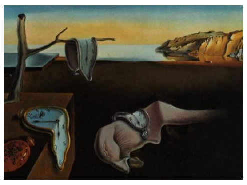
Journal of Statistical Software

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



# RprobitB: Bayes Estimation of Choice Behavior Heterogeneity in R

**Lennart Oelschläger**
Bielefeld University

**Dietmar Bauer**
Bielefeld University

**Abstract**

**RprobitB** is an R package for Bayes estimation of probit choice models, both in the cross-sectional and panel setting. The package can analyze binary, multinomial, ordered, and ranked choices, and places a special focus on modeling heterogeneity of choice behavior among deciders. The main functionality includes model fitting via Markov chain Monte Carlo methods, tools for convergence diagnostic, choice data simulation, in-sample and out-of-sample choice prediction, and model selection using information criteria and Bayes factors. The latent class model extension facilitates preference-based decider classification, where the number of latent classes can be inferred via the Dirichlet process or a weight-based updating scheme. This allows for flexible modeling of choice behavior without the need to impose structural constraints on the form of the heterogeneity. To demonstrate the application of the method, we reveal three different playing strategies of participants in an online chess tournament, where each chess player can be assigned a probability of following each of the strategies.

*Keywords*: discrete choice, probit model, choice behavior heterogeneity, Bayes estimation, decider classification, R.

## 1. Introduction

Discrete choice models aim to explain past and predict future choice behavior. They do so by connecting observed choices to observed covariates that influence the decision, such as attributes of the choice alternatives or decider's socio-demographic characteristics. While some influencing characteristics are easily measurable through surveys (like income and household size), others are not. For example, the preference for a low-emission car versus an SUV powered by a combustion engine will be influenced by the environmental concerns. The political attitude of the car buyer is hard to quantify, so it is typically not queried. But not accounting for such unobserved choice behavior heterogeneity generally leads to an inferior model fit.

The R (R Core Team 2017) package **RprobitB**[1] (Oelschläger and Bauer 2022) implements state-of-the-art tools for modeling taste heterogeneity in the context of discrete choices.

Discrete choice models are commonly interpreted as random utility models. This model class assumes that the deciders obtain utility values for the available alternatives which they seek to maximize. The utilities are unobserved by the researcher and hence modeled by a function of the covariates, in our case a linear combination of those, and a random error term. The error term distribution determines the model type, where **RprobitB** implements the probit model, assuming a joint normal distribution across alternatives (as opposed, e.g., to the logit model which assumes independent extreme value distributions). The coefficients of the linear combination represent the ceteris paribus effect of the covariates on the utility, and ratios of coefficients quantify substitution patterns, for example the willingness to pay more money for a lower CO2 emission rate.

Constant coefficients across deciders result in substitution patterns that again are constant. This can be unreasonable in certain scenarios, including the car purchase example from above. In such cases, explicit modeling of the decider's preferences would be preferable but limited by data availability. In the panel data case with many observations for each decider, fixed effect models provide options. For short panels or in cross-sectional data sets, the random effect model is a remedy, where the coefficients are assumed to be stemming from a multivariate normal (mixing) distribution. This specification allows for decider-specific coefficients and characterizes the taste heterogeneity, cf., Train (2009) and Bhat (2011). Some applications require even more flexibility: the empirical study Cirillo and Axhausen (2006) for example fits a bi-modal distribution to the values of travel time savings. For such cases, **RprobitB** provides the recently proposed approach of Oelschläger and Bauer (2020) that approximates any underlying mixing distributions by a mixture of Gaussian densities, leading to the latent class mixed probit model.

The latent class model extension also enables preference-based decider classification, i.e., identifying groups of deciders with similar preferences. This requires a reasonable specification of the total class number. While a trial-and-error strategy in conjunction with likelihood-based model selection is theoretically possible, it is numerically demanding. **RprobitB** offers two data-driven approaches that avoid the need to pre-specify the class number: weight-based class updates within the estimation routine (Oelschläger and Bauer 2020) and class updates based on the Dirichlet process, similar to Burda *et al.* (2008).

**RprobitB** implements Bayesian estimation via Markov chain Monte Carlo simulation, which has several benefits compared to the frequentist alternative for latent class mixed probit models: it does not need to compute the probit likelihood (which is not in closed form and hence would require approximation), it avoids numerical challenges associated with finding the likelihood optimum, it enables to impose prior believes on the model parameters, and it provides posterior parameter distributions instead of point estimates only. Additionally, the Bayesian approach was shown to be computationally faster with increasing number of random effects and normal mixing distributions than the maximum likelihood approach, cf., Train (2001) for a simulation study in the logit case.

There already exist several open-source implementations for discrete choice modeling: **Rchoice** (Sarrias 2016) and **mlogit** (Croissant 2020) are two R packages for maximum likelihood estimation (MLE) of the (mixed) probit and logit model. The Python (Van Rossum and Drake Jr

[1]The package name is a portmanteau of the language R, the probit model, and the Bayes framework.

1995) library **Biogeme** (Bierlaire 2020) can additionally estimate latent class (LC) models. In Julia (Bezanson *et al.* 2017), the **DiscreteChoiceModels.jl** package (Bhagat-Conway 2022) currently only supports the multinomial logit model. The R package **gmnl** (Sarrias and Daziano 2017) provides generalized and latent class mixed multinomial logit models, while the **apollo** package (Hess and Palma 2019) allows for more flexible logit and probit model specifications, with both maximum likelihood and Bayesian estimation. The R packages **bayesm** (Rossi 2019) and **MNP** (Imai and van Dyk 2022) are two alternatives for Bayesian estimation of the probit model, both implementing Markov chain Monte Carlo simulation methods similar to **RprobitB**. The **RprobitB** package complements this collection by implementing latent class mixed probit models in conjunction with class updating schemes, as outlined above. The software comparison is summarized in Table 1.

| | Probit | Logit | Bayes | MLE | Mixed | LC | LC update |
|---|---|---|---|---|---|---|---|
| **Rchoice** | ✓ | ✓ | | ✓ | ✓ | | |
| **mlogit** | ✓ | ✓ | | ✓ | ✓ | | |
| **Biogeme** | ✓ | ✓ | | ✓ | ✓ | ✓ | |
| **DiscreteChoiceModels.jl** | | ✓ | | ✓ | | | |
| **gmnl** | | ✓ | | ✓ | ✓ | ✓ | |
| **apollo** | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| **bayesm** | ✓ | ✓ | ✓ | | ✓ | | |
| **MNP** | ✓ | | ✓ | | | | |
| **RprobitB** | ✓ | | ✓ | | ✓ | ✓ | ✓ |

Table 1: Overview of open-source software for estimating discrete choice models.

This article provides a general description of choice modeling with **RprobitB** and is structured as follows. To fix our notation, Section 2 defines the probit model and formalizes the concepts of mixing distributions and latent classes. Sections 3 to 6 describe the package workflow, including choice data preparation, simulation, model fitting, choice prediction, preference classification, and model selection. The main functions for these tasks are visualized in the flowchart Figure 1. We illustrate the functions throughout the paper using three different real world data sets: the two stated-choice data sets `Train` and `Electricity` from **mlogit** allow for result comparison across packages and between the probit and logit model class, while we use a revealed-choice data set about online chess strategy (that is contained in **RprobitB**) to demonstrate flexible preference classification using the Dirichlet process. A simulation example demonstrates the accuracy of the model parameter estimates. Section 7 concludes and gives an outlook of anticipated package extensions.

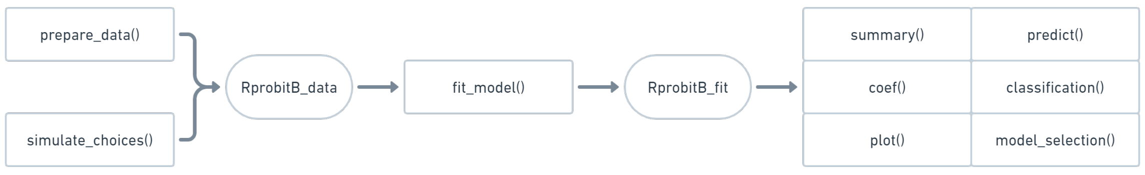


Figure 1: Flowchart of the main **RprobitB** functionalities (rectangles) and objects (ovals).

## 2. The probit model

Assume that we know the choices of $N$ deciders choosing between $J \geq 2$ alternatives at each of $T$ choice occasions.[2] Specific to each decider, alternative and choice occasion, we observe $P$ covariates, a linear combination of which eventually explains the latent random utility:

$$U_{ntj} = X_{ntj}^\top \tilde{\beta}_n + \varepsilon_{ntj} \tag{1}$$

for $n = 1, \dots, N$, $t = 1, \dots, T$ and $j = 1, \dots, J$. Here, $X_{ntj}$ is a (column) vector of $P$ characteristics specific to alternative $j$ as faced by decider $n$ at choice occasion $t$, $\tilde{\beta}_n \in \mathbb{R}^P$ is the coefficient vector encoding the preferences of $n$, and $(\varepsilon_{nt:}) = (\varepsilon_{nt1}, \dots, \varepsilon_{ntJ})^\top \sim \text{MVN}_J(0, \Sigma)$ is the model's error term vector for $n$ at $t$.

The value $U_{ntj}$ on the left-hand side of Equation 1 can be interpreted as the decider's utility. It is unobserved by the researcher, but we assume that the deciders know their utilities for each alternative and make a choice which is consistent with utility maximization.[3] Therefore,

$$y_{nt} = \underset{j=1,\dots,J}{\text{argmax}}\, U_{ntj}, \tag{2}$$

where $y_{nt} = j$ denotes the event that decider $n$ chooses $j$ at her $t$-th choice occasion.

Equation 1 has a decider-specific coefficient vector $\tilde{\beta}_n$. Some entries of $\tilde{\beta}_n$ can be fixed across deciders, in which case the coefficient vector is of the form $\tilde{\beta}_n^\top = (\alpha^\top, \beta_n^\top)^\top$, where $\alpha \in \mathbb{R}^{P_f}$ are $P_f$ coefficients that are constant across deciders and $\beta_n$ are $P_r$ decider-specific coefficients, $P_f + P_r = P$. The decider-specific coefficients are assumed to be realizations of an underlying mixing distribution and to be independent of the characteristics $X_{ntj}$ and the errors $(\varepsilon_{nt:})$. This distribution characterizes heterogeneity among the deciders and allows for individual sensitivities as motivated in the introduction.

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, pp. 136 ff.). Instead, **RprobitB** implements the approach of Oelschläger and Bauer (2020) to approximate any underlying mixing distribution by a mixture of $P_r$-variate Gaussian densities $\phi_{P_r}$ 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 $c = 1, \dots, C$ 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 allows for decider classifications, see Section 4.4 for an example.

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

[3]Utility maximizing behavior is a common assumption in econometric models. However, studies have shown that humans do not always decide in this rational sense (Tversky and Kahneman 1986; Hewig *et al.* 2011). See also (Thaler and Ganser 2015) for a survey of the history of the discipline of behavioral economics.

Finally, the probit model requires normalization, because any utility model is invariant towards the level and the scale of utility (Train 2009, Ch. 2). We therefore normalize the model by taking differences in Equation 1 and fixing one error term variance:

$$\tilde{U}_{ntj} = \tilde{X}_{ntj}^\top \tilde{\beta}_n + \tilde{\varepsilon}_{ntj}, \tag{3}$$

where (choosing some alternative $k \in \{1, \dots, J\}$ as the reference) $\tilde{U}_{ntj} = U_{ntj} - U_{ntk}$, $\tilde{X}_{ntj} = X_{ntj} - X_{ntk}$, and $\tilde{\varepsilon}_{ntj} = \varepsilon_{ntj} - \varepsilon_{ntk}$ for $j \neq k$. The error term differences $(\tilde{\varepsilon}_{nt:})$ again are multivariate normally distributed with mean 0 but transformed covariance matrix $\tilde{\Sigma}$, in which we fix one diagonal element to a positive number.[4]

## 3. Choice data

**RprobitB** requests that choice data sets are a) of class '`data.frame`', b) in wide format (i.e., each row provides the full information for one choice occasion), c) contain a column with unique identifiers for each decision maker (and optionally each choice occasion), 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, Ch. 2). Alternatives can be considered as ordered (e.g., the level of agreement on a Likert rating scale), and additionally full rankings of the alternatives can be provided (e.g., when asking the respondent to rank all available alternatives from best to worst).

### 3.1. Different types of covariates

Different covariate types can be considered: covariates that are constant across alternatives (e.g., a car buyer's income), covariates that are alternative specific (e.g., the car's price), covariates with the same coefficient for all alternatives (e.g., the car price may have the same impact on all alternatives), and covariates that have alternative specific coefficients (e.g., the range of an electric car might be of more importance than for other types of propulsion). To allow for these different types, we generalize Equation 1 to

$$U_{ntj} = \beta_{0j} + A_{ntj}\beta_1 + B_{nt}\beta_{2j} + C_{ntj}\beta_{3j} + \varepsilon_{ntj}. \tag{4}$$

Here, the covariates $A$ and $C$ depend on the alternative $j$, while $B$ is only decider and choice occasion specific. The coefficient $\beta_1$ for $A$ is constant (i.e., the same for each alternative), whereas $\beta_{2j}$ and $\beta_{3j}$ for $B$ and $C$ are alternative specific. The intercepts $\beta_{0j}$ are called alternative specific constants (ASCs). ASCs capture the average effect on the alternative's utility of all factors that are not included in the model.

Note that the full collections $(\beta_{0j})_{j=1,\dots,J}$ of ASCs and $(\beta_{2j})_{j=1,\dots,J}$ of coefficients for covariate type $B$ are not identified. This is because we took utility differences for model normalization (cf., Section 2), and hence one coefficient is a linear combination of the others, respectively.

[4]Fixing one element of $\tilde{\Sigma}$ determines the utility scale. Fixing one fixed effect (i.e., one entry of $\alpha$) serves the same purpose. Both alternatives are implemented, see Section 4.2.

To achieve identifiability, we fix $\beta_{0k}$ and $\beta_{2k}$ to 0 for one base alternative $k$. The coefficients $(\beta_{0j})_{j \neq k}$ and $(\beta_{2j})_{j \neq k}$ then have to be interpreted with respect to $k$.

### 3.2. Formula framework

Specifying Equation 4 in `R` requires a flexible formula framework. **RprobitB** can interpret a '`formula`' object of the form `choice ~ A | B | C`, where `choice` is the name of the dependent variable (the discrete choice we aim to explain), and `A`, `B`, and `C` are the different covariate types of Section 3.1.[5]

The framework has the following rules. ASCs are added to the model by default. 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.

### 3.3. Preparing data for estimation

Before model estimation, any `choice_data` set must pass the `prepare_data()` function together with a formula object `form` introduced in Section 3.2:

```
> 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()` (that we introduce in Section 4.2). The following arguments of `prepare_data()` are optional:

- `re`: A character vector of covariate names in `form` with random effects. Per default `re = NULL`, i.e., no random effects.
- `alternatives`: A character vector of the alternative names, defining the choice set. If not specified, all chosen alternatives in `choice_data` are considered.
- `ordered` and `ranked`: Two booleans, that are set to `FALSE` by default. If set to `TRUE`, the alternatives are interpreted as ordered, or the choices are interpreted as ranked, respectively.
- `base`: One element of `alternatives` specifying the base alternative (cf., Section 3.1). Per default, `base` 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 `choice_data`.

[5] We adapted this formula framework from the **mlogit** package.

- `standardize`: A character vector of variable names of `form` that get standardized, i.e., rescaled to have a mean of 0 and a standard deviation of 1 (none by default).
- `impute`: A character, specifying how to handle missing covariates in `choice_data`. Options are `"complete_cases"` (removing rows that contain missing entries, which is the default behavior), `"zero"` (replacing missing entries by 0), and `"mean"` (imputing missing entries by the covariate mean).

**Example 1: Train trips.** The **mlogit** package contains the data set `Train` with 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 `changes`. 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 Dutch 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 ...
```

For demonstration, we include all choice characteristics into our probit model, connect them to constant and fixed coefficients, and exclude ASCs:

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

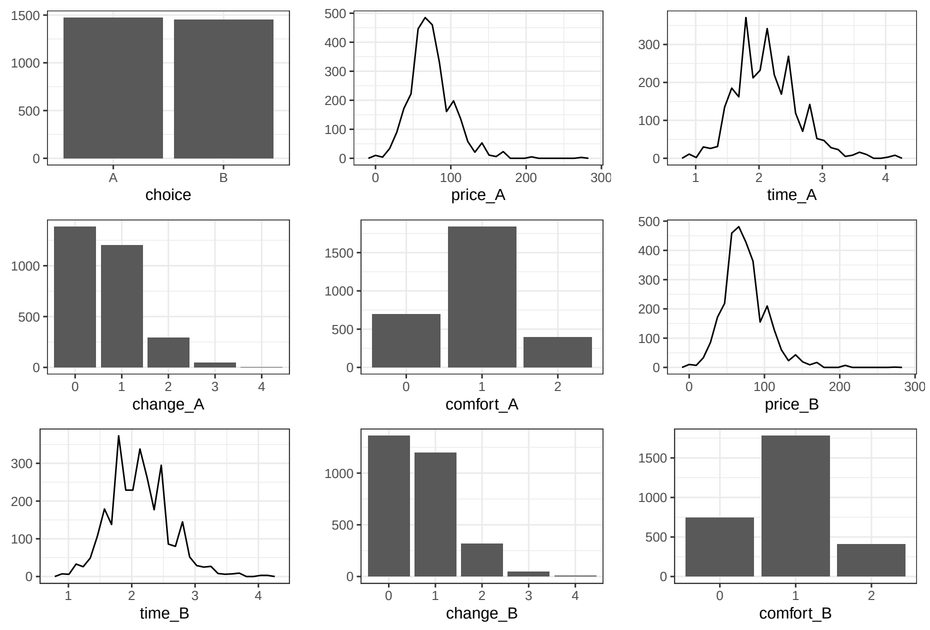
Passing `form` to `prepare_data()` returns an '`RprobitB_data`' object. We will feed this object into the estimation routine `fit_model()` in Section 4.

```
> data_train <- prepare_data(
+    form = form, choice_data = Train, id = "id", idc = "choiceid"
+  )
```

The `summary()` and `plot()` methods are useful for a first data overview:

```
> summary(data_train)

                  count
deciders            235
choice occasions   5-19
total choices      2929
alternatives          2
- 'A'              1474
- 'B'              1455
```

```
> plot(data_train)
```



## 3.4. Ordered alternatives

The two choice alternatives from the train trip example are unordered. If we had asked "rate your train trip from 1 (horrible) to 7 (great)", then the respondents would choose from a set of ordered alternatives. Such ordered alternatives can by analyzed by setting `ordered = TRUE` in `prepare_data()`. In this case, `alternatives` becomes a mandatory argument, where the alternatives must be named from worst to best.

Deciders obtaining separate utilities for each alternative is no longer a natural concept in this case (Train 2009, Ch. 7.4). Instead, we model only a single utility

$$U_{nt} = X_{nt}^\top \tilde{\beta}_n + \varepsilon_{nt}$$

per decider $n$ and choice occasion $t$. The value can be interpreted as the level of association that $n$ has with the choice question. It falls into discrete categories, which in turn are linked to the ordered alternatives $j = 1, \dots, J$. Formally, we replace the link in Equation 2 by

$$y_{nt} = \sum_{j=1,\dots,J} j \cdot I(\gamma_{j-1} < U_{nt} \leq \gamma_j),$$

where $\gamma_0 = -\infty$ and $\gamma_J = +\infty$. This implies that alternative $j$ is chosen, if the utility falls into the interval $(\gamma_{j-1}, \gamma_j]$. Monotonicity of the thresholds $(\gamma_j)_{j=1,\dots,J-1}$ is ensured by estimating logarithmic increments $d_j$ with $\gamma_j = \sum_{i \leq j} \exp(d_i)$, $j = 1, \dots, J-1$. For level normalization, we fix $\gamma_1 = 0$.

## 3.5. Ranked choices

Ranked choices are yet another model variation: rather than recording only the single most preferred alternative, some surveys ask for a full ranking of all the alternatives, which reveals far more about the underlying preferences. Ranked choices can by analyzed by setting `ranked = TRUE` in `prepare_data()`. The choice column of the data set must provide the full ranking for each choice occasion (from most preferred to least preferred), where the alternatives are separated by commas, see `help(prepare_data, package = "RprobitB")` for details.

The ranked probit model follows directly from the general unordered case noting that the ranking implies that the highest ranked alternative is chosen in any case, while the second highest ranked alternative is chosen, if the highest ranked alternative is not available and so forth. The only difference is that we take flexible utility differences such that the differenced utility vector is always negative, in contrast to Equation 3 where we differenced with respect to a fixed reference alternative. Thereby, we incorporate information of the full ranking (Train 2009, Ch. 7.3).

## 3.6. Simulating choice data

Simulation typically serves to assess the properties of the estimation algorithm either for research or in a bootstrap like fashion. The `simulate_choices()` function simulates choice data from a prespecified probit model. In order to simulate the choices of `N` deciders in `T` choice occasions[6] among `J` alternatives, we call

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

where `form` is a model formula (cf., Section 3.2). The following arguments are optional:

- `re`, `ordered`, `ranked`, `base`, `standardize`: Analogue to `prepare_data()`.
- `alternatives`: A character vector of length `J` with the names of the choice alternatives (by default the first `J` uppercase letters of the Roman alphabet).
- `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. Unspecified covariates are drawn from a standard normal distribution.
- `seed`: Optionally set a seed for the simulation.

[6] `T` can be either a single integer for a fixed number of choice occasions per decider, or a vector of length `N` with decision maker specific numbers of choice occasions.

By default, the model parameters are drawn randomly from the prior distributions specified in Section 4.1. Alternatively, they can be set via a named list for the function's `true_parameter` argument. The list can contain

- a numeric vector `alpha` with fixed effects,
- the number `C` of latent classes (`C = 1` by default),
- a numeric vector `s` of length `C` with class weights,
- a matrix `b` with class means as columns,
- a matrix `Omega` with vectorized class covariance matrices as columns,
- a matrix `Sigma_full` (`Sigma`), the (differenced) error term covariance matrix,
- a matrix `beta` with the decision maker specific coefficient vectors as columns,
- a numeric vector `z` of length `N` with elements in `1:C`, representing the class allocations,
- a numeric vector `d` of the logarithmic threshold increments in the ordered probit case.

**Example 2: Simulated choices.** For illustration, we simulate the choices of `N = 200` deciders at `T = 30` choice occasions between two fictitious alternatives:

```
> N <- 200
> T <- 30
> alternatives <- c("alt1", "alt2")
> form <- choice ~ var1 | var2 | var3
> re <- c("var2", "ASC")
```

The `overview_effects()` function provides an overview of the effect names (`effect`), whether the covariate has alternative specific values (`as_value`), whether the effect has alternative specific coefficients (`as_coef`), and whether the effect is random (`random`):

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

```
     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
```

The model has three fixed effects, consequently the vector `alpha` must be of length three, where the coefficients correspond to `var1`, `var3_alt1`, and `var3_alt2`, respectively. We set $\alpha = (-2, 0, 1)$. Additionally, the model has two random effects `var2_alt1` and `ASC_alt1`, hence the matrix `b` must be of dimension `2 x C`, with the mean effect in the different classes as columns. We specify `C = 3` latent classes with class weights $s = (0.6, 0.3, 0.1)$ and class means $b_1 = (-2, 1)$, $b_2 = (0, 2)$, and $b_3 = (2, -1)$. As class covariance matrices, we specify

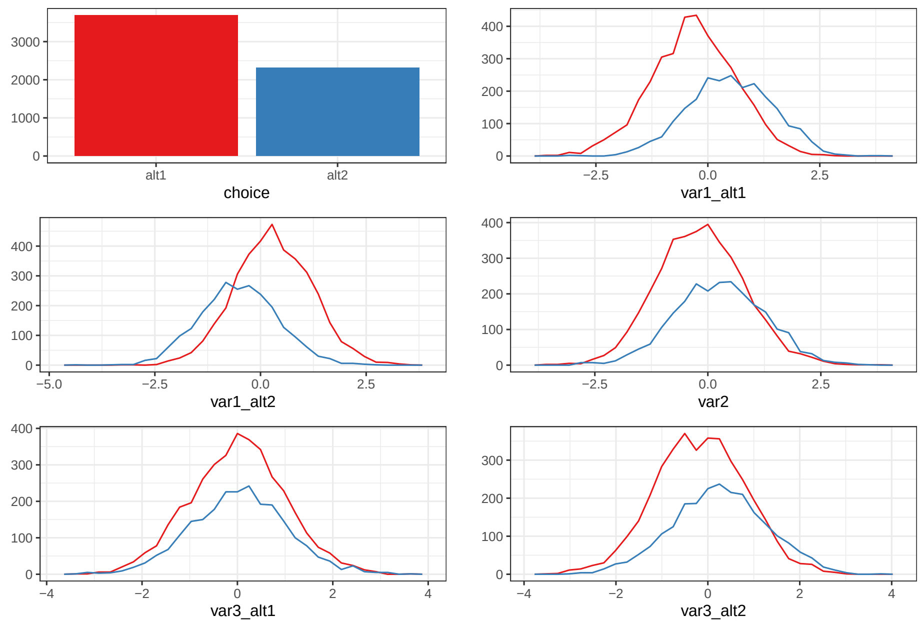
$$\Omega_1 = \begin{pmatrix} 0.3 & 0.7 \\ 0.7 & 1.9 \end{pmatrix}, \quad \Omega_2 = \begin{pmatrix} 1.3 & -0.2 \\ -0.2 & 0.9 \end{pmatrix}, \quad \Omega_3 = \begin{pmatrix} 0.6 & -0.9 \\ -0.9 & 2.4 \end{pmatrix}.$$

The differenced error term matrix $\tilde{\Sigma}$ is fixed to 1 here.

```
> true_parameter <- list(
+    alpha = c(-2,0,1), C = 3, s = c(0.6,0.3,0.1), Sigma = 1,
+    b = matrix(c(-2,1,0,2,2,-1), ncol = 3),
+    Omega = matrix(c(0.3,0.7,0.7,1.9,1.3,-0.2,-0.2,0.9,0.6,-0.9,-0.9,2.4),
+                   ncol = 3)
+  )
> data_sim <- simulate_choices(
+    form = form, N = N, T = T, J = 2, re = re, alternatives = alternatives,
+    seed = 1, true_parameter = true_parameter
+  )
```

Setting `by_choice = TRUE` in the `plot()` method of '`RprobitB_data`' objects visualizes the (randomly drawn) covariates grouped by the chosen alternatives:

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



The graphic is consistent with our model specification: for example, covariate `var1` was specified to have a negative effect on `alt1`, because the coefficient of `var1` (the first value of `alpha`) is negative ($-2$). Hence, higher values of `var1_alt1` correspond more frequently to choice `alt2` (top right panel).

# 4. Model fitting

**RprobitB** estimates the probit model in a Bayesian framework that 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 (Albert and Chib 1993), which treats the latent utilities in Equation 1 as additional parameters. Then, conditional on these parameters, the probit model constitutes a standard Bayesian linear regression setup. Its posterior distribution can be approximated via Gibbs sampling.

## 4.1. Prior and posterior distributions

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 $\text{MVN}_{P_f}$ denotes the $P_f$-dimensional normal distribution with mean $\psi$ and covariance $\Psi$,
- $b_c \sim \text{MVN}_{P_r}(\xi,\Xi)$, independent for all $c$,
- $\Omega_c \sim W^{-1}_{P_r}(\nu,\Theta)$, independent for all $c$, where $W^{-1}_{P_r}(\nu,\Theta)$ denotes the $P_r$-dimensional inverse Wishart distribution with $\nu$ degrees of freedom and scale matrix $\Theta$,
- $\tilde{\Sigma} \sim W^{-1}_{J-1}(\kappa,\Lambda)$,
- and $d \sim MVN_{J-2}(\zeta,Z)$.

These priors imply the following conditional posteriors (Oelschläger and Bauer 2020):

- 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 $m_c = \#\{n : z_n = c\}$ denotes the current absolute size of class $c$. The model is invariant to permutations of the class labels $1,\dots,C$. We therefore accept an update only if the ordering $s_1 > \dots > s_C$ still holds (thereby ensuring a unique class labeling).

- The allocation variables $(z_n)_n$ are updated independently for all $n$ via

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

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

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

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

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

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

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

- Independently for all $n, t$ and conditionally on the other components, the differenced utility vectors $(\tilde{U}_{nt:})$ follow a $(J-1)$-variate truncated 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 (analogue to Geweke (1998)):

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

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

$$\mu_{\tilde{U}_{ntj}} = \tilde{W}_{ntj}^{\top}\alpha + \tilde{X}_{ntj}^{\top}\beta_n - \Sigma_{\tilde{U}_{ntj}}(\tilde{\Sigma}^{-1})_{j(-j)}(\tilde{U}_{nt(-j)} - \tilde{W}_{nt(-j)}^{\top}\alpha - \tilde{X}_{nt(-j)}^{\top}\beta_n),$$

  where $(\tilde{\Sigma}^{-1})_{jj}$ denotes the $(j,j)$-th element of $\tilde{\Sigma}^{-1}$, $(\tilde{\Sigma}^{-1})_{j(-j)}$ the $j$-th row without the $j$-th entry, $\tilde{W}_{nt(-j)}$ and $\tilde{X}_{nt(-j)}$ the differenced covariate matrices connected to fixed and random effects, respectively, with the $j$-th column removed.

  In the ordered probit case, the (one-dimensional) utility in each choice occasion is drawn from a truncated normal distribution, where the truncation points are determined by the threshold increments $d$.

- Updating the fixed coefficient vector $\alpha$ is achieved by applying the formula for Bayesian linear regression of the regressors $\tilde{W}_{nt}$ on the regressands $(\tilde{U}_{nt:}) - \tilde{X}_{nt}^{\top}\beta_n$, i.e.,

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

$$\mu_\alpha = \Sigma_\alpha(\Psi^{-1}\psi + \textstyle\sum_{n=1,t=1}^{N,T} \tilde{W}_{nt}\tilde{\Sigma}^{-1}((\tilde{U}_{nt:}) - \tilde{X}_{nt}^{\top}\beta_n)),\ \Sigma_\alpha = (\Psi^{-1} + \textstyle\sum_{n=1,t=1}^{N,T} \tilde{W}_{nt}\tilde{\Sigma}^{-1}\tilde{W}_{nt}^{\top})^{-1}.$$

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

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

$$\mu_{\beta_n} = \Sigma_{\beta_n}(\Omega_{z_n}^{-1}b_{z_n} + \textstyle\sum_{t=1}^{T} \tilde{X}_{nt}\tilde{\Sigma}^{-1}((\tilde{U}_{nt:}) - \tilde{W}_{nt}^{\top}\alpha)),\ \Sigma_{\beta_n} = (\Omega_{z_n}^{-1} + \textstyle\sum_{t=1}^{T} \tilde{X}_{nt}\tilde{\Sigma}^{-1}\tilde{X}_{nt}^{\top})^{-1}.$$

- The covariance matrix $\tilde{\Sigma}$ of the error term differences is updated by means of

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

  where $S = \sum_{n=1,t=1}^{N,T} \tilde{\varepsilon}_{nt:}\tilde{\varepsilon}_{nt:}^{\top}$ and $\tilde{\varepsilon}_{nt:} = (\tilde{U}_{nt:}) - \tilde{W}_{nt}^{\top}\alpha - \tilde{X}_{nt}^{\top}\beta_n$.

- The utility thresholds `d` are updated by means of a Metropolis–Hastings algorithm with a normal proposal density.

The Gibbs samples obtained from this updating scheme (except for $s$ and $z$) lack identification with respect to the scale (cf., Section 2). Subsequent to the sampling and for the $i$-th updates in each iteration $i$, we therefore apply the normalization $\alpha^{(i)} \cdot \omega^{(i)}$, $b_c^{(i)} \cdot \omega^{(i)}$, $\tilde{U}_{nt}^{(i)} \cdot \omega^{(i)}$, $\beta_n^{(i)} \cdot \omega^{(i)}$, $\Omega_c^{(i)} \cdot (\omega^{(i)})^2$, $\tilde{\Sigma}^{(i)} \cdot (\omega^{(i)})^2$, and $\gamma^{(i)} \cdot \omega^{(i)}$, where either $\omega^{(i)} = (\text{const}/(\tilde{\Sigma}^{(i)})_{jj})^{0.5}$ with $(\tilde{\Sigma}^{(i)})_{jj}$ the $j$-th diagonal element of $\tilde{\Sigma}^{(i)}$, $1 \leq j \leq J-1$, or alternatively $\omega^{(i)} = \text{const}/\alpha_p^{(i)}$ for some coordinate $1 \leq p \leq P_f$ of the $i$-th draw for the coefficient vector $\alpha$. Here, 'const' is a constant to specify custom utility scales.

## 4.2. The estimation routine

The Gibbs sampling scheme can be executed via `fit_model(data = data)`, where `data` is an ‘`RprobitB_data`’ object (introduced in Section 3).[7] Optional arguments are:

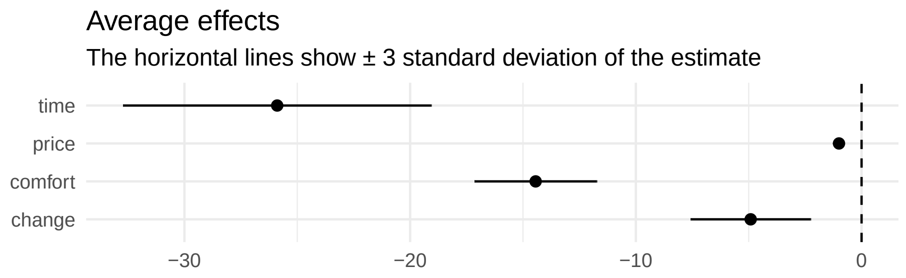
- `scale`: A character which determines the utility scale (cf., Section 2). It is of the form `"<parameter> := <value>"`, where `<parameter>` is either the name of a fixed effect or `Sigma_<j>,<j>` for the `<j>`-th diagonal element of `Sigma`, and `<value>` is the value of the fixed parameter (i.e., ’const’ from above). Per default `scale = "Sigma_1,1 := 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 = 1000`.
- `B`: The length of the burn-in period (`B = R/2` by default).[8]
- `Q`: The thinning factor for the Gibbs samples (`Q = 1` by default).
- `print_progress`: A boolean, determining whether to print the Gibbs sampler progress.
- `prior`: A named list of parameters for the prior distributions. Default values are documented in the `check_prior()` function, see `help(check_prior, package = "RprobitB")`.

**Example 1: Train trips (cont.).** Recall the `Train` data set of stated train trip alternatives, characterized by their `price`, `time`, number of `changes`, and level of `comfort`. From this data, we previously built the ‘`RprobitB_data`’ object `data_train`, which we now pass to the estimation routine. For scale normalization, we fix the `price` coefficient to `-1`, which has the advantage that we can interpret the other coefficients as monetary values:

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

We can visualize the estimated coefficients (the Gibbs sample means) as follows:

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



[7]The ‘`RprobitB_data`’ object already contains the model formula and the random effect specification.

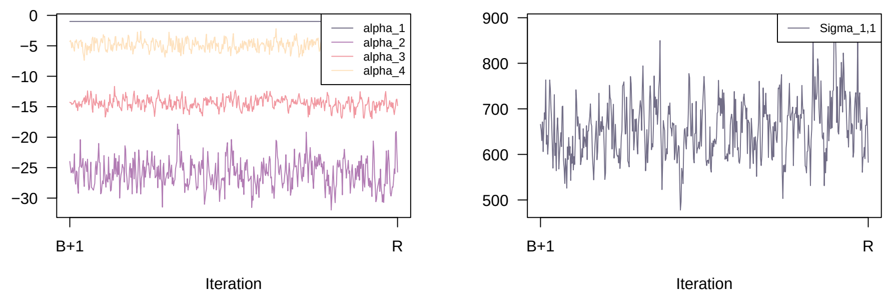
[8]The theory behind Gibbs sampling constitutes that the sequence of samples produced by the updating scheme is a Markov chain with stationary distribution equal to the desired joint posterior distribution. It takes a certain number of iterations for that stationary distribution to be approximated reasonably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

The results indicate that the deciders value one hour travel time by about 26 euros, an additional change by 5 euros, and a more comfortable class by 14 euros.[9] Calling the `summary()` method on the estimated ‘`RprobitB_fit`’ object yields additional information about the (transformed) Gibbs samples. The method receives a list `FUN` of arbitrary functions that can compute point estimates of the Gibbs samples, by default `mean()` for the arithmetic mean, `stats::sd()` for the standard deviation, and `R_hat()` for the Gelman-Rubin statistic[10] (Gelman and Rubin 1992):

```
> summary(model_train, FUN = c("mean" = mean, "sd" = stats::sd, "R^" = R_hat))

Probit model
Formula: choice ~ price + time + comfort + change | 0
R: 1000, B: 500, Q: 1
Level: Utility differences with respect to alternative 'B'.
Scale: Coefficient of effect 'price' (alpha_1) fixed to -1.

Gibbs sample statistics
          mean       sd       R^
 alpha

     1   -1.00     0.00     1.00
     2  -25.89     2.28     1.00
     3  -14.44     0.90     1.00
     4   -4.91     0.89     1.01

 Sigma

   1,1  656.92    63.81     1.04
```

The `plot()` method with the additional argument `type = "trace"` visualizes the trace of the transformed and thinned Gibbs samples after the burn-in period:

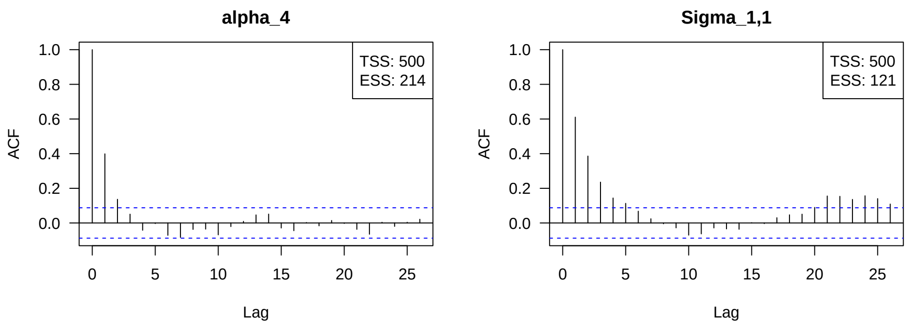
```
> par(mfrow = c(1,2))
> plot(model_train, type = "trace")
```



[9]We note that these results are consistent with the ones that are presented in a vignette of **mlogit** entitled "The random parameters (or mixed) logit model" on the same data set but using the logit model.
[10]A Gelman-Rubin statistic (a lot) greater than 1 indicates convergence issues of the Gibbs sampler.

Additionally, we can plot the autocorrelation of the Gibbs samples via `type = "acf"`, below exemplary for `alpha_4` and `Sigma_1,1`. The boxes in the plot's top right corner state the total sample size TSS, given by `(R - B) / Q`, and the effective sample size ESS. The effective sample size is the value $\text{TSS} \cdot \sigma^2 / f(0)$, where $f(0) = \sigma^2(1 + 2\sum_{k\geq 1} \rho_k)$ denotes the spectrum at frequency zero of the stationary output of the Gibbs sampler (Marin and Robert 2014). The spectrum is estimated via the `stats::spec.ar()` function.

```
> par(mfrow = c(1,2))
> plot(model_train, type = "acf", ignore = c("alpha_1", "alpha_2", "alpha_3"))
```



The `transform()` method can be used to reduce the autocorrelation in a subsample (commonly referred to as "thinning"):[11]

```
> model_train <- transform(model_train, Q = 5)
```

**Example 3: Electricity suppliers.** The `Electricity` data set from **mlogit** contains choices of residential electricity customers that decide between four hypothetical electricity suppliers. We expect heterogeneity in choice behavior here, because customers might value certain contract characteristics differently based on their living conditions. In particular, the contracts differed in six characteristics: their fixed price `pf` per kilowatt hour, their contract length `cf`, whether the supplier is a local company (`loc`), whether the supplier is a well known company (`wk`), whether the supplier offers a time-of-day electricity price which is higher during the day and lower during the night (`tod`), and whether the supplier's price is seasonal dependent (`seas`).

The following lines prepare the data set for estimation. We first use the convenience function `as_cov_names()` that relabels the data columns for alternative specific covariates into the required format `"<covariate>_<alternative>"`:

```
> data("Electricity", package = "mlogit")
> Electricity <- as_cov_names(
+    choice_data = Electricity,
+    cov = c("pf", "cl", "loc", "wk", "tod", "seas"),
+    alternatives = 1:4
+  )
```

[11]The function can also be used to change the length of the burn-in period with `transform(model_train, B = B_new)` or the utility scale, e.g., `transform(model_train, scale = "Sigma_1,1 := 1")`.

Via the `re` argument, we specify random effects for all but the price coefficient, which we again fix to `-1` for monetary interpretation:

```
> 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, scale = "pf := -1")
```

The `coef()` method returns the average effects and the estimated (marginal) variances:

```
> coef(model_elec)
```

```
        Estimate   (sd) Variance   (sd)
1   pf     -1.00 (0.00)       NA   (NA)
2   cl     -0.26 (0.03)     0.36 (0.06)
3  loc      2.88 (0.26)     7.20 (1.24)
4   wk      2.10 (0.21)     4.01 (0.75)
5  tod     -9.85 (0.24)    12.15 (2.01)
6 seas     -9.90 (0.19)     6.26 (0.95)
```

We deduce, for example, that a longer contract length has a negative effect on average (-0.26). However, our model shows that 33% of the customers still prefer to have a longer contract length. This share is estimated by computing the proportion under the mixing distribution that yields a positive coefficient for `cl`:

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

```
[1] 0.3316322
```

The estimated joint mixing distribution additionally allows to infer correlation patterns between effects. For example, we find a correlation of 0.79 between `loc` and `wk` (deciders that prefer local suppliers also prefer well known companies):

```
> round(cov_mix(model_elec, cor = TRUE), 2)
```

```
        cl   loc    wk   tod  seas
cl    1.00  0.09  0.06 -0.05 -0.11
loc   0.09  1.00  0.79  0.08 -0.01
wk    0.06  0.79  1.00  0.09 -0.03
tod  -0.05  0.08  0.09  1.00  0.55
seas -0.11 -0.01 -0.03  0.55  1.00
```

**Example 2: Simulated choices (cont.).** We previously simulated the ‘`RprobitB_data`’ object `data_sim` from a probit model with three latent classes. We now estimate the model parameters from the data generating process:

```
> model_sim <- fit_model(
+    data = data_sim, R = 1000, latent_classes = list("C" = 3), seed = 1
+  )
> summary(model_sim)

Probit model
Formula: choice ~ var1 | var2 | var3
R: 1000, B: 500, Q: 1
Level: Utility differences with respect to alternative 'alt2'.
Scale: Coefficient of the 1. error term variance fixed to 1.

Latent classes
C = 3

Gibbs sample statistics
          true    mean      sd      R^
 alpha

     1   -2.00   -2.00    0.13    1.21
     2    0.00   -0.04    0.03    1.10
     3    1.00    0.99    0.08    1.18

 s

     1    0.60    0.63    0.06    1.33
     2    0.30    0.27    0.06    1.10
     3    0.10    0.10    0.04    1.13

 b

   1.1   -2.00   -1.90    0.13    1.05
   1.2    1.00    0.93    0.23    1.70
   2.1    0.00    0.01    0.31    1.43
   2.2    2.00    2.04    0.36    1.40
   3.1    2.00    2.13    0.72    1.04
   3.2   -1.00   -0.84    0.83    1.06

 Omega

 1.1,1    0.30    0.61    0.22    1.82
 1.1,2    0.70    0.72    0.27    1.85
 1.2,2    1.90    2.11    0.52    1.68
 2.1,1    1.30    1.20    0.52    1.03
```

```
2.1,2   -0.20   -0.25    0.31    1.00
2.2,2    0.90    1.02    0.37    1.26
3.1,1    0.60    1.66    1.43    1.05
3.1,2   -0.90   -0.79    1.44    1.03
3.2,2    2.40    2.82    1.46    1.03

Sigma

  1,1    1.00    1.00    0.00    1.00
```

We note that more than $R = 1000$ iterations are required for proper convergence, as indicated by the Gelman-Rubin statistic.

## 4.3. Weight-based update of latent classes

Adding `"weight_update" = TRUE` to the list for the `latent_classes` argument of `fit_model()` executes the following weight-based updating scheme of the latent classes:

- Class $c$ is removed, if $s_c < \varepsilon_{\text{min}}$, i.e., if the class weight $s_c$ drops below some threshold $\varepsilon_{\text{min}}$. This case indicates that class $c$ has a negligible impact on the mixing distribution.
- Class $c$ is split 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 largest diagonal element of the class covariance matrix $\Omega_c$.
- Classes $c_1$ and $c_2$ are joined 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 the new class $c$ are assigned by adding the values of $s$ from $c_1$ and $c_2$ and averaging the values for $b$ and $\Omega$.

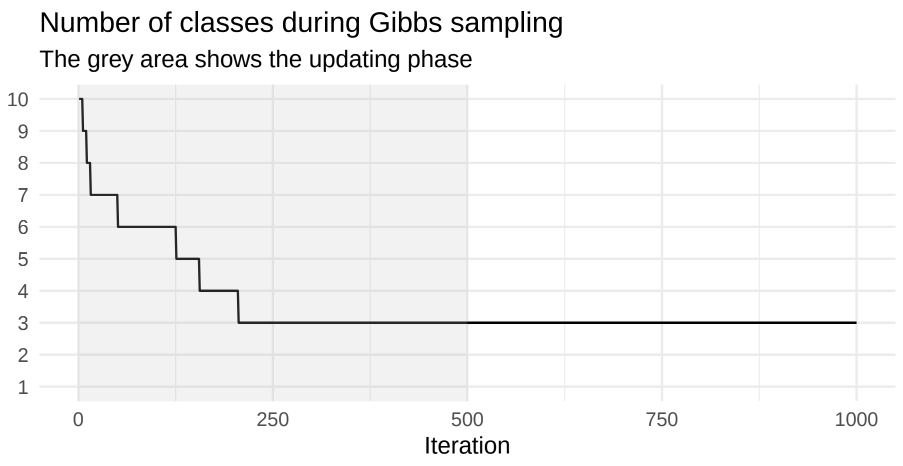
The values for $\varepsilon_{\text{min}}$, $\varepsilon_{\text{max}}$ and $\varepsilon_{\text{distmin}}$ can be specified via the `latent_classes` argument (by default `epsmin = 0.01`, `epsmax = 0.99`, and `distmin = 0.1`).

**Example 2: Simulated choices (cont.).** We apply the weight-based updating scheme to reproduce the true number $C = 3$ of latent classes. The scheme is initialized with `"C" = 10` classes and executed every `"buffer" = 5`-th iteration:

```
> model_sim <- fit_model(
+    data = data_sim, seed = 1,
+    latent_classes = list("C" = 10, "weight_update" = TRUE, "buffer" = 5)
+  )
```

The updating behavior can be visualized:

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



## 4.4. Dirichlet process-based update of latent classes

The Dirichlet process is a Bayesian nonparametric method that adds as many mixture components as needed for a good approximation. A priori, the mixture weights $(s_c)_c$ are Dirichlet distributed with concentration parameter $\delta/C$. Rasmussen (2000) shows that

$$\Pr(z \mid \delta) = \frac{\Gamma(\delta)}{\Gamma(N+\delta)} \prod_{c=1}^{C} \frac{\Gamma(m_c + \delta/C)}{\Gamma(\delta/C)}, \tag{5}$$

where $\Gamma(\cdot)$ denotes the gamma function. Crucially, Equation 5 is independent of the class weights $(s_c)_c$ (in contrast to the conditional posterior distribution stated in Section 4.1). From this equation, Li *et al.* (2019) shows that

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

for $C \to \infty$, and $z_{-n}$ denotes the vector $z$ without the $n$-th element. Now,

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

equals the probability that a new class for observation $n$ is created. This probability depends strictly monotonously increasing on the prior parameter $\delta$ (Neal 2000): a greater value for $\delta$ encourages the creation of new classes, smaller values increase the probability of an allocation to an already existing class. The total class number can theoretically rise to infinity but is practically bounded by $N$, since we delete unoccupied classes.

The Dirichlet process directly integrates to the existing Gibbs sampler: given $(\beta_n)_n$, we update the class means $b_c$ and covariance matrices $\Omega_c$ by means of their posterior predictive distribution. Mean vectors and covariance matrices for new classes are drawn from their prior predictive distribution, cf., Li *et al.* (2019).

**Example 4: Online chess strategy.** The **RprobitB** package contains revealed gambling preference data of chess players who participated in the "Yearly Bullet Arena 2022" on the platform www.lichess.org. The tournament had a special format: at the beginning of each game, both players can choose to trade half of their clock time[12] against the option to win an extra tournament point in case they win the game. The tournament lasted 4 hours, participants were paired again immediately after they finished a game, and the player with the most tournament points in the end won the event.

Several questions regarding the trade "clock time against a potential extra tournament point" (which the platform calls "berserking") arise: Do higher-rated chess players prefer to gamble? Does the remaining tournament time have an influence on the berserking choice? Can players be classified based on their revealed preferences to berserk? And if so, what are the class characteristics?

The `choice_berserk` data set provides the following information: whether a player berserked (`berserk = 1`, if yes), whether they had the `white` pieces (i.e., whether they made the first move, which is considered to be an advantage), their `rating` (a value provided by the platform, indicating the playing strength), the rating difference to the opponent (`rating_diff`), whether they lost the game (`lost = 1`, if yes), the remaining tournament time in minutes (`min_rem`), and whether they are currently on a winning `streak` (which gives extra points).

We additionally consider the lagged covariates `berserk.1` (the berserking choice in the previous game) and `lost.1` (the result of a player's previous game). They can be generated by the convenience function `create_lagged_cov()`:

```
> choice_berserk <- create_lagged_cov(
+    choice_data = choice_berserk,
+    column = c("berserk", "lost"),
+    id = "player_id"
+  )
```

We specify random effects for the rating difference and the result of the previous game:

```
> data <- prepare_data(
+    form = berserk ~ 0 | white + rating + rating_diff + min_rem + streak +
+      berserk.1 + lost.1 + 1,
+    re = c("rating_diff", "lost.1"),
+    choice_data = choice_berserk,
+    id = "player_id",
+    idc = "game_id",
+    standardize = c("rating", "rating_diff", "min_rem"),
+    impute = "zero"
+  )
```

[12]Players start each game with a time credit of one minute, which is consumend when it's their turn to make a move. A player whos time runs out looses the game automatically.

The Dirichlet process is applied for class updates:

```
> model_berserk <- fit_model(
+    data,
+    R = 5000,
+    latent_classes = list("dp_update" = TRUE, "C" = 10)
+  )
```

Estimating this model with $N = 6174$ deciders, $T = 1$ to 177 choice occasions and 126902 choices in total takes about 4 hours computation time. The estimated effects and class sizes are as follows:[13]

```
> coef(model_berserk)

                      Estimate   (sd) Variance   (sd)
1            white_0      0.04 (0.02)       NA   (NA)
2           rating_0      0.11 (0.01)       NA   (NA)
3          min_rem_0     -0.04 (0.01)       NA   (NA)
4           streak_0      0.27 (0.03)       NA   (NA)
5        berserk.1_0     -1.21 (0.02)       NA   (NA)
6              ASC_0      2.05 (0.03)       NA   (NA)
7  rating_diff_0 [1]     -0.11 (0.03)     0.07 (0.01)
8  rating_diff_0 [2]     -1.03 (0.05)     0.20 (0.06)
9  rating_diff_0 [3]     -1.60 (0.17)     1.71 (0.27)
10       lost.1_0 [1]     0.96 (0.14)     0.62 (0.14)
11       lost.1_0 [2]    -0.04 (0.05)     0.25 (0.06)
12       lost.1_0 [3]    -1.03 (0.20)     0.99 (0.23)

> est <- point_estimates(model_berserk)
> round(est[["s"]], 2)

   1    2    3
0.57 0.33 0.11
```

For interpretation of the model coefficients, remember that alternative `0` means denying the gambling opportunity. Overall, having the white pieces (i.e., starting the game), having a higher rating, and being on a streak discourages berserking (because the corresponding coefficients for "not berserking" are positive). This is as expected: starting the game with the white pieces and having the higher rating already gives an advantage which would be undermined by the berserking; berserking while being on a streak increases the risk of losing the streak benefits. In contrast, players more likely berserk towards the tournament end (which can be interpreted as the behavior to improve the standing as much as possible in the last minutes) and when they already berserked in the last round (continuity in behavior).

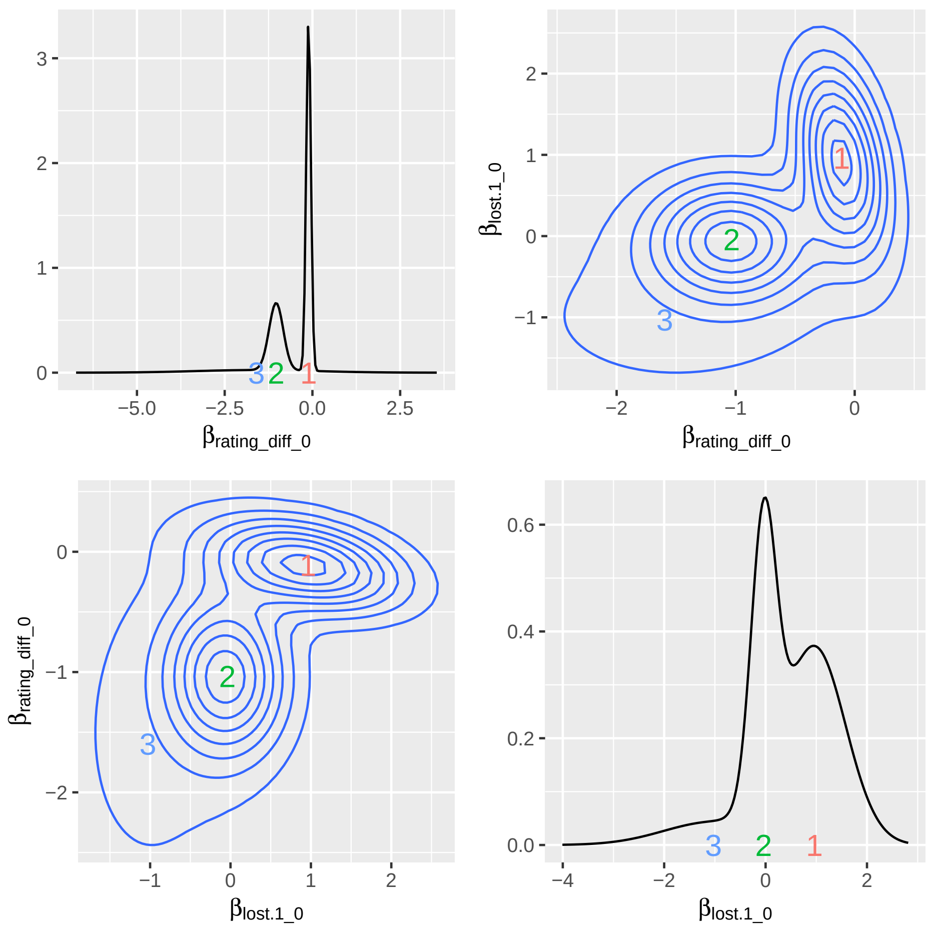
[13]Per default, point estimates are obtained as the posterior means. However, the `point_estimates()` function has the additional `FUN` argument for computing custom point estimates.

In terms of taste heterogeneity, we can characterize three types of players:

- Type one is rather risk averse: members don't often berserk against weaker opponents (`rating_diff_0 [1] = -0.1`) and most likely not when having lost in the previous round (`lost.1_0 [1] = 0.98`).
- Type two seems to make the decision independent from the result of the previous game (`lost.1_0 [2] = -0.03` with a notably small variance).
- Type three is willing to take risks: when they lost, they have an increased chance of berserking in the next round (`lost.1_0 [3] = -1.09`), and they like to berserk against weaker opponents (`rating_diff_0 [3] = -1.65`).

The different types can be visualized as follows (the numbers mark the class means):

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

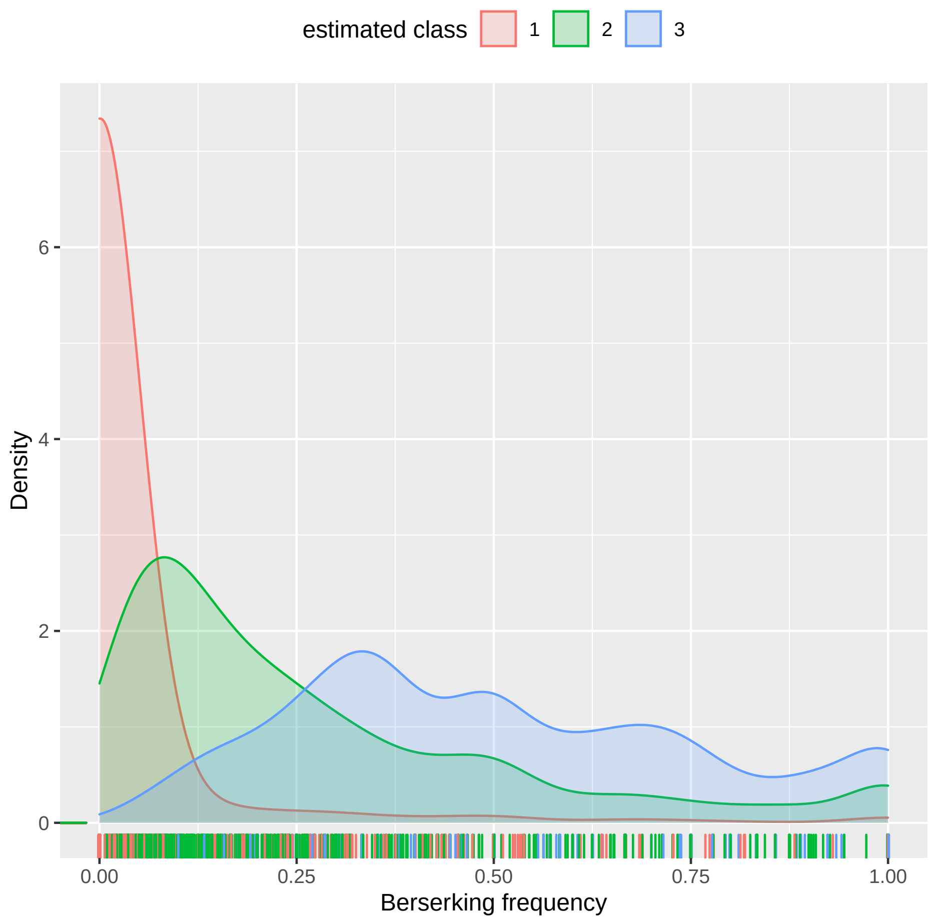


For each participant, we can compute probabilities for having followed each of the three strategies via the `classification()` function. The probabilities are computed as relative frequencies of the Gibbs samples for the class allocation variables `z`. We deduce that the winner of the tournament, Grandmaster Sergei Zhigalko (who berserked in 4 of his 113 games), and the unknown runner-up (who did not berserk at all) clearly followed different strategies:

```
> class <- classification(model_berserk)
> class[c("zhigalko_sergei", "serg_01"), ]
```

```
                     1      2      3 est
zhigalko_sergei 0.1844 0.7788 0.0368   2
serg_01         0.9380 0.0540 0.0080   1
```

Overall, the average berserking frequencies of the players in class 1 to 3 are 0.03, 0.28, and 0.51, respectively. The following plot visualizes kernel density estimates of the individual berserking frequencies grouped by the estimated classes. We see a strong overlap between the densities, which signals that the berserking frequency alone is not a sufficient indicator for identifying gambling preferences.



## 5. Choice prediction

**RprobitB** provides a `predict()` method for in-sample and out-of-sample prediction. The former case refers to reproducing the observed choices on the basis of the fitted model and subsequently using the deviations between prediction and reality as an indicator for the model performance. The latter means forecasting choice behavior for changes in the choice attributes. For illustration, we revisit our probit model of train travelers.

**Example 1: Train trips (cont.).** Per default, the `predict()` method returns a confusion matrix based on predicting the alternative with the largest choice probability, which gives an overview of the in-sample prediction performance:

```
> predict(model_train)
```

```
    predicted
true    A    B
   A 1033  441
   B  449 1006
```

The method can also return predictions on the level of individual choice occasions:[14]

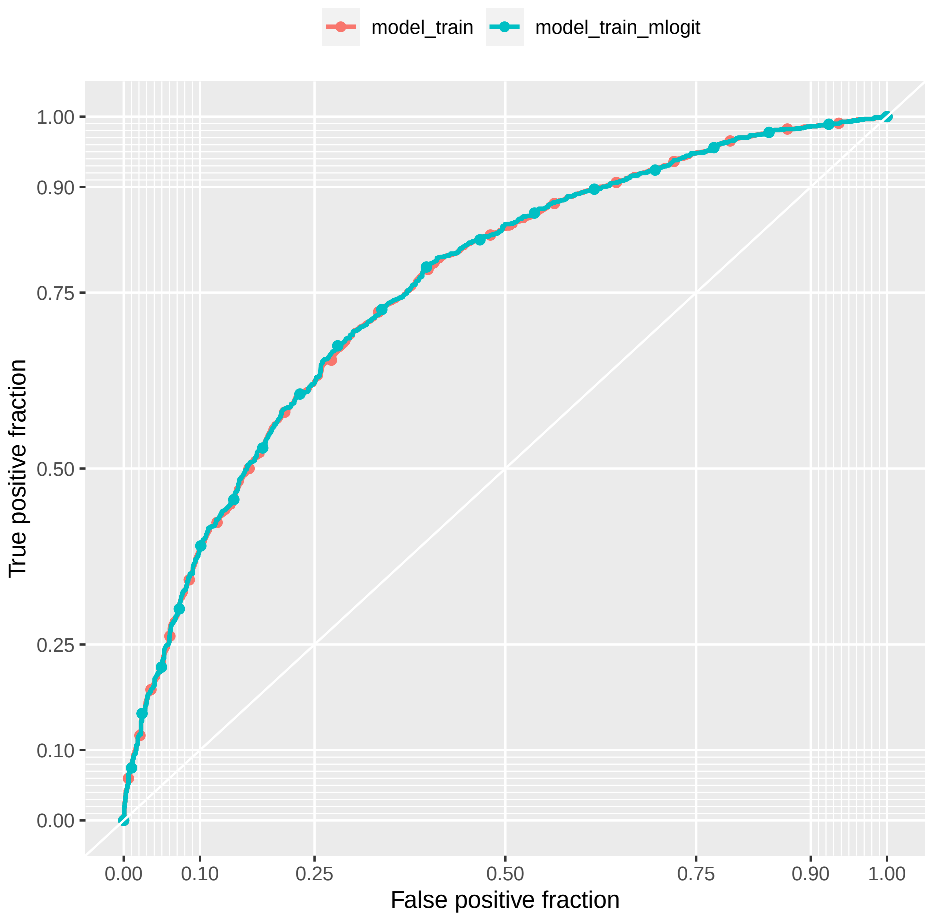
```
> pred <- predict(model_train, overview = FALSE)
> head(pred, n = 5)
```

```
  id choiceid    A    B true predicted correct
1  1        1 0.92 0.08    A         A    TRUE
2  1        2 0.64 0.36    A         A    TRUE
3  1        3 0.79 0.21    A         A    TRUE
4  1        4 0.18 0.82    B         B    TRUE
5  1        5 0.55 0.45    B         A   FALSE
```

The receiver operating characteristic (ROC) curve (Fawcett 2006) can be used to compare the prediction accuracy, for example between **RprobitB** and **mlogit**:

```
> library("mlogit")
> Train$choiceid <- 1:nrow(Train)
> Tr <- dfidx(
+    Train, choice = "choice", varying = 4:11, sep = "_",
+    opposite = c("price", "comfort", "time", "change"),
+    idx = list(c("choiceid", "id")), idnames = c("chid", "alt")
+  )
> Tr$price <- Tr$price / 100 * 2.20371
> Tr$time <- Tr$time / 60
> form <- choice ~ price + time + change + comfort | - 1
> model_train_mlogit <- mlogit(form, Tr)$probabilities
> plot_roc(model_train, model_train_mlogit)
```

[14]Incorrect predictions can be analyzed via the convenience function `get_cov()`, which extracts the characteristics of a particular choice situation.



The `predict()` method has an additional `data` argument for out-of-sample prediction. For example, assume that a train company wants to anticipate the effect of a price increase on their market share. By our model, increasing the ticket price from 100 to 110 euros (ceteris paribus) draws 15% of the customers to the competitor who does not increase their prices:

```
> predict(model_train, overview = FALSE,
+         data = data.frame("price_A" = c(100,110), "price_B" = c(100,100)))

  id choiceid    A    B prediction
1  1        1 0.50 0.50          A
2  2        1 0.35 0.65          B
```

However, a better comfort class compensates for the higher price and even results in a gain of 7% market share:

```
> predict(model_train, overview = FALSE,
+         data = data.frame("price_A" = c(100,110), "comfort_A" = c(1,0),
+                           "price_B" = c(100,100), "comfort_B" = c(1,1)))

  id choiceid    A    B prediction
1  1        1 0.50 0.50          A
2  2        1 0.57 0.43          A
```

## 6. Model selection

**RprobitB** provides several tools to identify the most appropriate model among competing ones, including the information criteria AIC (Akaike 1974), BIC (Schwarz 1978), and WAIC (Watanabe and Opper 2010), and the Bayes factor.

The WAIC is a Bayesian version of AIC and BIC and defined as $-2 \cdot \text{lppd} + 2 \cdot p_{\text{WAIC}}$, where $\text{lppd} = \sum_i \log\left(S^{-1} \sum_s p_{si}\right)$ is the log-pointwise predictive density, $p_{\text{WAIC}} = \sum_i \mathbb{V}_\theta \log(p_{si})$ a penalty term proportional to the variance in the posterior, and $p_{si} = \Pr(y_i \mid \theta_s)$ the probability of choice $y_i$ given the $s$-th set $\theta_s$ of parameter samples (McElreath 2020, p. 220).

The Bayes factor is an index of relative posterior model plausibility of one model over another (Marin and Robert 2014): given data $y$ and two models $M_1$ and $M_2$, it is defined as

$$BF(M_1, M_2) = \frac{\Pr(M_1 \mid y)}{\Pr(M_2 \mid y)} = \frac{\Pr(y \mid M_1)}{\Pr(y \mid M_2)} \bigg/ \frac{\Pr(M_1)}{\Pr(M_2)},$$

where by default $\Pr(M_1) = \Pr(M_2) = 0.5$. The value $\Pr(y \mid M)$ denotes the marginal model likelihood, which has no closed form and must be approximated numerically. **RprobitB** uses Gibbs samples to approximate the likelihood via the posterior harmonic mean estimator (Newton and Raftery 1994) in combination with the prior arithmetic mean estimator (Hammersley and Handscomb 1964). Both estimators converge with rising posterior samples to the marginal model likelihood by the law of large numbers. Convergence is fast if the prior and posterior distribution have a strong overlap (Gronau *et al.* 2017). **RprobitB** provides plotting methods for analyzing the convergence behavior, see `help(mml, package = "RprobitB")` for details.

**Example 1: Train trips (cont.).** As a competing model to `model_train`, we consider explaining the choices only by the alternative's price, i.e., the probit model with the formula `choice ~ price | 0`. The `update()` method helps in estimating such a nested model:

```
> model_train_sparse <- update(model_train, form = choice ~ price | 0)
```

**RprobitB** provides the convenience function `model_selection()`, which takes an arbitrary number of '`RprobitB_fit`' objects and returns a matrix of model selection criteria. The `criteria` input is a vector of `"npar"` (for the number of model parameters), `"LL"` (for the model's log-likelihood value, computed with the point estimates obtained from the Gibbs sample means), `"AIC"`, `"BIC"`, `"WAIC"`, `"MMLL"` (the marginal model log-likelihood), `"BF"` (for the Bayes factor), and `"pred_acc"` (the prediction accuracy). In order to compute WAIC, the marginal model likelihood, and the Bayes factor, the probabilities $p_{si} = \Pr(y_i \mid \theta_s)$ must be pre-computed:

```
> model_train <- compute_p_si(model_train)
> model_train_sparse <- compute_p_si(model_train_sparse)
> model_selection(
+    model_train, model_train_sparse,
+    criteria = c("npar", "LL", "AIC", "BIC", "WAIC", "MMLL", "BF", "pred_acc")
+  )
```

```
                         model_train model_train_sparse
npar                               4                  1
LL                          -1727.70           -1865.86
AIC                          3463.41            3733.73
BIC                          3487.34            3739.71
WAIC                         3464.17            3734.50
se(WAIC)                        0.19               0.09
pWAIC                           4.56               1.45
MMLL                        -1731.97           -1867.36
BF(*,model_train)                  1             < 0.01
BF(*,model_train_sparse)       > 100                  1
pred_acc                      69.61%             63.40%
```

The more complex model is clearly preferred.

## 7. Future developments

**RprobitB** implements Bayesian analysis of binary, multinomial, ordered, and ranked choices via the latent class mixed probit model. Working with the package follows a user-friendly workflow, from data management over model fitting to different types of model evaluations (Figure 1). This paper serves as a manual for R users who want to analyze choice behavior heterogeneity in their applications.

The inclusion of new package functionality is ongoing and can be followed on the project web page www.github.com/loelschlaeger. The following enhancements are currently under development, and we invite the community to suggest further features that we can implement in future package versions:

- Custom prior distributions — **RprobitB** currently only supports conjugate prior distributions due to their computational advantages (conjugate priors yield conditional posterior distributions in closed form which facilitates Gibbs sampling). An implementation with custom prior distributions would offer more flexibility. For example, prior distributions with strictly positive support enable sign restrictions that the user can impose on (some of) the (fixed) effects. However, custom prior distributions require a Metropolis–Hastings-type sampling scheme with increased risk of convergence problems and numerical cost.

- Imposing parameter constraints — **RprobitB** currently only offers parameter constraints by fixing for example the full vector of fixed effects or a full covariance matrix. In some applications, however, the researcher might want to fix only specific covariances, or impose equality restrictions between specific effects. We see a challenge in integrating such constraints in the Gibbs sampling scheme.

- Other types of mixing distributions — **RprobitB** currently is limited to the multivariate normal mixing distribution but offers approximation of any underlying mixing distribution by a mixture of Gaussian densities. Implementing other types of parametric mixing distributions (for example the log-normal or tent distribution) is possible by replacing the conditional posteriors for $\beta$, $b$, and $\Omega$, which is currently under development.

## Computational details

The results in this paper were obtained using R 4.2.1 with the **RprobitB** 1.1.0 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

Akaike H (1974). "A New Look at the Statistical Model Identification." *IEEE Transactions on Automatic Control*, **19**. `doi:10.1109/TAC.1974.1100705`.

Albert JH, Chib S (1993). "Bayesian Analysis of Binary and Polychotomous Response Data." *Journal of the American Statistical Association*, **88**. `doi:10.2307/2290350`.

Allenby GM, Rossi P (1998). "Marketing Models of Consumer Heterogeneity." *Journal of Econometrics*, **89**. `doi:10.1016/S0304-4076(98)00055-4`.

Bezanson J, Edelman A, Karpinski S, Shah VB (2017). "Julia: A Fresh Approach to Numerical Computing." *SIAM Review*, **59**(1). `doi:10.1137/141000671`.

Bhagat-Conway MW (2022). ***DiscreteChoiceModels.jl***. Julia Package Version 0.0.1, URL `https://github.com/mattwigway/DiscreteChoiceModels.jl`.

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**. `doi:10.1016/j.trb.2011.04.005`.

Bierlaire M (2020). "A Short Introduction to **PandasBiogeme**." *EPFL (Transport and Mobility Laboratory, ENAC)*.

Burda M, Harding M, Hausman J (2008). "A Bayesian Mixed Logit–Probit Model for Multinomial Choice." *Journal of Econometrics*, **147**(2). `doi:10.1016/j.jeconom.2008.09.029`.

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**. `doi:10.1016/j.tra.2005.06.007`.

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

Fawcett T (2006). "An Introduction to ROC Analysis." *Pattern Recognition Letters*, **27**(8). `doi:10.1016/j.patrec.2005.10.010`.

Gelman A, Rubin DB (1992). "Inference from Iterative Simulation Using Multiple Sequences." *Statistical Science*, **7**(4). `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." *Computing Science and Statistics*, **23**.

Gronau QF, Sarafoglou A, Matzke D, Ly A, Boehm U, Marsman M, Leslie DS, Forster JJ, Wagenmakers E, Steingroever H (2017). "A Tutorial on Bridge Sampling." *Journal of Mathematical Psychology*, **81**. doi:10.1016/j.jmp.2017.09.005.

Hammersley JM, Handscomb DC (1964). "General Principles of the Monte Carlo Method." *Springer Verlag*. doi:10.1007/978-94-009-5819-7_5.

Hess S, Palma D (2019). ***Apollo**: A Flexible, Powerful and Customisable Freeware Package for Choice Model Estimation and Application.* Choice Modelling Centre. R Package Version 0.2.7, URL http://www.ApolloChoiceModelling.com.

Hewig J, Kretschmer N, Trippe RH, Hecht H, Coles MGH, Holroyd CB, Miltner WHR (2011). "Why Humans Deviate from Rational Choice." *Psychophysiology*, **48**(4). doi:10.1111/j.1469-8986.2010.01081.x.

Imai K, van Dyk D (2022). ***MNP**: Fitting the Multinomial Probit Model.* R Package Version 3.1-3, URL https://CRAN.R-project.org/package=MNP.

Imai K, van Dyk DA (2005). "A Bayesian Analysis of the Multinomial Probit Model Using Marginal Data Augmentation." *Journal of Econometrics*.

Li Y, Schofield E, Gönen M (2019). "A Tutorial on Dirichlet Process Mixture Modeling." *Journal of Mathematical Psychology*, **91**. doi:10.1016/j.jmp.2019.04.004.

Marin J, Robert C (2014). *Bayesian Essentials with R.* Springer Verlag. doi:10.1007/978-1-4614-8687-9.

McCulloch R, Rossi P (1994). "An Exact Likelihood Analysis of the Multinomial Probit Model." *Journal of Econometrics*, **64**. doi:10.1016/0304-4076(94)90064-7.

McElreath R (2020). *Statistical Rethinking: A Bayesian Course with Examples in R and Stan.* Chapman and Hall/CRC. doi:10.1201/9780429029608.

Neal RM (2000). "Markov Chain Sampling Methods for Dirichlet Process Mixture Models." *Journal of Computational and Graphical Statistics*, **9**(2). doi:10.2307/1390653.

Newton MA, Raftery AE (1994). "Approximate Bayesian Inference with the Weighted Likelihood Bootstrap." *Journal of the Royal Statistical Society: Series B (Methodological)*, **56**(1).

Nobile A (1998). "A Hybrid Markov Chain for the Bayesian Analysis of the Multinomial Probit Model." *Statistics and Computing*, **8**. doi:10.1023/A:1008905311214.

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

Oelschläger L, Bauer D (2022). ***RprobitB**: Bayesian Probit Choice Modeling.* R Package Version 1.1.0, URL https://CRAN.R-project.org/package=RprobitB.

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

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Rossi P (2019). ***bayesm**: Bayesian Inference for Marketing/Micro-Econometrics*. R Package Version 3.1-4, URL https://CRAN.R-project.org/package=bayesm.

Sarrias M (2016). "Discrete Choice Models with Random Parameters in R: The **Rchoice** Package." *Journal of Statistical Software*, **74**(10). doi:10.18637/jss.v074.i10.

Sarrias M, Daziano R (2017). "Multinomial Logit Models with Continuous and Discrete Individual Heterogeneity in R: The **gmnl** Package." *Journal of Statistical Software*, **79**(2). doi:10.18637/jss.v079.i02.

Schwarz G (1978). "Estimating the Dimension of a Model." *The Annals of Statistics*, **6**.

Thaler RH, Ganser L (2015). *Misbehaving: The Making of Behavioral Economics*. WW Norton New York. doi:10.1080/00332747.2016.1222217.

Train K (2001). "A Comparison of Hierarchical Bayes and Maximum Simulated Likelihood for Mixed Logit." *University of California, Berkeley*.

Train K (2009). *Discrete Choice Methods with Simulation*. Cambridge University Press. doi:10.1017/CBO9780511805271.

Tversky A, Kahneman D (1986). "Rational Choice and the Framing of Decisions." *The Journal of Business*, **59**(4).

Van Rossum G, Drake Jr FL (1995). *Python Tutorial*. Centrum voor Wiskunde en Informatica Amsterdam, The Netherlands. URL http://www.python.org.

Watanabe S, Opper M (2010). "Asymptotic Equivalence of Bayes Cross Validation and Widely Applicable Information Criterion in Singular Learning Theory." *Journal of Machine Learning Research*, **11**(12).

**Affiliation:**

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