Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *doi: 10.18637/jss.v000.i00*

RprobitB: Bayes Estimation of Discrete Choice Behavior Heterogeneity via Probit Models in R

Lennart Oelschläger
Bielefeld University

Dietmar Bauer
Bielefeld University

Abstract

RprobitB is an R package for Bayes estimation of probit models with a special focus on modeling choice behavior heterogeneity. In comparison to competing packages it places a focus on approximating the mixing distribution via a latent mixture of Gaussian distributions and thereby providing a classification of deciders. It provides tools for data management, model estimation via Markov Chain Monte Carlo Simulation, diagnostics tools for the Gibbs sampling and a prediction function. This paper demonstrates the functionalities of **RprobitB** on known choice datasets and compares estimation results across packages.

Keywords: discrete choice, probit models, heterogeneity, Bayes estimation, R.

1. Introduction

The multinomial probit model is one of the most widely-used statistical models to explain the choices that individuals make among a discrete set of alternatives, which is of central interest in many scientific areas, for example in transportation and marketing. In many such choice scenarios it is reasonable to assume, that the preferences of the decision makers are non-homogeneous. Based on personal characteristics, deciders generally weight attributes like time and cost differently. Heterogeneity in choice behavior can be modeled using mixing distributions for the coefficients. Recently, Oelschlaeger and Bauer proposed a new instrument for approximating the underlying mixing distribution that combines Bayes estimation and semi-parametric methods. This paper presents the implementation of the methodology in the R package **RprobitB**.

Traditionally, discrete choice models are interpreted as random utility models, including the multinomial logit (MNL) and the multinomial probit (MNP) model as the most prominent members. The MNL model affords straightforward analysis but suffers from the well-known

independence of irrelevant alternatives assumption. In contrast, the MNP model avoids this assumption, which however comes at the price of more complex parameter estimation, cf. Train (2009). In their basic form, these models often fail to take into account heterogeneity of individual deciders, cf. Train (2009), Chapter 6, or Train (2016). A concrete example of heterogeneous preferences is constituted by the value of travel time, cf. Cirillo and Axhausen (2006). Modeling heterogeneity in preferences is indispensable in such cases and has been elaborated in both the MNL and the MNP model by imposing mixing distributions on the coefficients, cf. Train (2009) and Bhat (2011).

Specifying these mixing distributions is an important part of the model selection. In absence of alternatives, it has been common practice so far to try different types of standard parametric distributions (including the normal, log-normal, uniform and tent distribution) and to perform a likelihood value-based model selection, cf. Train (2009), Chapter 6. Aiming to capture correlation patterns across parameters, Fountas, Anastasopoulos, and Abdel-Aty (2018) and Fountas, Pantangi, Hulme, and Anastasopoulos (2019) apply multivariate normal mixing distributions in their probit models, which however comes at the price of imposing the rather strong normality assumption on their parameters.

In order to alleviate these restrictions Train (2016) proposes a non-parametric approach based on grid methods. Building on the ideas of Train (2016) and Bhat and Lavieri (2018) recently Bauer, Büscher, and Batram (2019) introduced procedures for non-parametrically estimating latent class mixed multinomial probit models where the number of classes is chosen iteratively in the algorithm. These procedures have been demonstrated to be useful in reasonable sized cross-sectional data sets. However, for large panel data sets with a significant number of choice occasions per person, the approach is numerically extremely demanding in particular due to its non-parametric nature and has to deal with the curse of dimensionality.

In the Bayesian framework Scaccia and Marcucci (2010) presents the idea to estimate latent class logit models with a fixed prespecified number of Gaussian components. This approach does not require the maximization of the likelihood while at the same time it allows for approximation of the underlying mixing distribution. The same idea has also been applied to probit models, cf. Xiong and Mannering (2013) for an analysis of adolescent driver-injury data. In both cases however, the specification of the number of latent classes is based only on a trial-and-error strategy.

Oelschlaeger and Bauer presents a more flexible approach that combines the ideas of a Bayesian framework, approximating the mixing distribution through a mixture of normal distributions and updates on the number of latent classes within the algorithm analogously to Bauer *et al.* (2019). As a consequence, the procedure unites the benefits of a reduced numerical complexity for the estimation compared to the non-parametric likelihood maximization approach and the ability to approximate any mixing distribution. Presenting simulation results on artificial test cases, it is shown that the approach is capable of approximating the underlying mixing distributions and thereby guiding the specification of mixing distributions for real-world applications.

This packages adds to the line of discrete choice software packages in R in the following way: Its focus is entirely on Bayesian estimation, thereby it differs from the packages Rchoice. Furthermore, it places a focus on modeling choice behaviour heterogeneity by approximating the underlying mixing distribution through a latent mixture of normal distributions. The method is explained in detail in Oelschlaeger and Bauer.

In this article we present the methodology, give an overview over the functionality of the package and apply the package to data sets. Some of them were already analysed and we aim to reconstruct their findings. In addition, we added two datasets that are especially appropriate for **RprobitB** in modeling choice behaviour heterogeneits. The first one is a dataset of contraception choice from the German family panel pairfam. It contains repeated observations of males and femals over several years having different social demographics and relationship status choosing different means of contraception. This choice a priori can be considered to be very hetereogenous and dependent on factors not directly observable by the researcher. The second application deals with the opening choice of chess players depending on their and their openents playing strenght measured in the popular measure system Elo, their gender and nationality. Like the contraception example, this choice a priori can be considers to depend on psychological factors that are not directly observable by the researcher. By applying the functionality of this package we demonstrate how we are able to classify the players into different categories of playing style.

1. With **RprobitB**, you can model the choices made by deciders among a discrete set of alternatives. For example, think of tourists that want to book a flight to their holiday destination. The knowledge why they prefer a certain route over another is of great value for airlines, especially the customer's willingness to pay for say a faster or more comfortable flight alternative.

2. Different deciders value different choice attributes differently. For example, it is imaginable that business people place a higher value on flight time and are willing to pay more for a faster route alternative than vacationers. Such choice behavior heterogeneity can be addressed by **RprobitB**. Furthermore, the package enables to identify groups of deciders that share similar preferences.

3. Finally, the package enables prediction of choice behavior when certain choice attributes change, for example the proportion of customers who will choose the competitor's product in the event of a price increase.

The functions of **RprobitB** can be grouped into ones for data management, model fitting, and model evaluation, see the flowchart below. The package can be used for two different purposes: (a) estimation of a model for given data and (b) estimation of a model for simulated data. Simulation typically serves to assess the properties of estimation algorithms either for research or in a bootstrap like fashion. **RprobitB** supports these functions.

2. The probit model

The probit model is a regression-type model where the dependent variable takes a finite number of values and the error term is normally distributed (Agresti 2015). Its most popular application are discrete choice scenarios, in which the dependent variable is one of finitely many and mutually exclusive alternatives and explanatory variables typically are characteristics of the deciders or the alternatives. This section defines the model via latent utilities, outlines an extension for modeling heterogeneity, and discusses necessary normalization for parameter identification.

2.1. Latent utilities

Assume that we know the choices of N deciders choosing between $J \geq 2$ alternatives at each

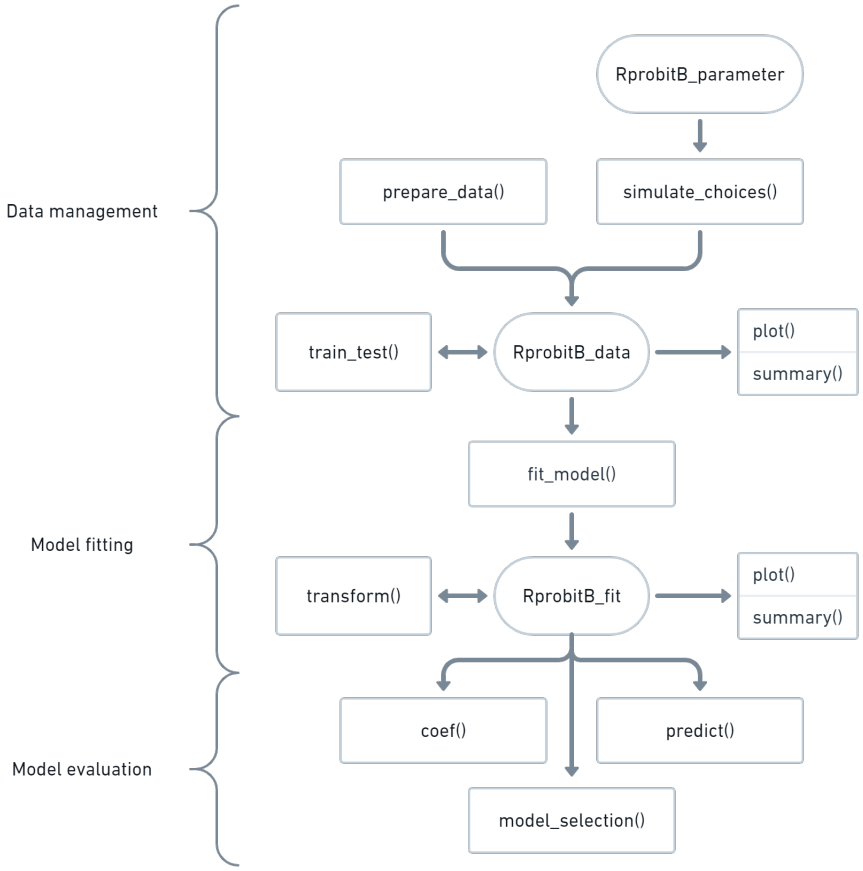


Figure 1: Package flowchart with main functions as rectangles and objects as ovals.

of T choice occasions.[1] Specific to each decision maker, alternative and choice occasion, we furthermore observe P covariates. The choices are explained by a linear combination of those. The linear combination is continuous and cannot be linked directly to the discrete choices but must take a detour over a latent variable. In the discrete choice setting, this variable can be interpreted as the decider's utility of a certain alternative. Formally, decider n's utility U_{ntj} for alternative j at choice occasion t is modeled as

$$U_{ntj} = X'_{ntj}\beta + \epsilon_{ntj} \tag{1}$$

for $n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J$. Here, X_{ntj} is a (column) vector of P characteristics of j as faced by n at t, $\beta \in \mathbb{R}^P$ is a vector of coefficients, and $(\epsilon_{nt:}) = (\epsilon_{nt1}, \dots, \epsilon_{ntJ})' \sim \text{MVN}_J(0, \Sigma)$ is the model's error term vector for n at t, which in the probit model is assumed to be multivariate normally distributed with zero mean and covariance matrix Σ.[2]

[1] For notational simplicity, the number of choice occasions T is assumed to be the same for each decision maker here. However, **RprobitB** allows for unbalanced panels, i.e. varying T. Of course, the cross-sectional case $T = 1$ is possible.

[2] The assumption about the error term distribution distinguishes the probit from the logit model. In the latter, each ϵ_{ntj} is assumed to be independently extreme value distributed.

Assuming utility maximizing behavior of the decision makers[3], we link the utilities to the choices via

$$y_{nt} = \underset{j=1,\ldots,J}{\operatorname{argmax}} U_{ntj}.$$

Here, $y_{nt} = j$ denotes the event that decider n chooses j at occasion t.

2.2. Choice behavior heterogeneity

Note that the coefficient vector β in equation (1) is constant across decision makers. This assumption is too restrictive for many applications[4] and can be relaxed by imposing a distribution on β. In this case, the model allows for choice behavior heterogeneity among deciders. Instead of estimating β directly, the modeler in this case aims to characterize its distribution. We formally define $\beta = (\alpha, \beta_n)$, where α are P_f coefficients that are constant across deciders and β_n are P_r decider-specific coefficients. Now if $P_r > 0$, β_n is distributed according to some P_r-variate distribution, the so-called mixing distribution.

Choosing an appropriate mixing distribution is a notoriously difficult task of the model specification. In many applications, different types of standard parametric distributions (including the normal, log-normal, uniform and tent distribution) are tried in conjunction with a likelihood value-based model selection (Train 2009). Instead, **RprobitB** implements the approach of Oelschläger and Bauer (2020) to approximate any underlying mixing distribution by a mixture of (multivariate) Gaussian densities. More precisely, the underlying mixing distribution for the random coefficients $(\beta_n)_n$ is approximated by a mixture of P_r-variate normal densities ϕ_{P_r} with mean vectors $b = (b_c)_c$ and covariance matrices $\Omega = (\Omega_c)_c$ using C components:

$$\beta_n \mid b, \Omega \sim \sum_{c=1}^{C} s_c \phi_{P_r}(\cdot \mid b_c, \Omega_c).$$

Here, $(s_c)_c$ are weights satisfying $0 < s_c \leq 1$ for $c = 1, \ldots, C$ and $\sum_c s_c = 1$. One interpretation of the latent class model is obtained by introducing variables $z = (z_n)_n$, allocating each decision maker n to class c with probability s_c, i.e.

$$\text{Prob}(z_n = c) = s_c \wedge \beta_n \mid z, b, \Omega \sim \phi_{P_r}(\cdot \mid b_{z_n}, \Omega_{z_n}).$$

This interpretation enables a classification of the deciders in terms of their preferences, see Section ... for an example.

2.3. Model normalization

Any utility model is invariant towards the level and the scale of utility, as Train (2009) points out. For identification of the model parameters, we therefore normalize the model by

[3]We note that utility maximizing behavior is a common assumption in econometric models. However, many studies have shown that humans do not decide in this rational sense in general, see for example Hewig, Kretschmer, Trippe, Hecht, Coles, Holroyd, and Miltner (2011).

[4]As an example, consider the case of modeling the choice of a means of transportation to work: It is easily imaginable that business people and pensioners do not share the same sensitivities towards cost and time.

considering only utility differences and fixing one error term variance. Formally, equation (1) is transformed to

$$\tilde{U}_{ntj} = \tilde{X}'_{ntj}\beta + \tilde{\epsilon}_{ntj},$$

$n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J-1$. Here and choosing J as the reference alternative, $\tilde{U}_{ntj} = U_{ntj} - U_{ntJ}$, $\tilde{X}_{ntj} = X_{ntj} - X_{ntJ}$, and $\tilde{\epsilon}_{ntj} = \epsilon_{ntj} - \epsilon_{ntJ}$, where $(\tilde{\epsilon}_{nt:}) = (\tilde{\epsilon}_{nt1}, ..., \tilde{\epsilon}_{nt(J-1)})' \sim \text{MVN}_{J-1}(0, \tilde{\Sigma})$ and $\tilde{\Sigma}$ denotes a covariance matrix with one diagonal element fixed to a positive number.[5]

3. Choice data

Choice data typically consist of the observed choices in conjunction with choice characteristics.[6] **RprobitB** requests that data sets are (a) of class '`data.frame`' and (b) in wide format (that means each row provides the full information for one choice occasion), (c) contain a column with unique identifiers for each decision maker[7], (d) contain a column with the observed choices (required for model fitting but not for prediction), and (e) contain columns for the values of alternative and/or decider specific covariates. The underlying set of choice alternatives is assumed to be mutually exclusive (one can choose one and only one alternative that are all different), exhaustive (the alternatives do not leave other options open), and finite (Train 2009).

This section introduces the formula framework of **RprobitB** for specifying the set of covariates entering a model. The framework is adapted from **mlogit**, which is flexible enough to allow for different types of covariates: covariates that are constant across alternatives (e.g. the deciders age), covariates that are alternative specific (e.g. the alternative's price), covariates with a generic coefficient (e.g. paying the same amount of money for train company A versus B should make no difference), and covariates that have alternative specific coefficients (e.g. spending time in a crowded train versus a private jet certainly makes a difference). Subsequently, we demonstrate how to pass such a formula to either `prepare_data()` for preparing empirical data for estimation or to `simulate_choices()` for simulating choice data.

3.1. Formula framework

We generalize equation (1) to allow for different types of covariates: Say we want to model the utility U_{ntj} of decider n at occasion t for alternative j via

$$U_{ntj} = A_{ntj}\beta_1 + B_{nt}\beta_{2j} + C_{ntj}\beta_{3j} + \epsilon_{ntj}. \tag{2}$$

Here, the covariates A and C depend on the alternative, whereas B is only choice occasion specific. The coefficient β_1 is generic (i.e. the same for each alternative), whereas β_{2j} and β_{3j}

[5] Restricting an element of $\tilde{\Sigma}$ determines the utility scale. **RprobitB** provides the alternative to fix an element of α which serves the same purpose, see see Section ...

[6] We only consider unordered alternatives (that is, alternatives cannot be ranked). Every decider may take one or repeated choices (called choice occasions).

[7] Additionally, it can contain a column with identifier for each choice occasion of each decider.

are alternative specific. Mind that not all $(\beta_{2j})_{j=1,\ldots,J}$ are identified. This is because we took utility differences for model normalization, see Section 2.3. We therefore fix β_{2k} to 0, where k is the base alternative that can be specified. The coefficients $(\beta_{2j})_{j\neq k}$ have to be interpreted with respect to alternative k.

Equation (2) can be entered into R via specifying the '`formula`' object `choice ~ A | B | C`, where `choice` is the dependent variable (the discrete choice we aim to explain). By default, alternative specific constants (ASCs)[8] are added to the model. They can be removed by adding `+ 0` in the second spot, e.g. `choice ~ A | B + 0 | C`. To exclude covariates of the backmost categories, use either `0`, e.g. `choice ~ A | B | 0` or just leave this part out and write `choice ~ A | B`. However, to exclude covariates of front categories, we have to use `0`, e.g. `choice ~ 0 | B`. To include more than one covariate of the same category, use `+`, e.g. `choice ~ A1 + A2 | B`. If we don't want to include any covariates of the second category but want to estimate ASCs, add `1` in the second spot, e.g. `choice ~ A | 1 | C`. The expression `choice ~ A | 0 | C` is interpreted as no covariates of the second category and no alternative specific constants.

3.2. Preparing data for estimation

Before model estimation, a choice data set `choice_data` must pass the `prepare_data()` function (cf. Figure 1) together with a formula object `form` introduced above:

```
> data <- prepare_data(form = form, choice_data = choice_data)
```

The function performs compatibility checks and data transformations and returns an object of class '`RprobitB_data`' that can be fed into the estimation routine `fit_model()`, see Section 4. The following arguments of `prepare_data()` are optional:

- `re`: A character vector of covariate names in `form`, whose coefficients we want to model using a mixing distribution, see Section 2.2. Per default `re = NULL`, i.e. no random effects. To have random effects for the ASCs, include `"ASC"` in `re`.
- `alternatives`: A character vector of alternative names, defining the choice set. If not specified, all alternatives chosen in the data set are considered.
- `base_alternative`: One element of `alternatives` specifying the base alternative for covariates that are not alternative specific. Per default, `base_alternative` is the last element of `alternatives`.
- `id` and `idc`: The names of the columns in `choice_data` that contain unique identifier for each decision maker and for each choice occasion, respectively. Per default, `id = "id"` and `idc = NULL`, in which case the choice occasion identifier are generated by the appearance of the choices in the data set.
- `standardize`: A character vector of variable names of `form` that get standardized. Per default, no covariate is standardized.

[8] ASCs capture the average effect on utility of all factors that are not included in the model. Again, we cannot estimate ASCs for all the alternatives due to identifiability. Therefore, they are added for all except for the base alternative.

- `impute`: A character, specifying how to handle missing data entries. Options are `"complete_cases"` (removing rows that contain missing entries, which is the default behavior), `"zero_out"` (replacing missing entries by 0), and `"mean"` (imputing missing entries by the covariate mean).

Example 1: Train We demonstrate the `prepare_data()` function using the choice data set `Train` from the **mlogit** package. The data set contains 2929 stated choices of 235 deciders between two fictional train trip alternatives `A` and `B`. The trip alternatives are characterized by their `price`, the travel `time`, the level of `comfort` (the lower the value the higher the comfort), and the number of `change`s. The data is in wide format, the columns `id` and `choiceid` identify the deciders and the choice occasions, respectively. The column `choice` provides the choices. For convenience, we transform `time` from minutes to hours and `price` from guilders to euros:

```
> data("Train", package = "mlogit")
> Train$price_A <- Train$price_A / 100 * 2.20371
> Train$price_B <- Train$price_B / 100 * 2.20371
> Train$time_A <- Train$time_A / 60
> Train$time_B <- Train$time_B / 60
> str(Train)

'data.frame':        2929 obs. of  11 variables:
 $ id       : int  1 1 1 1 1 1 1 1 1 1 ...
 $ choiceid : int  1 2 3 4 5 6 7 8 9 10 ...
 $ choice   : Factor w/ 2 levels "A","B": 1 1 1 2 2 2 2 2 1 1 ...
 $ price_A  : num  52.9 52.9 52.9 88.1 52.9 ...
 $ time_A   : num  2.5 2.5 1.92 2.17 2.5 ...
 $ change_A : num  0 0 0 0 0 0 0 0 0 0 ...
 $ comfort_A: num  1 1 1 1 1 0 1 1 0 1 ...
 $ price_B  : num  88.1 70.5 88.1 70.5 70.5 ...
 $ time_B   : num  2.5 2.17 1.92 2.5 2.5 ...
 $ change_B : num  0 0 0 0 0 0 0 0 0 0 ...
 $ comfort_B: num  1 1 0 0 0 0 1 0 1 0 ...
```

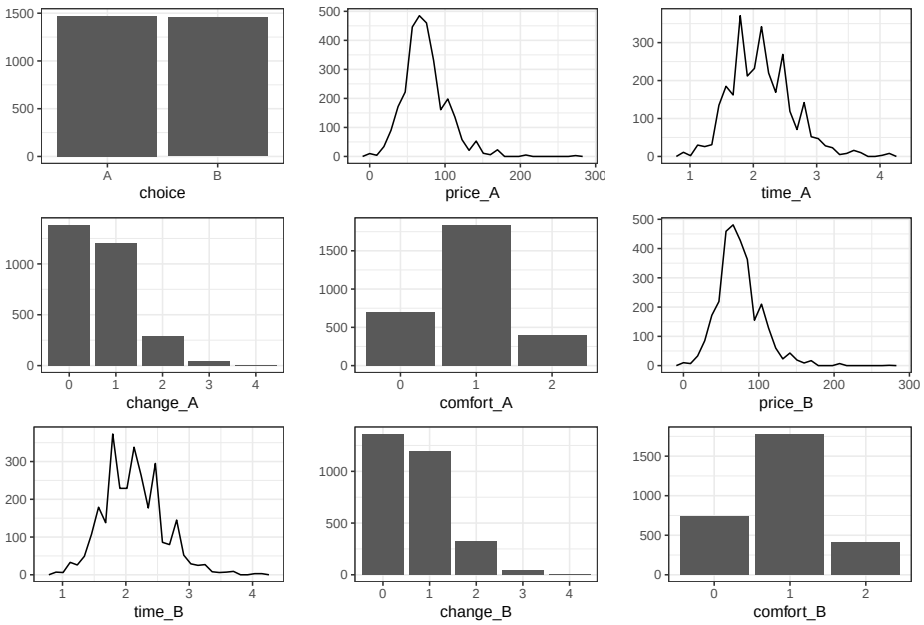
The naming convention `<covariate>_<alternative>` in the `Train` data set for alternative specific covariates is the required format for **RprobitB**. Say we want to include all of them in our probit model, connect them to generic coefficients, and exclude ASCs. Then we would specify the formula

```
> form <- choice ~ price + time + comfort + change | 0
```

Passing `form` to `prepare_data()` returns an '`RprobitB_data`' object. The object can be inspected via its `plot()` and `summary()` method:

```
> data_train <- prepare_data(
+     form = form,
```

```
+    choice_data = Train,
+    id = "id",
+    idc = "choiceid"
+    )
> summary(data_train)

  number deciders choice occasions choices total
1             235     5 to 19 each          2929

  alternative frequency
1           A      1474
2           B      1455

   effect as_value as_coef random
1   price     TRUE   FALSE  FALSE
2    time     TRUE   FALSE  FALSE
3 comfort     TRUE   FALSE  FALSE
4  change     TRUE   FALSE  FALSE

> plot(data_train)
```



This example will serve as an illustration of the estimation routine later.

3.3. Simulating choice data

The `simulate_choices()` function simulates discrete choice data from a probit model. Say we want to simulate the choices of `N` deciders in `T` choice occasions[9] among `J` alternatives. Together with a model formula `form`, we have to call

[9] `T` can be either a positive number, representing a fixed number of choice occasions for each decision maker, or a vector of length `N` with decision maker specific numbers of choice occasions

```
> data <- simulate_choices(form = form, N = N, T = T, J = J)
```

The function `simulate_choices()` has the following optional arguments:

- `re`: Analogue to `prepare_data()`.
- `alternatives`: A character vector of length `J` with the names of the choice alternatives. If not specified, the alternatives are labeled by the first `J` upper-case letters of the Roman alphabet.
- `base_alternative`: Analogue to `prepare_data()`.
- `covariates`: A named list of covariate values. Each element must be a vector of length equal to the number of choice occasions and named according to a covariate, or follow the naming convention for alternative specific covariates, i.e. `<covariate>_<alternative>`. Covariates for which no values are specified are drawn from a standard normal distribution.
- `standardize`: Analogue to `prepare_data()`.
- `seed`: Optionally set a seed for the simulation.

True model parameters are set at random per default or can be specified via the function's ellipsis argument:

- a numeric vector `alpha` with the fixed effects,
- an integer `C`, the number (greater or equal 1) of latent classes of decision makers (`C = 1` per default),
- a numeric vector `s` of length `C` with the class weights,
- a matrix `b` with the class means as columns,
- a matrix `Omega` with the class covariance matrices as columns,
- a matrix `Sigma`, the differenced error term covariance matrix, or `Sigma_full`, the full error term covariance matrix,
- a matrix `beta` with the decision-maker specific coefficient vectors as columns,
- a numeric vector of length `N`, the class allocation vector.

Example 2: Simulated choices For illustration, we simulate the choices of `N = 100` deciders at `T = 10` choice occasions between the fictitious alternatives `alt1` and `alt2`. The choices are explained by the alternative specific covariates `var1` and `var3` and the covariate `var2` which is only choice occasion specific (base alternative is `var2`) but connected to a random effect:

```
> N <- 100
> T <- 10
> alternatives <- c("alt1", "alt2")
> base_alternative <- "alt2"
> form <- choice ~ var1 | var2 | var3
> re <- c("ASC","var2")
```

RprobitB provides the function `overview_effects()` which can be used to get an overview of the effects for which parameters can be specified:

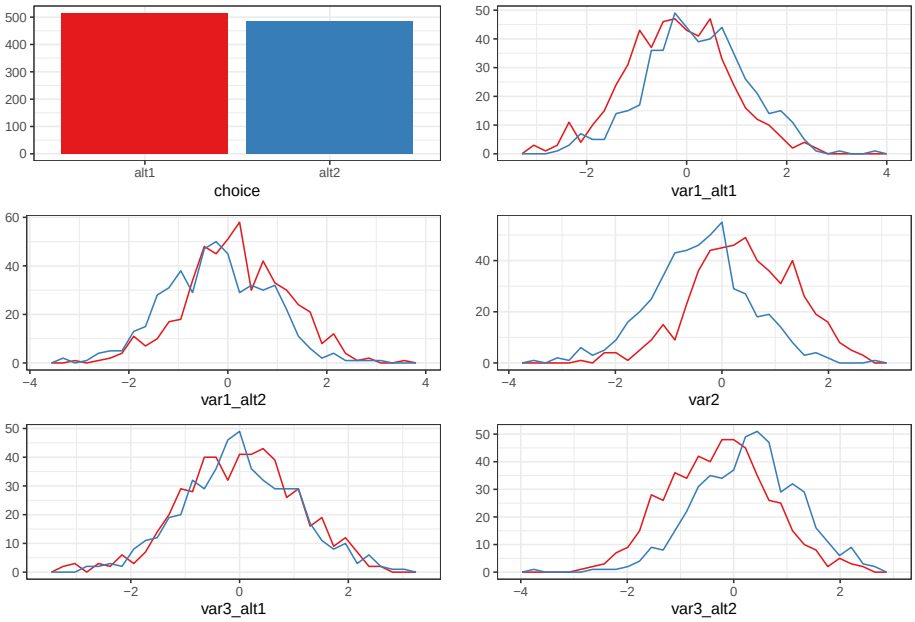
```
> overview_effects(
+    form = form,
+    re = re,
+    alternatives = alternatives,
+    base_alternative = base_alternative
+  )

     effect as_value as_coef random
1      var1     TRUE   FALSE  FALSE
2 var3_alt1     TRUE    TRUE  FALSE
3 var3_alt2     TRUE    TRUE  FALSE
4 var2_alt1    FALSE    TRUE   TRUE
5  ASC_alt1    FALSE    TRUE   TRUE
```

Two effects are fixed (`random = FALSE`), hence the vector `alpha` must be of length 3, where the elements 1 to 3 correspond to `var1`, `var3_alt1`, and `var3_alt2`, respectively. The matrix `b` must be of dimension `2 x C`, where row 1 and 2 correspond to `var2_alt1` and `ASC_alt1`, respectively.

```
> data_sim <- simulate_choices(
+    form = form,
+    N = N,
+    T = T,
+    J = 2,
+    re = re,
+    alternatives = alternatives,
+    base_alternative = base_alternative,
+    seed = 1,
+    alpha = c(-1,0,1),
+    C = 2,
+    s = c(0.7,0.3),
+    b = matrix(c(2,-0.5,1,1), ncol = 2)
+  )
```

The `plot()` method of '`RprobitB_data`' object has the argument `by_choice`, which allows for the visualization of covariates grouped by the chosen alternatives:

```
> plot(data_sim, by_choice = TRUE)
```



The graphic is consistent with our specification: for example, covariate `var1` has a negative effect on `alt1`, because the coefficient of `var1` (the first value of `alpha`) is negative (-1). Hence higher values of `var1_alt1` correspond more frequently to choice `alt2` (upper-right panel).

This example will serve as an illustration of the weight-based updating scheme later.

3.4. Train and test data set

RprobitB provides the function `train_test()` that can split an '`RprobitB_data`' object (i.e. the output of `prepare_data()` or `simulate_choices()`) in two parts: a train subset and a test subset. The model then is typically estimated on the train subset and applied to explain the test subset. This procedure can be useful to evaluate how the model generalizes to unseen data. For example, the following call puts 70% of deciders from `data_sim` into the train subsample and 30% of deciders in the test subsample:

```
> train_test(data_sim, test_proportion = 0.3, by = "N")

$train
Simulated data of 700 choices.

$test
Simulated data of 300 choices.
```

Alternatively, the following line puts 2 randomly chosen choice occasions per decider from `data` into the test subsample, the rest goes into the train subsample:

```
> train_test(data_sim, test_number = 2, by = "T", random = TRUE, seed = 1)
```

```
$train
Simulated data of 800 choices.

$test
Simulated data of 200 choices.
```

4. Model fitting

RprobitB estimates the probit model in a Bayesian framework. The framework builds upon the work of McCulloch and Rossi (1994), Nobile (1998), Allenby and Rossi (1998), and Imai and van Dyk (2005). A key ingredient is the concept of data augmentation, see Albert and Chib (1993): The idea is to treat the latent utilities in the model equation (1) as additional parameters. Then, conditional on the utilities, the probit model constitutes a standard Bayesian linear regression set-up. Its posterior distribution can be approximated by iteratively drawing and updating each model parameter conditional on the other parameters (the so-called Gibbs sampling approach).

This section lists the prior distributions for the model parameters and formulates the conditional posterior distributions. We subsequently introduce our estimation routine `fit_model()` and apply it to the two examples from the previous section. The remainder of the section presents how **RprobitB** estimates latent classes and updates the number of those either weight-based or using a Dirichlet process.

4.1. Prior and posterior distributions

This section follows closely Oelschläger and Bauer (2020). We a priori assume the following (conjugate) parameter distributions:

- $(s_1, \dots, s_C) \sim D_C(\delta)$, where $D_C(\delta)$ denotes the C-dimensional Dirichlet distribution with concentration parameter vector $\delta = (\delta_1, \dots, \delta_C)$,
- $\alpha \sim \text{MVN}_{P_f}(\psi, \Psi)$, where MVN_{P_f} denotes the P_f-dimensional normal distribution with mean ψ and covariance Ψ,
- $b_c \sim \text{MVN}_{P_r}(\xi, \Xi)$, independent for all c,
- $\Omega_c \sim W_{P_r}^{-1}(\nu, \Theta)$, independent for all c, where $W_{P_r}^{-1}(\nu, \Theta)$ denotes the P_r-dimensional inverse Wishart distribution with ν degrees of freedom and scale matrix Θ,
- and $\Sigma \sim W_{J-1}^{-1}(\kappa, \Lambda)$.

These priors imply the following conditional posterior distributions:

- The class weights are drawn from the Dirichlet distribution

$$(s_1, \dots, s_C) \mid \delta, z \sim D_C(\delta_1 + m_1, \dots, \delta_C + m_C),$$

  where for $c = 1, \dots, C$, $m_c = \#\{n : z_n = c\}$ denotes the current absolute class size. Mind that the model is invariant to permutations of the class labels $1, \dots, C$. For that reason, we accept an update only if the ordering $s_1 > \dots > s_C$ holds, thereby ensuring a unique labeling of the classes.

- Independently for all n, we update the allocation variables $(z_n)_n$ from their conditional distribution

$$\text{Prob}(z_n = c \mid s, \beta, b, \Omega) = \frac{s_c \phi_{P_r}(\beta_n \mid b_c, \Omega_c)}{\sum_c s_c \phi_{P_r}(\beta_n \mid b_c, \Omega_c)}.$$

- The class means $(b_c)_c$ are updated independently for all c via

$$b_c \mid \Xi, \Omega, \xi, z, \beta \sim \text{MVN}_{P_r}(\mu_{b_c}, \Sigma_{b_c}),$$

where $\mu_{b_c} = (\Xi^{-1} + m_c\Omega_c^{-1})^{-1}(\Xi^{-1}\xi + m_c\Omega_c^{-1}\bar{b}_c)$, $\Sigma_{b_c} = (\Xi^{-1} + m_c\Omega_c^{-1})^{-1}$, and $\bar{b}_c = m_c^{-1}\sum_{n:z_n=c}\beta_n$.

- The class covariance matrices $(\Omega_c)_c$ are updated independently for all c via

$$\Omega_c \mid \nu, \Theta, z, \beta, b \sim W_{P_r}^{-1}(\mu_{\Omega_c}, \Sigma_{\Omega_c}),$$

where $\mu_{\Omega_c} = \nu + m_c$ and $\Sigma_{\Omega_c} = \Theta^{-1} + \sum_{n:z_n=c}(\beta_n - b_c)(\beta_n - b_c)'$.

- Independently for all n and t and conditionally on the other components, the utility vectors $(U_{nt:})$ follow a $J-1$-dimensional truncated multivariate normal distribution, where the truncation points are determined by the choices y_{nt}. To sample from a truncated multivariate normal distribution, we apply a sub-Gibbs sampler, following the approach of Geweke (1998):

$$U_{ntj} \mid U_{nt(-j)}, y_{nt}, \Sigma, W, \alpha, X, \beta \sim \mathcal{N}(\mu_{U_{ntj}}, \Sigma_{U_{ntj}}) \cdot \begin{cases} 1(U_{ntj} > \max(U_{nt(-j)}, 0)) & \text{if } y_{nt} = j \\ 1(U_{ntj} < \max(U_{nt(-j)}, 0)) & \text{if } y_{nt} \neq j \end{cases},$$

where $U_{nt(-j)}$ denotes the vector $(U_{nt:})$ without the element U_{ntj}, $\mathcal{N}$ denotes the univariate normal distribution, $\Sigma_{U_{ntj}} = 1/(\Sigma^{-1})_{jj}$ and

$$\mu_{U_{ntj}} = W_{ntj}'\alpha + X_{ntj}'\beta_n - \Sigma_{U_{ntj}}(\Sigma^{-1})_{j(-j)}(U_{nt(-j)} - W_{nt(-j)}'\alpha - X_{nt(-j)}'\beta_n),$$

where $(\Sigma^{-1})_{jj}$ denotes the (j,j)th element of Σ^{-1}, $(\Sigma^{-1})_{j(-j)}$ the jth row without the jth entry, $W_{nt(-j)}$ and $X_{nt(-j)}$ the coefficient matrices W_{nt} and X_{nt}, respectively, without the jth column.

- Updating the fixed coefficient vector α is achieved by applying the formula for Bayesian linear regression of the regressors W_{nt} on the regressands $(U_{nt:}) - X_{nt}'\beta_n$, i.e.

$$\alpha \mid \Psi, \psi, W, \Sigma, U, X, \beta \sim \text{MVN}_{P_f}(\mu_\alpha, \Sigma_\alpha),$$

with the definitions $\mu_\alpha = \Sigma_\alpha(\Psi^{-1}\psi + \sum_{n=1,t=1}^{N,T} W_{nt}\Sigma^{-1}((U_{nt:}) - X_{nt}'\beta_n))$ and $\Sigma_\alpha = (\Psi^{-1} + \sum_{n=1,t=1}^{N,T} W_{nt}\Sigma^{-1}W_{nt}')^{-1}$.

- Analogously to α, the random coefficients $(\beta_n)_n$ are updated independently via

$$\beta_n \mid \Omega, b, X, \Sigma, U, W, \alpha \sim \text{MVN}_{P_r}(\mu_{\beta_n}, \Sigma_{\beta_n}),$$

$\mu_{\beta_n} = \Sigma_{\beta_n}(\Omega_{z_n}^{-1}b_{z_n} + \sum_{t=1}^{T} X_{nt}\Sigma^{-1}(U_{nt} - W_{nt}'\alpha))$ and $\Sigma_{\beta_n} = (\Omega_{z_n}^{-1} + \sum_{t=1}^{T} X_{nt}\Sigma^{-1}X_{nt}')^{-1}$.

- The error term covariance matrix Σ is updated by means of

$$\Sigma \mid \kappa, \Lambda, U, W, \alpha, X, \beta \sim W_{J-1}^{-1}(\kappa + NT, \Lambda + S),$$

where $S = \sum_{n=1,t=1}^{N,T} \varepsilon_{nt}\varepsilon_{nt}'$ and $\varepsilon_{nt} = (U_{nt:}) - W_{nt}'\alpha - X_{nt}'\beta_n$.

Samples obtained from the updating scheme described above lack identification (except for s and z draws), compare Section 2.3. Therefore, subsequent to the sampling, we apply the following normalization for the ith update in each iteration i:

- $\alpha^{(i)} \cdot \omega^{(i)}$,
- $b_c^{(i)} \cdot \omega^{(i)}$, $c = 1, \ldots, C$,
- $U_{nt}^{(i)} \cdot \omega^{(i)}$, $n = 1, \ldots, N$, $t = 1, \ldots, T$,
- $\beta_n^{(i)} \cdot \omega^{(i)}$, $n = 1, \ldots, N$,
- $\Omega_c^{(i)} \cdot (\omega^{(i)})^2$, $c = 1, \ldots, C$, and
- $\Sigma^{(i)} \cdot (\omega^{(i)})^2$,

where either $\omega^{(i)} = \sqrt{\text{const}/(\Sigma^{(i)})_{jj}}$ with $(\Sigma^{(i)})_{jj}$ the jth diagonal element of $\Sigma^{(i)}$, $1 \leq j \leq J-1$, or alternatively $\omega^{(i)} = \text{const}/\alpha_p^{(i)}$ for some coordinate $1 \leq p \leq P_f$ of the ith draw for the coefficient vector α. Here, const is any positive constant (typically 1). The preferences will be flipped if $\omega^{(i)} < 0$, which only is the case if $\alpha_p^{(i)} < 0$.

4.2. The estimation routine

The Gibbs sampling scheme described above can be executed by applying the function

```
> fit_model(data = data)
```

where `data` is an '`RprobitB_data`' object, see Section 3. The function has the following optional arguments:

- `scale`: A formula object which determines the utility scale (cf. Section 2.3). It is of the form `<parameter> ~ <value>`, where `<parameter>` is either the name of a fixed effect or `Sigma_<j>` for the `<j>`th diagonal element of `Sigma`, and `<value>` is the value of the fixed parameter (i.e. const introduced above). Per default `scale = Sigma_1 ~ 1`, i.e. the first error-term variance is fixed to 1.
- `R`: The number of iterations of the Gibbs sampler. The default is `R = 10000`.
- `B`: The length of the burn-in period[10], i.e. a non-negative number of samples to be discarded. The default is `B = R/2`.

[10]The theory behind Gibbs sampling constitutes that the sequence of samples produced by the updating scheme is a Markov chain with stationary distribution equal to the desired joint posterior distribution. It takes a certain number of iterations for that stationary distribution to be approximated reasonably well. Therefore, it is common practice to discard the first `B` out of `R` samples (the so-called burn-in period).

- `Q`: The thinning factor for the Gibbs samples[11], i.e. only every `Q`th sample is kept. The default is `Q = 1`.
- `print_progress`: A boolean, determining whether to print the Gibbs sampler progress.
- `prior`: A named list of parameters for the prior distributions (their default values are documented in the `check_prior()` function, see `help(check_prior, package = "RprobitB")`):
  - `eta`: The mean vector of length `P_f` of the normal prior for `alpha`.
  - `Psi`: The covariance matrix of dimension `P_f x P_f` of the normal prior for `alpha`.
  - `delta`: The concentration parameter of length 1 of the Dirichlet prior for `s`.
  - `xi`: The mean vector of length `P_r` of the normal prior for each `b_c`.
  - `D`: The covariance matrix of dimension `P_r` x `P_r` of the normal prior for each `b_c`.
  - `nu`: The degrees of freedom (a natural number greater than `P_r`) of the Inverse Wishart prior for each `Omega_c`.
  - `Theta`: The scale matrix of dimension `P_r` x `P_r` of the Inverse Wishart prior for each `Omega_c`.
  - `kappa`: The degrees of freedom (a natural number greater than `J-1`) of the Inverse Wishart prior for `Sigma`.
  - `E`: The scale matrix of dimension `J-1` x `J-1` of the Inverse Wishart prior for `Sigma`.
- `latent_classes`: A list of parameters specifying the number and the updating scheme of latent classes, see Section ...

Example 1: Train (cont.) Recall the `Train` data set of stated train trip alternatives, characterized by their `price`, `time`, number of `change`s, and level of `comfort`. From this data, we previously build the '`RprobitB_data`' object `data_train`, which we now pass to the estimation routine `fit_model()`.

For normalization, the first linear coefficient, the `price`, was fixed to `-1`, which allows to interpret the other coefficients as monetary values:

```
> model_train <- fit_model(data = data_train, scale = price ~ -1)
```

The model is saved in **RprobitB** and can be accessed via

```
> data(model_train, package = "RprobitB")
```

The estimated coefficients (using the mean of the Gibbs samples as a point estimate) can be printed via

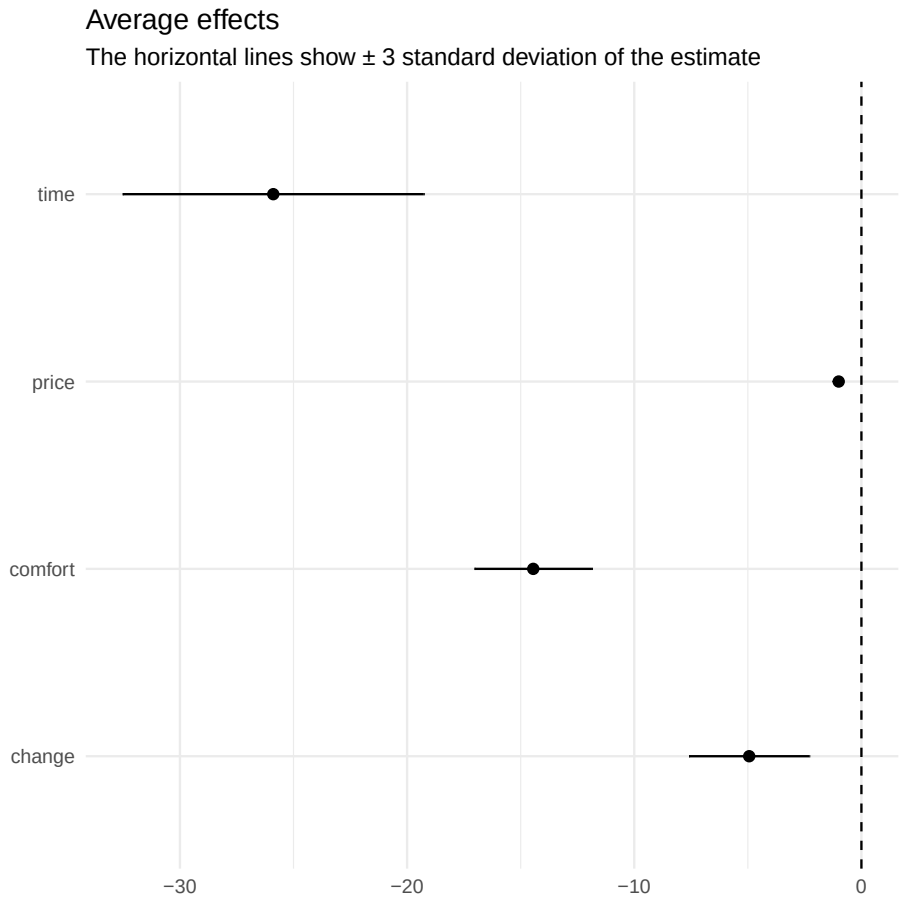
```
> coef(model_train)
```

[11] In order to obtain independent Gibbs samples, we consider only every `Q`th sample when computing parameter statistics.

```
           Estimate   (sd)
1   price     -1.00 (0.00)
2    time    -25.89 (2.21)
3  change     -4.94 (0.88)
4 comfort    -14.45 (0.86)
```

and visualized via

```
> plot(coef(model_train), sd = 3)
```



The results indicate that the deciders value one hour travel time by about 25 euros, an additional change by 5 euros, and a more comfortable class by 14 euros.[12]

The Gibbs samples are saved in list form in the 'RprobitB_fit' object at the entry "gibbs_samples", i.e.

```
> str(model_train$gibbs_samples, max.level = 2, give.attr = FALSE)
```

[12]We note that our results are consistent with the ones that are presented in a vignette of the **mlogit** package on the same data set but using the logit model.

```
List of 2
 $ gibbs_samples_raw:List of 2
  ..$ alpha: num [1:10000, 1:4] -0.00239 -0.02179 -0.02986 -0.03343 -0.03584 ...
  ..$ Sigma: num [1:10000, 1] 0.978 0.949 0.862 0.864 0.911 ...
 $ gibbs_samples_nbt:List of 2
  ..$ alpha: num [1:500, 1:4] -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 ...
  ..$ Sigma: num [1:500, 1] 649 541 662 691 623 ...
```

This object contains 2 elements: `gibbs_samples_raw` is a list of the raw samples from the Gibbs sampler and `gibbs_samples_nbt` are the Gibbs samples used for parameter estimates, i.e. the normalized and thinned Gibbs samples after the burn-in.

Calling the summary function on the estimated '`RprobitB_fit`' object yields additional information about the Gibbs samples `gibbs_samples_nbt`. You can specify a list `FUN` of functions that compute any point estimate of the Gibbs samples, for example

- `mean()` for the arithmetic mean,
- `stats::sd()` for the standard deviation,
- `R_hat()` for the Gelman-Rubin statistic (Gelman and Rubin 1992)[13],
- or custom statistics like the absolute difference between the median and the mean as an indicator for the symmetrie of the posterior distribution.

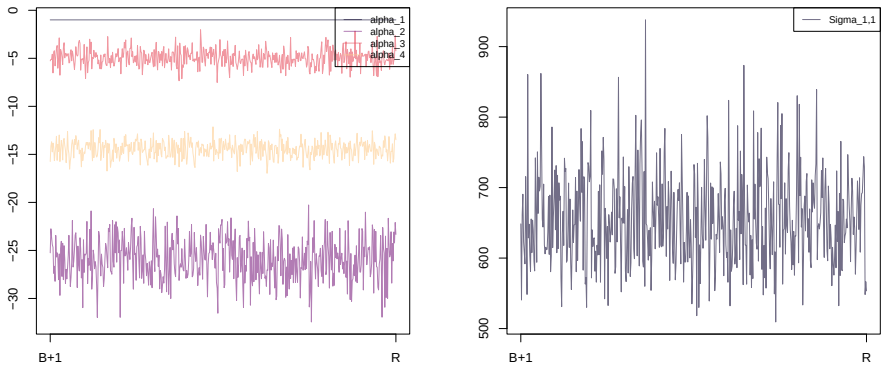
```
> summary(model_train,
+          FUN = c("mean"        = mean,
+                  "sd"          = stats::sd,
+                  "R^"          = R_hat,
+                  "custom_stat" = function(x) abs(mean(x) - median(x))
+                  )
+         )

Probit model
choice ~ price + time + change + comfort | 0
R: 10000
B: 5000
Q: 10

Utility normalization
Level: Utility differences with respect to alternative 'B'.
Scale: Coefficient of effect 'price' (alpha_1) fixed to -1.

Gibbs sample statistics
               mean           sd           R^  custom_stat
 alpha
```

[13] A Gelman-Rubin statistic close to 1 indicates that the chain of Gibbs samples converged to the stationary distribution.

```
      1        -1.00         0.00         1.00         0.00
      2       -25.89         2.21         1.00         0.02
      3        -4.94         0.88         1.01         0.01
      4       -14.45         0.86         1.00         0.02

Sigma

    1,1       655.56        65.79         1.00         7.28
```

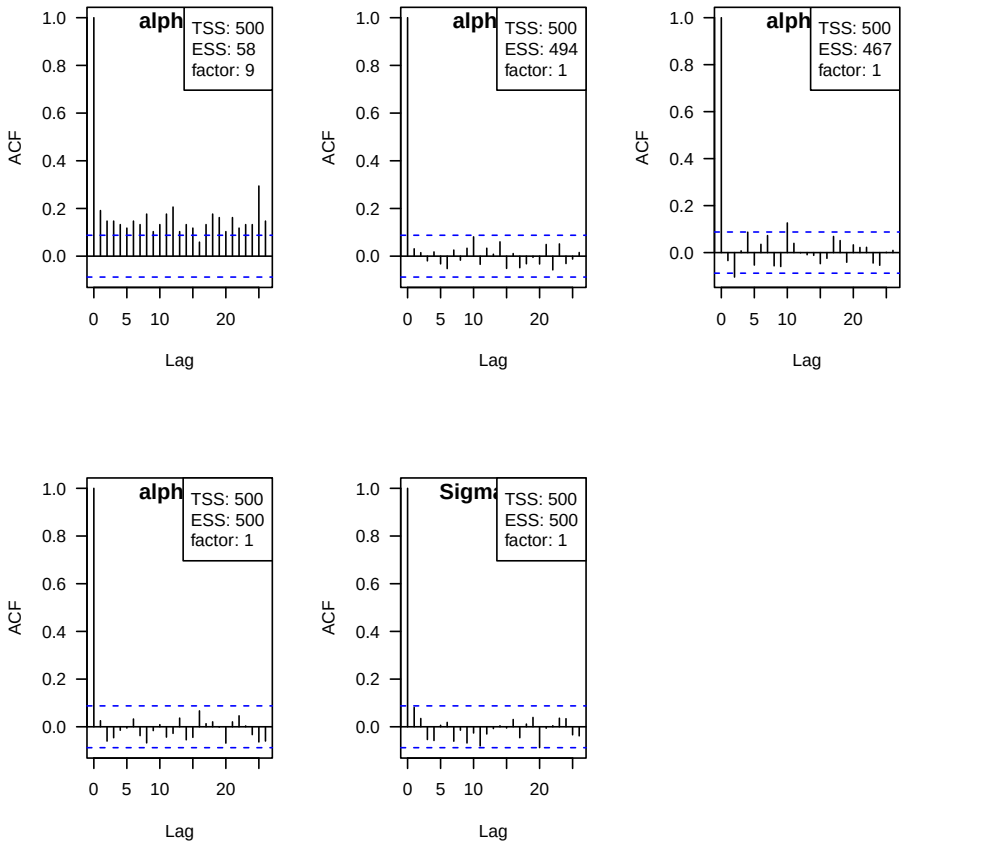
Calling the `plot()` method with the additional argument `type = "trace"` plots the trace of the Gibbs samples `gibbs_samples_nbt`:

```
> par(mfrow = c(1,2))
> plot(model_train, type = "trace")
```



Additionally, we can visualize the serial correlation of the Gibbs samples via the argument `type = "acf"`. The boxes in the top-right corner state the total sample size TSS (here `R - B` = 10000 - 5000 = 5000), the effective sample size ESS, and the factor by which TSS is larger than ESS. The effective sample size is the value $\text{TSS}/(1 + \sum_{k \geq 1} \rho_k)$, where ρ_k is the auto correlation between the chain offset by k positions. The auto correlations are estimated via the `stats::acf()` function.

```
> par(mfrow = c(2,3))
> plot(model_train, type = "acf")
```



The `transform()` method can be used to transform an '`RprobitB_fit`' object in three ways:

1. change the length `B` of the burn-in period, for example

```
> model_train <- transform(model_train, B = 1)
```

2. change the thinning factor `Q` of the Gibbs samples, for example

```
> model_train <- transform(model_train, Q = 100)
```

3. or change the model normalization `scale`, for example

```
> model_train <- transform(model_train, scale = Sigma_1 ~ 1)
```

4.3. Estimating a joint normal mixing distribution

We demonstrate how to estimate a joint normal mixing distribution in **RprobitB** using another real-data example.

Example 3: Electricity. The **mlogit** package (Croissant 2020) contains the data set `Electricity`, in which residential electricity customers were asked to decide between four hypothetical electricity suppliers. The suppliers differed in 6 characteristics:

1. their fixed price `pf` per kWh,
2. their contract length `cf`,
3. a boolean `loc`, indicating whether the supplier is a local company,
4. a boolean `wk`, indicating whether the supplier is a well known company,
5. a boolean `tod`, indicating whether the supplier offers a time-of-day electricity price (which is higher during the day and lower during the night), and
6. a boolean `seas`, indicating whether the supplier's price is seasonal dependent.

This constitutes a choice situation where choice behavior heterogeneity is expected: some customers might prefer a time-of-day electricity price (because they may be not at home during the day), while others can have the opposite preference. Ideally these differences in preferences should be modeled using characteristics of the deciders. In many cases (as in this data set) we do not have adequate information. Instead these differences in taste can be captured by means of a mixing distribution for the `tod` coefficient. This corresponds to the assumption of a random coefficient from the underlying mixing distribution to be drawn for each decider. We can use the estimated mixing distribution to determine for example the share of deciders that have a positive versus negative preference towards time-of-day electricity prices.

Additionally, we expect correlations between the random coefficients to certain covariates, for example a positive correlation between the influence of `loc` and `wk`: deciders that prefer local suppliers might also prefer well known companies due to recommendations and past experiences, although they might be more expensive than unknown suppliers. The fitted multivariate normal distribution will reveal these correlations.

The following lines prepare the `Electricity` data set for estimation. We use the convenience function `as_cov_names()` that relabels the data columns for alternative specific covariates into the required format `"<covariate>_<alternative>"`. Via the `re = c("cl","loc","wk","tod","seas")` argument, we specify that we want to model random effects for all but the price coefficient, which we will fix to `-1` to interpret the other estimates as monetary values.

```
> data("Electricity", package = "mlogit")
> Electricity <- as_cov_names(Electricity, c("pf","cl","loc","wk","tod","seas"), 1:4)
> data_elec <- prepare_data(
+    form = choice ~ pf + cl + loc + wk + tod + seas | 0,
+    choice_data = Electricity,
+    re = c("cl","loc","wk","tod","seas")
+    )
> model_elec <- fit_model(data_elec, R = 5000, scale = pf ~ -1)
```

The estimated model is saved in **RprobitB** and can be accessed via:

```
> data(model_elec, package = "RprobitB")
```

Calling the `coef()` method on the estimated model also returns the estimated (marginal) variances of the mixing distribution besides the average mean effects:

```
> coef(model_elec)
```

```
        Estimate   (sd) Variance   (sd)
1   pf     -1.00 (0.00)       NA   (NA)
2   cl     -0.25 (0.03)     0.23 (0.04)
3  loc      2.77 (0.24)     6.74 (1.19)
4   wk      2.02 (0.19)     3.48 (0.69)
5  tod     -9.70 (0.23)    10.88 (1.80)
6 seas     -9.87 (0.19)     5.90 (1.06)
```

By the sign of the estimates we can for example deduce, that the existence of the time-of-day electricity price `tod` in the contract has a negative effect. However, the deciders are very heterogeneous here, because the estimated variance of this coefficient is large (`12.37`). The same holds for the contract length `cl`. In particular, the estimated share of the population that prefers to have a longer contract length equals:

```
> cl_mu <- coef(model_elec)["cl","mean"]
> cl_sd <- sqrt(coef(model_elec)["cl","var"])
> pnorm(cl_mu / cl_sd)
```

```
[1] 0.2996708
```

The correlation between the covariates can be accessed as follows (setting `cor = FALSE` instead returns the estimated covariance matrix):

```
> cov_mix(model_elec, cor = TRUE)
```

```
            [,1]       [,2]        [,3]        [,4]        [,5]
[1,]  1.00000000 0.11846819  0.08690204 -0.05969293 -0.14077058
[2,]  0.11846819 1.00000000  0.80958767  0.12020817  0.01724159
[3,]  0.08690204 0.80958767  1.00000000  0.10642575 -0.02153803
[4,] -0.05969293 0.12020817  0.10642575  1.00000000  0.53831733
[5,] -0.14077058 0.01724159 -0.02153803  0.53831733  1.00000000
```

Here, we see the confirmation of our initial hypothesis about a high correlation between `loc` and `wk`.

4.4. Estimating latent classes

More generally, **RprobitB** allows to specify a Gaussian mixture $\sum_{c=1}^{C} \text{MVN}(b_c, \Omega_c)$ as the mixing distribution, see Section ... This specification allows for a) a better approximation of the true underlying mixing distribution and b) a preference based classification of the deciders. To estimate a latent mixture, specify a named list `latent_classes` with the following arguments and submit it to the estimation routine `fit_model()`:

- `C`, the fixed number (greater or equal 1) of latent classes, which is set to 1 per default,

- `weight_update`, a boolean, set to `TRUE` for a weight-based update of the latent classes, see below,
- `dp_update`, a boolean, set to `TRUE` for a Dirichlet process-based update of the latent classes, see below,
- `Cmax`, the maximum number of latent classes, set to `10` per default.

4.5. Weight-based update of the latent classes

The following weight-based updating scheme is analogue to Bauer *et al.* (2019) and executed within the burn-in period:

- We remove class c, if $s_c < \varepsilon_{\text{min}}$, i.e. if the class weight s_c drops below some threshold ε_{min}. This case indicates that class c has a negligible impact on the mixing distribution.

- We split class c into two classes c_1 and c_2, if $s_c > \varepsilon_{\text{max}}$. This case indicates that class c has a high influence on the mixing distribution whose approximation can potentially be improved by increasing the resolution in directions of high variance. Therefore, the class means b_{c_1} and b_{c_2} of the new classes c_1 and c_2 are shifted in opposite directions from the class mean b_c of the old class c in the direction of the highest variance.

- We join two classes c_1 and c_2 to one class c, if $\|b_{c_1} - b_{c_2}\| < \varepsilon_{\text{distmin}}$, i.e. if the euclidean distance between the class means b_{c_1} and b_{c_2} drops below some threshold $\varepsilon_{\text{distmin}}$. This case indicates location redundancy which should be repealed. The parameters of c are assigned by adding the values of s from c_1 and c_2 and averaging the values for b and Ω.

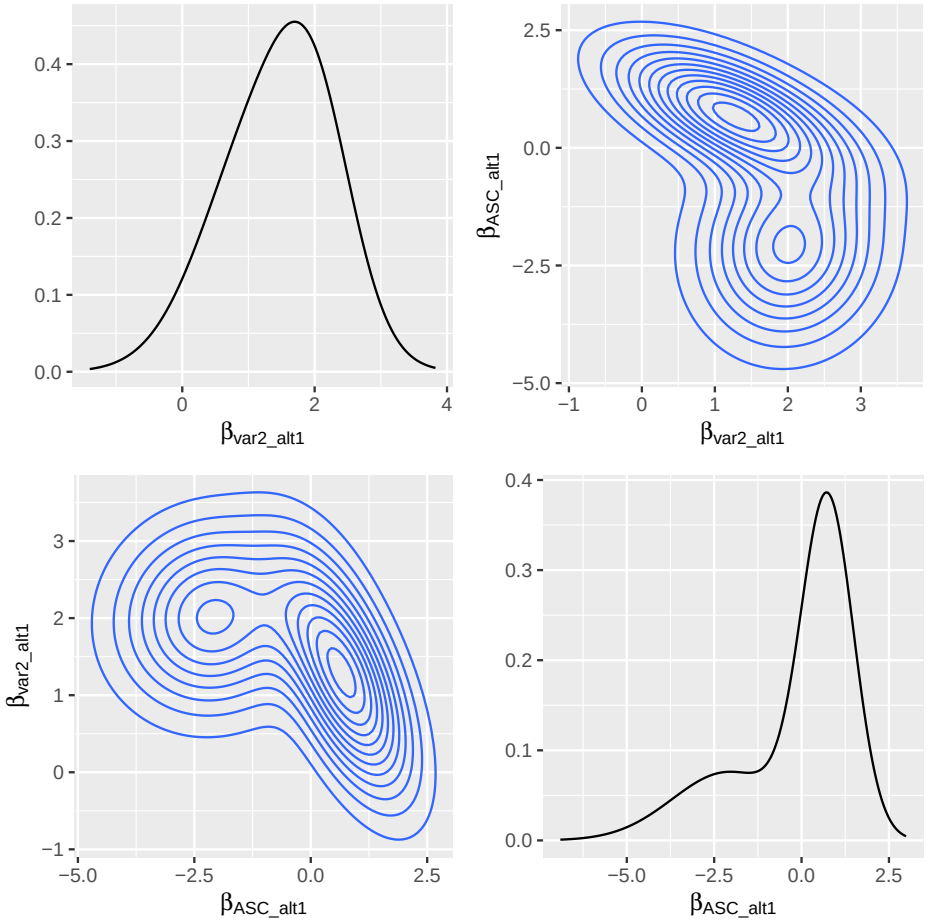
These rules contain choices on the values for ε_{min}, ε_{max} and $\varepsilon_{\text{distmin}}$. The adequate value for $\varepsilon_{\text{distmin}}$ depends on the scale of the parameters. Per default, **RprobitB** sets `epsmin = 0.01`, `epsmax = 0.99`, and `distmin = 0.1`. These values can be adapted through the `latent_class` list.

Example 2: Simulation (cont.). The pairwise mixing distributions can be visualized via calling the `plot()` method with the additional argument `type = mixture`:

```
> model_sim <- fit_model(data_sim, R = 1000, latent_classes = list("C" = 2))
> summary(model_sim)

Probit model
choice ~ var1 | var2 | var3
R: 1000
B: 500
Q: 1

Utility normalization
Level: Utility differences with respect to alternative 'alt2'.
Scale: Coefficient of the 1. error term variance fixed to 1.

Latent classes
C = 2
```

```
Gibbs sample statistics
          true    mean      sd      R^
 alpha

     1   -0.74   -0.79    0.10    1.08
     2    0.00    0.03    0.08    1.05
     3    0.74    0.83    0.12    1.00

 s

     1    0.70    0.70    0.10    1.22
     2    0.30    0.30    0.10    1.22

 b

   1.1    1.47    1.21    0.20    1.02
   1.2   -0.37    0.74    0.24    1.15
   2.1    0.74    2.01    0.53    1.50
   2.2    0.74   -2.10    0.69    1.20

 Omega

 1.1,1    0.17    0.87    0.38    1.16
 1.1,2    0.38   -0.48    0.20    1.05
 1.2,2    2.53    0.75    0.27    1.00
 2.1,1    0.90    0.57    0.38    1.03
 2.1,2   -0.65    0.05    0.58    3.44
 2.2,2    0.47    1.60    1.04    1.15

 Sigma

   1,1    1.00    1.00    0.00    1.00
```

```
> plot(model_sim, type = "mixture")
```



4.6. Dirichlet process-based update of the latent classes

As an alternative to the weight-based updating scheme to determine the correct number C of latent classes, **RprobitB** implemented the Dirichlet process. A similar approach in the context of discrete choice can be found in Burda, Harding, and Hausman (2008), where the Dirichlet process is applied to estimate a mixed logit-probit model. The Dirichlet Process is a Bayesian nonparametric method, where nonparametric means that the number of model parameters can theoretically grow to infinity. The method allows to add more mixture components to the mixing distribution if needed for a better approximation, see Neal (2000) for a documentation of the general case. The literature offers many representations of the method, including the Chinese Restaurant Process (Aldous 1985), the stick-braking metaphor (Sethuraman 1994), and the Polya Urn model (Blackwell and MacQueen 1973).

In our case, we face the situation to find a distribution g that explains the decider-specific coefficients $(\beta_n)_{n=1,\dots,N}$, where g is supposed to be a mixture of an unknown number C of Gaussian densities, i.e. $g = \sum_{c=1,\dots,C} s_c \text{MVN}(b_c, \Omega_c)$.

Let $z_n \in \{1, \dots, C\}$ denote the class membership of β_n. A priori, the mixture weights $(s_c)_c$ are given a Dirichlet prior with concentration parameter δ/C, i.e. $(s_c)_c \mid \delta \sim \text{D}_C(\delta/C, \dots, \delta/C)$. Rasmussen (2000) shows that

$$\Pr((z_n)_n \mid \delta) = \frac{\Gamma(\delta)}{\Gamma(N+\delta)} \prod_{c=1}^{C} \frac{\Gamma(m_c + \delta/C)}{\Gamma(\delta/C)},$$

where $\Gamma(\cdot)$ denotes the gamma function and $m_c = \#\{n : z_n = c\}$ the number of elements that are currently allocated to class c. Crucially, the last equation is independent of the class weights $(s_c)_c$, yet it still depends on the finite number C of latent classes. However, Li, Schofield, and Gönen (2019) shows that

$$\Pr(z_n = c \mid z_{-n}, \delta) = \frac{m_{c,-n} + \delta/C}{N - 1 + \delta},$$

where the notation $-n$ means excluding the nth element. We can let C approach infinity to derive:

$$\Pr(z_n = c \mid z_{-n}, \delta) \to \frac{m_{c,-n}}{N - 1 + \delta}.$$

Note that the allocation probabilities do not sum to 1, instead

$$\sum_{c=1}^{C} \frac{m_{c,-n}}{N - 1 + \delta} = \frac{N-1}{N - 1 + \delta}.$$

The difference to 1 equals

$$\Pr(z_n \neq z_m \; \forall \; m \neq n \mid z_{-n}, \delta) = \frac{\delta}{N - 1 + \delta}$$

and constitutes the probability that a new cluster for observation n is created. Neal (2000) points out that this probability is proportional to the prior parameter δ: A greater value for δ encourages the creation of new clusters, a smaller value for δ increases the probability of an allocation to an already existing class.

In summary, the Dirichlet process updates the allocation of each β coefficient vector one at a time, dependent on the other allocations. The number of clusters can theoretically rise to infinity, however, as we delete unoccupied clusters, C is bounded by N. As a final step after the allocation update, we update the class means b_c and covariance matrices Ω_c by means of their posterior predictive distribution. The mean and covariance matrix for a new generated cluster is drawn from the prior predictive distribution. The corresponding formulas are given in Li *et al.* (2019).

The Dirichlet process directly integrates into our existing Gibbs sampler. Given β values, it updated the class means b_c and class covariance matrices Ω_c. The Dirichlet process updating scheme is implemented in the function `update_classes_dp()`. In the following, we give a small example in the bivariate case `P_r = 2`. We sample true class means `b_true` and class covariance matrices `Omega_true` for `C_true = 3` true latent classes. The true (unbalanced) class sizes are given by the vector `N`, and `z_true` denotes the true allocations.

```
> set.seed(1)
> P_r <- 2
> C_true <- 3
```

```
> N <- c(100,70,30)
> (b_true <- matrix(replicate(C_true, rnorm(P_r)), nrow = P_r, ncol = C_true))

           [,1]       [,2]       [,3]
[1,] -0.6264538 -0.8356286  0.3295078
[2,]  0.1836433  1.5952808 -0.8204684

> (Omega_true <- matrix(replicate(C_true, rwishart(P_r + 1, 0.1*diag(P_r))$W, simplify = T
+                       nrow = P_r*P_r, ncol = C_true))

          [,1]        [,2]       [,3]
[1,] 0.3093652  0.14358543  0.2734617
[2,] 0.1012729 -0.07444148 -0.1474941
[3,] 0.1012729 -0.07444148 -0.1474941
[4,] 0.2648235  0.05751780  0.2184029

> beta <- c()
> for(c in 1:C_true) for(n in 1:N[c])
+    beta <- cbind(beta, rmvnorm(mu = b_true[,c,drop=F], Sigma = matrix(Omega_true[,c,drop
> z_true <- rep(1:3, times = N)
```

We specify the following prior parameters (for their definition see the vignette on model fitting):

```
> delta <- 0.1
> xi <- numeric(P_r)
> D <- diag(P_r)
> nu <- P_r + 2
> Theta <- diag(P_r)
```

Initially, we start with `C = 1` latent classes. The class mean `b` is set to zero, the covariance matrix `Omega` to the identity matrix:

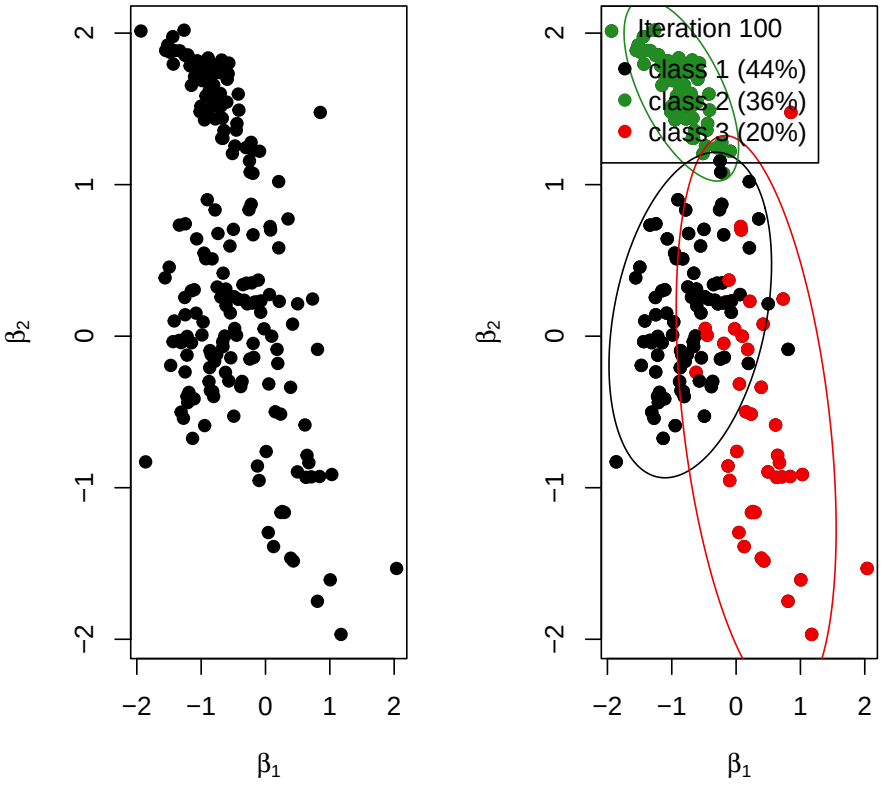
```
> z <- rep(1, ncol(beta))
> C <- 1
> b <- matrix(0, nrow = P_r, ncol = C)
> Omega <- matrix(rep(diag(P_r), C), nrow = P_r*P_r, ncol = C)
```

The following call to `update_classes_dp()` updates the latent classes in `100` iterations. Note that we specify the arguments `Cmax` and `s_desc`. The former denotes the maximum number of latent classes. This specification is not a requirement for the Dirichlet process per se, but rather for its implementation. Knowing the maximum possible class number, we can allocate the required memory space, which leads to a speed improvement. We later can verify that we won't exceed the number of `Cmax = 10` latent classes at any point of the Dirichlet process. Setting `s_desc = TRUE` ensures that the classes are ordered by their weights in a descending order to ensure identifiability.

```
> for(r in 1:100){
+    dp <- RprobitB:::update_classes_dp(
+      Cmax = 10, beta, z, b, Omega, delta, xi, D, nu, Theta, s_desc = TRUE
+      )
+    z <- dp$z
+    b <- dp$b
+    Omega <- dp$Omega
+  }
```

The Dirichlet process was able to infer the true number `C_true = 3` of latent classes:

```
> par(mfrow = c(1,2))
> plot(t(beta), xlab = bquote(beta[1]), ylab = bquote(beta[2]), pch = 19)
> RprobitB:::plot_class_allocation(beta, z, b, Omega, r = 100, perc = 0.95)
```



Computational details

Example 4: Chess. The results in this paper were obtained using R 4.1.0 with the **RprobitB** 1.0.0.9000 package. R itself and all packages used are available from the Comprehensive

R Archive Network (CRAN) at https://CRAN.R-project.org/.

Acknowledgments

This work has been financed partly by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - Projektnummer 356500581 which is gratefully acknowledged.

References

Agresti A (2015). *Foundations of Linear and Generalized Linear Models*. Wiley. ISBN 978-1-118-73003-4.

Albert JH, Chib S (1993). "Bayesian Analysis of Binary and Polychotomous Response Data." *Journal of the American Statistical Association*, **88**.

Aldous DJ (1985). "Exchangeability and related topics." **1117**, 1–198.

Allenby GM, Rossi P (1998). "Marketing models of consumer heterogeneity." *Journal of Econometrics*, **89**.

Bauer D, Büscher S, Batram M (2019). "Non-parameteric estiation of mixed discrete choice models." *Second International Choice Modelling Conference in Kobe*.

Bhat C (2011). "The Maximum Approximate Composite Marginal Likelihood (MACML) Estimation of Multinomial Probit-Based Unordered Response Choice Models." *Transportation Research Part B: Methodological*, **45**.

Bhat C, Lavieri P (2018). "A new mixed MNP model accommodating a variety of dependent non-normal coefficient distributions." *Theory and Decision*, **84**.

Blackwell D, MacQueen J (1973). "Ferguson distributions via Polya urn schemes." *The Annals of Statistics*, **1**, 353–355.

Burda M, Harding M, Hausman J (2008). "A Bayesian mixed logit–probit model for multinomial choice." *Journal of Econometrics*, **147**(2), 232–246. doi:10.1016/j.jeconom.2008.09.029. URL https://www.sciencedirect.com/science/article/pii/S0304407608001395.

Cirillo C, Axhausen K (2006). "Evidence on the distribution of values of travel time savings from a six-week diary." *Transportation Research Part A: Policy and Practice*, **40**.

Croissant Y (2020). "Estimation of Random Utility Models in R: The mlogit Package." *Journal of Statistical Software*, **95**(11), 1–41. doi:10.18637/jss.v095.i11.

Fountas G, Anastasopoulos Ph, Abdel-Aty M (2018). "Analysis of accident injury-severities using a correlated random parameters ordered probit approach with time variant covariates." *Analytic Methods in Accident Research*, **18**.

Fountas G, Pantangi SS, Hulme KF, Anastasopoulos PC (2019). "The effects of driver fatigue, gender, and distracted driving on perceived and observed aggressive driving behavior: A correlated grouped random parameters bivariate probit approach." *Analytic Methods in Accident Research*, **22**.

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

Geweke J (1998). "Efficient Simulation from the Multivariate Normal and Student-t Distributions Subject to Linear Constraints and the Evaluation of Constraint Probabilities."

Hewig J, Kretschmer N, Trippe RH, Hecht H, Coles MGH, Holroyd CB, Miltner WHR (2011). "Why humans deviate from rational choice." *Psychophysiology*, **48**(4), 507–514. doi:10.1111/j.1469-8986.2010.01081.x.

Imai K, van Dyk DA (2005). "A Bayesian analysis of the multinomial probit model using marginal data augmentation." *Journal of Econometrics*, **124**.

Li Y, Schofield E, Gönen M (2019). "A tutorial on Dirichlet process mixture modeling." *Journal of Mathematical Psychology*, **91**, 128–144. ISSN 0022-2496. doi:10.1016/j.jmp.2019.04.004. URL https://www.sciencedirect.com/science/article/pii/S0022249618301068.

McCulloch R, Rossi P (1994). "An exact likelihood analysis of the multinomial probit model." *Journal of Econometrics*, **64**.

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2), 249–265. ISSN 10618600. URL http://www.jstor.org/stable/1390653.

Nobile A (1998). "A hybrid Markov chain for the Bayesian analysis of the multinomial probit model." *Statistics and Computing*, **8**.

Oelschläger L, Bauer D (2020). "Bayes Estimation of Latent Class Mixed Multinomial Probit Models." *TRB Annual Meeting 2021*.

Rasmussen CE (2000). *The Infinite Gaussian Mixture Model*, volume 12. MIT Press.

Scaccia L, Marcucci E (2010). "Bayesian Flexible Modelling of Mixed Logit Models."

Sethuraman J (1994). "A constructive definition of dirichlet priors." *Statistica Sinica*, **4**(2), 639–650. URL http://www.jstor.org/stable/24305538.

Train K (2009). *Discrete choice methods with simulation*. 2. ed. edition. Cambridge Univ. Press.

Train K (2016). "Mixed logit with a flexible mixing distribution." *Journal of choice modelling*, **19**.

Xiong Y, Mannering FL (2013). "The heterogeneous effects of guardian supervision on adolescent driver-injury severities: A finite-mixture random-parameters approach." *Transportation Research Part B: Methodological*, **49**.

Affiliation:

Lennart Oelschläger
Department of Business Administration and Economics
Bielefeld University
Postfach 10 01 31
E-mail: lennart.oelschlaeger@uni-bielefeld.de

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
MMMMMM YYYY, Volume VV, Issue II *Submitted:* yyyy-mm-dd
doi:10.18637/jss.v000.i00 *Accepted:* yyyy-mm-dd