression with probabilistic modeling are generalized additive models (GAMs, Hastie and Tibshirani 1990), later extended to generalized additive models for location, scale, and shape (GAMLSS, Rigby and Stasinopoulos 2005), also known as Bayesian structured additive distributional regression (Klein, Kneib, Lang, and Sohn 2015c) which encompasses (generalized) linear models (GLMs, Nelder and Wedderburn 1972) as special cases. Bayesian inference in these models can be seen as a natural framework for penalizing flexible model terms and to overcome potential problems with p values and classical null hypothesis significance testing (Wasserstein and Lazar 2016). However, when fitting such models to big data – long and/or wide – classical estimation techniques using standard algorithms like iteratively weighted least squares (IWLS, Gamerman 1997) or Markov chain Monte Carlo (MCMC) might not be feasible. Instead, regularized estimation techniques such as lasso or boosting (Friedman, Hastie, and Tibshirani 2010; Mayr, Fenske, Hofner, Kneib, and Schmid 2012) might be necessary or further advanced custom algorithms (Wood 2017). Hence, to facilitate addressing all challenges and needs simultaneously – independent of a specific estimation strategy and/or fitting algorithm – the **bamlss** package for the R system for statistical computing (R Core Team 2021) implements a modular "Lego toolbox", extending the work of Umlauf, Klein, and Zeileis (2018). In this framework not only the response distribution is a "Lego brick" (as in a classical GLM) or the regression terms (as in a GAM) but also the estimation algorithm such as a specific MCMC sampler.

The idea of a "Lego toolbox" for regression models has of course been around for some time; in some implementations, Bayesian and frequentist, there is not only the possibility to easily implement new distributions, but also model terms, from splines to neural networks to regression trees. In some implementations optimization routines may also be exchanged. The following is a list of well-known packages for regression models in the R ecosystem, whose implementations are designed to be extremely flexible.

- *GAMs and GAMLSSs* are available in a number of packages, most notably the **mgcv** package (Wood 2017) and also the **gamlss** family of packages (Stasinopoulos, Rigby, Heller, Voudouris, and Bastiani 2017; Rigby, Stasinopoulos, Heller, and Bastiani 2019) and **VGAM** (Yee 2010). The latter two are notable for their support of a wide range of response distributions. While **VGAM** is restrictive with respect to the integration of flexible model terms, the **gamlss** package also supports (user-defined) smooth additive terms of general type (e.g., neural networks and regression trees), however, inference is mainly supported only for linear model terms. In contrast, **mgcv** excels at providing highly-optimized algorithms for general smooth models (Wood, Pya, and Säfken 2016), including inference, as well as the dedicated `bam()` function for big data that is long and/or wide (Wood, Li, Shaddick, and Augustin 2017). All these packages rely on frequentist estimation strategies. Moreover, the package provides sophisticated infrastructure for generating new classes of smooth terms (which is fully adopted by the **bamlss** package).

- *Bayesian inference* is not only an increasingly popular alternative to classical frequentist inference, it is also particularly attractive for hierarchical or multilevel models and for penalizing regression effects through suitable prior distributions. Also, fully Bayesian approaches using MCMC are appealing in flexible regression models for obtaining credible intervals from the posterior samples. The **brms** package (Bürkner 2017) is notable for providing a standard R workflow for estimating Bayesian multilevel models using **Stan** (Carpenter *et al.* 2017). Also, the above-mentioned **mgcv** package supports estimation of Bayesian GAMs via its `jagam()` function (Wood 2016) based on **JAGS** (Plummer 2003).

  For more flexibility, going beyond these capabilities, it is in principle possible to directly implement custom models using general purpose MCMC software like **JAGS**, **Stan**, or the BUGS family of packages (Lunn, Thomas, Best, and Spiegelhalter 2000; Goudie, Turner, Angelis, and Thomas 2020). However, for complex models – e.g., using large data sets, spatial effects, or higher-order interactions – sampling times from these generic MCMC engines can become long, sometimes prohibitively long. This has been addressed by dedicated packages for Bayesian additive models, e.g., with the standalone package **BayesX** (Brezger, Kneib, and Lang 2005; Belitz, Brezger, Klein, Kneib, Lang, and Umlauf 2015) being the first to provide highly-efficient sampling schemes for very large data sets as well as spatial/multilevel models and structured additive distributional regression. An R interface is available in **R2BayesX** (Umlauf, Adler, Kneib, Lang, and Zeileis 2015). Instead of fully Bayesian MCMC it is also possible to employ posterior mean estimation via the integrated nested Laplace approximation to estimate flexible Bayesian regression models. This is provided in the comprehensive R package **INLA** (Rue, Martino, and Chopin 2009), popular for estimating complex spatial Bayesian regression models (see e.g., Lindgren and Rue 2015; Bivand, Gómez-Rubio, and Rue 2015).

- *Regularized estimation* and explicit variable selection might be necessary, though, for going beyond the models described above, especially for large/wide data with many potential regressors and corresponding effects/interactions/etc. Widely-used approaches for this include the lasso, e.g., as available for GLM-type models in the R package **glmnet** (Friedman *et al.* 2010), or gradient boosting as available for GAMLSS-type models in the R package **gamboostLSS** (Hofner, Mayr, and Schmid 2016). However, many packages do not cover the Bayesian posterior estimation parts.

In summary, the discussion above highlights that many different packages with different strengths are already available in R. However, a package combining all the aspects above in a single framework is not readily available as there are typically limitations with respect to the inferential framework, the distributions and/or complexity of the models supported, or the estimation techniques and fitting algorithms. The package **bamlss**, available from the Comprehensive R Archive Network at `https://CRAN.R-project.org/package=bamlss`, tries to fill this gap with a modular "Lego" approach to flexible Bayesian regression providing:

- The usual R "look & feel" for regression modeling.
- Estimation of classic (GAM-type) regression models (Bayesian or frequentist).
- Estimation of flexible (GAMLSS-type) distributional regression models.

- An extensible "plug & play" approach for regression terms.
- Modular combinations of fitting algorithms and samplers.

Especially the last item is notable because the models in **bamlss** are not limited to a specific estimation algorithm but different engines can be plugged in without necessitating changes in other aspects of the model specification (such as response distributions or regression terms). By default **bamlss** is using IWLS-based backfitting for optimizing the model and IWLS-based MCMC for sampling from the posterior distribution. However, alternative optimizers and samplers are also implemented that support lasso or boosting, and more. Moreover, the package builds on the well-established **mgcv** infrastructure for smooth model terms, uses R's formula syntax for model specification, and provides standard extractor methods like `summary()`, `plot()`, `predict()`, etc.

The remainder of this paper is as follows. In Section 2, three motivating examples illustrate the first steps using **bamlss** and show cases the flexibility of the provided infrastructure. Section 3 introduces the flexible regression framework in more detail. A thorough introduction of the R package **bamlss**, describing the most important building blocks for developing families, model terms and estimation algorithms, is then given in Section 4. In Section 5 we highlight the unified modeling approach using a complex distributional regression model for lighting counts in complex terrain. Further details and examples about the **bamlss** package can be found online at `http://www.bamlss.org/`.

2. Motivating examples

This section gives a first quick overview of the functionality of the package. The first example demonstrates that the usual "look & feel" when using well-established model fitting functions like `glm()` is an elementary part of **bamlss**, i.e., first steps and basic handling of the package should be relatively simple. The second example shows that the package can deal with a variety of different model terms and that model fitting functions can easily be exchanged, here, we exemplify this feature by applying a lasso-type estimation engine. The third example then explains how full distributional regression models can be estimated and show cases once more the flexibility of the provided modeling infrastructure.

2.1. Basic Bayesian regression: Logit model

This example data is taken from the **AER** package (Kleiber and Zeileis 2008) and is about labor force participation (yes/no) of women in Switzerland 1981 (Gerfin 1996). The **bamlss** package and the data can be loaded with

```
R> library("bamlss")
R> data("SwissLabor", package = "AER")
```

The data frame contains 872 observations of 6 variables, where some of them might have a nonlinear influence on the response labor `participation`. Now, a standard Bayesian binomial logit model using the default MCMC algorithm can be fitted (sampler function `sam_GMCMC()`, see also Section 4 for other options). The MCMC algorithm uses iteratively weighted least squares (IWLS, Gamerman 1997, for more details see Section 3.2) proposals, which have very good mixing properties and computational advantages when using very large data sets (Lang, Umlauf, Wechselberger, Harttgen, and Kneib 2014). First, the model formula is specified with

```
R> f <- participation ~ income + age + education +
+    youngkids + oldkids + foreign + I(age^2)
```

Then, to reproduce the results the seed of the random number generator is set and the model is estimated by

```
R> set.seed(123)
R> b <- bamlss(f, family = "binomial", data = SwissLabor,
+    n.iter = 1200, burnin = 200, thin = 1)
```

Note that the default number of iterations (`n.iter`) for the MCMC sampler is 1200, the burnin-phase `burnin` is 200 and thinning (`thin`) is 1. The reason is that during the modeling process, users usually want to obtain first results rather quickly. Afterwards, if a final model is estimated the number of iterations of the sampler is usually set much higher to get close to i.i.d. samples from the posterior distribution. To obtain reasonable starting values for the MCMC sampler we run a backfitting algorithm that optimizes the posterior mode. Using the main model fitting function `bamlss()` all model fitting engines can be exchanged, which is explained in detail in Section 4 and the application Section 5. The default model fitting engines use family objects (see also Section 4), similar to the families that can be used with the `glm()` function, which enables easy implementation of new distributions (models).

Note that the model contains a quadratic term for variable `age` in order to capture nonlinearities. The resulting object `b` is of class `"bamlss"` for which standard extractor functions like `summary()`, `coef()`, `plot()`, `predict()`, etc. are available. The model summary output is printed by

```
R> summary(b)
```

```
Call:
bamlss(formula = f, family = "binomial", data = SwissLabor)
---
Family: binomial
Link function: pi = logit
*---
Formula pi:
---
participation ~ income + age + education + youngkids + oldkids +
    foreign + I(age^2)
-
Parametric coefficients:
                Mean     2.5%      50%    97.5% parameters
(Intercept)  6.15503  1.55586  5.99204 11.11051      6.196
income      -1.10565 -1.56986 -1.10784 -0.68652     -1.104
age          3.45703  2.05897  3.44567  4.79139      3.437
```

```
education    0.03354 -0.02175  0.03284  0.09223      0.033
youngkids   -1.17906 -1.51099 -1.17683 -0.83047     -1.186
oldkids     -0.24122 -0.41231 -0.24099 -0.08054     -0.241
foreignyes   1.16749  0.76276  1.17035  1.55624      1.168
I(age^2)    -0.48990 -0.65660 -0.49205 -0.31968     -0.488
-
Acceptance probability:
        Mean   2.5%    50% 97.5%
alpha 0.8759 0.3230 0.9941     1
---
Sampler summary:
-
DIC = 1033.325 logLik = -512.7258 pd = 7.8734
runtime = 1.115
---
Optimizer summary:
-
AICc = 1033.737 edf = 8 logLik = -508.7851
logPost = -571.3986 nobs = 872 runtime = 0.012
```

and is based on MCMC samples, which suggest "significant" effects for all covariates, except for variable `education`, since the 95% credible interval contains zero. In addition, the acceptance probabilities `alpha` are reported, i.e., the acceptance probability of the sample candidate based on the proposal and the posterior distribution which is calculated in each iteration, indicating proper behavior of the MCMC algorithm. The column `parameters` shows respective posterior mode estimates of the regression coefficients, which are calculated by the upstream optimizer algorithm (note that the column is named parameters, because optimizer functions can in principle return any type of parameters). Besides, more results from the optimizer are reported at the very end of the output: the corrected AIC (`AICc`, Hurvich and Tsai 1989; Cavanaugh 1997), the equivalent degrees of freedom (`edf`), the log-likelihood (`logLik`), etc. In addition, there are also extractor functions in **bamlss** for information criteria like the DIC (function `DIC()`) and the widely applicable information criterion (WAIC, Watanabe 2010, function `WAIC()`), or the out-of-sample continuous rank probability score (CRPS, Gneiting and Raftery 2007; Gneiting, Balabdaoui, and Raftery 2007, function `CRPS()`). Note that `CRPS()` approximates numerically, while the **scoringRules** package (Jordan, Krüger, and Lerch 2019) can compute the CRPS very efficiently for some distributions. The usage of the provided functions is similar to the generic `AIC()` and `BIC()`, e.g., the DIC can be computed with

```
R> DIC(b)
```

```
      DIC      pd
 1033.325 7.87343
```

Before proceeding the analysis, users usually perform additional convergence checks of the MCMC chains by looking at traceplots and autocorrelation (besides acceptance probabilities).

```
R> plot(b, which = c("samples", "max-acf"))
```

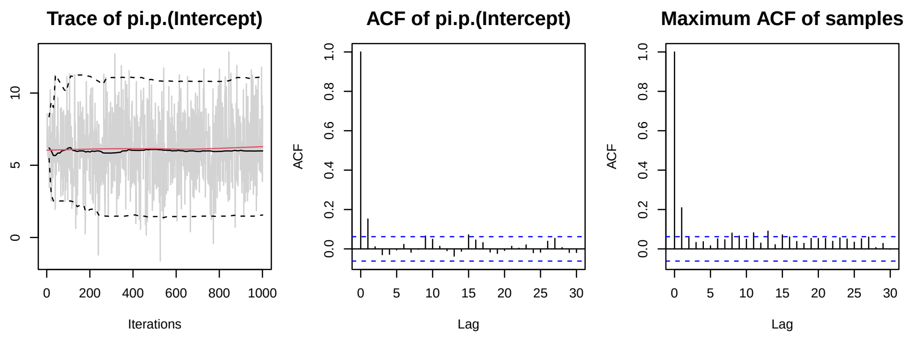

Figure 1: Logit model, MCMC trace (left panel), autocorrelation for the intercept (middle panel), maximum autocorrelation for all parameters (right panel).

These are visualized in Figure 1 and reveal approximate convergence of the MCMC chains, i.e., there is no visible trend, and the very low autocorrelation shown for the intercept and the maximum autocorrelation calculated as the maximum for each lag across all parameters suggest close to i.i.d. samples from the posterior distribution. As mentioned above, the user could also increase the number iterations and the burnin-phase, as well as adapt the thinning parameter (arguments `n.iter`, `burnin` and `thin`), to make the significant bar at lag one disappear. Note that the function call would compute all trace and autocorrelation plots, however, for convenience we only show plots for the intercept. In addition, samples can also be extracted using function `samples()`, which returns an object of class `"mcmc"`, a class provided by the **coda** package (Plummer, Best, Cowles, and Vines 2006) which includes a rich infrastructure for further convergence diagnostic checks, e.g., Gelman and Rubin's convergence diagnostic (Gelman and Rubin 1992; Brooks and Gelman 1998) or Heidelberger and Welch's convergence diagnostic (Heidelberger and Welch 1981, 1983).

Model predictions on the probability scale can be obtained by the `predict()` method, e.g., to visualize the effect of covariate `age` on the probability we can create a new data frame for prediction

```
R> nd <- data.frame(income = 11, age = seq(2, 6.2, length = 100),
+    education = 12, youngkids = 1, oldkids = 1, foreign = "no")
```

Afterwards, we predict for both cases of variable `foreign`

```
R> nd$p_swiss <- predict(b, newdata = nd, type = "parameter", FUN = c95)
R> nd$foreign <- "yes"
R> nd$p_foreign <- predict(b, newdata = nd, type = "parameter", FUN = c95)
```

The `predict()` method is applied on all MCMC samples and argument `FUN` specifies a function that can be applied on the predictor or distribution parameter samples. The default is the `mean()` function, however, in this case we additionally extract the empirical 2.5% and 97.5% quantiles using function `c95()` to obtain credible intervals (note, individual samples can be extracted by passing `FUN = identity`, i.e., this way users can easily generate their own statistics). Then, the estimated effect can be visualized with

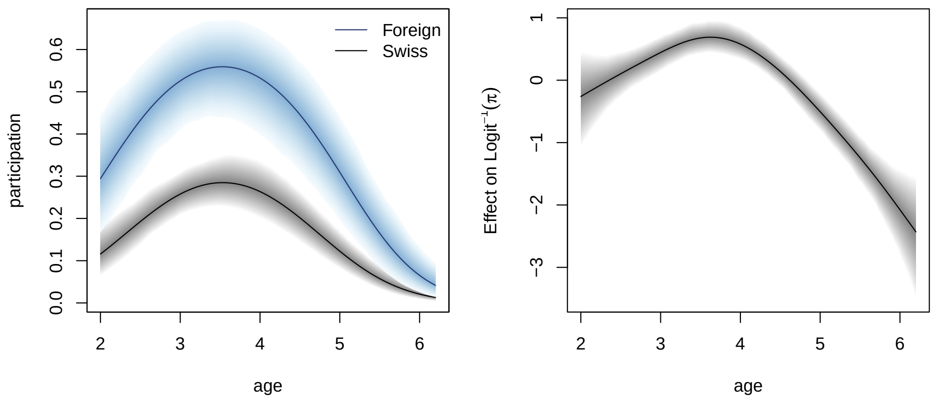

Figure 2: Left panel, quadratic polynomial effect of covariate `age` on estimated probabilities for both cases, `foreign "yes"` and `"no"`. Right panel, effect on $\text{Logit}^{-1}(\pi)$ of variable `age` using regression splines (see Section 2.2). The solid lines represent mean estimates, the shaded areas show 95% credible intervals.

```
R> blues <- function(n, ...) hcl.colors(n, "Blues", rev = TRUE)
R> plot2d(p_swiss ~ age, data = nd, ylab = "participation",
+    ylim = range(c(nd$p_swiss, nd$p_foreign)), fill.select = c(0, 1, 0, 1))
R> plot2d(p_foreign ~ age, data = nd, add = TRUE,
+    fill.select = c(0, 1, 0, 1), axes = FALSE,
+    s2.col = blues, col.lines = blues(1))
```

The estimates are shown in Figure 2 and suggest a clear difference for the effect of `age` between both cases of factor variable `foreign`.

2.2. Flexible model terms and estimators

Using the flexible infrastructure of **bamlss**, model terms can be easily exchanged. To give a first impression of the modeling capabilities, we again use the `SwissLabor` data and binomial logit model of Section 2.1, however, in this example we use regression splines to capture the nonlinear effect variable `age`.

As noted in the introduction, the **bamlss** package leverage the infrastructure from the R package **mgcv** (Wood 2021) for setting up the design and penalty matrices for smooth terms by calling **mgcv**'s `smooth.construct()` or `smoothCon()`, i.e., new user-defined smooth terms can also be added by providing new classes for the generic functions. To estimate a spline model instead of a polynomial model for variable `age` the model formula only needs to be slightly adapted

```
R> f <- participation ~ income + education +
+    youngkids + oldkids + foreign + s(age, k = 10)
```

The function `s()` is the smooth term constructor from the **mgcv** package, the default of `s()` are thin-plate regression splines with `k = 10` basis functions. The model is again fitted by

```
R> set.seed(123)
R> b <- bamlss(f, family = "binomial", data = SwissLabor)
```

Description	Formula
Linear effects: $\mathbf{X}\boldsymbol{\beta}$	`x1 + x2 + x3`
Nonlinear effects of continuous covariates: $f(\mathbf{x}) = f(x_1)$	`s(x1)`
Two-dimensional surfaces: $f(\mathbf{x}) = f(x_1, x_2)$	`s(x1,x2)`, `te(x1,x2)` or `ti(x1,x2)` (higher dimensional terms possible).
Spatially correlated effects: $f(\mathbf{x}) = f_{spat}(x_s)$	`s(xs, bs = "mrf", xt = list(penalty = K))`, where `xs` is a factor indicating the discrete regional information and `K` is a supplied penalty matrix. Other options within the `xt` argument are possible, please see the documentation of `smooth.construct.mrf.smooth.spec()`.
Varying coefficients: $f(\mathbf{x}) = x_1 f(x_2)$	`s(x2, by = x1)`
Spatially varying effects: $f(\mathbf{x}) = x_1 f_{spat}(x_s)$ or $f(\mathbf{x}) = x_1 f(x_2, x_3)$	`s(xs, bs = "mrf", xt = list(penalty = K), by = x1)`, `s(x2, x3, by = x1)` or `te(x2, x3, by = x1)`
Random intercepts with cluster index c: $f(\mathbf{x}) = \beta_c$	`s(id, bs = "re")`, where `id` is a factor of cluster indices.
Random slopes with cluster index c: $f(\mathbf{x}) = x_1 \beta_c$	`s(id, x1, bs = "re")`, as above with continuous covariate `x1`.

Table 1: Commonly used model term specifications with respective R formula syntax.

The estimated nonlinear effect can be plotted instantly by typing

```
R> plot(b, term = "s(age)")
```

The estimated effect based on regression splines is shown in the right panel of Figure 2 and reveals that the quadratic polynomial seems to capture the nonlinearity appropriately.

To give a better impression what type of model terms can be used with the **bamlss** framework Table 1 lists commonly-used specifications.

Besides the supported infrastructure from the **mgcv** package, it is also possible to implement completely new model terms that may follow different setups compared to the basis functions approach (see also Appendix B for an example using growth curves). Moreover, using **bamlss**, estimation engines can also be exchanged. To give an example we estimate the nonlinear `age` effect in the `SwissLabor` example using a fused lasso algorithm (see also Section 5 for a complex example using gradient boosting optimization). The algorithm performs variable selection in combination with factor fusion (clustering) and can also be used to identify interpretable nonlinearities. Methodological details on lasso-type penalization using **bamlss** are provided in Groll, Hambuckers, Kneib, and Umlauf (2019). To apply the fused lasso, the numeric variable `age` is categorized using empirical quantiles, e.g., with

```
R> SwissLabor$cage <- cut(SwissLabor$age,
+    breaks = quantile(SwissLabor$age, prob = seq(0, 1, length = 10)),
+    include.lowest = TRUE, ordered_result = TRUE)
```

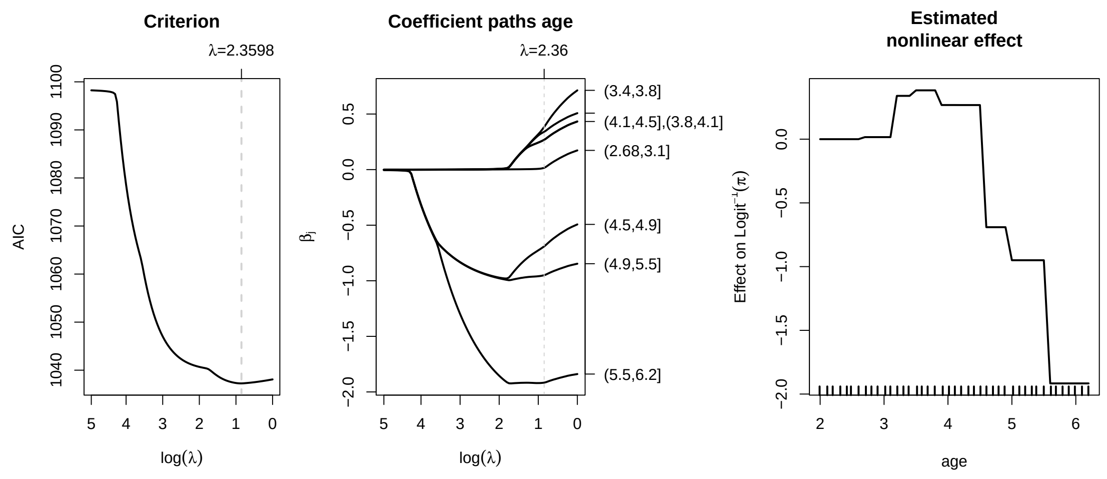

Figure 3: Left panel, AIC curve with optimum shrinkage parameter λ of the lasso example model. The middle panel shows the corresponding coefficient paths for variable `cage`. The right panel displays the respective estimated effect.

The formula for the fused lasso model is then specified with the special `la()` model term constructor function provided in **bamlss**:

```
R> f <- participation ~ income + education + youngkids + oldkids + foreign +
+    la(cage, fuse = 2)
```

where argument `fuse` specifies the type of fusion (nominal fusion `fuse = 1`, ordered fusion `fuse = 2`). To estimate the fused lasso model only the default `optimizer` function in the `bamlss()` wrapper function call needs to exchanged

```
R> b <- bamlss(f, family = "binomial", data = SwissLabor,
+    optimizer = opt_lasso, sampler = FALSE,
+    criterion = "AIC", upper = exp(5), lower = 1)
```

The optimum shrinkage parameter λ is selected by the AIC (another option is `criterion = "BIC"`). Arguments `upper` and `lower` determine the search interval of λ, per default `nlambda = 100` values are generated. Note that no MCMC sampling is used after the `opt_lasso()` estimation engine is applied, argument `sampler = FALSE` in the `bamlss()` call.

The AIC curve and the coefficient paths including the optimum shrinkage parameter λ can be visualized with

```
R> pathplot(b)
```

Figure 3 shows the AIC curve and coefficient paths for `cage`. The AIC curve assumes a minimum at the vertical gray dashed line. The coefficient paths obviously depict that the algorithm can either shrink categories out of the model (shrink to zero), or even fuses them. In the right panel of Figure 3, the estimated effect of the categorized variable `age` is shown. The effect is computed by predicting without intercept using the optimum stopping iteration, which is selected by AIC and can be extracted with function `lasso_stop()`. The stopping iteration is passed to the `predict()` method by specifying the `mstop` argument.

```
R> page <- predict(b, term = "cage", intercept = FALSE,
+    mstop = lasso_stop(b))
```

The figure is then created using the untransformed original covariate on the x-axis.

```
R> plot2d(page ~ age, data = SwissLabor, rug = TRUE)
```

Using the fused lasso estimation nonlinearities can be identified again, similar to the spline based estimate in the right panel of Figure 2.

2.3. Location-scale model

Here, we extend the framework and estimate a distributional regression model that not only captures the mean (or location) of the response variable but also its variance (or scale). As an example, we employ the number of weekly fatalities in Austria from 2000–2020 (up to week 46 in 2020) as obtained from the Eurostat data base (`https://ec.europa.eu/eurostat/`). The data is available in the **bamlss** package as `fatalities`, providing the number (`num`) of fatalities in each `year` and `week`. It can be loaded with

```
R> data("fatalities", package = "bamlss")
```

The idea of the subsequent analysis is to estimate a reference mortality model based on the data from 2000–2019 prior to the COVID-19 (Corona virus disease 2019) crisis in order to bring out graphically the excess mortality in 2020. Excess mortality is often employed for assessing the effects of exceptional events such as pandemics (Leon *et al.* 2020) or natural catastrophes (Fouillet *et al.* 2008). First, we split the data into the corresponding subsets.

```
R> d19 <- subset(fatalities, year <= 2019)
R> d20 <- subset(fatalities, year >= 2020)
```

To capture the long-term seasonal trend of the fatality number distribution, we employ a simple model here: log-fatalities are assumed to be normally distributed with smooth seasonal variations in both mean and variance. As shown below the log-transformation stabilizes skewness and variance in the data somewhat so that a normal model works sufficiently well. Cyclic splines with respect to the week of the year are employed to capture the smooth seasonal trends while assuring that the values at the beginning and the end of the year match. The model formula is now a list with elements for the mean of `log(num)` (corresponding to parameter `mu`) and standard deviation `sigma` of the normal distribution.

```
R> f <- list(
+    log(num) ~ s(week, bs = "cc", k = 20),
+    sigma    ~ s(week, bs = "cc", k = 20)
+  )
```

Function `s()` is again the smooth term constructor from the **mgcv** package (Wood 2021) and `bs = "cc"` specifies a penalized cyclic cubic regression spline. (Other smooth terms such as `te()` or `ti()` could be included in the same way.) Based on this `bamlss()` is used to estimate a full Bayesian regression model using the `NO()` normal family from the **gamlss.dist** package.

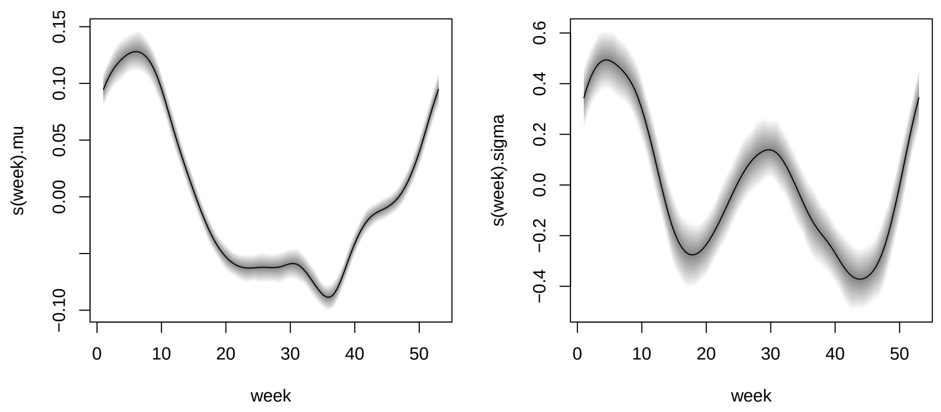

Figure 4: Estimated effects of `week` on the mean `mu` and standard deviation `sigma` of the normal model. The gray shaded areas represent 95% credible intervals.

```
R> library("gamlss.dist")
R> set.seed(456)
R> b <- bamlss(f, data = d19, family = NO)
```

The resulting estimated effects along with their 95% credible intervals can be easily visualized using the `plot()` method:

```
R> plot(b, model = c("mu", "sigma"))
```

The resulting displays are shown in Figure 4 depicting a clear nonlinear relationship for both distribution parameters. The left panel shows that mean log-fatalities are much higher in winter than in summer with a peak around February matching the highest risk for influenza and other viral infections in Austria. The right panel shows that the standard deviation is also highest at around the same time but that there is another local maximum in the summer months, possibly related to recurrent heat waves that can be quite stressful for the cardiovascular system (Fouillet *et al.* 2008).

Figure 5 shows the predicted 5%, 50%, and 95% quantiles (in black) of the corresponding normal distributions along with the observed fatalities in 2000–2019 (in light gray) and in 2020 (in red, up to week 46), respectively. Thus, the quantiles reflect the effects already conveyed by the predicted parameters in Figure 4. This shows that the fatalities in 2020 are above the median almost throughout all weeks and above the 95% quantile for a couple of weeks in spring and in the fall/winter, respectively. While the mortality in the spring period is only moderately increased, it is much higher than in previous years in fall/winter during the second COVID-19 wave in Austria.

In the following, we show how to draw Figure 5 using the **bamlss** infrastructure. First, we set up a new data frame and predict the distribution parameters for each week of the year.

```
R> nd <- data.frame(week = 1:53)
R> par <- predict(b, newdata = nd, type = "parameter")
```

Based on these, the fitted quantiles can be computed using the quantile function from the family of the model (see Section 4.2 for details). The `exp()` transformation maps the fitted values from the log-scale back to the original frequency scale.

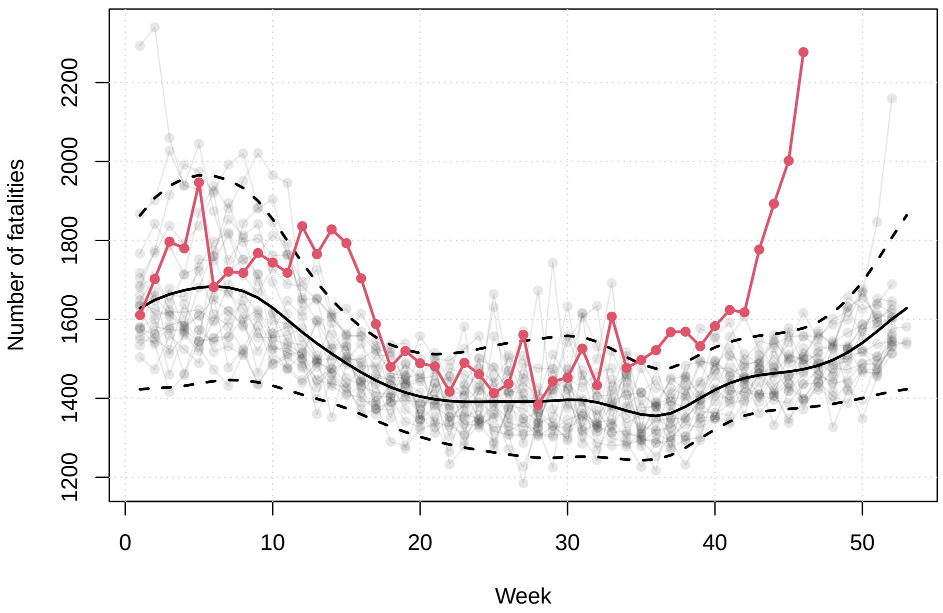

Figure 5: Predicted 5%, 50%, and 95% quantiles (in black) of the cyclic seasonal model along with the observed number of fatalities in 2000–2019 (in light gray) and in 2020 (in red).

```
R> nd$fit <- sapply(c(0.05, 0.5, 0.95),
+    FUN = function(p) { exp(family(b)$q(p, par)) })
```

Finally, the estimated quantiles and observed data can be visualized using `matplot()` after reshaping the data to "wide" format with a separate column for each year.

```
R> d19w <- reshape(d19, idvar = "week",
+    timevar = "year", direction = "wide")
R> matplot(d19w$week, d19w[, -1],
+    type = "o", lty = 1, pch = 16, col = gray(0.1, alpha = 0.1),
+    xlab = "Week", ylab = "Number of fatalities")
R> grid()
R> matplot(nd$week, nd$fit, type = "l", lty = c(2, 1, 2),
+    col = 1, lwd = 2, add = TRUE)
R> lines(num ~ week, data = d20, col = 2, lwd = 2, type = "o", pch = 16)
```

For judging how well the distributional model captures the observed data, the `plot()` method includes some diagnostic graphics such as histograms or quantile-quantile (Q-Q) plots of randomized quantile residuals (Dunn and Smyth 1996), shown in Figure 6.

```
R> plot(b, which = c("hist-resid", "qq-resid"), c95 = TRUE)
```

By setting `c95 = TRUE`, the Q-Q plot includes 95% credible intervals. Both plots show that the log-transformation of the fatality numbers only partially captures the right-skewed observations and that therefore the model fit is not ideal in the upper tail. In an accompanying online vignette at `http://www.bamlss.org/articles/fatalities.html` we show how to find

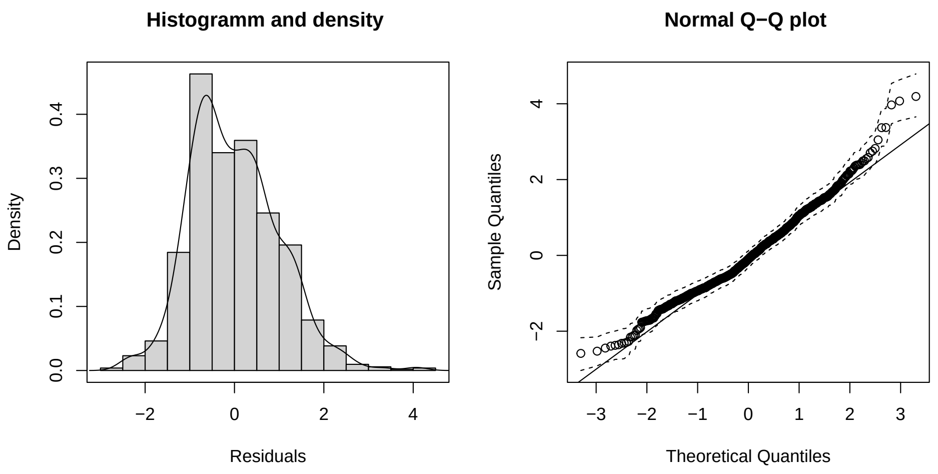

Figure 6: Histogram and Q-Q plot including 95% credible intervals (dashed lines) of the randomized quantile residuals for the distributional regression model.

a better probabilistic model by considering more flexible distributions (beyond the normal case) and general Box-Cox transformations (instead of hard-coding the log-transformation). While this improves the distributional model, the qualitative insights regarding the excess mortality in Austria during the COVID-19 crisis remain unchanged.

3. A flexible Bayesian model framework

This section briefly summarizes the BAMLSS modeling framework. For a detailed methodological description please refer to Umlauf *et al.* (2018), as well as to the references given below on page 15 that discuss various applications and extensions that are also implemented in **bamlss**. The following outlines the framework from the viewpoint of distributional regression models, however, please note that model classes like, e.g., GLMs and GAMs or even survival joint models (Köhler, Umlauf, Beyerlein, Winkler, Ziegler, and Greven 2017; Köhler, Umlauf, and Greven 2018) are special cases in this setup.

3.1. Model structure

Within the framework of GAMLSS or distributional regression models all parameters of the response distribution can be modeled by explanatory variables such that

$$y \sim \mathcal{D}\left(h_1(\theta_1) = \eta_1,\ h_2(\theta_2) = \eta_2, \ldots,\ h_K(\theta_K) = \eta_K\right), \tag{1}$$

where $\mathcal{D}$ denotes a parametric distribution for the response variable y with K parameters θ_k, $k = 1, \ldots, K$, that are linked to additive predictors using known monotonic and twice differentiable functions $h_k(\cdot)$. Note that the response may also be a q-dimensional vector $\mathbf{y} = (y_1, \ldots, y_q)^\top$, e.g., when $\mathcal{D}$ is a multivariate distribution (see, e.g., Klein, Kneib, Klasen, and Lang 2015a). The additive predictor for the k-th parameter is given by

$$\boldsymbol{\eta}_k = \eta_k(\mathbf{X}; \boldsymbol{\beta}_k) = f_{1k}(\mathbf{X}; \boldsymbol{\beta}_{1k}) + \ldots + f_{J_k k}(\mathbf{X}; \boldsymbol{\beta}_{J_k k}), \tag{2}$$

based on $j = 1, \dots, J_k$ unspecified (possibly nonlinear) functions $f_{jk}(\cdot)$, applied to each row of the generic data matrix $\mathbf{X}$, encompassing all available covariate information. The corresponding parameters $\boldsymbol{\beta}_k = (\boldsymbol{\beta}_{1k}, \dots, \boldsymbol{\beta}_{J_k k})^\top$ are typically regression coefficients pertaining to model matrices $\mathbf{X}_k = (\mathbf{X}_{1k}, \dots, \mathbf{X}_{J_k k})^\top$, whose structure only depend on the type of covariate(s) and prior assumptions about $f_{jk}(\cdot)$.

Usually, functions $f_{jk}(\cdot)$ are based on a basis function approach, where η_k then is a typical GAM-type or so-called structured additive predictor (STAR, Fahrmeir, Kneib, and Lang 2004). Similar to Stasinopoulos *et al.* (2017), Umlauf *et al.* (2018) relax this assumption and let $f_{jk}(\cdot)$ be an unspecified composition of covariate data and regression coefficients. For example, functions $f_{jk}(\cdot)$ could also represent nonlinear growth curves, a regression tree, a neural network or lasso-penalized model terms as shown in Section 2.2.

For full Bayesian inference, priors need to be assigned to the regression coefficients $\boldsymbol{\beta}_{jk}$. To be as flexible as possible, Umlauf *et al.* (2018) use the rather general prior $p_{jk}(\boldsymbol{\beta}_{jk}; \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{jk})$ for the j-th model term of the k-th parameter, where the form of $p_{jk}(\cdot)$ depends on the type of function $f_{jk}(\cdot)$. Here, $\boldsymbol{\tau} = (\boldsymbol{\tau}_{11}^\top, \dots, \boldsymbol{\tau}_{J_1 1}^\top, \dots, \boldsymbol{\tau}_{1K}^\top, \dots, \boldsymbol{\tau}_{J_K K}^\top)^\top$ is the vector of all assigned hyper-parameters, e.g., representing smoothing variances (shrinkage parameters). Similarly, $\boldsymbol{\alpha}_{jk}$ is the set of all fixed prior specifications, i.e., for GAM-type models $\boldsymbol{\alpha}_{jk}$ usually holds the so-called penalty matrices, amongst others. In most situations the prior $p_{jk}(\boldsymbol{\beta}_{jk}; \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{jk})$ is based on a multivariate normal kernel for $\boldsymbol{\beta}_{jk}$ and on inverse gamma distributions for each $\boldsymbol{\tau}_{jk} = (\tau_{1jk}, \dots, \tau_{L_{jk}jk})^\top$, but as indicated previously, in principle any type of prior can be used (see Gelman 2006; Polson and Scott 2012; Klein and Kneib 2016a; Umlauf *et al.* 2018 for more detailed discussions on priors for $\boldsymbol{\tau}_{jk}$ and principled constructions for such priors).

Examples of distributional models that fit well in this framework are the ones for:

- Univariate responses of any type, e.g., counts with zero-inflation and/or overdispersion as proposed in Klein, Kneib, and Lang (2015b); Herwartz, Klein, and Strumann (2016); Stasinopoulos and Rigby (2021a), continuous responses with spikes, skewness, heavy tails or bounded support as in Klein, Denuit, Lang, and Kneib (2014); Klein *et al.* (2015c); Stasinopoulos and Rigby (2021a), as well as responses for extreme events (Umlauf and Kneib 2018).
- Multivariate responses such as multivariate normal, multivariate t or Dirichlet regression (for analyzing compositional data, Klein *et al.* 2015a).
- Multivariate responses with more complex dependence structures modeled through copulas (Klein and Kneib 2016b).
- Survival data and joint modeling (Köhler *et al.* 2017, 2018).

3.2. Posterior estimation

Estimation typically requires to evaluate the log-likelihood $\ell(\boldsymbol{\beta}; \mathbf{y}, \mathbf{X})$ function and its derivatives w.r.t. all regression coefficients $\boldsymbol{\beta}$ a number of times. For Bayesian inference the log-posterior is either used for posterior mode estimation, or for solving high-dimensional integrals. e.g., for posterior mean estimation MCMC samples need to be computed.

Although the types of models that can be fitted within the flexible BAMLSS framework can be quite complex, Umlauf *et al.* (2018) show that there are a number of similarities between optimization and sampling concepts. Fortunately, and albeit the different model term complexity, algorithms for posterior mode and mean estimation can be summarized into a partitioned updating scheme with separate updating equations using leapfrog or zigzag iteration (Aitkin 1987; Smyth 1996), e.g., with updating equations

$$(\boldsymbol{\beta}_{jk}^{(t+1)}, \boldsymbol{\tau}_{jk}^{(t+1)}) = U_{jk}(\boldsymbol{\beta}_{jk}^{(t)}, \boldsymbol{\tau}_{jk}^{(t)}; \cdot) \qquad j = 1, \ldots, J_k, \quad k = 1, \ldots, K, \tag{3}$$

where function $U_{jk}(\cdot)$ is an updating function, e.g., for generating one Newton-Raphson step or for getting the next step in an MCMC simulation.

Rigby and Stasinopoulos (2005) showed that using a basis function approach, i.e., each function $f_{jk}(\cdot)$ can be represented by a linear combination of a design matrix and regression coefficients, the updating functions $U_{jk}(\cdot)$ for posterior mode (frequentist penalized likelihood) estimation for $\boldsymbol{\beta}_{jk}$ share an iteratively weighted least squares updating step (IWLS, Gamerman 1997)

$$\boldsymbol{\beta}_{jk}^{(t+1)} = U_{jk}(\boldsymbol{\beta}_{jk}^{(t)}; \cdot) = (\mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}\mathbf{X}_{jk} + \mathbf{G}_{jk}(\boldsymbol{\tau}_{jk}))^{-1}\mathbf{X}_{jk}^{\top}\mathbf{W}_{kk}(\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)}), \tag{4}$$

with weight matrices $\mathbf{W}_{kk}$ and working responses $\mathbf{z}_k$, similarly to the well-known IWLS updating scheme for generalized linear models (GLM, Nelder and Wedderburn 1972). In the same way, approximate full conditionals $\pi(\boldsymbol{\beta}_{jk}|\cdot)$ for MCMC are constructed with this updating step (Gamerman 1997; Fahrmeir *et al.* 2004; Brezger and Lang 2006; Klein and Kneib 2016b). The matrices $\mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})$ are derivative matrices of the priors $p_{jk}(\boldsymbol{\beta}_{jk}; \boldsymbol{\tau}_{jk}, \boldsymbol{\alpha}_{jk})$ w.r.t. the regression coefficients $\boldsymbol{\beta}_{jk}$, e.g., using basis function for $f_{jk}(\cdot)$ matrices $\mathbf{G}_{jk}(\boldsymbol{\tau}_{jk})$ can be a penalty matrices that penalize the complexity using a P-spline representation (Eilers and Marx 1996).

Even if the functions $f_{jk}(\cdot)$ are not based on a basis function approach, the updating scheme (4) can be further generalized to

$$\boldsymbol{\beta}_{jk}^{(t+1)} = U_{jk}\left(\boldsymbol{\beta}_{jk}^{(t)}, \mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)}; \cdot\right),$$

i.e., theoretically any updating function applied to the "partial residuals" $\mathbf{z}_k - \boldsymbol{\eta}_{k,-j}^{(t+1)}$ can be used (for detailed derivations see also Umlauf *et al.* 2018).

The great advantage of this modular architecture is that the concept does not limit to modeling of the distributional parameters θ_k in (1), e.g., as mentioned above, based on the survival function, Köhler *et al.* (2017) and Köhler *et al.* (2018) implement Bayesian joint models for survival and longitudinal data. Moreover, the updating schemes do not restrict to any particular estimation engine, e.g., Groll *et al.* (2019) use the framework to implement lasso-type penalization for GAMLSS and Simon, Fabsic, Mayr, Umlauf, and Zeileis (2018) investigate gradient boosting with stability selection algorithms (see also Section 5). Very recently, Klein, Simon, and Umlauf (2019) implement neural network distributional regression models.

3.3. Model choice and evaluation

Measures of performance

Model choice and variable selection is important in distributional regression due to the large number of candidate models. The following lists commonly-used tools:

- *Information criteria* can be used to compare different model specifications. For posterior mode estimation, the Akaike information criterion (AIC), or the corrected AIC, as well as the Bayesian information criterion (BIC), can be used. Estimation of model complexity is based on the so-called equivalent degrees of freedom (EDF), i.e., for each model term the trace of the smoother matrix is computed (see, e.g., Hastie and Tibshirani 1990) and the total degrees of freedom are approximated by the sum over all distributional parameters and model terms.

  For MCMC based estimation, model choice mainly relies on the deviance information criterion (DIC, Spiegelhalter, Best, Carlin, and Van der Linde 2002) and the widely applicable information criterion (WAIC, Watanabe 2010).

- *Quantile residuals* (Dunn and Smyth 1996) can be used to evaluate the model fit. The residuals can be assessed by quantile-quantile-plots, probability integral transforms (PIT) histograms (Gneiting *et al.* 2007) or worm plots (Van Buuren and Fredriks 2001).

- *Scoring rules*: Sometimes it is helpful to evaluate the performance on a test data set (or for instance based on cross validation). For this, proper scoring rules (Gneiting and Raftery 2007; Gneiting *et al.* 2007) can be utilized.

Evaluation and interpretation

- *Plotting*: Estimated functions $\hat{f}_{jk}(\cdot)$ are usually subject to a centering constraint (e.g., $\sum \hat{f}_{jk}(x_i) = 0$), therefore, simple effect plots are a straightforward method to evaluate individual model term importance and can also be used for respective interpretations. Sometimes it can be useful in distributional regression to look at transformations of the original model parameters such as expected value or variance of the response variable $\mathbf{y}$.

- *Predictions*: For obtaining such transformations model predictions need to be computed. This can be done either manually by the corresponding `predict()` method, or by the R package **distreg.vis** (Stadlmann 2021), which provides a graphical user interface for visualization of distributional regression models.

4. The bamlss package

The R package **bamlss** provides a modular software architecture for flexible Bayesian regression models (and beyond). The implementation follows the conceptional framework presented in Umlauf *et al.* (2018), which supports Bayesian and/or frequentist estimation engines using complex possibly nonlinear model terms of any type. The highlights of the package are:

- A unified model description where a `formula` specifies how to set up the predictors from the `data` and the `family`, which holds information about the response distribution, the model.

- A generic method for setting up model terms and a `model.frame()` for BAMLSS, the `bamlss.frame()`, along with the corresponding prior structures. A `transform()`

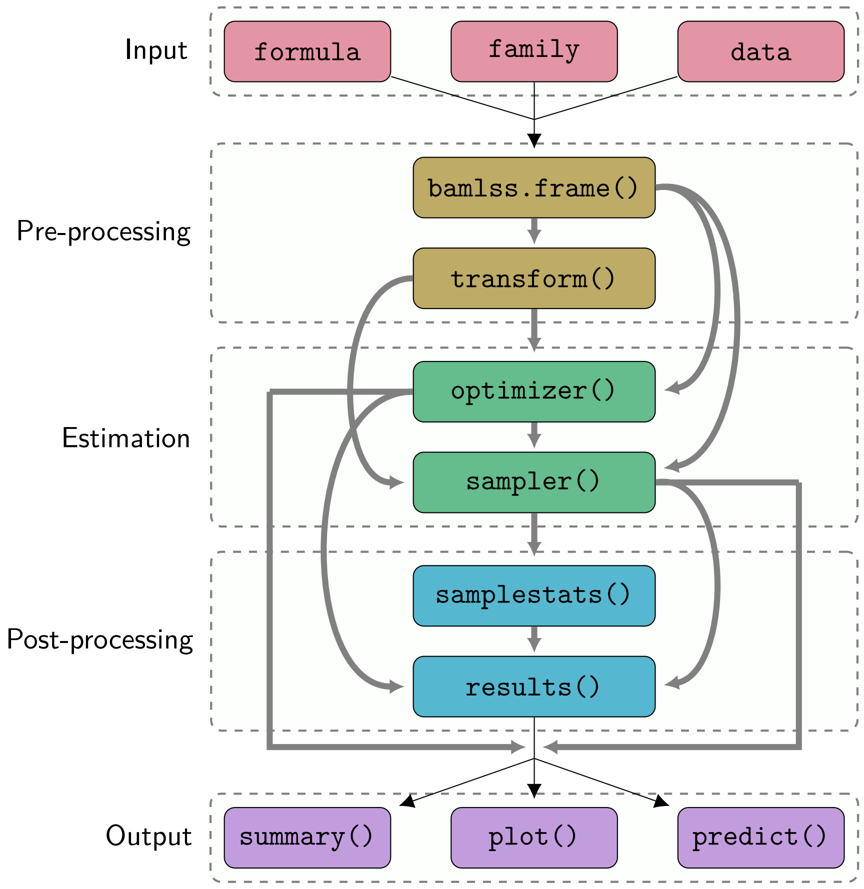

Figure 7: Flow chart of the **bamlss** modeling architecture. Thick gray lines represent optional paths, e.g., after building the `bamlss.frame()` the user can either run an `optimizer()` function prior running the `sampler()`, or run the `sampler()` function directly.

function can optionally set up modified terms, e.g., using mixed model representation for smooth terms.

- Support for modular and exchangeable updating functions or complete model fitting engines in order to optionally implement either algorithms for maximization of the log-posterior for posterior mode estimation or for solving high-dimensional integrals, e.g., for posterior mean or median estimation. First, an (optional) `optimizer()` function can be run, e.g., for computing posterior mode estimates. Second, a `sampler()` is employed for full Bayesian inference with MCMC, which uses the posterior mode estimates from the `optimizer()` as starting values. An additional step can be used for preparing the `results()`, e.g., for creating model term effect plots.
- Standard post-modeling extractor functions to create sampling statistics, visualizations, predictions, amongst others.

The modular architecture of **bamlss** is illustrated in Figure 7. As mentioned above, the first step in model development is to setup design and penalty matrices for a model that is specified by the `family` object. Therefore a `formula` is processed together with the `data` using

Step	Type	Function
Pre-processing	Parser	`bamlss.frame()`
	Transformer	`bamlss.engine.setup()`, `randomize()` `lasso_transform()`
Estimation	Optimizer	`opt_bfit()`, `opt_bbfit()`, `opt_boost()`, `opt_lasso()` `opt_Cox()`, `opt_JM()`
	Sampler	`sam_GMCMC()`, `sam_BayesX()`, `sam_JAGS()` `sam_Cox()`, `sam_JM()`
Post-processing	Stats & Results	`samplestats()`, `results.bamlss.default()`

Table 2: Current available functions that can be used for pre-processing, estimation and post-processing within the **bamlss** framework.

the `bamlss.frame()` function. In a second pre-processing step, the returned model frame may also be transformed. The BAMLSS model frame can then be used with `optimizer()` and/or `sampler()` functions in the estimation step. This is probably the main advantage of the architecture, users can easily exchange and integrate user defined estimation functions. The only requirement is to keep the structure of the `bamlss.frame()` function, as well for `optimizer()` and `sampler()` functions. Note that there is naming convention, optimizer functions start with prefix `opt_*` and sampler functions with `sam_*`. After the estimation step optional post-processing functions can be applied to create additional sampling statistics, function `samplestats()`, or results that can be used for plotting the estimated effects, function `results()`. The post-processing step is optional since it is not necessarily needed in the last output step, e.g., for computing predictions. This feature is especially important when using large data sets, because the run time for computing `samplestats()` or `results()` can be quite long or computations can even lead to memory problems.

In summary, besides implementing models using the family infrastructure (see Section 4.2) the architecture is very flexible such that also users interested in implementing new and non-standard models or algorithms only need to focus on the estimation step, i.e., write `optimizer()` or `sampler()` functions and get all post-processing and extractor functionalities "for free". This way, prototyping becomes relatively easy, but also the integration high-performance estimation engines is facilitated. Table 2 provides an overview of current available functions. Note that sampler functions `sam_BayesX()` and `sam_JAGS()` need installation of the **BayesXsrc** (Umlauf, Adler, Kneib, Lang, and Zeileis 2021) and the **rjags** (Plummer 2021) package. To have a better overview of various functionalities, function `engines()` provides information about which optimizer and sampler functions can be used with which family. For example, to look up compatibility for the normal distribution of the **gamlss.dist** package and the family implemented in **bamlss** the user can type

```
R> engines(NO, gaussian_bamlss, cox_bamlss)
```

```
                NO gaussian   cox
opt_bfit()     TRUE     TRUE FALSE
opt_boost()    TRUE     TRUE FALSE
opt_bbfit()    TRUE     TRUE FALSE
sam_GMCMC()    TRUE     TRUE FALSE
sam_BayesX()  FALSE     TRUE FALSE
```

```
sam_JAGS()    FALSE     TRUE FALSE
special_opt() FALSE    FALSE  TRUE
special_sam() FALSE    FALSE  TRUE
```

The table shows that the `NO()` family is compatible with all pure R implementations of optimizer and sampler functions, but not with special samplers like **BayesX** and **JAGS**. These can only be used with the `gaussian_bamlss()` family. In addition, neither `NO()` or `gaussian_bamlss()` has its own special optimizer or sampler implemented, such as the `cox_bamlss()` family.

To exemplify the presented "Lego toolbox", the following R code estimates the logit model using the `SwissLabor` data presented in Section 2.1. First, the data is loaded and the model formula is specified with

```
R> data("SwissLabor", package = "AER")
R> f <- participation ~ income + age + education +
+    youngkids + oldkids + foreign + I(age^2)
```

In the second step, the necessary design matrices are constructed using the model frame parser function `bamlss.frame()`

```
R> bf <- bamlss.frame(f, data = SwissLabor, family = "binomial")
```

Then, posterior mode estimates are obtained by using the implemented backfitting estimation function `opt_bfit()`

```
R> pm <- with(bf, opt_bfit(x, y, family))
```

The estimated parameters returned from function `opt_bfit()` can then be used as starting values for the MCMC sampler function `sam_GMCMC()`

```
R> set.seed(123)
R> samps <- with(bf, sam_GMCMC(x, y, family, start = pm$parameters))
```

Using the parameters samples returned from function `sam_GMCMC()`, statistics like the DIC are computed using the `samplestats()` function

```
R> stats <- with(bf, samplestats(samps, x, y, family))
R> print(unlist(stats))

    logLik        DIC         pd
-512.72579 1033.32501    7.87343
```

As one can see in the code above, estimation engines have common arguments `x` (holding the design and penalty matrices), `y` (the response data) and `family` (the **bamlss** family object). For implementing new estimation engines, users only need to keep the argument structures and the return values, i.e., for `optimizer()` functions a named numeric vector of estimated parameters and for `sampler()` functions parameter samples of class `"mcmc"` or `"mcmc.list"` (see package **coda**, Plummer *et al.* 2006). More details on the naming convention and the structure of the return value of `bamlss.frame()` are given in Section 4.1.

To ease the modeling process, all the single modeling steps presented in the above can be executed using the **bamlss** wrapper function `bamlss()`. The main arguments of `bamlss()` are

```
bamlss(formula, family = "gaussian", data = NULL,
  transform = NULL,                        ## Pre-processing
  optimizer = NULL, sampler = NULL,        ## Estimation
  samplestats = NULL, results = NULL, ...)  ## Post-processing
```

where the first line basically represents the standard model frame specifications (see Chambers and Hastie 1992). All other arguments represent functions presented in Table 2 and can be exchanged. Note that the default for argument `optimizer` is the backfitting estimation function `opt_bfit()` and the default for argument `sampler` is the `sam_GMCMC()` sampling function, which is a quite generic implementation. More specifically, `sam_GMCMC()` accepts proposal functions for each model term which do not necessarily have to be the same and can be exchanged, e.g., the core proposal function is implemented in C and is additionally optimized for large design and penalty matrices such that sampling using very large data sets is possible (see Lang *et al.* 2014 for details on algorithms, e.g., using `sam_JAGS()` is only suitable for moderate sized data and low complexity model terms). For more details on `sam_GMCMC()` please see the **bamlss** manual.

The returned fitted model object is a list of class "`bamlss`", which is supported by several standard methods and extractor functions, such as `plot()`, `summary()` and `predict()`.

As already exemplified in Section 2, using the model fitting wrapper function `bamlss()` it is straightforward to use different modeling approaches by simply exchanging the estimation engines. This feature can be particularly important in complex modeling situation, where good mixing of the MCMC algorithm requires very good starting values. One use case is presented in Section 5, where for stability reasons posterior mode estimates are obtained using the gradient boosting optimizer function `boost()`. Afterwards the MCMC sampling engine `sam_GMCMC()` is applied with the boosting estimates as starting values.

4.1. The BAMLSS model frame

Similar to the well-known `model.frame()` function that is used, e.g., by the linear model fitting function `lm()`, or for generalized linear models `glm()`, the `bamlss.frame()` function extracts a "model frame" for fitting distributional regression models. Internally, the function parses model formulae, one for each parameter of the distribution, using the **Formula** package infrastructure (Zeileis and Croissant 2010) in combination with `model.matrix()` processing for linear effects and `smooth.construct()` processing of the **mgcv** package to setup design and penalty matrices for unspecified smooth function estimation (Wood 2021, see also, e.g., the documentation of function `s()` and `te()`).

The most important arguments are

```
bamlss.frame(formula, data = NULL, family = "gaussian",
  weights = NULL, subset = NULL, offset = NULL,
  na.action = na.omit, contrasts = NULL, ...)
```

The argument `formula` can be a classical model formulae, e.g., as used by the `lm()` function, or an extended **bamlss** formula including smooth term specifications like `s()` or `te()`, that is internally parsed by function `bamlss.formula()`. Note that the **bamlss** package uses special `family` objects, that can be passed either as a character without the `"_bamlss"` extension of the **bamlss** family name (see the manual `?bamlss.family` for a list of available families), or the family function itself. In addition, all families of the **gamlss** (Stasinopoulos and Rigby 2021a) and **gamlss.dist** (Stasinopoulos and Rigby 2021b) package are supported, i.e., there is a transformer function that reads all necessary components and then transfers them into a family object for **bamlss**.

The returned object, a named list of class `"bamlss.frame"`, can be employed with the model fitting engines listed in Table 2. The most important elements used for estimation are:

- `x`: A named list, the elements correspond to the parameters that are specified within the `family` object. For each distribution parameter, the list contains all design and penalty matrices needed for modeling (see the upcoming example).
- `y`: The response data.
- `family`: The processed **bamlss** `family`.

To better understand the structure of the `"bamlss.frame"` object a print method is provided. For illustration, we simulate data

```
R> set.seed(111)
R> d <- GAMart()
```

and set up a `"bamlss.frame"` object for a Gaussian distributional regression model including smooth terms. First, a model formula is needed

```
R> f <- list(
+    num ~ x1 + s(x2) + s(x3) + te(lon,lat),
+    sigma ~ x1 + s(x2) + s(x3) + te(lon,lat)
+  )
```

Afterwards the model frame can be computed with

```
R> bf <- bamlss.frame(f, data = d, family = "gaussian")
```

To keep the overview, there is also an implemented print method for `"bamlss.frame"` objects.

```
R> print(bf)
```

```
'bamlss.frame' structure:
  ..$ call
  ..$ model.frame
  ..$ formula
  ..$ family
  ..$ terms
  ..$ x
  .. ..$ mu
  .. .. ..$ formula
  .. .. ..$ fake.formula
  .. .. ..$ terms
  .. .. ..$ model.matrix
  .. .. ..$ smooth.construct
  .. ..$ sigma
  .. .. ..$ formula
  .. .. ..$ fake.formula
  .. .. ..$ terms
  .. .. ..$ model.matrix
  .. .. ..$ smooth.construct
  ..$ y
  .. ..$ num
  ..$ delete
```

For writing a new estimation engine, the user can directly work with the `model.matrix` elements, for linear effects, and the `smooth.construct` list, for smooth effects respectively. The `smooth.construct` is a named list which is compiled using the `smoothCon()` function of the **mgcv** package using the generic `smooth.construct()` method for setting up smooth terms.

```
R> print(names(bf$x$mu$smooth.construct))

[1] "s(x2)"       "s(x3)"       "te(lon,lat)"
```

In this example, the list contains three smooth term objects for parameter `mu` and `sigma`.

As shown in Appendix B the `bamlss.frame()` function can also process special model terms, i.e., model terms that are not necessarily represented by a linear matrix vector product.

4.2. Family objects

Family objects are important building blocks in the design of BAMLSS models. The implementation in **bamlss** follows the well-established structures for family objects that are supported, e.g., by the base R model fitting function `glm()`, or family objects of the **gamlss** and **VGAM** package. This means, that users can also easily write new family objects to be used with **bamlss**. Such family objects specify the distribution by collecting functions of the density, respective log-likelihood, first-order derivatives of the log-likelihood w.r.t. predictors (the score function), and (optionally) second-order derivatives of the log-likelihood w.r.t. predictors or their expectation (the Hessian). Commonly used distributions are already implemented in **bamlss**; and note that the ones from the **gamlss** and **gamlss.dist** package can also be accessed through the **bamlss** package (see Section 2.3 for an example).

We illustrate how to build a **bamlss** family by hand along the Gaussian distribution, with density

$$f(y \mid \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} \cdot \exp\left(\frac{-(y-\mu)^2}{2\sigma^2}\right),$$

and log-likelihood function

$$\ell(\mu, \sigma \mid y) = -\frac{1}{2}\log(2\pi) - \log(\sigma) - \frac{(y-\mu)^2}{2\sigma^2},$$

for an individual observation. The sum of the log-likelihood function over all observations is the target function of the optimization problem.

In the distributional regression framework the parameters are linked to predictors by link functions,

$$\mu = \eta_\mu, \qquad \log(\sigma) = \eta_\sigma.$$

For the Gaussian μ and σ are linked to η_μ and η_σ by the identity function and the logarithm, respectively.

The score functions in **bamlss** are the first derivatives of the log-likelihood w.r.t. the predictors:

$$s_\mu = \frac{\partial \ell}{\partial \eta_\mu} = \frac{\partial \ell}{\partial \mu} \cdot \frac{\partial \mu}{\partial \eta_\mu} = \frac{y-\mu}{\sigma^2},$$

and

$$s_\sigma = \frac{\partial \ell}{\partial \eta_\sigma} = \frac{\partial \ell}{\partial \sigma} \cdot \frac{\partial \sigma}{\partial \eta_\sigma} = -1 + \frac{(y-\mu)^2}{\sigma^2}.$$

For the second derivative of the log-likelihood we are able to obtain the negative expectation,

$$\mathsf{E}(-\partial^2 \ell / \partial \eta_\mu^2) = \sigma^{-2},$$

and

$$\mathsf{E}(-\partial^2 \ell / \partial \eta_\sigma^2) = 2.$$

In more detail, the default **bamlss** estimation engines are based on IWLS updating functions and do not require the mixed elements of the Hessian, i.e., the backfitting optimizer function `opt_bfit()` uses leapfrog or zizag iterations (Smyth 1996) and the MCMC sampler function `sam_GMCMC()` also only updates one model term $f_{jk}(\cdot)$ at a time, hence, only the diagonal elements of the Fisher information matrix are needed. Furthermore, it is not mandatory to use the expected Fisher information, but for numerical stability it is recommended. If the information on the second derivatives is not provided the `bamlss.frame()` will set up approximate versions by numerical differentiation of the score functions, the same mechanism is applied for first order derivatives. Hence, in quite a few cases implementing a new family can only be based on the specification of the density function, however, in terms of optimization runtime this is certainly not the most efficient choice. For distributions for which the expectation of the second derivative is intractable or does not exist, the user can rely on two options: the first option is to simply take the Hessian evaluated at observations and corresponding predictors rather than computing the theoretical expectation analytically for filling the diagonals of the weight matrices $\mathbf{W}_{kk}$. The second option is to find a good approximation for the expectation (see, e.g., Klein *et al.* 2015b, for the case of the overdispersion parameter of the negative binomial distribution).

Now we have to write a function that returns a `family.bamlss` object (S3) which encapsulates functions for density, score and Hessian, and the names of the family, parameter and link functions. The required elements are listed in Table 3. Note that there are no other specifications to follow, for example one could also build a family that allows for flexible link functions (like the families from the **gamlss** package).

Merely all functions take as first argument the response `y` and as second argument a named list holding the evaluated parameters `par` of the distribution. The example implementation is shown in Appendix A.

Name of element	Value
`family`	Character string with the name of the family.
`names`	Vector of character strings with the names of the parameters.
`links`	Vector of character strings with the names of the link functions
`d`	A function returning the density with arguments `d(y, par, log = FALSE)` (see below).
`p`	The cumulative distribution function `p(y, par, ...)`.
`score`	A list with functions (one for each parameter) returning the first derivatives of the log-likelihood w.r.t. predictors.
`hess`	A list with functions (one for each parameter) returning the negative second derivatives of the log-likelihood w.r.t. predictors.

Table 3: Elements of the Gaussian distribution `"bamlss.family"` object.

Optionally, the `"family.bamlss"` object can be extended by functions for

- the quantile function (the inverse cdf) `q(p, par)`,
- a random number generator `r(n, par)`,
- the log-likelihood `loglik(y, par)`,
- the expectation `mu(par, ...)`,
- initial values for optimization, which has to be a list containing a function for each parameter,
- a customized `predict()` function which will be called by `predict.bamlss()`, e.g., as implemented in the family `cox_bamlss()`,
- similarly, a customized `residuals()` function that should be used by `residuals.bamlss()`,

which can help to speed up optimization, or be convenient for predictions and simulations.

When all formulas for a family are worked out, it usually takes about an hour to create a new family object. Of course, this also depends on the complexity of the density function. With some families it can be meaningful for speed reasons to port the functions for example to C. In our experience, programming then takes only slightly longer, about 2 to 3 hours.

For a list of all implemented families, please see the documentation of `?bamlss.family`.

4.3. Estimation engines

Estimation engines in **bamlss** are usually based on the model frame setup function `bamlss.frame()` (see Section 4.1), i.e., the functions all have a `x` argument, which contains all the necessary model and penalty matrices, and a `y` argument, which is the response (univariate or multivariate). In addition, an estimation engine usually has a `family` argument, which specifies the model to be estimated. However, this is not a mandatory argument, i.e., one could write an estimation function that is designed for one specific problem, only. As mentioned at the beginning of Section 4, there is naming convention, optimizer functions start with prefix `opt_*` and sampler functions with `sam_*`. The naming convention is not mandatory, but it gives the user a better overview of the many functions of the package.

The modeling setup is best explained by looking at the main estimation engines provided by **bamlss**. The default optimizer using the `bamlss()` wrapper function is `opt_bfit()`, which is a backfitting routine. The most important arguments are

```
  opt_bfit(x, y, family, start = NULL, weights = NULL, offset = NULL, ...)
```

The default sampling engine in **bamlss** is `sam_GMCMC()`, again the most important arguments are

```
  sam_GMCMC(x, y, family, start = NULL, weights = NULL, offset = NULL, ...)
```

So basically, the arguments of the optimizer and the sampling function are the same, the main difference is the return value. In **bamlss** optimizer functions usually return a vector of estimated regression coefficients (parameters), while sampling functions return a matrix of parameter samples of class `"mcmc"` or `"mcmc.list"` (for details see the documentation of the **coda** package).

Internally, what the optimizer or sampling function is actually processing is not important for the `bamlss()` wrapper function as long as a vector or matrix of parameters is returned. For optimizer functions the return value needs to be named list with an element `"parameters"`, the vector (also a matrix, e.g., for `lasso()` and `boost()` optimizers) of estimated parameters. The most important requirement to make use of all extractor functions like `summary.bamlss()`, `predict.bamlss()`, `plot.bamlss()`, `residuals.bamlss()`, etc., is to follow the naming convention of the returned estimates. The parameter names are based on the names of the distribution parameters as specified in the family object. For example, the family object `gaussian_bamlss()` has parameter names `"mu"` and `"sigma"`

```
R> gaussian_bamlss()$names

[1] "mu"    "sigma"
```

Then, each distributional parameter can be modeled by parametric (linear) and nonlinear smooth effect terms. The parametric part is indicated with `"p"` and the smooth part with `"s"`. The names of the parametric coefficients are the names of the corresponding model matrices as returned from `bamlss.frame()`. E.g., if two linear effects, with variables `"x1"` and `"x2"`, enter the model for distributional parameter `"mu"`, then the final names are `"mu.p.x1"` and `"mu.p.x2"`. Similarly for the smooth parts, if we model a variable `"x3"` using a regression spline as provided by the `s()` function of the **mgcv** package, the name is based on the names that are used by `bamlss.frame()` for the `smooth.construct()` object. In this case the parameter names start with `"mu.s.s(x3)"`. If this smooth term has 10 regression coefficients, then the final name must be

```
R> paste0("mu.s.s(x3)", ".b", 1:10)

 [1] "mu.s.s(x3).b1"  "mu.s.s(x3).b2"  "mu.s.s(x3).b3"
 [4] "mu.s.s(x3).b4"  "mu.s.s(x3).b5"  "mu.s.s(x3).b6"
 [7] "mu.s.s(x3).b7"  "mu.s.s(x3).b8"  "mu.s.s(x3).b9"
[10] "mu.s.s(x3).b10"
```

i.e., all smooth term parameters are named with "b" and a numerated.

An example of how to setup an estimation engine for **bamlss** for linear regression models is given in Appendix C. The example also provides details on the naming convention and return values of optimizer and sampler functions.

5. Flexible count regression for lightning reanalysis

This section illustrates the workflow with **bamlss** along a small case study. We want to build a statistical model linking positive counts of cloud-to-ground lightning discharges to atmospheric quantities from a reanalysis dataset.

5.1. Motivation and data

The region we focus on are the European Eastern Alps. Cloud-to-ground lightning discharges – detected by the Austrian Lightning Detection and Information System (ALDIS, Schulz, Cummins, Diendorfer, and Dorninger 2005) – are counted on grids with a mesh size of 32 *km*. The lightning observations are available for the period 2010–2018. The reanalysis data come from ERA5, the fifth generation of the ECMWF (European Centre for Medium-Range Weather Forecasts) atmospheric reanalyses of the global climate (Copernicus Climate Change Service 2017; Hersbach and et al. 2020). ERA5 provides globally complete and consistent pseudo-observations of the atmosphere using the laws of physics. The horizontal resolution is approx. 32 *km*, while the temporal resolution is hourly and covers the years from 1979 to present. In this example application we work only with a small subset of the data, which can be assessed from the accompanying R package **FlashAustria** (Simon 2021). The data are loaded with

```
R> data("FlashAustria", package = "FlashAustria")
R> head(FlashAustriaTrain)[, 1:6]

  counts      d2m   q_prof_PC1 cswc_prof_PC4 t_prof_PC1 v_prof_PC2
1      2 291.3184 -0.011472293  7.168725e-06  15.922548  2.5646172
2     16 283.5004  0.001007288  1.612870e-05  -9.758380  0.7955608
3      1 291.0506 -0.005590341 -3.226052e-06  20.274007  7.5535312
4      7 288.0358 -0.006293043  3.715074e-05  14.258116  5.8523424
5     41 288.4433 -0.006315605  3.509800e-05   8.757239  8.3675943
6      1 286.6035 -0.001597900 -3.195042e-06  -3.433136 -3.4291366

R> nrow(FlashAustriaTrain)

[1] 12000
```

The motivation for this application is as follows: lightning counts are not modeled within the atmospheric reanalyses, as their spatial resolution is too coarse for resolving convective events that lead to lightning discharges. Homogeneous lightning observations are only available for the period in the order of a decade, here 2010–2018. Thus, based on a probabilistic statistical model, lightning counts for the time before 2010 could be fitted, thus enabling the analysis

Abbreviation	Description
`d2m`	2 metre dewpoint temperature is a measure of the humidity of the air. The temperature to which the air, at 2 metres above the surface of the Earth, would have to be cooled for saturation to occur.
`q_prof_PC1`	The vertical profile of specific humidity `q` has been decomposed by principal component analysis (PCA). This is the first principal component.
`cswc_prof_PC4`	The vertical profile of specific snow water content `cswc` has been decomposed by PCA. This is the forth principal component.
`t_prof_PC1`	The vertical profile of temperature `t` has been decomposed by PCA. This is the first principal component.
`v_prof_PC2`	The vertical profile of the v-component of the wind `v` has been decomposed by PCA. This is the second principal component.
`sqrt_cape`	The square root of convective available potential energy. This is an indication of the (in)stability of the atmosphere.
`sqrt_lsp`	Large-scale precipitation. Accumulated liquid and frozen water, comprising rain and snow, which is generated by the cloud scheme of the numerical model.

Table 4: Quantities taken directly or derived from ERA5.

of lightning events in the past for which no observations are available. On the one hand this will increase our knowledge about physical processes leading to such events, and on the other it will enable quantification how these extreme short-term events are affected by changing climate (Westra *et al.* 2014).

Table 4 lists the covariates considered which are based on a small subset of ERA5 quantities (Copernicus Climate Change Service 2017; Hersbach and et al. 2020) and include variables that are known to be good predictors for convective events (e.g., Simon *et al.* 2018).

5.2. Model specification

The response of our statistical model are positive counts, with a mean of 13.61, and a variance of 1180.63. Thus, we are facing a truncated count data distribution which is highly overdispersive (Cameron and Trivedi 2013). Simon, Mayr, Umlauf, and Zeileis (2019) employed a zero-truncated negative binomial distribution, which is specified by two parameters $\mu > 0$ and $\theta > 0$. μ is the expectation of the underlying untruncated negative binomial, and θ modifies the variance of the untruncated negative binomial by $\text{VAR}(\tilde{Y}) = \mu + \mu^2/\theta$, where $\tilde{Y}$ is a latent random variable following the underlying untruncated negative binomial distribution.

The spatial and temporal scale of aggregation of the lightning discharges here differs from the one in Simon *et al.* (2019). Therefore, it is worth comparing the zero-truncated negative binomial against other distributions that could capture the truncation of the count data and its overdispersion. Hence, we also consider the zero-truncated Sichel distribution which can also capture skewed count data.

The zero-truncated negative binomial distribution is implemented as `ztnbinom_bamlss()` within **bamlss** while the Sichel is available as `SICHEL()` within **gamlss.dist**. Using the **gamlss.tr** package the latter is truncated at zero so that it can be readily plugged into the `family` argument of `bamlss()`.

```
R> library("gamlss.dist")
R> library("gamlss.tr")
R> ztSICHEL <- trun(0, family = "SICHEL", local = FALSE)
```

In the following we illustrate how to model the lightning counts with one of the two distributions. To specify smooth terms for all distributional parameters – for `ztnbinom_bamlss()` parameters μ and θ and for `ztSICHEL()` parameters μ, σ and ν – we set up a `list` of three formulas. Smooth P-splines (Eilers and Marx 1996), known for their good sampling properties, are employed for all predictors in the formula for μ. For the (over)dispersion model large-scale precipitation is used in the second formula (without a parameter name on the left-hand side in order to be applicable to both distributional models). Finally, for the Sichel distribution a constant shape parameter is added in the third formula (which is ignored when using the formula list with the zero-truncated negative binomial distribution).

```
R> f <- list(
+    counts ~ s(d2m, bs = "ps") + s(q_prof_PC1, bs = "ps") +
+             s(cswc_prof_PC4, bs = "ps") + s(t_prof_PC1, bs = "ps") +
+             s(v_prof_PC2, bs = "ps") + s(sqrt_cape, bs = "ps"),
+           ~ s(sqrt_lsp, bs = "ps"),
+           ~ 1
+  )
```

Now, we have all ingredients on hand to feed the standard interface for statistical models in R: a formula `f`, families `ztnbinom_bamss()`, `ztSICHEL()`, and a data set `FlashAustriaTrain`. Within the `bamlss()` call we also provide arguments which are passed forward to the optimizer and the sampler. We choose the gradient boosting optimizer `opt_boost()` in order to find initial values for the default sampler `sam_GMCMC()`. Gradient boosting proved to offer a very stable method for finding regression coefficients that serve as initial values for a MCMC sampler (Simon *et al.* 2019). In the following, we illustrate the estimation of the models with the `ztSICHEL()` family. We set the number of iterations to `maxit = 1000`. For the sampling we allow 1000 iterations as burn-in phase, and apply a thinning of the resulting chain of 3. Running `n.iter = 2000` iterations on 3 cores in parallel leads to 1000 MCMC samples in the end (note that parallel chains are started using function `mclapply()` of the base R **parallel** package by setting argument `cores`, see the manual of sampler function `sam_MCMC()`).

```
R> set.seed(123)
R> flash_model_ztSICHEL <- bamlss(f, data = FlashAustriaTrain,
+    family = ztSICHEL, binning = TRUE,
+    optimizer = opt_boost, maxit = 1000,
+    thin = 3, burnin = 1000, n.iter = 2000,
+    light = TRUE, cores = 3)

logLik -36636.9 eps 0.0003 iteration 1000 qsel 7
elapsed time: 28.99min
Starting the sampler...
|********************| 100%  0.00sec 115.67min
```

The model was fitted on three Intel Xeon CPU E5-4660 v4 cores with 2.20GHz on which the boosting took about 28.99 minutes and the average sampling time about 1.9 hours. This is relatively slow as the `ztSICHEL()` uses a rather generic high-level R implementation. To fit the model `flash_model_ztnbinom` we rather used `family = ztnbinom_bamlss()` than `family = ztSICHEL`, all other specifications for optimization left untouched.

```
> set.seed(123)
> flash_model_ztnbinom <- bamlss(f, data = FlashAustriaTrain,
+   family = "ztnbinom", binning = TRUE,
+   optimizer = opt_boost, maxit = 1000,
+   thin = 3, burnin = 1000, n.iter = 2000,
+   light = TRUE, cores = 3)

logLik -36700.7 eps 0.0006 iteration 1000 qsel 7
elapsed time:  7.47min
Starting the sampler...
|********************| 100%  0.00sec  9.12min
```

Computation times for the more efficient `ztnbinom_bamlss()` family are much lower with about 7 and 9 minutes for boosting and sampling, respectively. By setting the argument `binning = TRUE` only unique observations are being used internally to create the design matrices. This can have two advantages, (a) for large data sets one does not run into memory problems so easily, (b) the estimation algorithms are sometimes much faster, because fewer floating point operations need to be performed. In addition, when setting `light = TRUE`, no design or penalty matrices, model frames, results, etc., are stored. Hence, memory requirements when storing the model objects are much lower so that the fitted model objects can be provided within the **FlashAustria** package.

```
R> data("FlashAustriaModel", package = "FlashAustria")
```

The corresponding R code is provided in the supplemental materials.

5.3. Model diagnostics

To select one of the two models, we examine their calibration using a worm plot (Van Buuren and Fredriks 2001). The worm plot is implemented within **bamlss** for objects of class `bamlss.residuals` and can be selected via the `which` argument of the plot method for these objects:

```
R> resids <- c(
+    "ztnbinom" = residuals(flash_model_ztnbinom),
+    "ztSICHEL" = residuals(flash_model_ztSICHEL)
+  )
R> plot(resids, which = "wp", main = "Worm plot")
```

The worm plots (Figure 8) reveal that both `ztSICHEL` and `ztnbinom` somewhat underestimate the mass of the upper tail. However, for `ztSICHEL` the effect is less pronounced and overall

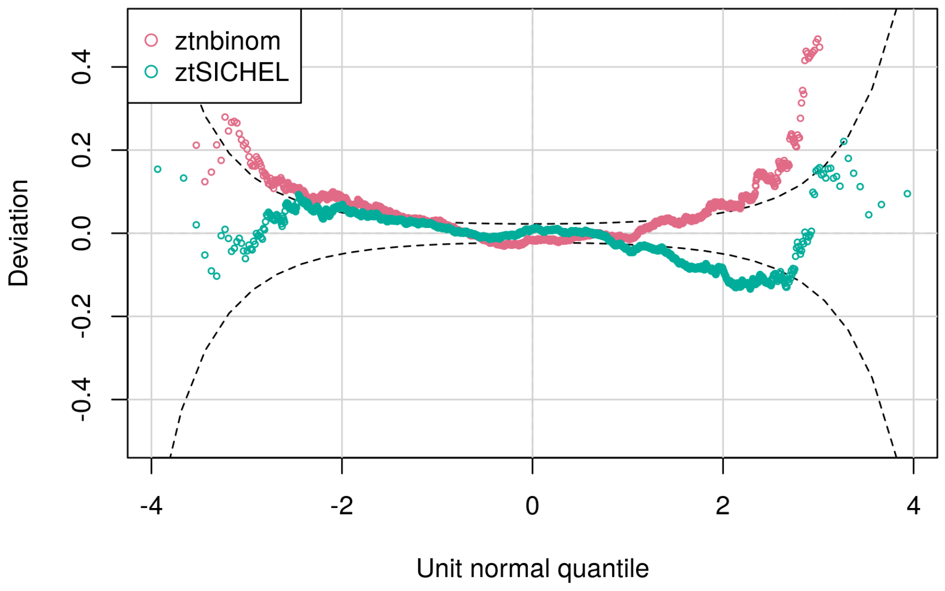

Figure 8: Worm plots for the two distributional models.

calibration is much better than for `ztnbinom`. Hence, we focus on the `ztSICHEL()` model but remark that most qualitative insights are very similar for `ztnbinom`.

As a next diagnostic we check the log-likelihood contributions of the individual terms during the boosting optimization (Figure 9).

```
R> pathplot(flash_model_ztSICHEL, which = "loglik.contrib")
```

After 1000 iterations the term `s(q_prof_PC1).mu` has the highest contribution to the log-likelihood with 144 followed by `s(sqrt_cape).mu` with 115 and `s(d2m).mu` with 50. Overall contributions to the log-likelihood at the end of the boosting procedure are very small signaling that the algorithm approach a state that is suitable for initializing the MCMC sampling.

The MCMC chains can be assessed by visualizations of their traces and autocorrelation functions (ACFs), exemplified in Figure 10 for the term `s(q_prof_PC1)` (for parameter μ of the Sichel distribution).

```
R> plot(flash_model_ztSICHEL, model = "mu", term  = "s(q_prof_PC1)",
+    which = "samples")
```

The traces reveal samples around stables means, confirming that the 1000 boosting iterations and the 1000 burn-in samples were sufficient. The ACFs reveal quite some autocorrelation after the thinning, suggesting that sampling efforts should be increased further in a final model run.

5.4. Predictions and visualizations

As the boosting summary (Figure 9) reveals that the terms `s(q_prof_PC1)`, `s(sqrt_cape)` and `s(d2m)` have the largest contribution for improving the fit, the corresponding effects

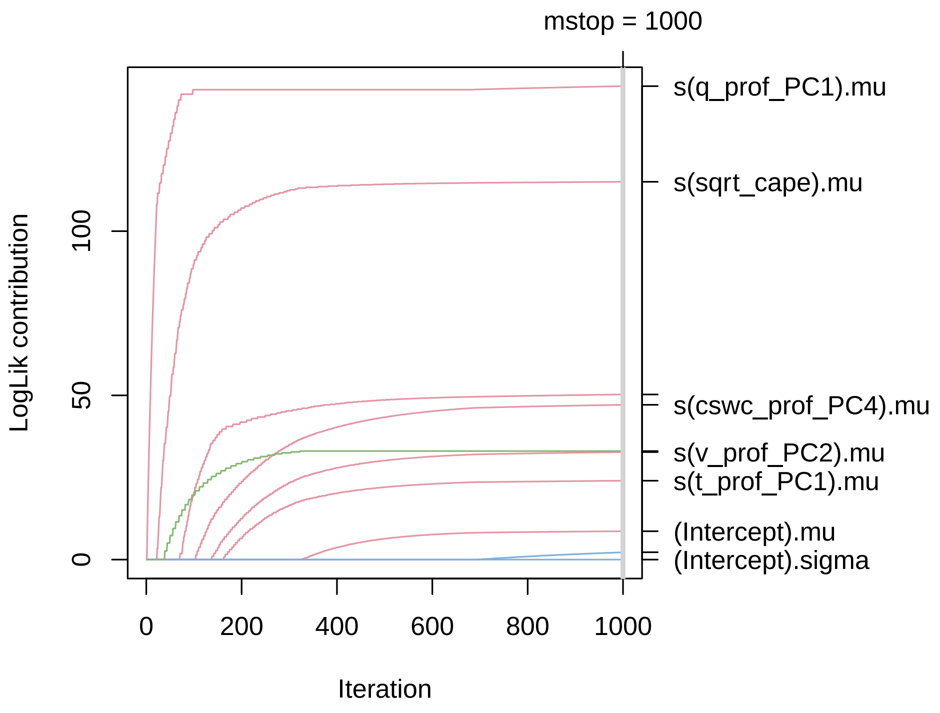

Figure 9: Contribution to the log-likelihood of individual terms during gradient boosting.

are shown in Figure 11 to illustrate how the atmospheric quantities of the reanalyses are related to lightning events. The effects are presented on the scale of the additive predictor of the distributional parameter μ, i.e., the log scale. A higher $\log(\mu)$ would result in a higher expectation of the count data distribution.

```
R> plot(flash_model_ztSICHEL, model = "mu",
+    term  = c("s(q_prof_PC1)", "s(sqrt_cape)", "s(d2m)"))
```

`s(q_prof_PC1)` shows a clear decrease. As `q_prof_PC1` is the leading principal component of the vertical profile of specific humidity, one has to consider the corresponding spatial mode (not shown) for interpretation: positive values of `q_prof_PC1` are linked to more moisture in the lower atmosphere (below 850 hPa) and less moisture in the mid atmosphere (between 850 hPa and 600 hPa). Thus, smaller values of the principal component mean that more moisture is available in the mid atmosphere, a source of *latent energy*, energy that becomes free when water transfers from the gas to the liquid phase. This energy supports the occurrence of deep convection and thus of heavy lightning events. `s(sqrt_cape)` reveals an increasing shape. This means a higher *convective available potential energy* (CAPE) increases μ, which increases the expectation of the distribution and thus is associated with higher probabilities for events with high counts. Physically the shape of the effect is meaningful as more convective available potential energy has the potential to lead to heavier lightning events. The similar is true for the increasing effect of `s(d2m)`. Finally, the model is leveraged to predict a case for the period before 2010, for which no lightning data are available. The case of interest is a front moving from the West to the East on the Northern side of the Alps on 2001-09-15 and 2001-09-16. The case data `FlashAustriaCase` contains additional columns containing time and

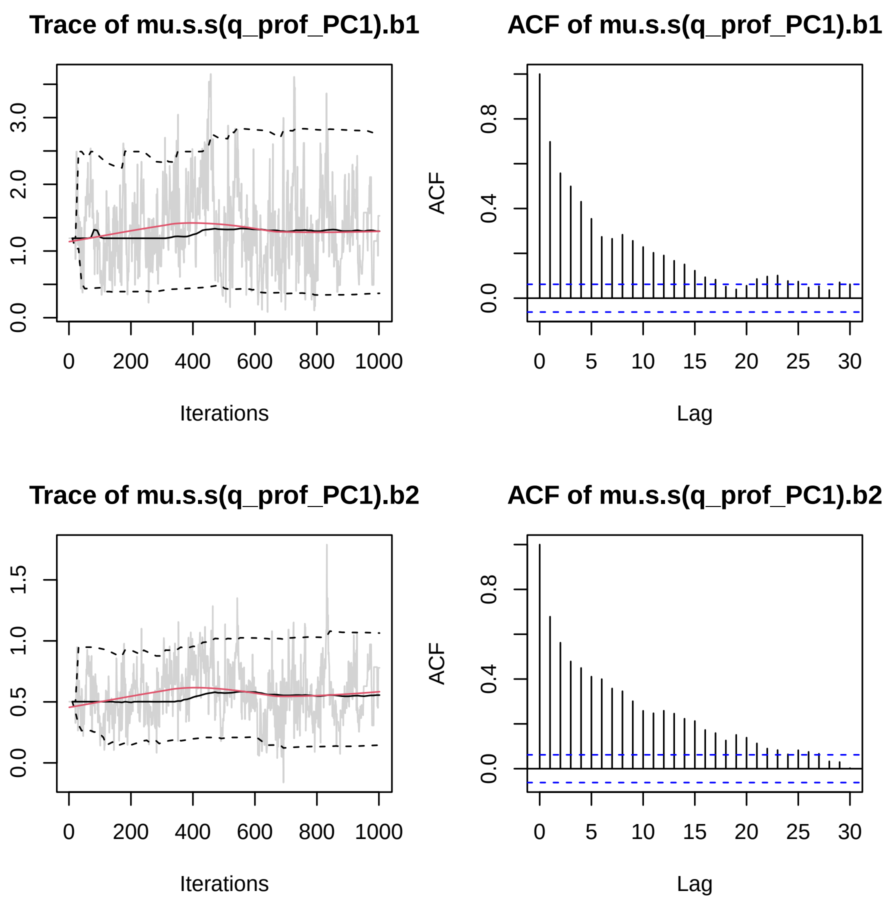

Figure 10: MCMC trace (left panels) and autocorrelation (right panels) for two parameters from the term `s(q_prof_PC1)` of the model `mu`.

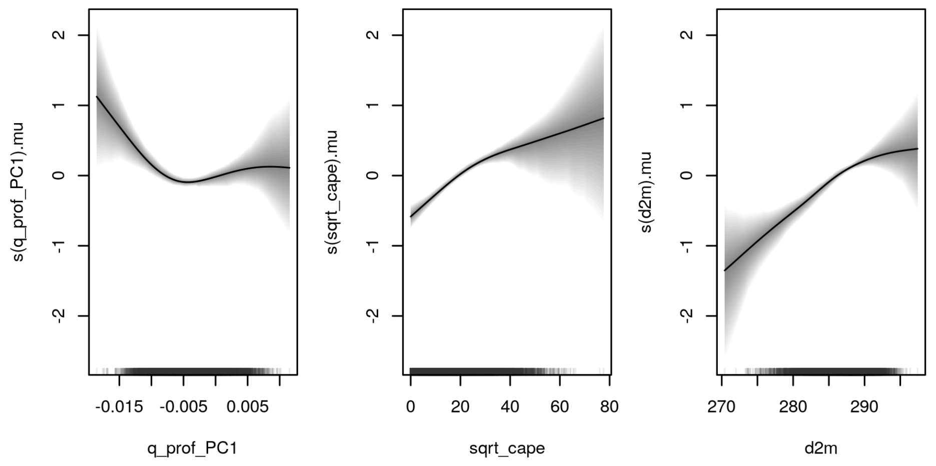

Figure 11: Effect of the terms `s(q_prof_PC1)`, `s(sqrt_cape)`, and term `s(d2m)` from model `mu`. Credible intervals derived from MCMC samples.

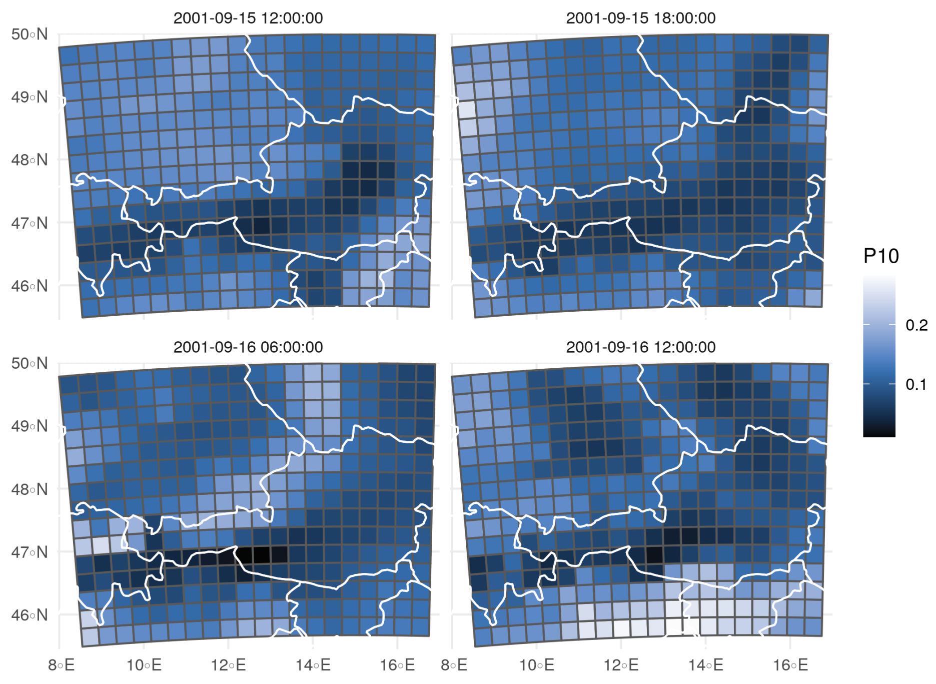

Figure 12: A probabilistic reconstruction of lightning counts occurred on September 15 2001 at 12 UTC and at 18 UTC and on September 16 2001 at 6 UTC and 12 UTC, i.e., the probability of having observed 10 or more counts within one grid box.

space information, and is of class `sf` (Pebesma 2018). We predict the parameters for this case, and derive the probability of observing 10 or more flashes within a grid box conditioned on thunderstorm activity, by applying the cumulative distribution function `...$p()` of the family which can be extracted from the fitted model using `family()`. The family contains functions to map the predictors to the parameter scale, density, cumulative distribution function, log-likelihood, and scores and Hessian. We apply the cdf to compute the probability of observing more than 10 flashes in a box and hour given a lightning event. The function `...$p()` takes the quantile as first argument, and the `list` with the parameters, as returned by `predict()`, as a second argument.

```
R> library("sf")
R> fit <- predict(flash_model_ztSICHEL,
+    newdata = FlashAustriaCase, type = "parameter")
R> fam <- family(flash_model_ztSICHEL)
R> FlashAustriaCase$P10 <- 1 - fam$p(9, fit)
```

We visualize this case by employing `ggplot()` (Wickham 2016), and the Oslo color scale from the **colorspace** package (Zeileis *et al.* 2020). The country borders `world` are retrieved from the **rnaturalearth** package (South 2017).

```
R> library("ggplot2")
R> world <- rnaturalearth::ne_countries(scale = "medium", returnclass = "sf")
R> ggplot() + geom_sf(aes(fill = P10), data = FlashAustriaCase) +
+     scale_fill_continuous_sequential("Oslo", rev = TRUE) +
+     geom_sf(data = world, col = "white", fill = NA) +
+     coord_sf(xlim = c(7.95, 17), ylim = c(45.45, 50), expand = FALSE) +
+     facet_wrap(~time) + theme_minimal()
```

The case gives an example for a reconstructed lightning event: On 2001-09-15 (top row of Figure 12) it reveals moderate probabilities for observing 10 or more lightning counts within a grid cell and hour. Over night the situation is still unstable, around 6 UTC on 2001-09-16 there is still quite high activity on the northern parts of the Alps. (bottom left panel of Figure 12). Finally, strong lightning events were reconstructed for the afternoon of the second day (2001-09-16) in particular on the Southern side of the main Alpine ridge (bottom right panel of Figure 12).

6. Conclusion

The R package **bamlss** is a very comprehensive software to estimate Bayesian distributional regression models. The package is primarily based on the typical R "look & feel", which makes it easy to get started with the package. Similar to other implementations, **bamlss** is a modular "Lego toolbox", however, the package stands out from others in that it makes complete sampling and/or optimizer functions exchangeable, so that users who are interested in extensions can easily set up new models, where all elaborate data processing infrastructure and extractor functions are completely provided by the package. Several examples illustrate the functionality of the package. For the future it is planned to provide algorithms for Gigadata, a topic that the package so far treats only very superficially. In addition, it is planned to expand the family infrastructure to support families from other implementations more easily.

Acknowledgments

We thank two anonymous reviewers for critically reading the manuscript and suggesting substantial improvements. Nadja Klein acknowledges support through the Emmy Noether grant KL 3037/1-1 of the German research foundation (DFG). Thorsten Simon acknowledges the funding by the Austrian Science Fund (FWF, grant no. P 31836)

References

Aitkin M (1987). "Modelling Variance Heterogeneity in Normal Regression Using GLIM." *Journal of the Royal Statistical Society C*, **36**(3), 332–339. doi:10.2307/2347792.

Belitz C, Brezger A, Klein N, Kneib T, Lang S, Umlauf N (2015). ***BayesX*** *– Software for Bayesian Inference in Structured Additive Regression Models*. Version 3.0.2, URL http://www.BayesX.org/.

Bivand RS, Gómez-Rubio V, Rue H (2015). "Spatial Data Analysis with R-**INLA** with Some Extensions." *Journal of Statistical Software*, **63**(20), 1–31. doi:10.18637/jss.v063.i20.

Brezger A, Kneib T, Lang S (2005). "**BayesX**: Analyzing Bayesian Structured Additive Regression Models." *Journal of Statistical Software*, **14**(11), 1–22. doi:10.18637/jss.v014.i11.

Brezger A, Lang S (2006). "Generalized Structured Additive Regression Based on Bayesian P-Splines." *Computational Statistics & Data Analysis*, **50**, 947–991. doi:10.1016/j.csda.2004.10.011.

Brooks SP, Gelman A (1998). "General Methods for Monitoring Convergence of Iterative Simulations." *Journal of Computational and Graphical Statistics*, **7**(4), 434–455. doi:10.1080/10618600.1998.10474787.

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

Cameron AC, Trivedi PK (2013). *Regression Analysis of Count Data*. Econometric Society Monographs, 2nd edition. Cambridge University Press, Cambridge.

Carpenter B, Gelman A, Hoffman MD, Lee D, Goodrich B, Betancourt M, Brubaker M, Guo J, Li P, Riddell A (2017). "Stan: A Probabilistic Programming Language." *Journal of Statistical Software*, **76**(1), 1–32. doi:10.18637/jss.v076.i01.

Cavanaugh JE (1997). "Unifying the Derivations for the Akaike and Corrected Akaike Information Criteria." *Statistics & Probability Letters*, **33**(2), 201–208. doi:10.1016/S0167-7152(96)00128-9.

Chambers JM, Hastie TJ (eds.) (1992). *Statistical Models in S*. Chapman & Hall, London.

Copernicus Climate Change Service (2017). "ERA5: Fifth Generation of ECMWF Atmospheric Reanalyses of the Global Climate." Copernicus Climate Change Service Climate Date Store (CDS). Date of access: June 2019, https://cds.climate.copernicus.eu/cdsapp#!/home.

Dunn PK, Smyth GK (1996). "Randomized Quantile Residuals." *Journal of Computational and Graphical Statistics*, **5**(3), 236–244. doi:10.2307/1390802.

Eilers PHC, Marx BD (1996). "Flexible Smoothing Using B-Splines and Penalized Likelihood." *Statistical Science*, **11**, 89–121. doi:10.1214/ss/1038425655.

Fahrmeir L, Kneib T, Lang S (2004). "Penalized Structured Additive Regression for Space Time Data: A Bayesian Perspective." *Statistica Sinica*, **14**, 731–761. doi:10.1007/978-3-642-34333-9_9.

Fouillet A, Rey G, Wagner V, Laaidi K, Empereur-Bissonnet P, Le Tertre A, Frayssinet P, Bessemoulin P, Laurent F, De Crouy-Chanel P, Jougla E, Hémon D (2008). "Has the Impact of Heat Waves on Mortality Changed in France since the European Heat Wave of Summer 2003? A Study of the 2006 Heat Wave." *International Journal of Epidemiology*, **37**(2), 309–317. ISSN 0300-5771. doi:10.1093/ije/dym253.

Friedman J, Hastie T, Tibshirani R (2010). "Regularization Paths for Generalized Linear Models via Coordinate Descent." *Journal of Statistical Software*, **33**(1), 1–22. doi:10.18637/jss.v033.i01.

Gamerman D (1997). "Sampling from the Posterior Distribution in Generalized Linear Mixed Models." *Statistics and Computing*, **7**(1), 57–68. doi:10.1023/a:1018509429360.

Gelman A (2006). "Prior Distributions for Variance Parameters in Hierarchical Models (Comment on Article by Browne and Draper)." *Bayesian Analysis*, **1**(3), 515–534. doi:10.1214/06-ba117a.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4), 457–472. doi:10.1214/ss/1177011136.

Gerfin M (1996). "Parametric and Semi-Parametric Estimation of the Binary Response Model of Labour Market Participation." *Journal of Applied Econometrics*, **11**(3), 321–339. doi:10.1002/(SICI)1099-1255(199605)11:3<321::AID-JAE391>3.0.CO;2-K.

Gneiting T, Balabdaoui F, Raftery AE (2007). "Probabilistic Forecasts, Calibration and Sharpness." *Journal of the Royal Statistical Society B*, **69**(2), 243–268. doi:10.1111/j.1467-9868.2007.00587.x.

Gneiting T, Raftery AE (2007). "Strictly Proper Scoring Rules, Prediction, and Estimation." *Journal of the American Statistical Association*, **102**(477), 359–378. doi:10.1198/016214506000001437.

Goudie RJB, Turner RM, Angelis DD, Thomas A (2020). "**MultiBUGS**: A Parallel Implementation of the BUGS Modeling Framework for Faster Bayesian Inference." *Journal of Statistical Software*, **95**(7), 1–20. doi:10.18637/jss.v095.i07.

Groll A, Hambuckers J, Kneib T, Umlauf N (2019). "LASSO-Type Penalization in the Framework of Generalized Additive Models for Location, Scale and Shape." *Computational Statistics & Data Analysis*, **140**, 59–74. doi:10.1016/j.csda.2019.06.005.

Hastie T, Tibshirani R (1990). *Generalized Additive Models*. Chapman & Hall/CRC, New York.

Heidelberger P, Welch PD (1981). "A Spectral Method for Confidence Interval Generation and Run Length Control in Simulations." *Communications of the ACM*, **24**(4), 233–245. doi:10.1145/358598.358630.

Heidelberger P, Welch PD (1983). "Simulation Run Length Control in the Presence of an Initial Transient." *Operations Research*, **31**(6), 1109–1144. doi:10.1287/opre.31.6.1109.

Hersbach H, et al (2020). "The ERA5 Global Reanalysis." *Quarterly Journal of the Royal Meteorological Society*, **146**(730), 1999–2049. doi:10.1002/qj.3803.

Herwartz H, Klein N, Strumann C (2016). "Modelling Hospital Admission and Length of Stay by Means of Generalised Count Data Models." *Journal of Applied Econometrics*, **31**(6), 1159–1182. doi:10.1002/jae.2454.

Hofner B, Mayr A, Schmid M (2016). "**gamboostLSS**: An R Package for Model Building and Variable Selection in the GAMLSS Framework." *Journal of Statistical Software*, **74**(1), 1–31. doi:10.18637/jss.v074.i01.

Hurvich CM, Tsai CL (1989). "Regression and Time Series Model Selection in Small Samples." *Biometrika*, **76**(2), 297–307. doi:10.1093/biomet/76.2.297.

Jordan A, Krüger F, Lerch S (2019). "Evaluating Probabilistic Forecasts with **scoringRules**." *Journal of Statistical Software*, **90**(12), 1–37. doi:10.18637/jss.v090.i12.

Kleiber C, Zeileis A (2008). *Applied Econometrics with R*. Springer-Verlag, New York. URL https://CRAN.R-project.org/package=AER.

Klein N, Denuit M, Lang S, Kneib T (2014). "Nonlife Ratemaking and Risk Management with Bayesian Generalized Additive Models for Location, Scale, and Shape." *Insurance: Mathematics and Economics*, **55**, 225 – 249. doi:10.1016/j.insmatheco.2014.02.001.

Klein N, Kneib T (2016a). "Scale-Dependent Priors for Variance Parameters in Structured Additive Distributional Regression." *Bayesian Analysis*, **11**(4), 1071–1106. doi:10.1214/15-ba983.

Klein N, Kneib T (2016b). "Simultaneous Inference in Structured Additive Conditional Copula Regression Models: A Unifying Bayesian Approach." *Statistics and Computing*, **26**(4), 841–860. doi:10.1007/s11222-015-9573-6.

Klein N, Kneib T, Klasen S, Lang S (2015a). "Bayesian Structured Additive Distributional Regression for Multivariate Responses." *Journal of the Royal Statistical Society C*, **64**, 569–591. doi:10.1111/rssc.12090.

Klein N, Kneib T, Lang S (2015b). "Bayesian Generalized Additive Models for Location, Scale and Shape for Zero-Inflated and Overdispersed Count Data." *Journal of the American Statistical Association*, **110**(509), 405–419. doi:10.1080/01621459.2014.912955.

Klein N, Kneib T, Lang S, Sohn A (2015c). "Bayesian Structured Additive Distributional Regression with an Application to Regional Income Inequality in Germany." *Annals of Applied Statistics*, **9**, 1024–1052. doi:10.1214/15-aoas823.

Klein N, Simon T, Umlauf N (2019). "Neural Network Regression with an Application to Leukaemia Survival Data – An Unstructured Distributional Approach." In *Proceedings of the 34th International Workshop on Statistical Modelling, Guimarães, Portugal*, volume 1, pp. 157–160. Statistical Modelling Society.

Köhler M, Umlauf N, Beyerlein A, Winkler C, Ziegler AG, Greven S (2017). "Flexible Bayesian Additive Joint Models with an Application to Type 1 Diabetes Research." *Biometrical Journal*, **59**(6), 1144–1165. doi:10.1002/bimj.201600224.

Köhler M, Umlauf N, Greven S (2018). "Nonlinear Association Structures in Flexible Bayesian Additive Joint Models." *Statistics in Medicine*, **37**(30), 4771–4788. doi:10.1002/sim.7967.

Lang S, Umlauf N, Wechselberger P, Harttgen K, Kneib T (2014). "Multilevel Structured Additive Regression." *Statistics and Computing*, **24**(2), 223–238. doi:10.1007/s11222-012-9366-0.

Leon DA, Shkolnikov VM, Smeeth L, Magnus P, Pechholdová M, Jarvis CI (2020). "COVID-19: A Need for Real-Time Monitoring of Weekly Excess Deaths." *The Lancet*, **395**(10234), e81. doi:https://doi.org/10.1016/S0140-6736(20)30933-8.

Lindgren F, Rue H (2015). "Bayesian Spatial Modelling with R-**INLA**." *Journal of Statistical Software*, **63**(19), 1–25. doi:10.18637/jss.v063.i19.

Lunn DJ, Thomas A, Best N, Spiegelhalter D (2000). "**WinBUGS** – A Bayesian Modelling Framework: Concepts, Structure, and Extensibility." *Statistics and Computing*, **10**, 325–337. doi:10.1023/a:1008929526011.

Mayr A, Fenske N, Hofner B, Kneib T, Schmid M (2012). "Generalized Additive Models for Location, Scale and Shape for High Dimensional Data: A Flexible Approach Based on Boosting." *Journal of the Royal Statistical Society C*, **61**(3), 403–427. doi:10.1111/j.1467-9876.2011.01033.x.

Neal RM (2003). "Slice Sampling." *The Annals of Statistics*, **31**(3), 705–767. doi:10.1214/aos/1056562461.

Nelder JA, Wedderburn RWM (1972). "Generalized Linear Models." *Journal of the Royal Statistical Society A*, **135**, 370–384. doi:10.2307/2344614.

Pebesma E (2018). "Simple Features for R: Standardized Support for Spatial Vector Data." *The R Journal*, **10**(1), 439–446. doi:10.32614/RJ-2018-009.

Plummer M (2003). "**JAGS**: A Program for Analysis of Bayesian Graphical Models Using Gibbs Sampling." In K Hornik, F Leisch, A Zeileis (eds.), *Proceedings of the 3rd International Workshop on Distributed Statistical Computing, Vienna, Austria*. URL http://www.ci.tuwien.ac.at/Conferences/DSC-2003/Proceedings/.

Plummer M (2021). ***rjags**: Bayesian Graphical Models using MCMC*. R package version 4-12, URL https://CRAN.R-project.org/package=rjags.

Plummer M, Best N, Cowles K, Vines K (2006). "**coda**: Convergence Diagnosis and Output Analysis for MCMC." *R News*, **6**(1), 7–11. URL https://cran.r-project.org/doc/Rnews/Rnews_2006-1.pdf.

Polson NG, Scott JG (2012). "On the Half-Cauchy Prior for a Global Scale Parameter." *Bayesian Analysis*, **7**(4), 887–902. doi:10.1214/12-ba730.

R Core Team (2021). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rigby R, Stasinopoulos M, Heller G, Bastiani F (2019). *Distributions for Modeling Location, Scale, and Shape: Using GAMLSS in R*. Chapman & Hall/CRC. doi:10.1201/9780429298547.

Rigby RA, Stasinopoulos DM (2005). "Generalized Additive Models for Location, Scale and Shape." *Journal of the Royal Statistical Society C*, **54**(3), 507–554. doi:10.1111/j.1467-9876.2005.00510.x.

Rue H, Martino S, Chopin N (2009). "Approximate Bayesian Inference for Latent Gaussian Models by Using Integrated Nested Laplace Approximations." *Journal of the Royal Statistical Society B*, **71**(2), 319–392. doi:10.1111/j.1467-9868.2008.00700.x.

Schulz W, Cummins K, Diendorfer G, Dorninger M (2005). "Cloud-to-Ground Lightning in Austria: A 10-Year Study Using Data from a Lightning Location System." *Journal of Geophysical Research: Atmospheres*, **110**(D9). doi:10.1029/2004JD005332.

Simon T (2021). ***FlashAustria**: Data and Model for Reanalyzing Flash Counts in Austria.* R package version 0.4-2/r4223, URL https://R-Forge.R-project.org/projects/bayesr/.

Simon T, Fabsic P, Mayr GJ, Umlauf N, Zeileis A (2018). "Probabilistic Forecasting of Thunderstorms in the Eastern Alps." *Monthly Weather Review*, **146**, 2999–3009. doi:10.1175/MWR-D-17-0366.1.

Simon T, Mayr GJ, Umlauf N, Zeileis A (2019). "NWP-Based Lightning Prediction Using Flexible Count Data Regression." *Advances in Statistical Climatology, Meteorology and Oceanography*, **5**(1), 1–16. doi:10.5194/ascmo-5-1-2019.

Smyth GK (1996). "Partitioned Algorithms for Maximum Likelihood and Other Non-Linear Estimation." *Statistics and Computing*, **6**(3), 201–216. doi:10.1007/bf00140865.

South A (2017). ***rnaturalearth**: World Map Data from Natural Earth.* R package version 0.1.0, URL https://CRAN.R-project.org/package=rnaturalearth.

Spiegelhalter DJ, Best NG, Carlin BP, Van der Linde A (2002). "Bayesian Measures of Model Complexity and Fit." *Journal of the Royal Statistical Society B*, **64**(4), 583–639. doi:10.1111/1467-9868.00353.

Stadlmann S (2021). ***distreg.vis**: Framework for the Visualization of Distributional Regression Models.* R package version 1.7.2, URL https://CRAN.R-project.org/package=distreg.vis.

Stasinopoulos DM, Rigby RA (2021a). ***gamlss**: Generalised Additive Models for Location, Scale and Shape.* R package version 5.3-4, URL https://CRAN.R-project.org/package=gamlss.

Stasinopoulos DM, Rigby RA (2021b). ***gamlss.dist**: Distributions for Generalized Additive Models for Location, Scale and Shape.* R package version 5.3-2, URL https://CRAN.R-project.org/package=gamlss.dist.

Stasinopoulos MD, Rigby RA, Heller GZ, Voudouris V, Bastiani FD (2017). *Flexible Regression and Smoothing: Using GAMLSS in R.* CRC Press. doi:10.1201/b21973.

Umlauf N, Adler D, Kneib T, Lang S, Zeileis A (2015). "Structured Additive Regression Models: An R Interface to **BayesX**." *Journal of Statistical Software*, **63**(21), 1–46. doi:10.18637/jss.v063.i21.

Umlauf N, Adler D, Kneib T, Lang S, Zeileis A (2021). ***BayesXsrc**: R Package Distribution of the **BayesX** C++ Sources.* R package version 3.0-1.1, URL https://CRAN.R-project.org/package=BayesXsrc.

Umlauf N, Klein N, Zeileis A (2018). "BAMLSS: Bayesian Additive Models for Location, Scale, and Shape (and Beyond)." *Journal of Computational and Graphical Statistics*, **27**(3), 612–627. doi:10.1080/10618600.2017.1407325.

Umlauf N, Kneib T (2018). "A Primer on Bayesian Distributional Regression." *Statistical Modelling*, **18**(3-4), 219–247. doi:10.1177/1471082X18759140.

Van Buuren S, Fredriks M (2001). "Worm Plot: A Simple Diagnostic Device for Modelling Growth Reference Curves." *Statistics in Medicine*, **20**(8), 1259–1277. doi:10.1002/sim.746.

Wasserstein RL, Lazar NA (2016). "The ASA Statement on p Values: Context, Process, and Purpose." *The American Statistician*, **70**(2), 129–133. doi:10.1080/00031305.2016.1154108.

Watanabe S (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *The Journal of Machine Learning Research*, **11**, 3571–3594. URL http://jmlr.org/papers/v11/watanabe10a.html.

Westra S, Fowler HJ, Evans JP, Alexander LV, Berg P, Johnson F, Kendon EJ, Lenderink G, Roberts NM (2014). "Future Changes to the Intensity and Frequency of Short-Duration Extreme Rainfall." *Reviews of Geophysics*, **52**(3), 522–555. doi:10.1002/2014RG000464.

Wickham H (2016). ***ggplot2**: Elegant Graphics for Data Analysis.* Springer-Verlag, New York. URL https://ggplot2.tidyverse.org.

Wood SN (2016). "Just Another Gibbs Additive Modeler: Interfacing **JAGS** and **mgcv**." *Journal of Statistical Software*, **75**(7), 1–15. doi:10.18637/jss.v075.i07.

Wood SN (2017). *Generalized Additive Models: An Introduction with R*. 2nd edition. Chapman & Hall/CRC, Boca Raton.

Wood SN (2021). ***mgcv**: Mixed GAM Computation Vehicle with Automatic Smoothness Estimation.* R package version 1.8-38, URL https://CRAN.R-project.org/package=mgcv.

Wood SN, Li Z, Shaddick G, Augustin NH (2017). "Generalized Additive Models for Gigadata: Modelling the UK Black Smoke Network Daily Data." *Journal of the American Statistical Association*, **112**(519), 1199–1210. doi:10.1080/01621459.2016.1195744.

Wood SN, Pya N, Säfken B (2016). "Smoothing Parameter and Model Selection for General Smooth Models." *Journal of the American Statistical Association*, **111**(516), 1548–1563. doi:10.1080/01621459.2016.1180986.

Yee TW (2010). "The **VGAM** Package for Categorical Data Analysis." *Journal of Statistical Software*, **32**(10), 1–34. doi:10.18637/jss.v032.i10.

Zeileis A, Croissant Y (2010). "Extended Model Formulas in R: Multiple Parts and Multiple Responses." *Journal of Statistical Software*, **34**(1), 1–13. doi:10.18637/jss.v034.i01.

Zeileis A, Fisher JC, Hornik K, Ihaka R, McWhite CD, Murrell P, Stauffer R, Wilke CO (2020). "**colorspace**: A Toolbox for Manipulating and Assessing Colors and Palettes." *Journal of Statistical Software*, **96**(1), 1–49. doi:10.18637/jss.v096.i01.

A. Gaussian family object

The following R code shows an example implementation of the Gaussian distribution as presented in Section 4.2.

```
R> Gauss_bamlss <- function(...) {
+    f <- list(
+      "family" = "mygauss",
+      "names"  = c("mu", "sigma"),
+      "links"  = c(mu = "identity", sigma = "log"),
+      "d" = function(y, par, log = FALSE) {
+        dnorm(y, mean = par$mu, sd = par$sigma, log = log)
+      },
+      "p" = function(y, par, ...) {
+        pnorm(y, mean = par$mu, sd = par$sigma, ...)
+      },
+      "r" = function(n, par) {
+        rnorm(n, mean = par$mu, sd = par$sigma)
+      },
+      "q" = function(p, par) {
+        qnorm(p, mean = par$mu, sd = par$sigma)
+      },
+      "score" = list(
+        mu = function(y, par, ...) {
+          drop((y - par$mu) / (par$sigma^2))
+        },
+        sigma = function(y, par, ...) {
+          drop(-1 + (y - par$mu)^2 / (par$sigma^2))
+        }
+      ),
+      "hess" = list(
+        mu = function(y, par, ...) {
+          drop(1 / (par$sigma^2))
+        },
+        sigma = function(y, par, ...) {
+          rep(2, length(y))
+        }
+      )
+    )
+    class(f) <- "family.bamlss"
+    return(f)
+  }
```

B. Special model terms

The default estimation engines `opt_bfit()` and `sam_GMCMC()` (also the gradient boosting optimizer function `boost()`) in **bamlss** provide support for the implementation of special model terms, i.e., model terms that cannot be represented by the **mgcv** smooth term constructor infrastructure. One simple example of such a special model term is a nonlinear growth curve, e.g., a nonlinear Gompertz curve

$$f(x; \boldsymbol{\beta}) = \beta_1 \cdot \exp(-\beta_2 \cdot \exp(-\beta_3 \cdot x)),$$

but also the lasso model term constructor `la()` presented in Section 2.2 is a special **bamlss** model term. The special model term constructor is needed in this case, since the growth curve is nonlinear in the parameters $\boldsymbol{\beta}$, hence, the default backfitting and sampling strategies cannot be applied. Fortunately, estimation algorithms in distributional regression can be split into separate updating equations (see also Section 3.2). This means that each model term can have its own updating function. The user interested in this feature only needs to write a new `smooth.construct()` and `Predict.matrix()` method.

The following R code implements a Gompertz growth model term which can be used by the default optimizer function `opt_bfit()` and sampling function `sam_GMCMC()` of the **bamlss** package. The new `smooth.construct()` method is

```
R> smooth.construct.gc.smooth.spec <- function(object, data, knots)
+  {
+    object$X <- matrix(as.numeric(data[[object$term]]), ncol = 1)
+    center <- if(!is.null(object$xt$center)) {
+      object$xt$center
+    } else TRUE
+    object$by.done <- TRUE
+    if(object$by != "NA")
+      stop("by variables not supported!")
+
+    ## Begin special elements to be used with opt_bfit() and sam_GMCMC().
+    object$fit.fun <- function(X, b, ...) {
+      f <- b[1] * exp(-b[2] * exp(-b[3] * drop(X)))
+      if(center)
+        f <- f - mean(f)
+      f
+    }
+    object$update <- bfit_optim
+    object$propose <- GMCMC_slice
+    object$prior <- function(b) { sum(dnorm(b, sd = 1000, log = TRUE)) }
+    object$fixed <- TRUE
+    object$state$parameters <- c("b1" = 0, "b2" = 0.5, "b3" = 0.1)
+    object$state$fitted.values <- rep(0, length(object$X))
+    object$state$edf <- 3
+    object$special.npar <- 3 ## Important!
+    ## End special elements.
+
+    ## Important, This is a special smooth constructor!
+    class(object) <- c("gc.smooth", "no.mgcv", "special")
+
+    object
+  }
```

In principle, the setup is very similar to the smooth constructor functions provided by the **mgcv** package. Only few elements need to be added:

- `fit.fun()`: A function of the data `X` and parameter vector `b` that evaluates the fitted values.
- `update()`: An updating function to be used with optimizer `opt_bfit()`.
- `propose()`: A MCMC propose function to be used with sampler `sam_GMCMC()`.
- `prior()`: Function of the parameters `b` that evaluates the log-prior. Note, additional functions can be `grad()` and `hess` that evaluate the first and second derivative of the log-prior w.r.t. the parameters `b`.
- `fixed`: Is the number of degrees of freedom fixed or not?
- `state`: This is a named list with starting values for the `"parameters"`, the `"fitted.values"` and degrees of freedom `"edf"`. Note that regression coefficients are always named with `"b*"` and shrinkage or smoothing variances with `"tau2*"` in the `"parameters"` vector.
- `special.npar`: How many parameters does this model term have in total? This is needed for internal setup, because the Gompertz function has three parameters but the design matrix only one column.

To compute predictions of this model term a new method for the `Predict.matrix()` function needs to be implemented, too.

```
R> Predict.matrix.gc.smooth <- function(object, data, knots)
+  {
+    X <- matrix(as.numeric(data[[object$term]]), ncol = 1)
+    X
+  }
```

Special model terms can then be used with the constructor function `s2()`. To illustrate the this feature in **bamlss**, we simulate heteroskedastic growth data with

$$\texttt{y} \sim \mathcal{N}(\mu = 2 + 1/(1 + \exp(0.5 \cdot (15 - \texttt{time}))), \log(\sigma) = -3 + 2 \cdot \cos(\texttt{time}/30 \cdot 6 - 3))$$

and subsequently estimate the model with slice sampling (Neal 2003) for $\boldsymbol{\beta}$ in the MCMC algorithm using the following R code

```
R> set.seed(111)
R> d <- data.frame("time" = 1:30)
R> d$y <- 2 + 1 / (1 + exp(0.5 * (15 - d$time))) +
+    rnorm(30, sd = exp(-3 + 2 * cos(d$time/30 * 6 - 3)))
R> f <- list(
+    y ~ s2(time, bs = "gc"),
+    sigma ~ s(time)
+  )
R> b <- bamlss(f, data = d, optimizer = opt_bfit, sampler = sam_GMCMC)
R> plot(b)
```

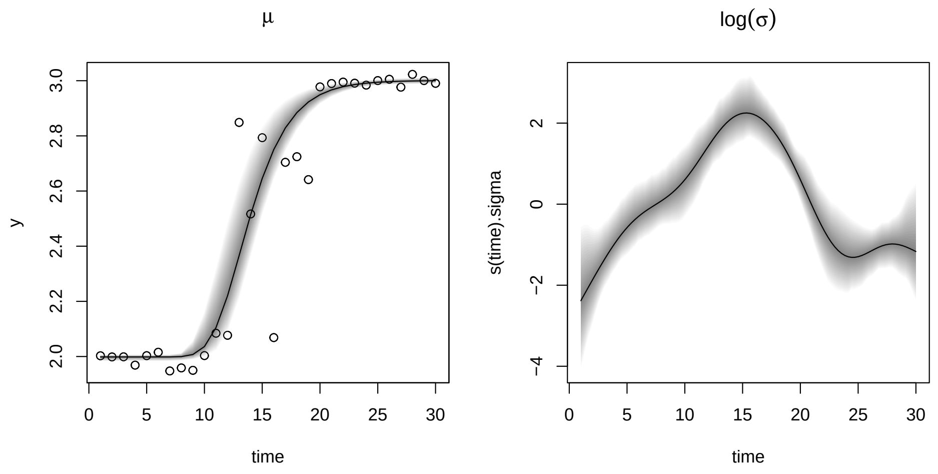

Figure 13: Estimated nonlinear effects on parameter μ and σ of the simulated growth curve example. Gray shaded areas represent 95% credible intervals.

The estimated effects are shown in Figure 13. The growth curve mean function estimate seems to fit the data quite well. Also, the nonlinear relationship for parameter σ could be captured by the model.

In summary, in order to build up special **bamlss** model terms only a few things have to be considered. The example R code for the Gompertz smooth constructor given here is a good starting point for readers interested in using this feature.

C. Model fitting engines for linear regression

In the following, to explain the setup and the naming convention of estimation engines in more detail, we implement

- a new family object for simple linear models $y = x^\top \boldsymbol{\beta} + \varepsilon$ with $\varepsilon \sim N(0, \sigma^2)$,
- and set up an optimizer function,
- and additionally a MCMC sampling function.

For illustration, the family object is kept very simple, we only model the mean function in terms of covariates.

```
R> lm_bamlss <- function(...) {
+    f <- list(
+      "family" = "LM",
+      "names" = "mu",
+      "links" = "identity",
+      "d" = function(y, par, log = FALSE) {
+        sigma <- sqrt(sum((y - par$mu)^2) / (length(y) - .lm_bamlss.p))
+        dnorm(y, mean = par$mu, sd = sigma, log = log)
+      },
+       "p" = function(y, par, ...) {
+         sigma <- sqrt(sum((y - par$mu)^2) / (length(y) - .lm_bamlss.p))
+         pnorm(y, mean = par$mu, sd = sigma, ...)
+       }
+     )
+     class(f) <- "family.bamlss"
+     return(f)
+   }
```

Now, for setting up the estimation functions we first simulate some data using the `GAMart()` function, afterwards the necessary `"bamlss.frame"` can be created with

```
R> d <- GAMart()
R> bf <- bamlss.frame(num ~ x1 + x2, data = d, family = "lm")
R> print(bf)

'bamlss.frame' structure:
  ..$ call
  ..$ model.frame
  ..$ formula
  ..$ family
  ..$ terms
  ..$ x
  .. ..$ mu
  .. .. ..$ formula
  .. .. ..$ fake.formula
  .. .. ..$ terms
  .. .. ..$ model.matrix
  ..$ y
  .. ..$ num
  ..$ delete
```

As noted above, the object is a named list with elements `"x"` and `"y"`, which will be passed to the estimation functions. For the moment, since we only implement a linear model, we need to work with the linear model matrix that is part of the `bf` object.

```
R> head(bf$x$mu$model.matrix)

  (Intercept)        x1           x2
1           1 0.8900343 0.7833063691
2           1 0.5579411 0.0009382977
3           1 0.9245978 0.7372933284
4           1 0.4532554 0.3819316742
5           1 0.5021393 0.0152245569
6           1 0.4715666 0.1129542917
```

and the response `y`

```
R> head(bf$y)

          num
1 -0.2649470
2 -0.4907917
3 -0.5134658
4 -0.1233677
5 -0.2206055
6  0.3200742
```

to setup the optimizer function with:

```
R> opt_LM <- function(x, y, ...)
+  {
+    ## Only univariate response.
+    y <- y[[1L]]
+
+    ## For illustration this is easier to read.
+    X <- x$mu$model.matrix
+
+    ## Estimate model parameters.
+    par <- drop(chol2inv(chol(crossprod(X))) %*% crossprod(X, y))
+
+    ## Set parameter names.
+    names(par) <- paste0("mu.p.", colnames(X))
+
+    ## Return estimated parameters and fitted values.
+    rval <- list(
+      "parameters" = par,
+      "fitted.values" = drop(X %*% par),
+      "edf" = length(par),
+      "sigma" = drop(sqrt(crossprod(y - X %*% par) / (length(y) - ncol(X))))
+    )
+
+    ## Set edf within .GlobalEnv for the
+    ## loglik() function in the lm_bamlss() family.
+    .lm_bamlss.p <<- length(par)
+
+    return(rval)
+  }
```

This optimizer function can already be used with the `bamlss()` wrapper function and all extractor functions are readily available.

```
R> f <- num ~ x1 + poly(x2, 5) + poly(x3, 5)
R> b <- bamlss(f, data = d, family = "lm", optimizer = opt_LM, sampler = FALSE)
R> summary(b)
```

```
Call:
bamlss(formula = f, family = "lm", data = d, optimizer = lm.opt,
    sampler = FALSE)
---
Family: LM
Link function: mu = identity
*---
Formula mu:
---
num ~ x1 + poly(x2, 5) + poly(x3, 5)
-
Parametric coefficients:
             parameters
(Intercept)       0.193
x1               -0.599
poly(x2, 5)1     -1.402
poly(x2, 5)2      2.300
poly(x2, 5)3      0.612
poly(x2, 5)4     -1.388
poly(x2, 5)5      0.861
poly(x3, 5)1     -0.107
poly(x3, 5)2      3.581
poly(x3, 5)3     -0.203
poly(x3, 5)4     -0.276
poly(x3, 5)5      0.087
---
Optimizer summary:
-
edf = 12 sigma = 0.2215
---
```

```
R> nd <- data.frame("x2" = seq(0, 1, length = 100))
R> nd$p <- predict(b, newdata = nd, term = "x2")
```

Plot the estimated effect of variable x2.

```
R> plot2d(p ~ x2, data = nd)
```

The next step is to setup a full Bayesian MCMC sampling function. Fortunately, if we assume multivariate normal priors for the regression coefficients and an inverse Gamma prior for the variance, a Gibbs sampler with multivariate normal and inverse Gamma full conditionals can be created. The MCMC algorithm consecutively samples for $t = 1, \ldots, T$ from the full conditionals

$$\boldsymbol{\beta}^{(t)}| \cdot \sim N\left(\boldsymbol{\mu}_{\beta}^{(t-1)}, \boldsymbol{\Sigma}_{\beta}^{(t-1)}\right)$$

and

$$\sigma^{2^{(t)}}| \cdot \sim IG\left(a'^{(t-1)}, b'^{(t-1)}\right),$$

where $IG(\cdot)$ is the inverse Gamma distribution for sampling the variance parameter. The covariance matrix for $\boldsymbol{\beta}$ is given by

$$\boldsymbol{\Sigma}_\beta = \left(\frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{X} + \frac{1}{\sigma^2}\mathbf{M}^{-1}\right)^{-1}$$

and the mean

$$\boldsymbol{\mu}_\beta = \boldsymbol{\Sigma}_\beta\left(\frac{1}{\sigma^2}\mathbf{X}^\top\mathbf{y} + \frac{1}{\sigma^2}\mathbf{M}^{-1}\mathbf{m}\right),$$

where $\mathbf{m}$ is the prior mean and $\mathbf{M}$ the prior covariance matrix. Similarly, for σ^2 parameters a' and b' are computed by

$$a' = a + \frac{n}{2} + \frac{p}{2}$$

and

$$b' = b + \frac{1}{2}(\mathbf{y} - \mathbf{X}\boldsymbol{\beta})^\top(\mathbf{y} - \mathbf{X}\boldsymbol{\beta}) + \frac{1}{2}(\boldsymbol{\beta} - \mathbf{m})^\top\mathbf{M}^{-1}(\boldsymbol{\beta} - \mathbf{m}),$$

where a and b are usually set small, e.g., with $a = 1$ and $b = 0.0001$, such that the prior is flat and uninformative.

We can implement the MCMC algorithm in the following sampling function

```
R> sam_LM <- function(x, y, start = NULL,
+    n.iter = 12000, burnin = 2000, thin = 10,
+    m = 0, M = 1e+05,
+    a = 1, b = 1e-05,
+    verbose = TRUE, ...)
+  {
+    ## How many samples are saved?
+    itrthin <- seq.int(burnin, n.iter, by = thin)
+    nsaves <- length(itrthin)
+
+    ## Only univariate response.
+    y <- y[[1L]]
+
+    ## For illustration this is easier to read.
+    X <- x$mu$model.matrix
+
+    ## Again, set edf within .GlobalEnv for the
+    ## loglik() function in the lm_bamlss() family.
+    .lm_bamlss.p <<- ncol(X)
+
+    ## Number of observations and parameters.
+    n <- length(y)
+    p <- ncol(X)
+
+    ## Matrix saving the samples.
+    samples <- matrix(0, nsaves, p + 1L)
+
+    ## Stick to the naming convention.
+    pn <- paste0("mu.p.", colnames(X))
+    colnames(samples) <- c(
+      pn,      ## Regression coefficients and
+      "sigma"  ## variance samples.
+    )
+
+    ## Setup coefficient vector,
+    ## again, use correct names.
+    beta <- rep(0, p)
+    names(beta) <- pn
+    sigma <- sd(y)
+
+    ## Check for starting values obtained,
+    ## e.g., from lm.opt() from above.
+    if(!is.null(start)) {
+      sn <- names(start)
+      for(j in names(beta)) {
+        if(j %in% sn)
+          beta[j] <- start[j]
+      }
+    }
+
+    ## Process prior information.
+    m <- rep(m, length.out = p)
+    if(length(M) < 2)
+      M <- rep(M, length.out = p)
+    if(!is.matrix(M))
+      M <- diag(M)
+    Mi <- solve(M)
+
+    ## Precompute cross products.
+    XX <- crossprod(X)
+    Xy <- crossprod(X, y)
+
+    ## Inverse gamma parameter.
+    a <- a + n / 2 + p / 2
+
+    ## Start sampling.
+    ii <- 1
+    for(i in 1:n.iter) {
+      ## Sampling sigma
+      b2 <- b + 1 / 2 * t(y - X %*% beta) %*% (y - X %*% beta) +
+        1 / 2 * t(beta - m) %*% Mi %*% (beta - m)
+      sigma2 <- sqrt(1 / rgamma(1, a, b2))
+
+      ## Sampling beta.
+      sigma2i <- 1 / sigma2
+       Sigma <- chol2inv(chol(sigma2i * XX + sigma2i * Mi))
+       mu <- Sigma %*% (sigma2i * Xy + sigma2i * Mi %*% m)
+       beta <- MASS::mvrnorm(1, mu, Sigma)
+
+       if(i %in% itrthin) {
+         samples[ii, pn] <- beta
+         samples[ii, "sigma"] <- sqrt(sigma2)
+         ii <- ii + 1
+       }
+       if(verbose) {
+         if(i %% 1000 == 0)
+           cat("iteration:", i, "\n")
+       }
+     }
+
+     ## Convert to "mcmc" object.
+     samples <- as.mcmc(samples)
+
+     return(samples)
+   }
```

The new sampling function can be directly used with the `bamlss()` wrapper

```
R> b <- bamlss(f, data = d, family = "lm", optimizer = opt_LM,
+    sampler = sam_LM)

iteration: 1000
iteration: 2000
iteration: 3000
iteration: 4000
iteration: 5000
iteration: 6000
iteration: 7000
iteration: 8000
iteration: 9000
iteration: 10000
iteration: 11000
iteration: 12000

R> summary(b)

Call:
bamlss(formula = f, family = "lm", data = d, optimizer = lm.opt,
    sampler = lm.mcmc)
---
Family: LM
Link function: mu = identity
```

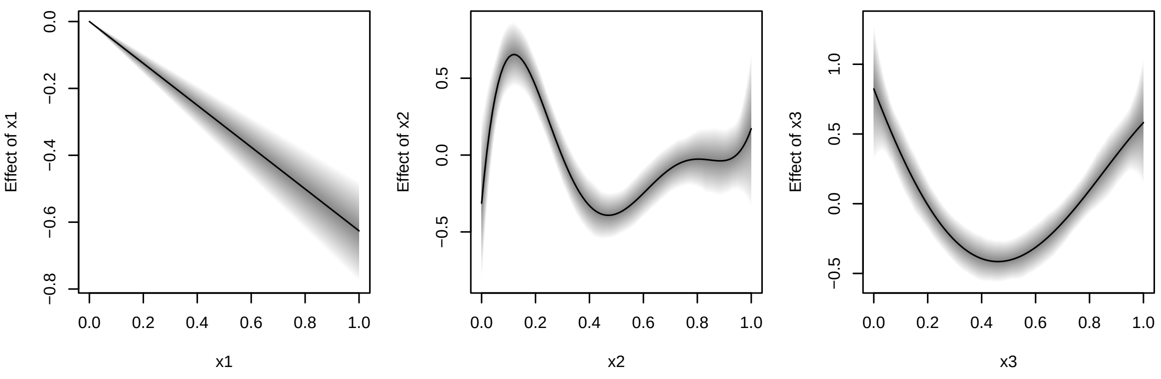

Figure 14: Estimated effects for covariates `x1`, `x2` and `x3` in the simulated data example.

```
*---
Formula mu:
---
num ~ x1 + poly(x2, 5) + poly(x3, 5)
-
Parametric coefficients:
                    Mean      2.5%       50%     97.5% parameters
(Intercept)     0.193433  0.104863  0.192858  0.281712      0.193
x1             -0.599433 -0.747700 -0.600058 -0.451035     -0.599
poly(x2, 5)1 -1.387117 -2.351893 -1.390560 -0.429834     -1.402
poly(x2, 5)2  2.333106  1.418701  2.334373  3.264135      2.300
poly(x2, 5)3  0.622568 -0.341720  0.614777  1.577756      0.612
poly(x2, 5)4 -1.401156 -2.319579 -1.412909 -0.510414     -1.388
poly(x2, 5)5  0.880996 -0.002737  0.899331  1.742269      0.861
poly(x3, 5)1 -0.115377 -1.107490 -0.108267  0.802904     -0.107
poly(x3, 5)2  3.578458  2.634068  3.594275  4.531447      3.581
poly(x3, 5)3 -0.215304 -1.158019 -0.216776  0.763266     -0.203
poly(x3, 5)4 -0.293656 -1.166910 -0.318136  0.673257     -0.276
poly(x3, 5)5  0.087389 -0.817009  0.093475  1.120255      0.087
---
Sampler summary:
-
DIC = 7.0055 pd = 53.6796 runtime = 2.513
---
Optimizer summary:
-
edf = 12 sigma = 0.2215
---
```

Predict for all terms including 95% credible intervals:

```
R> nd$x1 <- nd$x3 <- seq(0, 1, length = 100)
```

```
R> for(j in c("x1", "x2", "x3")) {
+    nd[[paste0("p.", j)]] <- predict(b, newdata = nd, term = j,
+      FUN = c95, intercept = FALSE)
+  }
```

The estimated effects are shown in Figure 14 and can be plotted with:

```
R> par(mfrow = c(1, 3))
R> plot2d(p.x1 ~ x1, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))
R> plot2d(p.x2 ~ x2, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))
R> plot2d(p.x3 ~ x3, data = nd, fill.select = c(0, 1, 0, 1), lty = c(2, 1, 2))
```

Affiliation:

Nikolaus Umlauf, Achim Zeileis, Thorsten Simon
Department of Statistics
Faculty of Economics and Statistics
Universität Innsbruck
Universitätsstr. 15
6020 Innsbruck, Austria
E-mail: Nikolaus.Umlauf@uibk.ac.at,
Achim.Zeileis@R-project.org,
Thorsten.Simon@uibk.ac.at
URL: https://eeecon.uibk.ac.at/~umlauf/,
https://www.zeileis.org/

Nadja Klein
Humboldt Universität zu Berlin
School of Business and Economics
Applied Statistics
Unter den Linden 6
10099 Berlin, Germany
E-mail: nadja.klein@hu-berlin.de
URL: https://hu.berlin/NK

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
November 2021, Volume 100, Issue 4 *Submitted:* 2019-09-27
doi:10.18637/jss.v100.i04 *Accepted:* 2021-06-10