BayesMallows: An R Package for the Bayesian Mallows Model

by Øystein Sørensen, Marta Crispino, Qinghua Liu, and Valeria Vitelli

Abstract **BayesMallows** is an R package for analyzing preference data in the form of rankings with the Mallows rank model, and its finite mixture extension, in a Bayesian framework. The model is grounded on the idea that the probability density of an observed ranking decreases exponentially with the distance to the location parameter. It is the first Bayesian implementation that allows wide choices of distances, and it works well with a large amount of items to be ranked. **BayesMallows** handles non-standard data: partial rankings and pairwise comparisons, even in cases including non-transitive preference patterns. The Bayesian paradigm allows coherent quantification of posterior uncertainties of estimates of any quantity of interest. These posteriors are fully available to the user, and the package comes with convienient tools for summarizing and visualizing the posterior distributions.

Introduction

Preference data are usually collected when individuals are asked to rank a set of items according to a certain preference criterion. The booming of internet-related activities and applications in recent years has led to a rapid increase of the amount of data that have ranks as their natural scale, however often in the form of partial or indirect rankings (pairwise preferences, ratings, clicks). The amount of readily available software handling preference data has consequently increased consistently in the last decade or so, but not many packages are flexible enough to handle all types of data mentioned above. The typical tasks when analyzing preference data are rank aggregation, classification or clustering, and prediction, where the latter task refers to the estimation of the individual rankings of the assessors when completely or partially missing. These tasks can be addressed either via model-based inference or via heuristic machine learning algorithms, with or without uncertainty quantification. However, very few methods allow handling diverse data types, combining several inferential tasks with proper propagation of the uncertainties, while also providing individualized predictions. Our proposal goes exactly in this direction, thus making the scopes of BayesMallows broad when it comes to data handling and individual-level inference.

The R package **BayesMallows** is the first software conceived to answer the needs mentioned above in a unified framework: it implements full Bayesian inference for ranking data, and performs all of the tasks above in the framework of the Bayesian Mallows model (BMM) (Mallows, 1957; Vitelli et al., 2018). More specifically, **BayesMallows** allows for data in the forms of complete rankings, partial rankings, as well as pairwise comparisons, including the case where some comparisons are inconsistent. In these situations, it provides all Bayesian inferential tools for rank modeling with the BMM: it performs rank aggregation (estimation of a consensus ranking of the items), it can cluster together the assessors providing similar preferences (estimating both cluster specific model parameters, and individual cluster assignments, with uncertainty), it performs data augmentation for estimating the latent assessor-specific full ranking of the items in all missing data situations (partial rankings, pairwise preferences). The latter in particular, i.e., the possibility of predicting individual preferences for unranked items, enables the model to be used as a probabilistic recommender system. **BayesMallows** also enlarges the pool of distances that can be used in the Mallows model, and it supports the rank distances most used in the literature: Spearman's footrule (henceforth footrule), Spearman's rank correlation (henceforth Spearman), Cayley, Hamming, Kendall, and Ulam distances (we refer to Diaconis, 1988; Marden, 1995, for details on these). Finally, **BayesMallows** implements the Iterative Proportional Fitting Procedure (IPFP) algorithm for computing the partition function for the Mallows model (MM) (Mukherjee, 2016) and the importance sampling algorithm described in Vitelli et al. (2018). In addition to being used in the MM, these functions may be of interest in their own right.

Comparing with other available inferential software, we notice that not many packages allow for such flexibility, very few in combination with full Bayesian inference, and none when using the MM as outlined in Section 4 below. Often machine learning approaches focus on either rank aggregation (i.e., consensus estimation), or individual rank prediction, while **BayesMallows** handles both. Since the BMM is fully Bayesian, all posterior quantities of interest are automatically available from **BayesMallows** for the first time for the MM. In addition, the package also has tools for visualizing posterior distributions, and hence, posterior quantities as well as their associated uncertainties. Uncertainty quantification is often critical in real applications: for recommender systems, the model should not spam the users with very uncertain recommendations; when performing subtype identification for cancer patients, a very uncertain cluster assignment might have serious consequences for the clinical practice, for example in treatment choice. The package also works well with a fairly large number of items, thanks to computational approximations and efficient programming. In conclusion, **BayesMallows** provides the first fully probabilistic inferential tool for the MM with many different distances. It is flexible in the types of data it handles, and computationally efficient. We therefore think that this package will gain popularity, and prove its usefulness in many practical situations, many of which we probably cannot foresee now.

The paper is organized as follows. The BMM for ranking data is briefly reviewed in Section 2, as well as its model extensions to different data types and to mixtures. Section 3 includes details on the implementation of the inferential procedure. For a thorough discussion of both the model and its implementation we refer interested readers to Vitelli et al. (2018). An overview of existing R packages implementing the Mallows model (MM) is given in Section 4. The use of the **BayesMallows** package is presented, in the form of three case studies, in Sections 5, 6, and 7. Section 8 concludes the paper, also discussing model extensions that will come with new releases of the package.

Background: the Bayesian Mallows model for rankings

In this section we give the background for understanding the functions in the **BayesMallows** package. More details can be found in Vitelli et al. (2018) and Liu et al. (2019). The section is organized as follows: we first clarify the notations that we will use throughout the paper (Section 2.1). We then briefly describe the BMM for complete ranking data (Section 2.2), also focusing on the relevance of the choice of distance (Section 2.3). The last two sections focus on model extensions: partial and pairwise data (Section 2.4), non-transitive pairwise comparisons (Section 2.5), and mixtures (Section 2.6).

Notation

Let us denote with $\mathcal{A} = \{A_1, ..., A_n\}$ the finite set of labeled items to be ranked, and with $\mathcal{P}_n$ the space of n-dimensional permutations. A complete ranking of n items is then a mapping $\boldsymbol{R} : \mathcal{A} \to \mathcal{P}_n$ that attributes a rank $R_i \in \{1, ..., n\}$ to each item, according to some criterion. We here denote a generic complete ranking by $\boldsymbol{R} = (R_1, ..., R_n)$, where R_i is the rank assigned to item A_i. Given a pair of items $\{A_i, A_k\}$, we denote a pairwise preference between them as $(A_i \prec A_k)$, meaning that item A_i is preferred to item A_k. Note the intimate relation that exists between a ranking and pairwise preferences. Given a full ranking $\boldsymbol{R} \in \mathcal{P}_n$, it is immediate to evince all the possible $n(n-1)/2$ pairwise preferences between the items taken in pairs, since the item in the pair having the lower rank is the preferred one:

$$(A_{t_1} \prec A_{t_2}) \iff R_{t_1} < R_{t_2}, \qquad t_1, t_2 = 1, ..., n, \quad t_1 \neq t_2.$$

Conversely, obtaining a full ranking from a set of pairwise preferences is not straightforward. Pairwise preference data are typically incomplete, meaning that not all pairwise preferences necessary to determine each individual ranking are present. They can contain non-transitive patterns, that is, one or more pairwise preferences contradict what is implied by other pairwise preferences. In this package we can handle partial and possibly non-transitive pairwise preferences.

The BMM for Complete Rankings

The MM for ranking data (Mallows, 1957) specifies the probability density for a ranking $\mathbf{r} \in \mathcal{P}_n$ as follows

$$P(\mathbf{r}|\alpha, \boldsymbol{\rho}) = \frac{1}{Z_n(\alpha)} \exp\left[-\frac{\alpha}{n} d(\mathbf{r}, \boldsymbol{\rho})\right] 1_{\mathcal{P}_n}(\mathbf{r}) \tag{1}$$

where $\boldsymbol{\rho} \in \mathcal{P}_n$ is the location parameter representing the consensus ranking, $\alpha \geq 0$ is the scale parameter (precision), $Z_n(\alpha)$ is the normalizing function (or partition function), $d(\cdot, \cdot)$ is a right-invariant distance among rankings (Diaconis, 1988), and $1_{\mathcal{P}_n}(\mathbf{r})$ is an indicator function for the set $\mathcal{P}_n$ which equals one when $\mathbf{r} \in \mathcal{P}_n$ and zero otherwise.

In the complete data case, N assessors have provided complete rankings of the n items in $\mathcal{A}$ according to some criterion, yielding the permutation $\mathbf{R}_j = \left(R_{1j}, \ldots, R_{nj}\right)$ for assessor j, $j = 1, \ldots, N$. The likelihood of the N observed rankings $\mathbf{R}_1, \ldots, \mathbf{R}_N$, assumed conditionally independent given α and $\boldsymbol{\rho}$, is

$$P(\mathbf{R}_1, \ldots, \mathbf{R}_N|\alpha, \boldsymbol{\rho}) = \frac{1}{Z_n(\alpha)^N} \exp\left[-\frac{\alpha}{n} \sum_{j=1}^{N} d\left(\mathbf{R}_j, \boldsymbol{\rho}\right)\right] \prod_{j=1}^{N} 1_{\mathcal{P}_n}\left(\mathbf{R}_j\right). \tag{2}$$

According to the BMM introduced in Vitelli et al. (2018), prior distributions have to be elicited on every parameter of interest. A truncated exponential prior distribution was specified for α

$$\pi\left(\alpha|\lambda\right)=\frac{\lambda\exp\left(-\lambda\alpha\right)1_{[0,\alpha_{\max}]}\left(\alpha\right)}{1-\exp\left(-\lambda\alpha_{\max}\right)}, \tag{3}$$

where λ is a rate parameter, small enough to ensure good prior dispersion, and $\alpha_{\max}$ is a cutoff, large enough to cover reasonable α values. A uniform prior $\pi\left(\boldsymbol{\rho}\right)$ on $\mathcal{P}_n$ was assumed for $\boldsymbol{\rho}$. It follows that the posterior distribution for α and $\boldsymbol{\rho}$ is

$$P\left(\alpha,\boldsymbol{\rho}|\mathbf{R}_1,\ldots,\mathbf{R}_N\right)\propto\frac{1_{\mathcal{P}_n}\left(\boldsymbol{\rho}\right)}{Z_n\left(\alpha\right)^N}\exp\left[-\frac{\alpha}{n}\sum_{j=1}^{N}d\left(\mathbf{R}_j,\boldsymbol{\rho}\right)-\lambda\alpha\right]1_{[0,\alpha_{\max}]}\left(\alpha\right). \tag{4}$$

Inference on the model parameters is based on a Metropolis-Hastings (M-H) Markov Chain Monte Carlo (MCMC) algorithm, described in Vitelli et al. (2018). Some details relevant for a correct use of this package are also given in Section 3.1.

Distance measures and partition function

The partition function $Z_n\left(\alpha\right)$ in (1), (2) and (4) does not depend on the latent consensus ranking $\boldsymbol{\rho}$ when the distance $d\left(\cdot,\cdot\right)$ is right-invariant, meaning that it is unaffected by a relabelling of the items (Diaconis, 1988). Right-invariant distances play an important role in the MM, and **BayesMallows** only handles right-invariant distances. The choice of distance affects the model fit to the data and the results of the analysis, and is crucial also because of its role in the partition function computation. Some right-invariant distances allow for analytical computation of the partition function, and for this reason they have become quite popular. In particular, the MM with Kendall (Meilă and Chen, 2010; Lu and Boutilier, 2014), Hamming (Irurozki et al., 2014) and Cayley (Irurozki et al., 2016b) distances have a closed form of $Z_n\left(\alpha\right)$ due to Fligner and Verducci (1986). There are however important and natural right-invariant distances for which the computation of the partition function is NP-hard, in particular the footrule (l_1) and the Spearman (l_2) distances. For precise definitions of all distances involved in the MM we refer to Marden (1995). **BayesMallows** handles the footrule, Spearman, Cayley, Hamming, Kendall, and Ulam distances.

Partial rankings and transitive pairwise comparisons

When complete rankings of all items are not readily available, the BMM can still be used by applying data augmentation techniques. Partial rankings can occur because ranks are missing at random, because the assessors have only ranked their top-k items, or because they have been presented with a subset of items. In more complex situations, data do not include ranks at all, but the assessors have only compared pairs of items and given a preference between the two. The Bayesian data augmentation scheme can still be used to handle such pairwise comparison data, thus providing a solution that is fully integrated into the Bayesian inferential framework. The following paragraphs provide a summary of Sections 4.1 and 4.2 of Vitelli et al. (2018), which we refer to for details.

Let us start by considering the case of top-k rankings. Suppose that each assessor j has chosen a set of preferred items $\mathcal{A}_j\subseteq\mathcal{A}$, which were given ranks from 1 to $n_j=|\mathcal{A}_j|$. Now $R_{ij}\in\{1,\ldots,n_j\}$ if $A_i\in\mathcal{A}_j$, while for $A_i\in\mathcal{A}_j^c$, R_{ij} is unknown, except for the constraint $R_{ij}>n_j$, $j=1,\ldots,N$. The augmented data vectors $\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_N$ are introduced in the model to include the missing ranks, which are estimated as latent parameters. Let $\mathcal{S}_j=\{\tilde{\mathbf{R}}_j\in\mathcal{P}_n:\tilde{R}_{ij}=R_{ij}\text{ if }A_i\in\mathcal{A}_j\}$, $j=1,\ldots,N$ be the set of possible augmented random vectors, including the ranks of the observed top-n_j items together with the unobserved ranks, which are assigned a uniform prior on the permutations of $\{n_j+1,\ldots,n\}$. The goal is to sample from the posterior distribution

$$P\left(\alpha,\boldsymbol{\rho}|\mathbf{R}_1,\ldots,\mathbf{R}_N\right)=\sum_{\tilde{\mathbf{R}}_1\in\mathcal{S}_1}\cdots\sum_{\tilde{\mathbf{R}}_N\in\mathcal{S}_N}P\left(\alpha,\boldsymbol{\rho},\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_N|\mathbf{R}_1,\ldots,\mathbf{R}_N\right). \tag{5}$$

The augmentation scheme amounts to alternating between sampling α and $\boldsymbol{\rho}$ given the current values of the augmented ranks using the posterior given in (4), and sampling the augmented ranks given the current values of α and $\boldsymbol{\rho}$. For the latter task, once α, $\boldsymbol{\rho}$ and the observed ranks $\mathbf{R}_1,\ldots,\mathbf{R}_N$ are fixed, one can see that $\tilde{\mathbf{R}}_1,\ldots,\tilde{\mathbf{R}}_N$ are conditionally independent, and that each $\tilde{\mathbf{R}}_j$ only depends on the corresponding $\mathbf{R}_j$. As a consequence, the update of new augmented vectors is performed independently, for each $j=1,\ldots,N$.

The above procedure can also handle more general situations where missing rankings are not necessarily the bottom ones, and where each assessor is asked to provide the mutual ranking of some possibly random subset $\mathcal{A}_j \subseteq \mathcal{A}$ consisting of $n_j \leq n$ items. Note that the only difference from the previous formulation is that each latent rank vector $\tilde{\mathbf{R}}_j$ takes values in the set

$$\mathcal{S}_j = \left\{ \tilde{\mathbf{R}}_j \in \mathcal{P}_n : \left(R_{i_1 j} < R_{i_2 j} \right) \wedge \left(A_{i_1}, A_{i_2} \in \mathcal{A}_j \right) \Rightarrow \tilde{R}_{i_1 j} < \tilde{R}_{i_2 j} \right\}.$$

Also in this case the prior for $\tilde{\mathbf{R}}_j$ is assumed uniform on $\mathcal{S}_j$.

In the case of pairwise comparison data, let us call $\mathcal{B}_j$ the set of all pairwise preferences stated by assessor j, and let $\mathcal{A}_j$ be the set of items appearing at least once in $\mathcal{B}_j$. Note that the items in $\mathcal{A}_j$ are not necessarily fixed to a given rank but may only be given some partial ordering. For the time being, we assume that the observed pairwise orderings in $\mathcal{B}_j$ are transitive, i.e., mutually compatible, and define by $\text{tc}\left(\mathcal{B}_j\right)$ the transitive closure of $\mathcal{B}_j$, which contains all pairwise orderings of the elements in $\mathcal{A}_j$ induced by $\mathcal{B}_j$. The model formulation remains the same as in the case of partial rankings, with the prior for the augmented data vectors $\tilde{\mathbf{R}}_1, \ldots, \tilde{\mathbf{R}}_N$ being uniform on the set $\mathcal{S}_j$ of rankings that are compatible with the observed data.

Non-transitive pairwise comparisons

It can happen in real applications that individual pairwise comparison data are non-transitive, that is, they may contain a pattern of the form $x \prec y,\ y \prec z$ but $z \prec x$. This is typically the case of data collected from internet user activities, when the pool of items is very large: non-transitive patterns can arise for instance due to assessors' inattentiveness, uncertainty in their preferences, and actual confusion, even when one specific criterion for ranking is used. Another frequent situation is when the number of items is not very large, but the items are perceived as very similar by the assessors. This setting is discussed in Crispino et al. (2019), where the model for transitive pairwise comparisons of Section 2.4 is generalized to handle situations where non-transitivities in the data occur. Note that the kind of non-transitivity that is considered in Crispino et al. (2019) considers only the individual level preferences. A different type of non-transitivity, which we do not consider here, arises when aggregating preferences across assessors, as under Condorcet (Marquis of Condorcet, 1785) or Borda (de Borda, 1781) voting rules.

The key ingredient of this generalization consists of adding one layer of latent variables to the model hierarchy, accounting for the fact that assessors can make mistakes. The main assumption is that the assessor makes pairwise comparisons based on her latent full rankings $\tilde{\boldsymbol{R}}$. A mistake is defined as an inconsistency between one of the assessor's pairwise comparisons and $\tilde{\boldsymbol{R}}$. Suppose each assessor $j = 1, \ldots, N$ has assessed M_j pairwise comparisons, collected in the set $\mathcal{B}_j$, and assume the existence of latent ranking vectors $\tilde{\boldsymbol{R}}_j, j = 1, \ldots, N$. Differently from Section 2.4, since $\mathcal{B}_j$ is allowed to contain non-transitive pairwise preferences, the transitive closure of $\mathcal{B}_j$ is not defined, and the posterior density (5) cannot be evaluated. In this case, the posterior takes the form,

$$\begin{aligned} P\left(\alpha, \boldsymbol{\rho} | \mathcal{B}_1, ..., \mathcal{B}_N\right) &= \sum_{\tilde{\boldsymbol{R}}_1 \in \mathcal{P}_n} \cdots \sum_{\tilde{\boldsymbol{R}}_N \in \mathcal{P}_n} P\left(\alpha, \boldsymbol{\rho}, \tilde{\boldsymbol{R}}_1, ..., \tilde{\boldsymbol{R}}_N | \mathcal{B}_1, ..., \mathcal{B}_N\right) = \\ &= \sum_{\tilde{\boldsymbol{R}}_1 \in \mathcal{P}_n} \cdots \sum_{\tilde{\boldsymbol{R}}_N \in \mathcal{P}_n} P(\alpha, \boldsymbol{\rho} | \tilde{\boldsymbol{R}}_1, ..., \tilde{\boldsymbol{R}}_N) P\left(\tilde{\boldsymbol{R}}_1, ..., \tilde{\boldsymbol{R}}_N | \mathcal{B}_1, ..., \mathcal{B}_N\right) \end{aligned} \tag{6}$$

where the term $P\left(\tilde{\boldsymbol{R}}_1, ..., \tilde{\boldsymbol{R}}_N | \mathcal{B}_1, ..., \mathcal{B}_N\right)$ models the presence of mistakes in the data, while in the case of transitive pair comparisons it was implicitly assumed equal to 1 if each augmented ranking $\tilde{\boldsymbol{R}}_j$ was compatible with the partial information contained in $\mathcal{B}_j$, and 0 otherwise.

Two models for (6) are considered in Crispino et al. (2019): the Bernoulli model, which accounts for random mistakes, and the Logistic model, which lets the probability of making a mistake depend on the similarity of the items being compared. The Bernoulli model states that:

$$P\left(\tilde{\boldsymbol{R}}_1, ..., \tilde{\boldsymbol{R}}_N | \theta, \mathcal{B}_1, ..., \mathcal{B}_N\right) \propto \theta^M (1-\theta)^{\sum_j M_j - M}, \quad \theta \in [0, 0.5) \tag{7}$$

where M counts the number of times the observed preferences contradict what is implied by the ranking $\tilde{\boldsymbol{R}}_j$, M_j is the number of pairwise comparisons reported by assessor j, and the parameter θ is the probability of making a mistake in a single pairwise preference. θ is *a priori* assigned a truncated Beta distribution on the interval $[0, 0.5)$ with given hyper-parameters κ_1 and κ_2, conjugate to the Bernoulli model (7). The Logistic model is a generalization of (7) where, instead of assigning a constant value θ to the probability of making a mistake, it depends on the distance between the ranks of the two items under comparison. In Crispino et al. (2019) the Logistic model gave results very similar to the Bernoulli model, and currently only the Bernoulli model is available in **BayesMallows**. The sampling scheme is similar to the one used for the case of transitive pairwise preferences, apart from an additional step for updating θ, and the augmentation scheme for $\tilde{\boldsymbol{R}}_j$, which is slightly different. We refer to Crispino et al. (2019) for details.

Clustering

The assumption, implicit in the discussion so far, that there exists a unique consensus ranking shared by all assessors is unrealistic in most real applications: the BMM thus handles the case where the rankings provided by all assessors can be modeled as a finite mixture of MMs. In the following brief discussion we assume that the data consist of complete rankings, but **BayesMallows** can fit a mixture based on any kinds of preference data described so far. See Section 4.3 of Vitelli et al. (2018) for details.

Let $z_1, \ldots, z_N \in \{1, \ldots, C\}$ assign each assessor to one of C clusters, and let the rankings within each cluster $c \in \{1, \ldots, C\}$ be described by an MM with parameters α_c and $\boldsymbol{\rho}_c$. The likelihood of the observed rankings $\mathbf{R}_1, \ldots, \mathbf{R}_N$ is given by

$$P\left(\mathbf{R}_1, \ldots, \mathbf{R}_N \middle| \{\boldsymbol{\rho}_c, \alpha_c\}_{c=1,\ldots,C}, \left\{z_j\right\}_{j=1,\ldots,N}\right) = \prod_{j=1}^{N} \frac{1_{\mathcal{P}_n}\left(\mathbf{R}_j\right)}{Z_n\left(\alpha_{z_j}\right)} \exp\left[-\frac{\alpha_{z_j}}{n} d\left(\mathbf{R}_j, \boldsymbol{\rho}_{z_j}\right)\right],$$

where conditional independence is assumed across the clusters. We also assume independent truncated exponential priors for the scale parameters and independent uniform priors for the consensus rankings. The cluster labels $z_1, \ldots, z_N$ are a priori assumed conditionally independent given the clusters mixing parameters $\tau_1, ..., \tau_C$, and are assigned a uniform multinomial. Finally $\tau_1, \ldots, \tau_C$ (with $\tau_c \geq 0$, $c = 1, \ldots, C$ and $\sum_{c=1}^{C} \tau_c = 1$) are assigned the standard symmetric Dirichlet prior of parameter Ψ, thus implying a conjugate scheme. The posterior density is then given by

$$P\left(\{\boldsymbol{\rho}_c, \alpha_c, \tau_c\}_{c=1}^{C}, \left\{z_j\right\}_{j=1}^{N} \middle| \mathbf{R}_1, \ldots, \mathbf{R}_N\right) \propto \left[\prod_{c=1}^{C} e^{-\lambda \alpha_c} \tau_c^{\Psi - 1}\right] \left[\prod_{j=1}^{N} \frac{\tau_{z_j} e^{-\frac{\alpha_{z_j}}{n} d\left(\mathbf{R}_j, \boldsymbol{\rho}_{z_j}\right)}}{Z_n\left(\alpha_{z_j}\right)}\right]. \quad (8)$$

Computational considerations

In this section we briefly give some additional details regarding the implementation of the models described in Section 2. The BMM implementation is thoroughly described in Vitelli et al. (2018).

Details on the MCMC procedures

In order to obtain samples from the posterior density of equation (4), **BayesMallows** implements an MCMC scheme iterating between (i) updating $\boldsymbol{\rho}$ and (ii) updating α (Algorithm 1 of Vitelli et al., 2018). The leap-and-shift proposal distribution, which is basically a random local perturbation of width L of a given ranking, is used for updating $\boldsymbol{\rho}$ in step (i). The L parameter of the leap-and-shift proposal controls how far the proposed ranking is from the current one, and it is therefore linked to the acceptance rate. The recommendation given in Vitelli et al. (2018) is to set it to $L = n/5$, which is also the default value in **BayesMallows**, but the user is allowed to choose a different value. For updating α in step (ii), a log-normal density is used as proposal, and its variance σ_α^2 can be tuned to obtain a desired acceptance rate.

As mentioned in Section 2.4, the MCMC procedure for sampling from the posterior densities corresponding to the partial data cases (Algorithm 3 of Vitelli et al., 2018) has an additional M-H step to account for the update of the augmented data rankings $\left\{\tilde{\mathbf{R}}_j\right\}_{j=1}^{N}$. In the case of partial rankings, for updating the augmented data $\tilde{\mathbf{R}}_j$, $j = 1, ..., N$ we use a uniform proposal on the set of rankings compatible with the partial data, $\mathcal{S}_j$. In the case of pairwise preferences, due to the increased sparsity in the data, we instead implemented a modified parameter-free leap-and-shift proposal distribution, which proposes a new augmented ranking by locally permuting the ranks in $\tilde{\mathbf{R}}_j$ within the constraints given by $\mathcal{B}_j$ (Vitelli et al., 2018, Section 4.2). The generalization to non-transitive pairwise comparisons, outlined in Section 4 of Crispino et al. (2019), requires further considerations. First, in the M-H step for updating the augmented data rankings, the modified parameter-free leap-and-shift proposal has to be replaced by a Swap proposal, whose tuning parameter $L^\star$ is the maximum allowed distance between the ranks of the swapped items. Second, the Bernoulli model for mistakes makes it necessary to add a a Gibbs step for the update of θ. The MCMC algorithm for sampling from the mixture model posterior (8) (Algorithm 2 of Vitelli et al., 2018) alternates between updating $\{\boldsymbol{\rho}_c, \alpha_c\}_{c=1}^{C}$ in an M-H step, and $\left\{\tau_c, z_j\right\}_{c=1,j=1}^{C,N}$ in a Gibbs sampling step, in addition to the necessary M-H steps for data

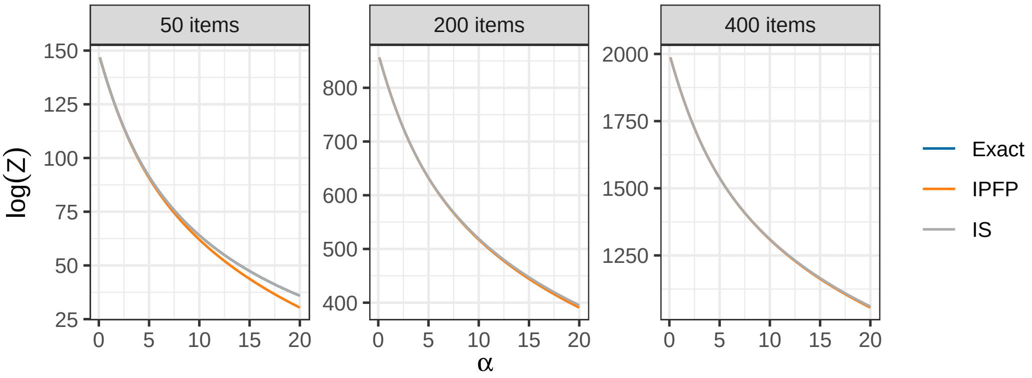


Figure 1: Estimates of the log partition function for the BMM with footrule distance computed using exact calculation (only in the case of 50 items), asymptotic estimation with the IPFP algorithm, and Monte Carlo simulation with the IS algorithm. Exact and IS estimates are perfectly overlapping in the case of 50 items. The bias of the IPFP estimates decreases when the number of items increases.

augmentation or estimation of error models, as outlined above.

Partition Function

When the distance in the BMM is footrule or Spearman, the partition function $Z_n(\cdot)$ does not have a closed form. In these situations **BayesMallows** allows for three different choices, which the user may employ depending on the value of n: (a) exact calculation, (b) Importance Sampling (IS), and (c) the asymptotic approximation due to Mukherjee (2016).

The package contains integer sequences for exact calculation of the partition function with footrule distance for up to $n = 50$ items, and with the Spearman distance for up to $n = 14$ items (see Vitelli et al., 2018, Section 2.1). These sequences are downloaded from the On-Line Encyclopedia of Integer Sequences (Sloane, 2017).

The IS procedure can be used to compute an off-line approximation $\hat{Z}_n(\alpha)$ of $Z_n(\alpha)$ for the specific value of n which is needed in the application at hand. The IS estimate $\hat{Z}_n(\alpha)$ is computed on a grid of α values provided by the user, and then a smooth estimate obtained via a polynomial fit is returned to the user, who can also select the degree of the polynomial function. Finally, the user should set the number K of IS iterations, and we refer to Vitelli et al. (2018) for guidelines on how to select a large enough value for K. The procedure might be time-consuming, depending on K, n, and on how the grid for α is specified. In our experience, values of n larger than approximately 100 might require K to be as large as 10^8 in order for the IS to provide a good estimate, and hence a long computing time.

The IPFP algorithm (Mukherjee, 2016, Theorem 1.8) yields a numeric evaluation of $Z_{\text{lim}}(\cdot)$, the asymptotic approximation to $Z_n(\cdot)$. In this case the user needs to specify two parameters: the number of iterations m to use in the IPFP, and the number of grid points K of the grid approximating the continuous domain where the limit is computed. Values of m and K have been suggested by Mukherjee (2016), and we refer to the Supplementary Material of Vitelli et al. (2018) for more details.

A simulation experiment was conducted comparing the methods for estimating $\log(Z(\alpha))$ with footrule distance for $\alpha = 0.1, 0.2, \ldots, 20$. Let $\log(Z(\alpha))^K$ denote the IS estimate obtained with K iterations, and define the absolute relative difference between two IS estimates obtained with K_2 and K_1 iterations as

$$\rho(K_2, K_1) = \max_\alpha \left\{ \frac{\left|\log(Z(\alpha))^{K_2} - \log(Z(\alpha))^{K_1}\right|}{\left|\log(Z(\alpha))^{K_1}\right|} \right\}.$$

The IS algorithm was run with 10^5, 10^6, and 10^7 iterations, and we obtained $\rho\left(10^6, 10^5\right) < 0.3\%$ and $\rho\left(10^7, 10^6\right) < 0.15\%$, suggesting that using $K = 10^7$ yields low Monte Carlo variation over this range of α. The IPFP algorithm was used to estimate $\log(Z_{\text{lim}}(\alpha))$, with $K = 10^3$ and $m = 10^3$. The results are summarized in Figure 1, in which also exact computation is included in the case of 50 items. With 50 items, the IS estimate perfectly overlaps the exact estimate, while the asymptotic estimate has a bias that increases with increasing α. As expected, the bias of the asymptotic estimate decreases when the number of items increases, as this brings it closer to the asymptotic limit.

Sampling from the Bayesian Mallows Model

To obtain random samples from the MM with Cayley, Hamming, Kendall, or Ulam distance and fixed α and $\boldsymbol{\rho}$, we suggest using the **PerMallows** (Irurozki et al., 2016a) package, which is optimized for this task. We instead provide a procedure for sampling from the MM with footrule and Spearman. The procedure to generate a random sample of size N from the MM is straightforward, and described in Appendix C of Vitelli et al. (2018). Basically we run a Metropolis-Hastings algorithm with fixed $\boldsymbol{\rho}$ and α and accept/reject the sample based on its acceptance probability. We then take N rankings with a large enough interval between each of them to achieve independence.

Packages implementing the Mallows model

This section gives an overview of existing packages for fitting the MM.

- **PerMallows** is the package that comes closest in functionality to **BayesMallows**. It contains functions for learning and sampling from the frequentist versions of the MM and generalized Mallows model (GMM) (Fligner and Verducci, 1986). Compared to **BayesMallows**, it lacks support for footrule or Spearman distance, it is not Bayesian, and does not compute uncertainty ranges for the estimated parameters. In addition **PerMallows** handles only complete rankings, and does not provide functionality for computation of mixture models. According to Irurozki et al. (2016a, Table 1), computing the maximum likelihood estimates (MLE) of α and ρ using the function `lmm` is possible when $n < 80$ for Kendall, $n < 250$ for Cayley, $n < 90$ for Hamming and $n < 100$ for Ulam. Our experiments suggest that these estimates are conservative, and that even larger numbers of items are fit rapidly. Hence, **PerMallows** seems to be a good choice for modeling with complete data without clusters, when the supported distance measures are appropriate and uncertainty estimates are not sought. **PerMallows** also has very efficient functions for sampling from the MM with Cayley, Hamming, Kendall, and Ulam distances.
- **pmr** (Lee and Yu, 2013) provides summary statistics, visualization, and model fitting tools for complete ranking data in the MM, as well as other models. The function `dbm` returns the MLE of α together with its variance. The MLE of ρ, however, is not returned, but printed to the console, and no uncertainty estimates are given. Internally, `dbm` generates a matrix of size $n! \times n$ containing all possible permutations of the n items. As a result, it quickly runs into memory issues. In our tests, `pmr` was not able to handle a ranking dataset with $n = 10$ items.
- **rankdist** (Qian, 2018) implements distance-based probability models for ranking data as described in Alvo and Yu (2014), returning MLEs for α and ρ, but no uncertainty estimates. The package handles a large number of distances and supports mixture models, but in our experiments a warning was issued when using mixtures with all distances except Kendall. **rankdist** also implements the GMM (Fligner and Verducci, 1986). However, for Cayley, footrule, Hamming, and Spearman distances, it generates an $n! \times n$ matrix internally, causing our R session to crash with $n \geq 10$ items, hence limiting its applicability. For Kendall, on the other hand, **rankdist** appears to work fine both with a large number of items, and with mixtures.

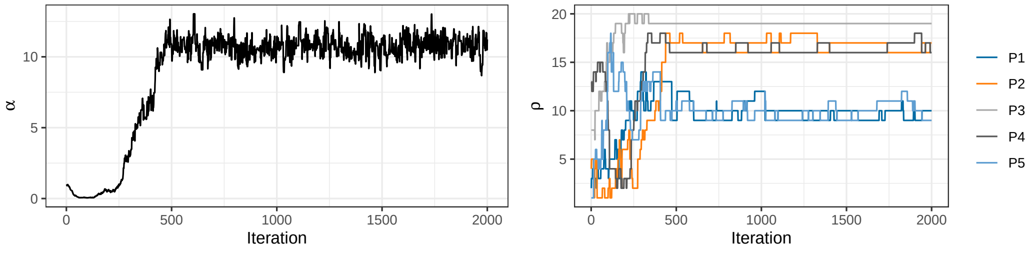
BayesMallows provides many new functionalities not implemented in these packages, as will be illustrated in the use cases of the following three sections.

Analysis of complete rankings with BayesMallows

We illustrate the case of complete rankings with the potato datasets described in Liu et al. (2019, Section 4). In short, a bag of 20 potatoes was bought, and 12 assessors were asked to rank the potatoes by weight, first by visual inspection, and next by holding the potatoes in hand. These datasets are available in **BayesMallows** as matrices with names `potato_weighing` and `potato_visual`, respectively. The true ranking of the potatoes' weights is available in the vector `potato_true_ranking`. In general, `compute_mallows` expects ranking datasets to have one row for each assessor and one column for each item. Each row has to be a proper permutation, possibly with missing values. We are interested in the posterior distribution of both the level of agreement between assessors, as described by α, and in the latent ranking of the potatoes, as described by $\boldsymbol{\rho}$. We refer to the attached replication script for random number seeds for exact reproducibility.

First, we do a test run to check convergence of the MCMC algorithm, and then get trace plots with `assess_convergence`.

```
bmm_test <- compute_mallows(potato_visual)
assess_convergence(bmm_test)
```



(a) Trace of α.

(b) Trace of ρ.

Figure 2: Trace plots of α and ρ for the MCMC algorithm with the `potato_visual` dataset. The plots indicating good mixing for both parameters after about 500 iterations.

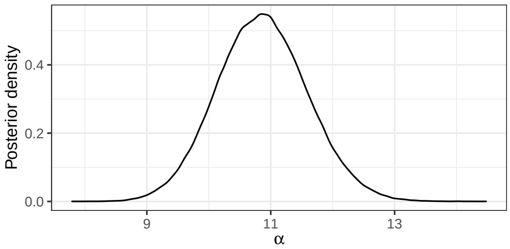


Figure 3: Posterior distribution of α with the `potato_visual` dataset. The posterior mass is symmetrically centered between 9 and 13, with a mean around 11.

By default, `assess_convergence` returns a trace plot for α, shown in Figure 2a. The algorithm seems to be mixing well after around 500 iterations. Next, we study the convergence of $\boldsymbol{\rho}$. To avoid overly complex plots, we pick potatoes $1 - 5$ by specifying this in the `items` argument.

```
assess_convergence(bmm_test, parameter = "rho", items = 1:5)
```

The corresponding plot is shown in Figure 2b, illustrating that the MCMC algorithm seems to have converged after around 1,000 iterations.

From the trace plots, we decide to discard the first 1,000 MCMC samples as burn-in. We rerun the algorithm to get 500,000 samples after burn-in. The object `bmm_visual` has `S3` class "BayesMallows", so we plot the posterior distribution of α with `plot.BayesMallows`.

```
bmm_visual <- compute_mallows(potato_visual, nmc = 501000)
bmm_visual$burnin <- 1000 # Set burn-in to 1000
plot(bmm_visual) # Use S3 method for plotting
```

The plot is shown in Figure 3. We can also get posterior credible intervals for α using `compute_posterior_intervals`, which returns both highest posterior density intervals (HPDI) and central intervals in a `tibble` (Müller and Wickham, 2018). **BayesMallows** uses `tibble`s rather than `data.frame`s, but both are accepted as function inputs. We refer to `tibble`s as dataframes henceforth.

```
compute_posterior_intervals(bmm_visual, decimals = 1L)

# A tibble: 1 x 6
  parameter  mean median conf_level hpdi        central_interval
  <ch      <dbl>  <dbl> <ch      <ch      <chr>
1 alpha       10.9   10.9 95 %       [9.4,12.3] [9.5,12.3]
```

Next, we can go on to study the posterior distribution of $\boldsymbol{\rho}$.

```
plot(bmm_visual, parameter = "rho", items = 1:20)
```

If the `items` argument is not provided, and the number of items exceeds five, five items are picked at random for plotting. To show all potatoes, we explicitly set `items = 1:20`. The corresponding plots are shown in Figure 4.

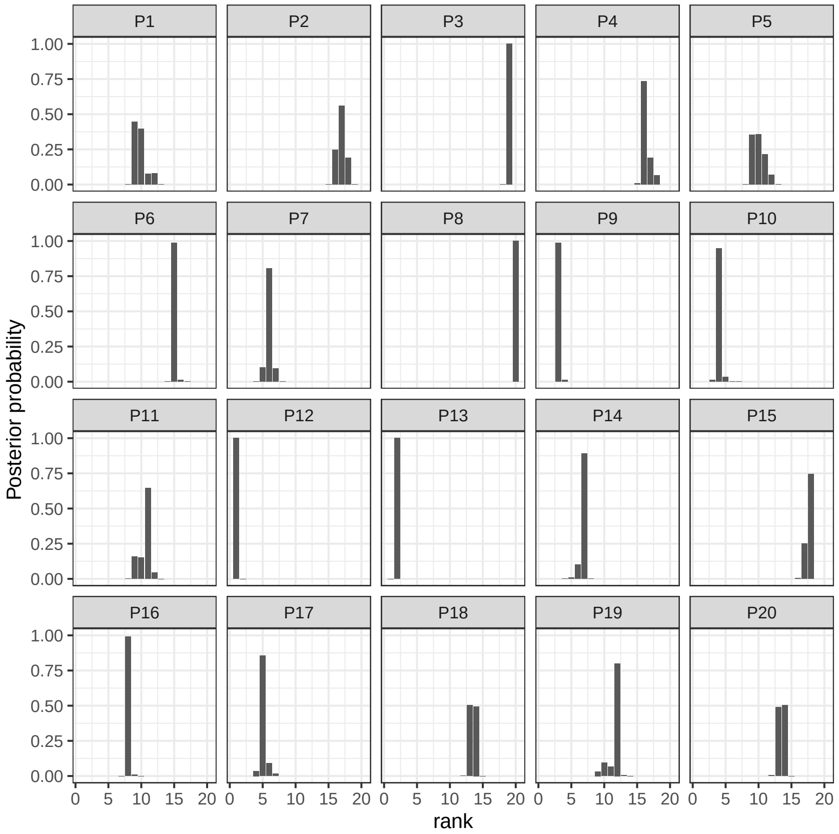


Figure 4: Posterior distribution of latent ranks $\boldsymbol{\rho}$ with the `potato_visual` dataset. Most potatoes have highly peaked posterior distributions, indicating low uncertainty about their ranking.

Jumping over the scale parameter

Updating α in every step of the MCMC algorithm may not be necessary, as the number of posterior samples typically is more than large enough to obtain good estimates of its posterior distribution. With the `alpha_jump` argument, we can tell the MCMC algorithm to update α only every `alpha_jump`-th iteration. To update α every 10th update of $\boldsymbol{\rho}$, we do

```
bmm <- compute_mallows(potato_visual, nmc = 501000, alpha_jump = 10)
```

On a MacBook Pro 2.2 GHz Intel Core i7 running R version 3.5.1, the above call ran in 2.0 seconds on average over 1,000 replications using **microbenchmark** (Mersmann, 2018), while it took 4.2 seconds using the default value `alpha_jump = 1`, i.e., updating α less frequently more than halved the computing time.

Other distance metrics

By default, `compute_mallows` uses the footrule distance, but the user can also choose to use Cayley, Kendall, Hamming, Spearman, or Ulam distance. Running the same analysis of the potato data with Spearman distance is done with the command

```
bmm <- compute_mallows(potato_visual, metric = "spearman", nmc = 501000)
```

For the particular case of Spearman distance, **BayesMallows** only has integer sequences for computing the exact partition function with 14 or fewer items. In this case a precomputed importance sampling estimate is part of the package, and used instead.

Analysis of preference data with BayesMallows

Unless the argument `error_model` to `compute_mallows` is set, pairwise preference data are assumed to be consistent within each assessor. These data should be provided in a dataframe with the following three columns, with one row per pairwise comparison.

- assessor is an identifier for the assessor; either a numeric vector containing the assessor index, or a character vector containing the unique name of the assessor.
- bottom_item is a numeric vector containing the index of the item that was disfavored in each pairwise comparison.
- top_item is a numeric vector containing the index of the item that was preferred in each pairwise comparison.

A dataframe with this structure can be given in the preferences argument to compute_mallows. compute_mallows will generate the full set of implied rankings for each assessor using the function generate_transitive_closure, as well as an initial ranking matrix consistent with the pairwise preferences, using the function generate_initial_ranking.

We illustrate with the beach preference data containing stated pairwise preferences between random subsets of 15 images of beaches, by 60 assessors (Vitelli et al., 2018, Section 6.2). This dataset is provided in the dataframe beach_preferences.

Transitive closure and initial ranking

We start by generating the transitive closure implied by the pairwise preferences.

```
beach_tc <- generate_transitive_closure(beach_preferences)
```

The dataframe beach_tc contains all the pairwise preferences in beach_preferences, with all the implied pairwise preferences in addition. The latter are preferences that were not specifically stated by the assessor, but instead are implied by the stated preferences. As a consequence, the dataframe beach_tc has 2921 rows, while beach_preferences has 1442 rows. Initial rankings, i.e., a set of full rankings for each assessor that are consistent with the implied pairwise preferences are then generated, and we set the column names of the initial ranking matrix to "Beach 1", "Beach 2", ..., "Beach 15" in order have these names appear as labels in plots and output.

```
beach_init_rank <- generate_initial_ranking(beach_tc)
colnames(beach_init_rank) <- paste("Beach", 1:ncol(beach_init_rank))
```

If we had not generated the transitive closure and the initial ranking, compute_mallows would do this for us, but when calling compute_mallows repeatedly, it may save time to have these precomputed and saved for future re-use. In order to save time in the case of big datasets, the functions for generating transitive closures and initial rankings from transitive closures can all be run in parallel, as shown in the examples to the compute_mallows function. The key to the parallelization is that each assessor's preferences can be handled independently of the others, and this can speed up the process considerably with large dataset.

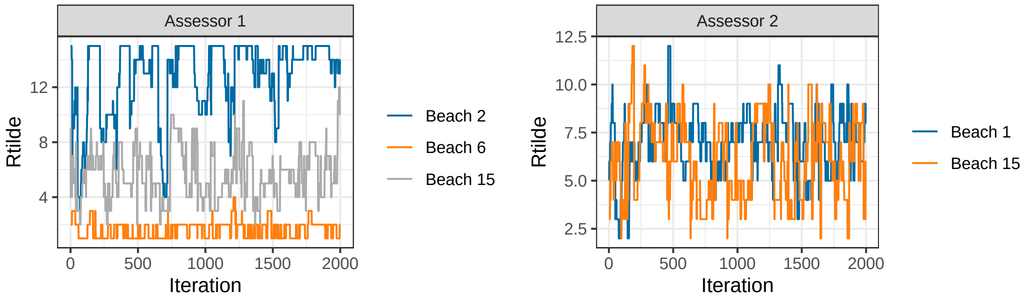
As an example, let us look at all preferences stated by assessor 1 involving beach 2. We use filter from **dplyr** (Wickham et al., 2018) to obtain the right set of rows.

```
library("dplyr")
# All preferences stated by assessor 1 involving item 2
filter(beach_preferences, assessor == 1, bottom_item == 2 | top_item == 2)

# A tibble: 1 x 3
  assessor bottom_item top_item
     <dbl>       <dbl>    <dbl>
1        1           2       15
```

Assessor 1 has performed only one direct comparison involving beach 2, in which the assessor stated that beach 15 is preferred to beach 2. The implied orderings, on the other hand, contain two preferences involving beach 2:

```
# All implied orderings for assessor 1 involving item 2
filter(beach_tc, assessor == 1, bottom_item == 2 | top_item == 2)

  assessor bottom_item top_item
1        1           2        6
2        1           2       15
```

In addition to the statement that beach 15 is preferred to beach 2, all the other orderings stated by assessor 1 imply that this assessor prefers beach 6 to beach 2.



(a) Beaches 2, 6, and 15 for assessor 1. The ordering of the traces is always consistent with the constraints set by assessor 1's preferences.

(b) Beaches 1 and 15 for assessor 2. No orderings between beaches 1 and 15 are implied by assessor 1's preferences, and the traces are hence free to cross.

Figure 5: Trace plots of augmented ranks $\tilde{R}$

Convergence diagnostics

As with the potato data, we can do a test run to assess the convergence of the MCMC algorithm. However, this time we provide the initial rankings beach_init_rank to the rankings argument and the transitive closure beach_tc to the preferences argument of compute_mallows. We also set save_aug = TRUE to save the augmented rankings in each MCMC step, hence letting us assess the convergence of the augmented rankings.

```
bmm_test <- compute_mallows(rankings = beach_init_rank,
                            preferences = beach_tc, save_aug = TRUE)
```

Running assess_convergence for α and ρ shows good convergence after 1000 iterations (not shown). To check the convergence of the data augmentation scheme, we need to set parameter = "Rtilde", and also specify which items and assessors to plot. Let us start by considering items 2, 6, and 15 for assessor 1, which we studied above.

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = c(2, 6, 15), assessors = 1)
```

The resulting plot is shown in Figure 5a. It illustrates how the augmented rankings vary, while also obeying their implied ordering.

By further investigation of beach_tc, we would find that no orderings are implied between beach 1 and beach 15 for assessor 2. With the following command, we create trace plots to confirm this:

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = c(1, 15), assessors = 2)
```

The resulting plot is shown in Figure 5b. As expected, the traces of the augmented rankings for beach 1 and 15 for assessor 2 do cross each other, since no ordering is implied between them. Ideally, we should look at trace plots for augmented ranks for more assessors to be sure that the algorithm is close to convergence. We can plot assessors 1-8 by setting assessors = 1:8. We also quite arbitrarily pick items 13-15, but the same procedure can be repeated for other items.

```
assess_convergence(bmm_test, parameter = "Rtilde",
                   items = 13:15, assessors = 1:8)
```

The resulting plot is shown in Figure 6, indicating good mixing.

Posterior distributions

Based on the convergence diagnostics, and being fairly conservative, we discard the first 2,000 MCMC iterations as burn-in, and take 100,000 additional samples.

```
bmm_beaches <- compute_mallows(rankings = beach_init_rank, preferences = beach_tc,
                               nmc = 102000, save_aug = TRUE)
bmm_beaches$burnin <- 2000
```

The posterior distributions of α and ρ can be studied as shown in the previous sections. Posterior intervals for the latent rankings of each beach are obtained with compute_posterior_intervals:

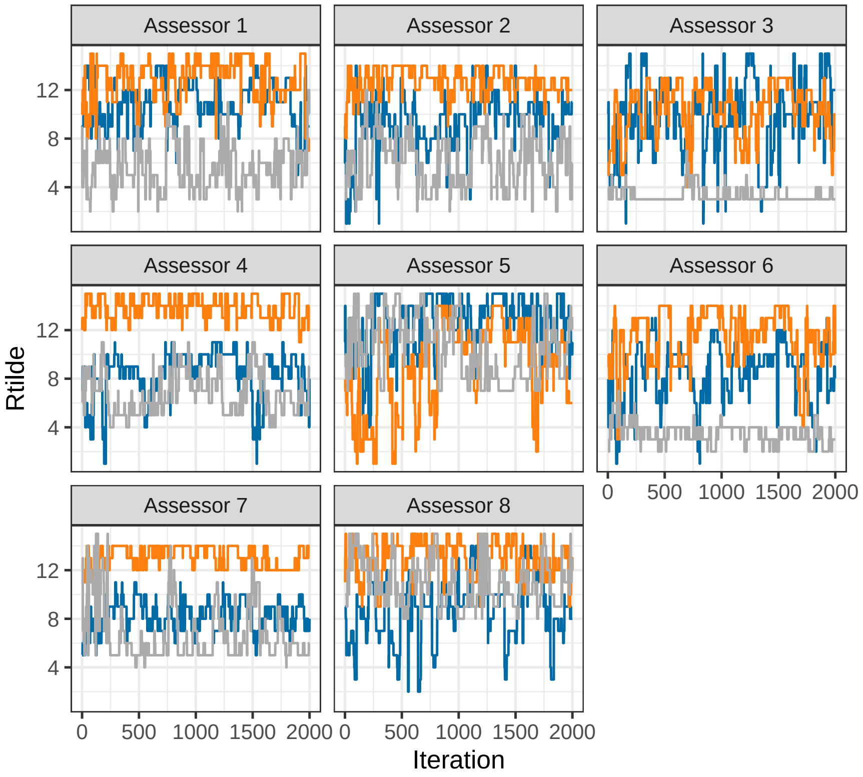


Figure 6: Trace plots of augmented ranks $\tilde{R}$ for beaches 13-15 and assessors 1-8, indicating that the MCMC algorithm obtains good mixing after a low number of iterations.

```
compute_posterior_intervals(bmm_beaches, parameter = "rho")

# A tibble: 15 x 7
   item      parameter  mean median conf_level hpdi    central_interval
   <fct>     <chr>     <dbl>  <dbl> <chr>      <chr>   <chr>
 1 Beach 1   rho           7      7 95 %       [7]     [6,7]
 2 Beach 2   rho          15     15 95 %       [15]    [15]
 3 Beach 3   rho           3      3 95 %       [3,4]   [3,4]
 4 Beach 4   rho          12     12 95 %       [11,13] [11,14]
 5 Beach 5   rho           9      9 95 %       [8,10]  [8,10]
 6 Beach 6   rho           2      2 95 %       [1,2]   [1,2]
 7 Beach 7   rho           9      8 95 %       [8,10]  [8,10]
 8 Beach 8   rho          12     11 95 %       [11,13] [11,14]
 9 Beach 9   rho           1      1 95 %       [1,2]   [1,2]
10 Beach 10  rho           6      6 95 %       [5,6]   [5,7]
11 Beach 11  rho           4      4 95 %       [3,4]   [3,5]
12 Beach 12  rho          13     13 95 %       [12,14] [12,14]
13 Beach 13  rho          10     10 95 %       [8,10]  [8,10]
14 Beach 14  rho          13     14 95 %       [12,14] [11,14]
15 Beach 15  rho           5      5 95 %       [5,6]   [4,6]
```

We can also rank the beaches according to their cumulative probability (CP) consensus (Vitelli et al., 2018, Section 5.1) and their maximum posterior (MAP) rankings. This is done with the function compute_consensus, and the following call returns the CP consensus:

```
compute_consensus(bmm_beaches, type = "CP")

# A tibble: 15 x 3
   ranking item     cumprob
     <dbl> <chr>      <dbl>
 1       1 Beach 9    0.896
 2       2 Beach 6    1
 3       3 Beach 3    0.738
```

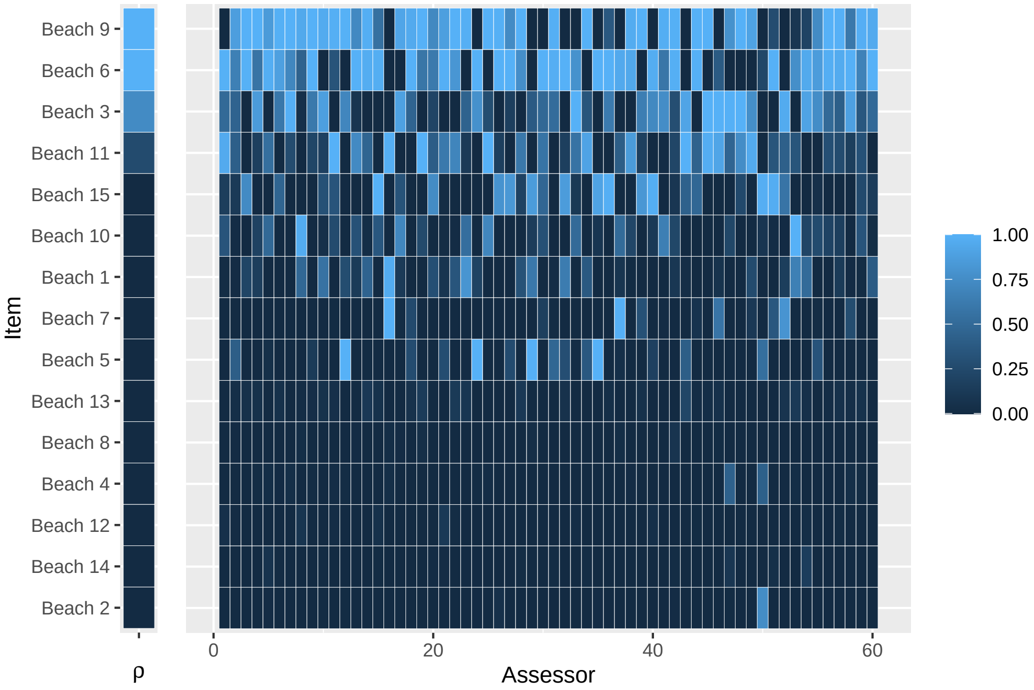


Figure 7: Probability of being ranked top-3 for each beach in the beach preference example (left) and the probability that each assessor ranks the given beach among top-3 (right). Beaches 6 and 9 are most popular overall, but the assessor differ considerably in their preference for these beaches, as can be seen by the varying pattern of light and dark blue.

```
 4        4 Beach 11   0.966
 5        5 Beach 15   0.953
 6        6 Beach 10   0.971
 7        7 Beach 1    1
 8        8 Beach 7    0.528
 9        9 Beach 5    0.887
10       10 Beach 13   1.00
11       11 Beach 8    0.508
12       12 Beach 4    0.717
13       13 Beach 12   0.643
14       14 Beach 14   0.988
15       15 Beach 2    1
```

The column cumprob shows the probability of having the given rank or lower. Looking at the second row, for example, this means that beach 6 has probability 1 of having latent ranking 2 or lower. Next, beach 3 has probability 0.738 of having latent rank 3 or lower. This is an example of how the Bayesian framework can be used to not only rank items, but also to give posterior assessments of the uncertainty of the rankings. The MAP consensus is obtained similarly, by setting type = "MAP".

Keeping in mind that the ranking of beaches is based on sparse pairwise preferences, we can also ask: for beach i, what is the probability of being ranked top-k by assessor j, and what is the probability of having latent rank among the top-k. The function plot_top_k plots these probabilities. By default, it sets k = 3, so a heatplot of the probability of being ranked top-3 is obtained with the call:

```
plot_top_k(bmm_beaches)
```

The plot is shown in Figure 7. The left part of the plot shows the beaches ranked according to their CP consensus, and the probability $P(\rho_i) \leq 3$ for each beach i. The right part of the plot shows, for each beach as indicated on the left axis, the probability that assessor j ranks the beach among top-3. For example, we see that assessor 1 has a very low probability of ranking beach 9 among her top-3, while assessor 3 has a very high probability of doing this. The function predict_top_k returns a dataframe with all the underlying probabilities. For example, in order to find all the beaches that are among the top-3 of assessors 1-5 with more than 90 % probability, we would do:

```
predict_top_k(bmm_beaches) %>%
  filter(prob > 0.9, assessor %in% 1:5)
```

```
# A tibble: 6 x 3
# Groups:   assessor [4]
  assessor item      prob
     <dbl> <chr>    <dbl>
1        1 Beach 11 0.955
2        1 Beach 6  0.997
3        3 Beach 6  0.997
4        3 Beach 9  1
5        4 Beach 9  1.00
6        5 Beach 6  0.979
```

Note that assessor 2 does not appear in this table, i.e., there are no beaches for which we are at least 90 % certain that the beach is among assessor 2's top-3.

Clustering with BayesMallows

BayesMallows comes with a set of sushi preference data, in which 5,000 assessors each have ranked a set of 10 types of sushi (Kamishima, 2003). It is interesting to see if we can find subsets of assessors with similar preferences. The sushi dataset was analyzed with the BMM by Vitelli et al. (2018), but the results in that paper differ somewhat from those obtained here, due to a bug in the function that was used to sample cluster probabilities from the Dirichlet distribution.

Computing mixtures of Mallows distributions

The function `compute_mallows_mixtures` computes multiple Mallows models with different numbers of mixture components. It returns a list of models of class `BayesMallowsMixtures`, in which each list element contains a model with a given number of mixture components. Its arguments are `n_clusters`, which specifies the number of mixture components to compute, an optional parameter `cl` which can be set to the return value of the `makeCluster` function in the **parallel** package, and an ellipsis (...) for passing on arguments to `compute_mallows`.

Hypothesizing that we may not need more than 10 clusters to find a useful partitioning of the assessors, we start by doing test runs with 1, 4, 7, and 10 mixture components in order to assess convergence. We set the number of Monte Carlo samples to 5,000, and since this is a test run, we do not save cluster assignments nor within-cluster distances from each MCMC iteration and hence set `save_clus = FALSE` and `include_wcd = FALSE`. We also run the computations in parallel on four cores, one for each mixture component.

```
library("parallel")
cl <- makeCluster(4)
bmm <- compute_mallows_mixtures(n_clusters = c(1, 4, 7, 10),
                                rankings = sushi_rankings, nmc = 5000,
                                save_clus = FALSE, include_wcd = FALSE, cl = cl)
stopCluster(cl)
```

Convergence diagnostics

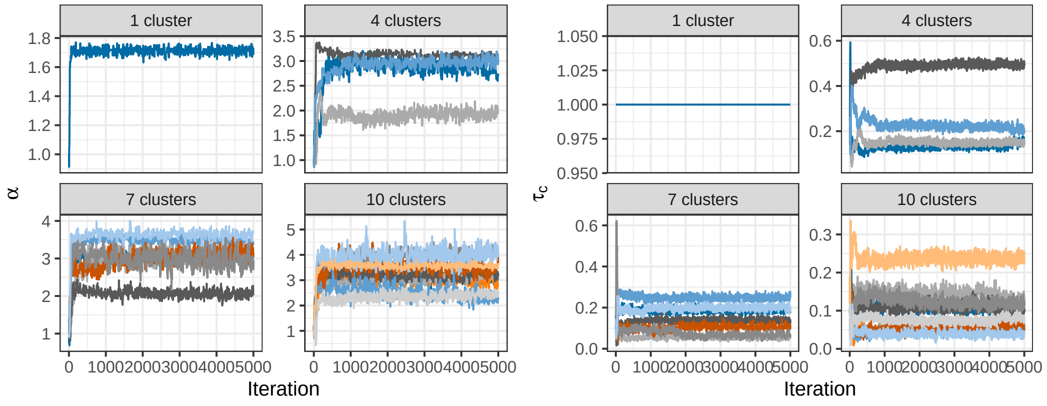
The function `assess_convergence` automatically creates a grid plot when given an object of class `BayesMallowsMixtures`, so we can check the convergence of α with the command

```
assess_convergence(bmm)
```

The resulting plot is given in Figure 8a, showing that all the chains seem to be close to convergence quite quickly. We can also make sure that the posterior distributions of the cluster probabilities τ_c, $(c = 1, \dots, C)$ have converged properly, by setting `parameter = "cluster_probs"`.

```
assess_convergence(bmm, parameter = "cluster_probs")
```

The trace plots for each number of mixture components are shown in Figure 8b. Note that with only one cluster, the cluster probability is fixed at the value 1, while for other number of mixture components, the chains seem to be mixing well.



(a) Trace of α. **(b)** Trace of τ_c.

Figure 8: Trace plot of α and τ_c for the sushi dataset with 1, 4, 7, and 10 mixture components, respectively. Both trace plots indicate good mixing after a few thousand iterations.

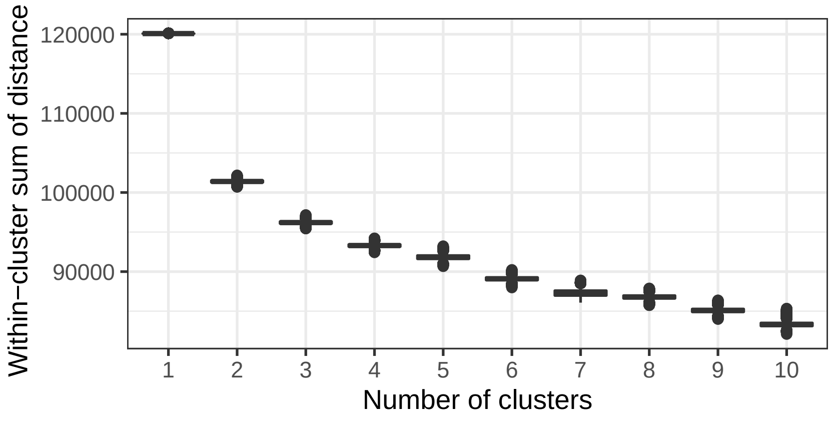


Figure 9: Elbow plot for the sushi mixture models. While it is not entirely clear where the elbow occurs, we choose the mixture distribution with five clusters.

Deciding on the number of mixtures

Given the convergence assessment of the previous section, we are fairly confident that a burn-in of 5,000 is sufficient. We run 95,000 additional iterations, and try from 1 to 10 mixture components. Our goal is now to determine the number of mixture components to use, and in order to create an elbow plot, we set `include_wcd = TRUE` to compute the within-cluster distances in each step of the MCMC algorithm. Since the posterior distributions of ρ_c ($c = 1, \ldots, C$) are highly peaked, we save some memory by only saving every 10th value of $\boldsymbol{\rho}$ by setting `rho_thinning = 10`.

```
cl <- makeCluster(4)
bmm <- compute_mallows_mixtures(n_clusters = 1:10, rankings = sushi_rankings,
                                nmc = 100000, rho_thinning = 10, save_clus = FALSE,
                                include_wcd = TRUE, cl = cl)
stopCluster(cl)
plot_elbow(bmm, burnin = 5000) # Create elbow plot
```

The resulting elbow plot is a notched boxplot (Mcgill et al., 1978; Wickham, 2016) shown in Figure 9, for which the barely visible upper and lower whiskers represent approximate 95 % confidence intervals. Although not clear-cut, we see that the within-cluster sum of distances levels off at around 5 clusters, and hence we choose to use 5 clusters in our model.

Posterior distributions

Having chosen 5 mixture components, we go on to fit a final model, still running 95,000 iterations after burnin. This time we call `compute_mallows` and set `n_clusters = 5`. We also set `save_clus =`

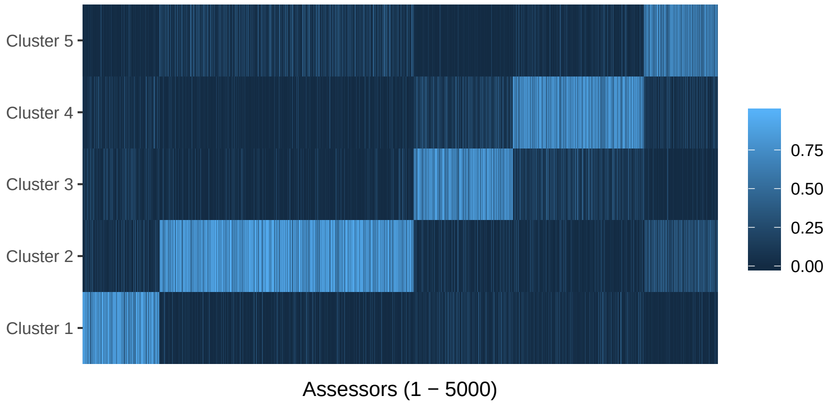


Figure 10: Posterior probabilities of assignment to each cluster for each of the 5000 assessors in the sushi dataset. The scale to the right shows the color coding of probabilities. The blocks of light colors along the anti-diagonal show the clusters to which the assessors were assigned. Darker colors within these blocks indicate assessors whose cluster assignment is uncertain.

TRUE and `clus_thin = 10` to save the cluster assignments of each assessor in every 10th iteration, and `rho_thinning = 10` to save the estimated latent rank every 10th iteration.

```
bmm <- compute_mallows(rankings = sushi_rankings, n_clusters = 5, save_clus = TRUE,
                       clus_thin = 10, nmc = 100000, rho_thinning = 10)
bmm$burnin <- 5000
```

We can plot the posterior distributions of α and ρ in each cluster using `plot.BayesMallows` as shown preivously for the potato data. We can also show the posterior distributions of the cluster probabilities, using:

```
plot(bmm, parameter = "cluster_probs")
```

Using the argument `parameter = "cluster_assignment"`, we can visualize the posterior probability for each assessor of belonging to each cluster:

```
plot(bmm, parameter = "cluster_assignment")
```

The resulting plot is shown in Figure 10. The underlying numbers can be obtained using the function `assign_cluster`.

We can find clusterwise consensus rankings using `compute_consensus`. The following call finds the CP consensuses, and then uses `select` from **dplyr** and `spread` from **tidyr** (Wickham and Henry, 2018) to create one column for each cluster. The result is shown in Table 1.

```
library("tidyr")
compute_consensus(bmm) %>%
    select(-cumprob) %>%
    spread(key = cluster, value = item)
```

	Cluster 1	Cluster 2	Cluster 3	Cluster 4	Cluster 5
1	shrimp	fatty tuna	fatty tuna	fatty tuna	fatty tuna
2	sea eel	sea urchin	sea eel	tuna	sea urchin
3	egg	salmon roe	tuna	shrimp	shrimp
4	squid	sea eel	shrimp	tuna roll	tuna
5	salmon roe	tuna	tuna roll	squid	salmon roe
6	fatty tuna	shrimp	squid	salmon roe	squid
7	tuna	tuna roll	egg	egg	tuna roll
8	tuna roll	squid	cucumber roll	cucumber roll	sea eel
9	cucumber roll	egg	salmon roe	sea eel	egg
10	sea urchin	cucumber roll	sea urchin	sea urchin	cucumber roll

Table 1: CP consensus for each of the clusters found for sushi data.

Note that for estimating cluster specific parameters, label switching is a potential problem that needs to be handled. **BayesMallows** ignores label switching issues inside the MCMC, because it has been shown that this approach is better for ensuring full convergence of the chain (Jasra et al., 2005; Celeux et al., 2000). MCMC iterations can be re-ordered after convergence is achieved, for example by using the implementation of Stephens' algorithm (Stephens, 2000) provided by the R package **label.switching** (Papastamoulis, 2016). A full example of how to assess label switching after running `compute_mallows` is provided by running the following command:

```
help("label_switching")
```

For the sushi data analyzed in this section, no label switching is detected by Stephen's algorithm.

Discussion

In this paper we discussed the methodological background and computational strategies for the **BayesMallows** package, implementing the inferential framework for the analysis of preference data based on the Bayesian Mallows model, as introduced in Vitelli et al. (2018). The package aims at providing a general probabilistic tool, capable of performing various inferential tasks (estimation, classification, prediction) with a proper uncertainty quantification. Moreover, the package widens the applicability of the Mallows model, by providing reliable algorithms for approximating the associated partition function, which has been the bottleneck for a successful use of this general and flexible model so far. Finally, it handles a variety of preference data types (partial rankings, pairwise preferences), and it could possibly handle many others which can lie in the above mentioned categories (noisy continuous measurements, clicking data, ratings).

One of the most important features of the **BayesMallows** package is that, despite implementing a Bayesian model, and thus relying on MCMC algorithms, its efficient implementation makes it possible to manage large datasets. The package can easily handle up to hundreds of items, and thousands of assessors; an example is the Movielens data analyzed in Section 6.4 of (Vitelli et al., 2018). By using the log-sum-exp trick, the implementation of the importance sampler is able to handle at least ten thousand items without numerical overflow. We believe that all these features make the package a unique resource for fitting the Mallows model to large data, with the benefits of a fully probabilistic interpretation.

Nonetheless, we also recognize that the **BayesMallows** package can open the way for further generalizations. The Bayesian Mallows model for time-varying rankings that has been introduced in Asfaw et al. (2017) will be considered for a future release. Some further extensions which we might consider to implement in the **BayesMallows** in the future include: fitting an infinite mixture of Mallows models for automatically performing model selection; allowing for a non-uniform prior for $\boldsymbol{\rho}$; performing automatic item selection; estimating the assessors' quality as rankers; and finally including covariates, both on the assessors and on the items. In addition, since the data augmentation steps in the MCMC algorithm are independent across assessors, potential speedup in the case of missing data or pairwise preferences can be obtained by updating the augmented data in parallel, and this is likely to be part of a future package update.

Acknowledgments

The authors would like to thank Arnoldo Frigessi and Elja Arjas for fruitful discussions.

Bibliography

M. Alvo and P. L. Yu. *Statistical Methods for Ranking Data*. Frontiers in Probability and the Statistical Sciences. Springer, New York, NY, USA, 2014. URL https://doi.org/10.1007/978-1-4939-1471-5. [p10]

D. Asfaw, V. Vitelli, Ø. Sørensen, E. Arjas, and A. Frigessi. Time-varying rankings with the Bayesian Mallows model. *Stat*, 6(1):14–30, 2017. URL https://doi.org/10.1002/sta4.132. [p20]

G. Celeux, M. Hurn, and C. Robert. Computational and inferential difficulties with mixture posterior distribution. *Journal of the American Statistical Association*, 95(451):957–970, 2000. URL https://doi.org/10.2307/2669477. [p20]

M. Crispino, E. Arjas, V. Vitelli, N. Barrett, and A. Frigessi. A Bayesian Mallows approach to nontransitive pair comparison data: How human are sounds? *The Annals of Applied Statistics*, 13(1):492–519, Mar. 2019. URL https://doi.org/10.1214/18-aoas1203. [p7, 8]

J. C. de Borda. Mémoire sur les élections au scrutin, histoire de l'académie royale des sciences. *Paris, France*, 1781. [p7]

P. Diaconis. *Group Representations in Probability and Statistics*, volume 11 of *Lecture Notes - Monograph Series*. Institute of Mathematical Statistics, Hayward, CA, USA, 1988. [p4, 5, 6]

M. A. Fligner and J. S. Verducci. Distance based ranking models. *Journal of the Royal Statistical Society: Series B (Methodological)*, 48(3):359–369, July 1986. URL https://doi.org/10.1111/j.2517-6161.1986.tb01420.x. [p6, 10]

E. Irurozki, B. Calvo, and A. Lozano. Sampling and learning the Mallows and weighted Mallows models under the Hamming distance. *Technical Report*, 2014. URL https://addi.ehu.es/handle/10810/11240. [p6]

E. Irurozki, B. Calvo, and J. A. Lozano. PerMallows: An R Package for Mallows and Generalized Mallows Models. *Journal of Statistical Software*, 71(12), 2016a. URL https://doi.org/10.18637/jss.v071.i12. [p10]

E. Irurozki, B. Calvo, and J. A. Lozano. Sampling and learning Mallows and generalized Mallows models under the Cayley distance. *Methodology and Computing in Applied Probability*, 20(1):1–35, June 2016b. URL https://doi.org/10.1007/s11009-016-9506-7. [p6]

A. Jasra, C. C. Holmes, and D. A. Stephens. Markov chain Monte Carlo methods and the label switching problem in Bayesian mixture modeling. *Statistical Science*, 20(1):50–67, Feb. 2005. URL https://doi.org/10.1214/088342305000000016. [p20]

T. Kamishima. Nantonac collaborative filtering: Recommendation based on order responses. In *Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '03, pages 583–588, New York, NY, USA, 2003. ACM. ISBN 1-58113-737-0. URL https://doi.org/10.1145/956750.956823. [p17]

P. H. Lee and P. L. Yu. An R package for analyzing and modeling ranking data. *BMC Medical Research Methodology*, 13(1):65, May 2013. ISSN 1471-2288. doi: 10.1186/1471-2288-13-65. URL https://doi.org/10.1186/1471-2288-13-65. [p10]

Q. Liu, M. Crispino, I. Scheel, V. Vitelli, and A. Frigessi. Model-based learning from preference data. *Annual Review of Statistics and Its Application*, 6(1):329–354, 2019. URL https://doi.org/10.1146/annurev-statistics-031017-100213. [p5, 10]

T. Lu and C. Boutilier. Effective sampling and learning for Mallows models with pairwise-preference data. *Journal of Machine Learning Research*, 15:3783–3829, 2014. [p6]

C. L. Mallows. Non-null ranking models. i. *Biometrika*, 44(1-2):114–130, 1957. URL https://doi.org/10.1093/biomet/44.1-2.114. [p4, 5]

J. I. Marden. *Analyzing and Modeling Rank Data*, volume 64 of *Monographs on Statistics and Applied Probability*. Chapman & Hall, Cambridge, MA, USA, 1995. [p4, 6]

M. J. A. N. d. C. Marquis of Condorcet. Essai sur l'application de l'analyse à la probabilité des décisions rendues à la pluralité des voix. *Paris: De l'imprimerie royale*, 1785. [p7]

R. Mcgill, J. W. Tukey, and W. A. Larsen. Variations of box plots. *The American Statistician*, 32(1):12–16, 1978. URL https://doi.org/10.1080/00031305.1978.10479236. [p18]

M. Meilă and H. Chen. Dirichlet process mixtures of generalized Mallows models. In *Proceedings of the Twenty-Sixth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-10)*, pages 358–367, Corvallis, OR, USA, 2010. AUAI Press. [p6]

O. Mersmann. *microbenchmark: Accurate Timing Functions*, 2018. URL https://CRAN.R-project.org/package=microbenchmark. R package version 1.4-6. [p12]

S. Mukherjee. Estimation in exponential families on permutations. *The Annals of Statistics*, 44(2): 853–875, 2016. URL https://doi.org/doi:10.1214/15-AOS1389. [p4, 9]

K. Müller and H. Wickham. *tibble: Simple Data Frames*, 2018. URL https://CRAN.R-project.org/package=tibble. R package version 1.4.2. [p11]

P. Papastamoulis. label.switching: An R package for dealing with the label switching problem in MCMC outputs. *Journal of Statistical Software*, 69, 2016. URL https://doi.org/10.18637/jss.v069.c01. [p20]

Z. Qian. *rankdist: Distance Based Ranking Models*, 2018. URL https://CRAN.R-project.org/package=rankdist. R package version 1.1-3. [p10]

N. J. A. Sloane. The On-Line Encyclopedia of Integer Sequences, 2017. URL http://oeis.org. [p9]

M. Stephens. Dealing with label switching in mixture models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 62(4):795–809, Nov. 2000. URL https://doi.org/10.1111/1467-9868.00265. [p20]

V. Vitelli, Ø. Sørensen, M. Crispino, A. Frigessi, and E. Arjas. Probabilistic preference learning with the Mallows rank model. *Journal of Machine Learning Research*, 18(1):5796–5844, Jan. 2018. [p4, 5, 6, 8, 9, 10, 13, 15, 17, 20]

H. Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN 978-3-319-24277-4. URL http://ggplot2.org. [p18]

H. Wickham and L. Henry. *tidyr: Easily Tidy Data with 'spread()' and 'gather()' Functions*, 2018. URL https://CRAN.R-project.org/package=tidyr. R package version 0.8.1. [p19]

H. Wickham, R. François, L. Henry, and K. Müller. *dplyr: A Grammar of Data Manipulation*, 2018. URL https://CRAN.R-project.org/package=dplyr. R package version 0.7.7. [p13]

Øystein Sørensen
Center for Lifespan Changes in Brain and Cognition
Department of Psychology
University of Oslo
ORCID: 0000-0003-0724-3542
oystein.sorensen@psykologi.uio.no

Marta Crispino
Univ. Grenoble Alpes, Inria, CNRS, LJK
38000 Grenoble, France
crispino.marta8@gmail.com

Qinghua Liu
Department of Mathematics
University of Oslo
qinghual@math.uio.no

Valeria Vitelli
Oslo Centre for Biostatistics and Epidemiology
Department of Biostatistics
University of Oslo
OORCID: 0000-0002-6746-0453
valeria.vitelli@medisin.uio.no