MPTinR: Analysis of Multinomial Processing Tree Models in R*

Henrik Singmann
University of Zurich
Albert-Ludwigs-Universität Freiburg

David Kellen
University of Basel
Albert-Ludwigs-Universität Freiburg

July 26, 2015

Abstract

We introduce MPTinR, a software package developed for the analysis of multinomial processing tree (MPT) models. MPT models represent a prominent class of cognitive measurement models for categorical data with applications in a wide variety of fields. MPTinR is the first software for the analysis of MPT models in the statistical programming language R, providing a modeling framework that is more flexible than standalone software packages. MPTinR also introduces important features such as a) the ability to calculate the Fisher information approximation measure of model complexity for MPT models, b) the ability to fit models for categorical data outside the MPT model class, like signal-detection models, c) a function for model selection across a set of nested and non-nested candidate models (using several model selection indices), and d) multi-core fitting. MPTinR is available from the Comprehensive R Archive Network at http://cran.r-project.org/web/packages/MPTinR/

Keywords: MPT models, R, software, Fisher information

1 Introduction

Multinomial processing tree (MPT) models represent a prominent class of cognitive measurement models for categorical data (**???**). They describe the observed response frequencies from a finite set of response categories (i.e., responses following a multinomial distribution) with a finite number of latent states. Each latent state is reached by particular combinations of cognitive processes; processes that are assumed to take place in an all-or-nothing fashion (e.g., either a previously seen item is remembered as having been seen or not). The probability of a latent state being reached depends on the probabilities that the different cognitive processes associated to it successfuly take place. The latent states usually follow each other in a serial order that can be displayed in a tree-like structure (see Figure 1, this model is described in more detail below).

MPT models exist in a wide range of fields, such as memory, reasoning, perception, categorization, and attitude measurement (for reviews see **??**) where they provide a superior data analysis strategy compared to the usually employed ad-hoc models (e.g., ANOVA). MPT models allow the decomposition of observed responses into latent cognitive processes with a psychological interpretation, whereas ad-hoc models only permit the testing of hypotheses concerning the observed data, and the model parameters do not have a psychological interpretation nor does the model provide any insight into the underlying cognitive processes. In this paper we present a package for the analysis of MPT models for the statistical programming language R (**?**) called **MPTinR** that offers several advantages over comparable software (see **?**, for a comparison of software for the analysis of MPTs).

The remainder of this paper is organized as follows. In the next section we will introduce a particular MPT model as an example. In the section thereafter we will provide a general overview of model representation, parameter estimation, and statistical inference in the MPT model class. This overview is by no means exhaustive, but it gives several references that provide in-depth analysis. In the section thereafter, we

*This is an extended and updated version of our manuscript on MPTinR. The code given in this document can be executed as is and the output given here is produced from a real R session (using **knitr** and `sweave`). The original paper was published as: Singmann, H., & Kellen, D. (2013). MPTinR: Analysis of multinomial processing tree models in R. *Behavior Research Methods*, 45(2), 560-575. `http://doi.org/10.3758/s13428-012-0259-0` introduce MPTinR and its functionalities and provide an example session introducing the most important functions for model fitting, model selection and simulation. Furthermore, an overview of all functions in MPTinR is given. Finally, the Appendix contains a description of the algorithms used by MPTinR.

2 An example MPT: The Two-High-Threshold Model of Recognition Memory

Consider the model depicted in Figure 1, which describes the responses produced in a simple recognition memory experiment consisting of two phases: A learning-phase in which participants study a list of items (e.g., words), and subsequently a test-phase in which a second list is presented and participants have to indicate which items were previously studied (old items) and which were not (new items) by responding “Old” or “New”, respectively.

This particular MPT model for the recognition task — the Two-High Threshold Model (2HTM; **?**) — has been chosen because of its simplicity. It consists of two trees, with the item-type associated with each tree (old and new items) specified at the tree’s root. Response categories are specified at the leaves of the trees. Cognitive processes are specified in a sequential manner by the tree nodes and their outcomes (successful occurrence or not) are represented by the branches that emerge from these nodes. The probability of each cognitive process successfully occurring is defined by a parameter.

Let us first consider the old-item tree. When presented with an old item at test, a state of successful remembering is reached with probability D_o (= detect old), and the “Old” response is then invariably given. If the item is not remembered (with probability $(1 - D_o)$), the item is guessed to be “Old” with probability g (= guessing), or to be “New” with probability $(1 - g)$. Regarding the new-item tree, items can be detected as non-studied with probability D_n (= detect new), leading to the items’ rejection (response “New”). If the new item is not detected (with probability $(1 - D_n)$), then it is guessed to be “Old” or “New” with probabilities g and $(1 - g)$, respectively. The predicted response probabilities for each observable response category can be represented by a set of equations:

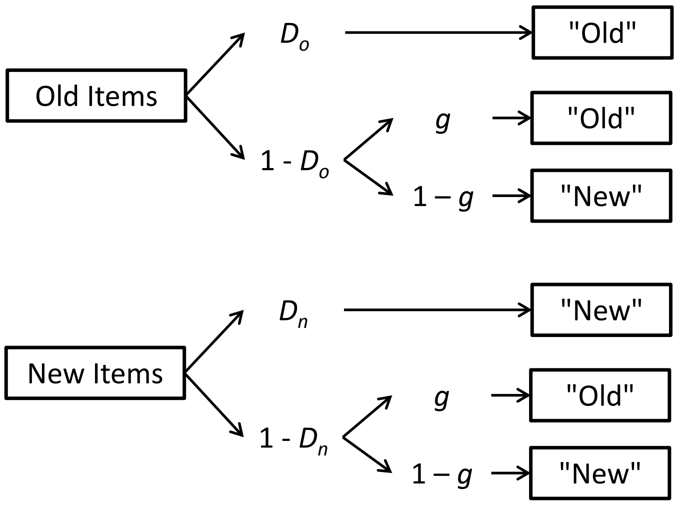


Figure 1: A MPT model (2HTM) for recognition memory. On the left side are the two different item types, old and new items, respectively, each represented by one tree. On the right side are the observed responses “Old” and “New”. In between are the assumed latent states with the probabilities leading to these states. Each tree is traversed from left to right. D_o = Detect an old item as old, D_n = detect a new item as new, g = guess an item as old.

$$\begin{aligned}
P(\text{"Old"}|\text{old item}) &= D_o + (1 - D_o) \times g && (1)\\
P(\text{"New"}|\text{old item}) &= (1 - D_o) \times (1 - g) && (2)\\
P(\text{"Old"}|\text{new item}) &= (1 - D_n) \times g && (3)\\
P(\text{"New"}|\text{new item}) &= D_n + (1 - D_n) \times (1 - g) && (4)
\end{aligned}$$

These equations are constructed by concatenating all branches leading to the same observable response category (e.g., "Old") within one tree. For example, the first line concatenates all branches leading to response "Old" in the old-item tree. As stated above, for old items the response "Old" is given either when an item is successfully remembered as being old (D_o) or, if it is nor remembered as being old, by guessing ($+(1 - D_o) \times g$). Note that the responses associated to the equations above only provide two degrees of freedom, while the model equations assume three free parameters (D_o, D_n, and g), which means that the model is in the present form non-identifiable (see **?**). This issue will be discussed in greater detail below.

The model presented above describes observed responses in terms of a set of unobservable latent cognitive processes, namely a mixture of 1) memory retrieval (D_o), 2) distractor detection (D_n), and 3) guessing (g). Whereas the memory parameters are specific for the item types (i.e., D_o is only part of the old item tree and D_n is only part of the new item tree), the same guessing parameter is present in both trees. This means that it is assumed that guessing (or response bias) is identical whether or not an item is old or new reflecting that the status of each item (old or new) is completely unknown to the participants when guessing. Note that this psychological interpretation of the parameters requires validation studies. In these studies it needs to be shown that certain experimental manipulations expected to selectively affect certain psychological processes are reflected in the resulting model parameters, with changes only being reliably found in the parameters representing those same processes (see **?**, for validation studies of 2HTM parameters).

The contribution of each of the assumed cognitive processes can be assessed by finding the parameter values (numerically or analytically) that produce the minimal discrepancies between predicted and observed responses. The discrepancies between predicted and observed responses can be quantified by a divergence statistic (**?**). As we discuss in more detail below, discrepancies between models and data can be used to evaluate the overall adequacy of the model and to test focused hypotheses on parameters (e.g., parameters have the same values across conditions).

3 Representation, Estimation and Inference in MPT Models: A Brief Overview

3.1 Model Specification and Parameter Estimation

Following the usual formalization (**?**) of MPT models, let $\Theta = \{\theta_1, ..., \theta_S\}$, with $0 \leq \theta_s \leq 1$, θ_s, $s = 1, ..., S$ denote the vector of S model parameters representing the different cognitive processes. For tree k, the probability of a branch i leading to response category j given Θ corresponds to:

$$p_{i,j,k}(\Theta) = c_{i,j,k} \prod_{s=1}^{S} \theta_s^{a_{s,i,j,k}} (1 - \theta)_s^{b_{s,i,j,k}}, \tag{5}$$

where $a_{s,i,j,k}$ and $b_{s,i,j,k}$ represent the number of times each parameter θ_s and its complement $(1 - \theta_s)$ are respectively represented at each branch i leading to category j of tree k, and $c_{i,j,k}$ represents the product of constants on the tree links, if the latter are present. The probability of category j, k given Θ corresponds to $p_{j,k}(\Theta) = \sum_{i=1}^{I_{j,k}} p_{i,j,k}(\Theta)$ (i.e., the sum of all branches ending in one response category per tree), with $\sum_{j=1}^{J_k} p_{j,k}(\Theta) = 1$ (i.e., the sum of all probabilities per tree is 1).

Let $\mathbf{n}$ be a vector of observed category frequencies, with $n_{j,k}$ denoting the frequency of response category j in tree k, with $N_k = \sum_{j=1}^{J_k} n_{j,k}$ and $N = \sum_{k=1}^{K} N_k$. The likelihood function of $\mathbf{n}$ given model parameter vector Θ is:

$$L(\mathbf{n} \mid \Theta) = \prod_{k=1}^{K} \left[\binom{N_k}{n_{1,k}, ..., n_{J_k,k}} \prod_{j=1}^{J_k} [p_{j,k}(\Theta)^{n_{j,k}}] \right] \tag{6}$$

The parameter values that best describe the observed responses correspond to the ones that maximize the likelihood function in Equation 6. These maximum-likelihood parameter estimates (denoted by $\hat{\Theta}$) can sometimes be obtained analytically (e.g., **?**), but in the vast majority of cases they can only be found by means of iterative methods such as the EM algorithm (see **?**). Regarding the variability of the maximum-likelihood parameter estimates, confidence intervals can be obtained by means of the Fisher Information Matrix (the matrix of second order partial derivatives of the likelihood function with respect to Θ; **?**) or via bootstrap simulation (**?**).

The search for the parameters that best describe the data (maximize the likelihood function) requires that the model is *identifiable*. Model identifiability concerns the property that a set of predicted response probabilities can only be obtained by a single set of parameter values. Let Θ and Θ' be model parameter vectors, with $p(\Theta)$ and $p(\Theta')$ as their respective predicted response probabilities. A model is globally identifiable if $\Theta \neq \Theta'$ implies $p(\Theta) \neq p(\Theta')$ across the entire parameter space, and locally identifiable if it holds in the region of parameter space where $\hat{\Theta}$ lies (**?**). An important aspect is that the degrees-of-freedom provided by a dataset provide the upper bound for the number of potentially identifiable free parameters in an model, that is $S \leq \sum_{k=1}^{K}(J_k - 1)$. Local identifiability is sufficient for most purposes, and it can be shown to hold by checking that the Fisher Information Matrix for $\hat{\Theta}$ has rank equal to the number of free parameters (the rank of the Fisher Information Matrix is part of the standard output produced by MPTinR). For a detailed discussion on model identifiability in the MPT model class, see **?**.

3.2 Null-Hypothesis Testing

The discrepancies between predicted and observed response frequencies when taking the maximum-likelihood parameter estimates ($\hat{\Theta}$) are usually summarized by the G^2 statistic:

$$G^2 = 2 \sum_{k=1}^{K} \sum_{j=1}^{J_k} n_{j,k} \left[ln(n_{j,k}) - ln(N_k p_{j,k}) \right] \tag{7}$$

The smaller G^2, the smaller the discrepancies.[1] An important aspect of the G^2 statistic is that it follows a chi-square distribution with degrees-of-freedom equal to the number of independent response categories minus the number of free parameters $\left(\left[\sum_{k=1}^{K}(J_k - 1)\right] - S\right)$. This means that the quality of the account provided by the model can be assessed through null-hypothesis testing. Parameter-equality restrictions (e.g., $\theta_1 = \theta_2$ or $\theta_1 = 0.5$) can also be tested by means of null-hypothesis testing. The difference in G^2 between the unrestricted and restricted models also follows a chi-square distribution with degrees-of-freedom equal to the difference in free parameters between the two models (**?**). It should be noted that parameter-inequality restrictions (e.g., $\theta_1 \leq \theta_2$; see Knapp & Batchelder, 2004) can also be tested, but the difference in G^2 no longer follows a chi-square distribution but a particular mixture of chi-square distributions that in many cases needs to be determined via simulation (**?**, for an example see **?**).

[1]Parameter estimation in MPTinR can only be made using the maximum-likelihood method (Equation 6), which can be obtained by minimizing the discrepancies between observed and expected response frequencies as measured by the G^2 statistic (Equation 7; **?**). Instead of minimizing the G^2 statistic, other discrepancy statistics can be used, in particular one of the many possible statistics coming from the power-divergence family (**?**), from which the G^2 is a special case. Studies (e.g., **??**) have shown that some of these statistics can be advantageous when dealing with sparse data and attempting to minimize a model's sensitivity to outliers. Still, the use of alternatives to the G^2 statistic coming from the power-divergence family has several shortcomings: First, it is not clear which particular statistic would be more advantageous for a specific type of MPT model and data, a situation that would require an extensive evaluation of different alternative statistics. Second, the use of an alternative to the G^2 statistic represents a dismissal of the maximum-likelihood method, which in turn compromises the use of popular model selection measures such as the Akaike Information Criterion, the Bayesian Information Criterion, or the Fisher Information Approximation (which will be discussed in detail later), all of which assume the use of the maximum-likelihood method. Given these disadvantages, and the almost-ubiquitous use of maximum-likelihood estimation in MPT modeling (for reviews, see **??**) the current version of **MPTinR** only implements the maximum-likelihood method for parameter estimation and the G^2 statistic for quantification of model misfit.

3.3 Model Selection

It is important to note that G^2 is a measure that only summarizes the models' goodness-of-fit, and which can only be used to test between nested models (when the restricted model is a special case of the unrestricted model). Furthermore, it ignores possible flexibility differences between the models, that is differences in their inherent ability to fit data in general. The more flexible a model is, the better it will fit any data pattern, regardless of the appropriateness of the model. The best model is not necessarily the one that better fits the data, as it is also important that a model's range of predictions closely follows the observations made and that it can produce accurate predictions regarding future observations (**?**). Model selection analyses attempt to find the model that strikes the best balance between goodness-of-fit and model flexibility (for discussions on different model selection approaches, see **??**), which makes G^2 an unsuitable measure for the comparison of non-nested models. In order to compare both nested and non-nested models in a single framework, as well as to account for potential differences in model flexibility, measures such as the Akaike Information Criterion (AIC; **?**) and the Bayesian Information Criterion (BIC; **?**) are used:[2]

$$\text{AIC} = G^2 + 2S \tag{8}$$
$$\text{BIC} = G^2 + ln(N)S \tag{9}$$

AIC and BIC correct models' fit-results by introducing a punishment factor (the second term in the formulas) that penalizes them for their flexibility (S is the number of parameters). The lower the AIC/BIC the better the account. For the case of AIC and BIC, the number of free parameters is used as a proxy for model flexibility, a solution that is convenient to use but that ignores differences in the model's functional form and is rendered useless when used to compare models that have the same number of parameters (**?**). For example, consider the structurally identical models A and B with two parameters θ_1 and θ_2, with the sole difference between both models that for model B the restriction $\theta_1 \leq \theta_2$ holds. According to AIC and BIC the models are equally flexible despite the fact that the inequality restriction halves model B's parameter space and therefore its flexibility.

A measure that provides a more precise quantification of model flexibility is the Fisher Information Approximation (FIA), a measure that stems from the Minimum Description Length framework (for an introduction, see **?**):

$$\text{FIA} = \frac{1}{2}G^2 + \frac{S}{2} ln\frac{N}{2\pi} + ln\int \sqrt{\det I(\Theta)}\, d\Theta \tag{10}$$

where $I(\Theta)$ is the Fisher Information Matrix for sample size 1 (for details, see **?**). The third additive term of Equation 10 is the penalty factor that accounts for the functional form of the model, providing a more accurate depiction of a model's flexibility. Unlike AIC and BIC, FIA can account for flexibility differences in models that have the same number of parameters. Despite its advantages, FIA is a measure whose computation is far from trivial given the integration of the determinant of the Fisher Information Matrix across a multidimensional parameter space. Due to the recent efforts of **??** the computation of FIA for the MPT model class has become more accessible.

3.4 Context-Free Language for MPTs

Also of interest is the context-free language for the MPT model class developed by **?**, called L_{BMPT}. In L_{BMPT}, each MPT model is represented by a string, called a word, consisting only of symbols representing parameters (θ) or categories (C). The word in L_{BMPT} representing each tree is created by recursively performing the following operations:

1. visit the root

[2]In the AIC and BIC formulas, the first term corresponds to the model's goodness of fit, and the second additive term to the model's penalty factor. As noted by one of the reviewers, we use the G^2 as the first term, contrary to other implementations that use the models' log-likelihood (LL_M) instead. G^2 corresponds to $2 \times (LL_S - LL_M)$, with LL_S being the log-likelihood of a saturated model that perfectly describes the data. In this sense, the definitions of AIC and BIC given in the main body of text can be viewed as differences in AIC and BIC between these two models, making the notation ΔAIC and ΔBIC more appropriate. We nevertheless use the notation AIC and BIC given that we use the notation ΔAIC and ΔBIC when referring to differences between different candidate models other than the saturated model that perfectly describes the data.

2. traverse the upper subtree

3. traverse the lower subtree

During these operations the word is built in the following manner: Whenever a parameter (and *not* its converse) is encountered, add the parameter to the string. Whenever a response category (i.e., leaf) is reached, add the category to the string. The word is complete when reaching the last response category. The structure for the trees in Figure 1 in $\mathrm{L_{BMPT}}$ is thus:

$$\theta C \theta C C \tag{11}$$

By assigning indices one obtains a word in $\mathrm{L_{BMPT}}$ for each tree in Figure 1:

$$\begin{gathered} \theta_{D_o} C_{Old} \theta_g C_{Old} C_{New} \\ \theta_{D_n} C_{New} \theta_g C_{Old} C_{New} \end{gathered} \tag{12}$$

In order to create a single MPT model of the two trees in Figure 1 one needs to assume a joining parameter θ_{join} whose branches connect the two trees into a single one. In this case, the values of θ_{join} and (1-θ_{join}) would be fixed a priori as they represent the proportion of times that old and new items occur during test. The resulting full model for the recognition memory experiment in $\mathrm{L_{BMPT}}$ is:

$$\theta_{join} \theta_{D_o} C_{Old} \theta_g C_{Old} C_{New} \theta_{D_n} C_{New} \theta_g C_{Old} C_{New} \tag{13}$$

The context-free language of **?** is extremely useful as it allows the statement and proof of propositions regarding the MPT model class. One application of this language is in the computation of FIA (Wu et al., 2010a, 2010b).

4 General Overview of MPTinR

MPTinR offers five main advantages over comparable software (cf. **?**). First, **MPTinR** is fully integrated into the R language, an open-source implementation of the S statistical programming language (**?**), that is becoming the lingua franca of statistics (**??**). Besides being free (as it is part of the GNU project, see http://www.gnu.org) and platform independent, R's major strength is the combination of being extremely powerful (it is a programming language) with the availability of a wide variety of statistical and graphical techniques. Couching **MPTinR** within this environment lets it benefit from these strengths. For example, data usually needs to be preprocessed before fitting an MPT model. In addition to fitting an MPT model, one may want to visualize the parameter estimates, run hypothesis-tests on particular parameter restrictions or perform simulations validating certain aspects of the model such as the parameter estimates or the identifiability of the model. When using **MPTinR** all of those processes that can be done within one single environment without the need to ever move data between programs.

Second, **MPTinR** was developed with the purpose of being easy to use, improving some of the more cumbersome features of previous programs, such as the ones concerning model representation. MPT models are represented in most programs such as GPT (**?**), HMMTree (**?**), or multiTree (**?**) by means of .EQN model files. Model specification in .EQN files need to follow a certain structure that could lead to errors, and diverge from the model equations (e.g., Equations 1-4) that are normally used to represent these models in scientific manuscripts. These requirements can become especially cumbersome when handling MPT models comprised of trees with numerous branches (e.g., **?**). Furthermore, most programs require parameter restrictions to be specified "by hand" every time the program is used (for an exception, see Moshagen, 2010), or different model files implementing the parameter restrictions have to be created. **MPTinR** overcomes these inconvenient features: Models are specified in a way that is virtually equivalent to the equations used to represent models (i.e., Equations 1-4), and model restrictions are intuitively specified. Furthermore, model and restrictions can be specified in external files or directly within an R script. In addition, **MPTinR** automatically detects whether single or multiple datasets are fitted and adjusts the output accordingly. For multiple datasets, the summed results (e.g. summed G^2 values) as well as the results for the aggregate data (i.e., summed response frequencies across datasets) are per default provided in the output.

Third, **MPTinR** provides different model selection measures, namely AIC, BIC, and FIA. As previously referred, the computation of FIA is not trivial, and only very recently has it become available for the MPT model class (**??**).

Fourth, **MPTinR** is able to translate an MPT model into a string representation according to the context-free language developed by Purdy and Batchelder (2009) as well as construct model equations from the string representation. Given that the manual translation of MPTs can be rather difficult and tedious, the possibility of an automatic translation will likely encourage the use of this context-free language, which has shown great potential in the assessment of model flexibility (see **?**).

Fifth, although being specifically designed for MPTs, **MPTinR** can also be used to fit a wide range of other cognitive models for categorical data, for example models based on signal detection theory (SDT; **??**). This essentially makes **MPTinR** a framework for fitting many types of cognitive models for categorical data and to facilitate their comparison. As this last point is outside the scope of this article we refer interested readers to the documentation for the functions `fit.model` and `fit.mptinr` which contain detailed examples of how to fit different SDT models.

4.1 Getting Started

MPTinR is a package for the R programming language and therefore needs to be used within the R environment by using the functions described below. For users familiar with commercial statistic packages such as SPSS, the handling of R may be uncommon as it does not come with a graphical user interface (but see **?**, for an overview, and **?**, for a powerful graphical user interface for R). Instead, all commands have to be entered at the prompt. **MPTinR** comes with a manual describing all functions in detail (available also via http://cran.r-project.org/web/packages/MPTinR/MPTinR.pdf) and has a website with more information on important features such as model files and restrictions (see http://www.psychologie.uni-freiburg.de/Members/singmann/R/mptinr/modelfile). To obtain the documentation for any function of **MPTinR** simply enter the function name at the prompt preceded by a `?` (e.g., `?fit.mpt` to obtain the detailed documentation containing examples for the main function `fit.mpt`). The documentation for each function contains a detailed description of its use and the arguments that need to be passed to it. Here we only present the most relevant arguments of each function.

MPTinR is available via the Comprehensive R Archive Network (CRAN; http://cran.r-project.org/) and can therefore be installed from within any R session with the following command (given an active Internet connection):

```
install.packages("MPTinR")
```

Note that you might need an up-to-date version of R to install MPTinR. After successful installation (which only needs to be done once), MPTinR needs to be loaded into the current R session with the following command (this needs to be done each time you start a new R session):

```
library("MPTinR")
```

4.2 Format of Models, Restrictions, and Data

The basis of all analyses of MPT models in **MPTinR** is the representation of the model via a model file. Whereas MPTinR can read the well-known .EQN model files (e.g., **?**) it offers an alternative, the `easy` format. To specify a model in the `easy` format the model file needs to contain the right hand sides of the equations defining an MPT model (e.g., Equations 1-4) with the equations for each tree separated by at least a single empty line. In other words, for each tree all branches ending in the same response category need to be written in a single line concatenated by +. Note that only trees with binary branching can be specified in **MPTinR** (for an exception, see **?**). The model file for the 2HTM model from Figure 1 in the `easy` format could be:

```
# Tree for old items: First 'yes', then 'no'
Do + (1 - Do) * g
(1-Do)*(1-g)

#Tree for new items: First 'yes', then 'no'
(1-Dn) * g
Dn + (1-Dn) * (1 - g)
```

As can be seen, MPTinR allows for comments in the model file. Everything to the right of the number sign `#` will be ignored and lines containing only a comment count as empty.[3] Also, additional whitespace within the equations is ignored. Note that the parameter names used in the model files need to be valid R variable names (for details, type `?make.names` and `?reserved` at the command prompt).

The format of restrictions files is similar to the format of model files. Each restriction needs to be specified on one line. Furthermore, the following rules apply: Inequality restrictions needs to be placed before equality restrictions and can only be specified using the smaller than operator `<` (note that inequality restrictions containing `<` actually represent the weak inequality $\leq$). If a variable appears not as the rightmost element in a restriction it can only appear as the rightmost element in any other restriction (in other words in a set of restrictions a variable can appear multiple times, but only once not as the rightmost element). In addition to simple equality and inequality restrictions MPTinR can also deal with order restrictions involving more than two parameters. For example, $Y1 = Y2 = 0.5$ will set both parameters $Y1$ and $Y2$ to 0.5. Similarly, $W1 < W2 < W3$ will be correctly interpreted as $W1 < W2$ and $W2 < W3$. A valid restrictions file (for a fictitious MPT model) could be (note that we have added quotes at the beginnign and end for display purposes, those need to be removed in an actual file):

```
" # quotes need to be removed
W1 < W2 < W3
X4 = X3
Y1 = Y3 = 0.5
Z = 0 #Restrictions may also contain comments.
"
```

Note that it is also possible to specify model and restriction within an R script (as compared to in an external file), using for example the `textConnection` function included in the base R package. Restrictions can also be specified within a script as a `list` of strings. We will use this functionality in the following.

MPTinR contains the function `check.mpt` that may help in writing model and restrictions files. It has the format `check.mpt(model.filename, restrictions.filename = NULL)` and will return a list with the following information: a logical value indicating whether or not the probabilities on each tree sum to one, the number of trees and the number of response categories the model has, the number of independent response categories the model provides, and the name and number of the parameters in the tree. `model.filename` is the only mandatory argument and needs to be a character string specifying the location and name of the model file[4]. `restrictions.filename` is an optional argument specifying the location and name of the restrictions file.

For example, calling `check.mpt` on the 2HTM model above will return an output indicating that the probabilities in each tree sum to one (if not, the function will pinpoint the misspecified trees), that the number of trees in the model is two, the number of response categories is four, the number of independent response categories is two, and the three parameters are D_n, D_o, and g:

[3]Note that the way MPTinR deals with comments has changed. As is common in R (and other programming languages) everything to the right of the comment symbol # is ignored. In previous versions or MPTinR (prior to version 0.9.2) a line containing a # at any position was ignored completely.

[4]R looks for files in the current working directory. To find out what is the current working directory type `getwd()` at the R prompt. You can change the working directory using either the R menu or using function `setwd`. Additionally, models and restrictions can also be specified within an R script (i.e., not in a file) using a `textConnection`, see the examples in `?fit.mpt` and `?fit.model`.

```
mod_2htm_1 <- "
# Tree for old items: First 'yes', then 'no'
Do + (1 - Do) * g
(1-Do)*(1-g)

#Tree for new items: First 'yes', then 'no'
(1-Dn) * g
Dn + (1-Dn) * (1 - g)
"
check.mpt(textConnection(mod_2htm_1))
```

```
## $probabilities.eq.1
## [1] TRUE
##
## $n.trees
## [1] 2
##
## $n.model.categories
## [1] 4
##
## $n.independent.categories
## [1] 2
##
## $n.params
## [1] 3
##
## $parameters
## [1] "Dn" "Do" "g"
```

The data on which an MPT model will be fitted needs to be passed as a numeric data object, either as a `vector`, `matrix`, or `data.frame`. The mapping of data to response category is done via position. The ordinal position (rank) of each equation in the model file needs to correspond to the response category that is represented at that position/rank in the data `vector` or to the column with that number if the data object is a `matrix` or `data.frame`. If a `matrix` or `data.frame` contains more than one row, each row is considered as one dataset and the MPT model is fitted separately for each dataset and the data summed across rows is considered as another dataset (called `aggregated` dataset) which is also fitted. The data can be entered directly into R or by loading it using one of the data import functions (e.g., `read.table`, see also **?**). The `aggregated` data of the dataset described below could be entered as vector `d.broeder.agg` as:

```
d.broeder.agg <- c(145, 95, 170, 1990, 402, 198, 211, 1589, 868, 332,
                   275, 925, 1490, 310, 194, 406, 1861, 299, 94, 146)
```

4.3 An Example Session

Before estimating model parameters, it is important to see whether a model is identifiable. As previously pointed out the 2HTM as presented in Figure 1 has three parameters, while the "Old"/"New" responses for both old and new items only provide two independent categories (i.e., independent data points to be fitted) as given in the output of `check.mpt`. This means that the 2HTM with three paramaters is not identifiable in the current form. There are two ways of achieving identifiability for this model: 1) by imposing the restriction $D_n = D_o$ (**?**), and/or 2) by including additional sets of observed categorical responses and subsequently extend the model in order to account for them.

The extension proposed by the second option can be implemented by fitting the model to responses obtained across different bias conditions, where individuals assumed distinct tendencies to respond "Old" or

"New". These different response biases or tendencies can be induced by changing the proportion of old items in the test phase (e.g., 10% vs. 90%), for example. According to the theoretical principles underlying underlying the 2HTM, the guessing parameter g would be selectively affected by a response-bias manipulation, with D_o and D_n remaining unchanged. For example, **?** used the second solution sketched above by implementing five separate test phases, each with different proportions of old items (10%, 25%, 50%, 75%, and 90%). Consider the resulting model `mod_2htm_2` and the corresponding output from `check.mpt` showing that there are more degrees-of-freedom than free parameters:

```
mod_2htm_2 <- "
# Tree for old items (10%): First 'yes', then 'no'
Do + (1 - Do) * g1
(1-Do)*(1-g1)

#Tree for new items (90%): First 'yes', then 'no'
(1-Dn) * g1
Dn + (1-Dn) * (1 - g1)

# Tree for old items (25%): First 'yes', then 'no'
Do + (1 - Do) * g2
(1-Do)*(1-g2)

#Tree for new items  (75%): First 'yes', then 'no'
(1-Dn) * g2
Dn + (1-Dn) * (1 - g2)

# Tree for old items (50%): First 'yes', then 'no'
Do + (1 - Do) * g3
(1-Do)*(1-g3)

#Tree for new items  (50%): First 'yes', then 'no'
(1-Dn) * g3
Dn + (1-Dn) * (1 - g3)

# Tree for old items (75%): First 'yes', then 'no'
Do + (1 - Do) * g4
(1-Do)*(1-g4)

#Tree for new items  (25%): First 'yes', then 'no'
(1-Dn) * g4
Dn + (1-Dn) * (1 - g4)

# Tree for old items (90%): First 'yes', then 'no'
Do + (1 - Do) * g5
(1-Do)*(1-g5)

#Tree for new items  (10%): First 'yes', then 'no'
(1-Dn) * g5
Dn + (1-Dn) * (1 - g5)
"
```

```
check.mpt(textConnection(mod_2htm_2))

## $probabilities.eq.1
## [1] TRUE
```

```
##
## $n.trees
## [1] 10
##
## $n.model.categories
## [1] 20
##
## $n.independent.categories
## [1] 10
##
## $n.params
## [1] 7
##
## $parameters
## [1] "Dn" "Do" "g1" "g2" "g3" "g4" "g5"
```

Now, consider a 40 × 20 matrix named `d.broeder` containing the individual data of the 40 participants from **?** Experiment 3.[5] Each participant was tested across five different base-rate conditions (10%, 25%, 50%, 75%, and 90% old items). In this data matrix, each row corresponds to one participant, and the columns correspond to the different response categories in the same order as in the model file. We just show the first 6 lines (i.e., participants) here:

```
data("d.broeder")
head(d.broeder)

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    1    5    1   53   10    5    2   43   22     8    10    20    39
## [2,]    6    0    6   48   11    4    1   44   25     5     2    28    39
## [3,]    3    3    3   51   10    5    1   44   15    15     7    23    40
## [4,]    3    3    2   52   10    5    3   42   22     8     3    27    32
## [5,]    5    1    2   52   12    3   12   33   23     7    10    20    40
## [6,]    6    0    0   54   13    2    0   45   28     2     0    30    44
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     6     5    10    49     5     4     2
## [2,]     6     2    13    48     6     1     5
## [3,]     5     3    12    42    12     2     4
## [4,]    13     1    14    39    15     3     3
## [5,]     5     9     6    51     3     6     0
## [6,]     1     0    15    49     5     0     6
```

MPTinR provides two main functions for model fitting and selection, `fit.mpt` and `select.mpt`:

`fit.mpt` is the major function for fitting MPT models to data, returning results such as the obtained log-likelihood and G^2 value, the information criteria AIC, BIC, and FIA (if requested), parameter estimates and respective confidence intervals, and predicted response frequencies. Optionally one can specify model restrictions or request the estimation of the FIA. It has the basic format `fit.mpt(data, model.filename, restrictions.filename = NULL, n.optim = 5, fia = NULL)`. Two arguments in the `fit.mpt` function are of note: First, `fit.mpt` by default returns the best of of five fitting runs for each dataset, a number that can be changed with the `n.optim` argument. Second, FIA is calculated using Monte Carlo methods (see Wu et al., 2010a, 2010b), with the number of samples to be used being specified by the `fia` argument. Following the recommendations of Wu et al., the number of samples should not be below 200,000 for real applications (we use smaller numbers in this vignette for demonstration purposes only).

`select.mpt` is a function that should aid in model selection (e.g., **?**) and takes multiple results from `fit.mpt` as the argument and produces a table comparing the models based on the information criteria AIC,

[5]We thank Arndt Bröder for providing this dataset which also comes with **MPTinR**. The other files necessary to fit this data (i.e., the model and restriction files) also come with MPTinR, see `?fit.mpt`.

BIC, and FIA. It has the basic format `select.mpt(mpt.results, output = c("standard", "full"))`, where `mpt.results` is a `list` of results returned by `fit.mpt`.

In order to exemplify the use of the `fit.mpt` and `select.mpt` functions, two MPT models are fitted to the data from **?** Experiment 3, the 2HTM model described above, and a restricted version of the 2HTM in which the `g1 < g2 < g3 < g4 < g5` constraint was imposed[6]. As noted before, in real applications one should use more than 25,000 FIA samples (usually values such as 200,000).

```
br.2htm <- fit.mpt(d.broeder, textConnection(mod_2htm_2), fia = 25000)

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:02"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:02"
## Time difference of 0.3709061 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:02"

## Optimization routine for dataset(s) 2 7
##  did not converge succesfully.  Tried again with use.gradient == FALSE.
## Optimization for dataset(s) 2 7
##  using numerically estimated gradients produced better results.  Using those results.
##  Old results saved in output == 'full' [['optim.runs']].

## [1] "Model fitting stopped at 2015-07-26 21:35:06"
## Time difference of 3.419285 secs
```

```
br.2htm.ineq <- fit.mpt(d.broeder, textConnection(mod_2htm_2),
                        list("g1 < g2 < g3 < g4 < g5"), fia = 25000)

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:06"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:13"
## Time difference of 7.222584 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:13"

## Optimization routine for dataset(s) 6 7
##  did not converge succesfully.  Tried again with use.gradient == FALSE.
## Optimization for dataset(s) 6 7
##  using numerically estimated gradients produced better results.  Using those results.
##  Old results saved in output == 'full' [['optim.runs']].

## [1] "Model fitting stopped at 2015-07-26 21:35:25"
## Time difference of 11.88893 secs
```

As can be seen from the output, fitting a model produces various status messages and occasional warnings. The warnings stem from the numerical optimization algorithm employed (a quasi-Newton method, see Appendix for more details). Per default **MPTinR** uses the analytic gradient in the optimization process. If the optimization algorithm indicates it did not converge successful, fitting is restartet but this time using the numerical gradient. The warning details which method produced a better result (i.e., larger maximum likelihood) and is consequently used. This warning usually indicates that data is some cells are missing (as is the case here) and some parameters might not be identified for those participants.

[6]To fit only the aggregated data entered before as `d.broeder.agg` simply replace `d.broeder` with `d.broeder.agg` as in `br.2htm.2 <- fit.mpt(d.broeder.agg, "2htm.model")`.

```
d.broeder[c(2, 6, 7),]

##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    6    0    6   48   11    4    1   44   25     5     2    28    39
## [2,]    6    0    0   54   13    2    0   45   28     2     0    30    44
## [3,]    6    0    0   54   13    2    3   42   28     2     1    29    44
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20]
## [1,]     6     2    13    48     6     1     5
## [2,]     1     0    15    49     5     0     6
## [3,]     1     1    14    51     3     1     5
```

fit.mpt returns a list with the following elements (for a detailed description see ?fit.mpt):

```
str(br.2htm, 1)

## List of 6
##  $ goodness.of.fit     :List of 3
##  $ information.criteria:List of 3
##  $ model.info          :List of 2
##  $ parameters          :List of 3
##  $ data                :List of 2
##  $ fitting.runs        :List of 2
```

Element model.info can be used for further model diagnostic. As described before, a necessary condition that all parameters are identified is that the rank of the Fisher Matrix is equal to the number of free parameters. As expected from the warning during fitting, this is not the case for participants 2 and 7, while participant 6 showed no problems here.

```
br.2htm$model.info

## $individual
##    rank.fisher n.parameters n.independent.categories
## 1            7            7                       10
## 2            6            7                       10
## 3            7            7                       10
## 4            7            7                       10
## 5            7            7                       10
## 6            7            7                       10
## 7            6            7                       10
## 8            7            7                       10
## 9            7            7                       10
## 10           7            7                       10
## 11           7            7                       10
## 12           7            7                       10
## 13           7            7                       10
## 14           7            7                       10
## 15           7            7                       10
## 16           7            7                       10
## 17           7            7                       10
## 18           7            7                       10
## 19           7            7                       10
## 20           7            7                       10
## 21           7            7                       10
## 22           7            7                       10
## 23           7            7                       10
## 24           7            7                       10
## 25           7            7                       10
## 26           7            7                       10
## 27           7            7                       10
## 28           7            7                       10
## 29           7            7                       10
## 30           7            7                       10
## 31           7            7                       10
## 32           7            7                       10
## 33           7            7                       10
## 34           7            7                       10
## 35           7            7                       10
## 36           7            7                       10
## 37           7            7                       10
## 38           7            7                       10
## 39           7            7                       10
## 40           7            7                       10
##
## $aggregated
##   rank.fisher n.parameters n.independent.categories
## 1           7            7                       10
```

Let us now look at the 2HTM results for aggregated data (which replicate the results from Bröder and Schütz, 2009, Table 4):

```
br.2htm[["goodness.of.fit"]][["aggregated"]]

##   Log.Likelihood G.Squared df   p.value
## 1      -5376.127  2.835736  3 0.4176509

br.2htm[["parameters"]][["aggregated"]]

##    estimates lower.conf upper.conf
## Dn 0.4450308  0.3716383  0.5184232
## Do 0.5561326  0.5102111  0.6020540
## g1 0.1411012  0.1129961  0.1692063
## g2 0.2152983  0.1736149  0.2569817
## g3 0.3998479  0.3388262  0.4608696
## g4 0.6045703  0.5446874  0.6644531
## g5 0.6895514  0.6424590  0.7366437
```

Next we may want to compare the parameter estimates for the aggregated data between the original and inequality restricted model:

```
br.2htm[["parameters"]][["aggregated"]][,"estimates"]

## [1] 0.4450308 0.5561326 0.1411012 0.2152983 0.3998479 0.6045703 0.6895514

br.2htm.ineq[["parameters"]][["aggregated"]][,"estimates"]

## [1] 0.4450308 0.5561326 0.1411012 0.2152983 0.3998479 0.6045703 0.6895514
```

As these results show, the order restriction on the guessing parameters held for the aggregated datasets (as the parameter values are identical between the two models) and the parameters values are within reasonable ranges. Note that order of the parameters is alphabetical (i.e., D_n, D_o, g_1, g_2, g_3, g_4, g_5). This ordering is based on the current locale (a fact that can lead to unexpected beahviors when e.g., moving between operating systems employing different locales such as Linux and Windows)! Before comparing the models to decide which to select based on the performance on this dataset we might want to check if all models provided a reasonable account of the data by inspecting the goodness of fit statistics. To this end we not only inspect the aggregated data but also the summed individual fits.

```
br.2htm[["goodness.of.fit"]][2:3]

## $sum
##   Log.Likelihood G.Squared  df   p.value
## 1      -4926.139  106.3814 120 0.8081909
##
## $aggregated
##   Log.Likelihood G.Squared df   p.value
## 1      -5376.127  2.835736  3 0.4176509

br.2htm.ineq[["goodness.of.fit"]][2:3]

## $sum
##   Log.Likelihood G.Squared  df   p.value
## 1      -4942.507  139.1176 120 0.1118981
##
## $aggregated
##   Log.Likelihood G.Squared df   p.value
## 1      -5376.127  2.835736  3 0.4176509
```

The results show that the 2HTM is not grossly misfitting the data as none of the likelihood ratio tests is rejected (i.e., ps $> .05$). Furthermore, as implied from the previous results, there are no differences for the aggregated data between the 2HTM with or without the order restriction applied to the guessing parameters. This, however, is not the case for the summed individual data. The log-likelihood and G^2 values for the 2HTM with the order restriction is slightly worse than these values for the original 2HTM indicating that at least for some datasets the order restriction does not completely hold.

From these findings the question arises of whether or not the restrictions on the guessing parameters are justified when taking both model fit and model flexibility into account. To this end we now also consider the information criteria and again only look at the summed individual results as well as the results of the aggregated data.[7]

```
br.2htm$information.criteria[2:3]

## $sum
##        FIA      AIC      BIC FIA.penalty
## 1 531.5117 666.3814 2736.327     478.321
##
## $aggregated
##        FIA      AIC      BIC FIA.penalty
## 1 26.28697 16.83574 68.58437     24.8691

br.2htm.ineq$information.criteria[2:3]
```

[7]In previous versions of **MPTinR** the summed BIC values were reported incorrectly. Due to the non-linear nature of the $ln(N)$ terms in the BIC formula (Equation 9), the individual values cannot simply be summed (which is what previous versions reported). Instead, the G^2 values, the Ns, and the parameters, need to be summed first and BIC calculated from these summed values. From **MPTinR** version 1.9.2 onwards both `fit.mpt` et al. and `select.mpt` report correctly summed BIC values. Although FIA also contains a $ln(N)$ term, the FIA values *can* simply be summed (this can be easily checked by comparing the FIA of multiple individuals with a super-model in which all individuals are fit jointly with separate parameters for each individual). MPTinR version 1.9.2 incorrectly did not do so, but summed parts of the penalty term separately (see Equation 10) producing incorrect results. From version 1.10.2 FIA values are again correctly summed across multiple data sets.

```
## $sum
##        FIA      AIC      BIC FIA.penalty
## 1 359.2473 699.1176 2769.063    289.6886
##
## $aggregated
##        FIA      AIC      BIC FIA.penalty
## 1 21.57116 16.83574 68.58437    20.15329
```

An alternative way of comparing models using information criteria is via function `select.mpt` (note that we call `select.mpt` with `output = "full"` to obtain the model selection table for both individual and aggregated data, the default `output = "standard"` only returns a table comparing the individual results).

```
select.mpt(list(br.2htm, br.2htm.ineq), output = "full")

##          model n.parameters G.Squared.sum df.sum    p.sum p.smaller.05
## 1      br.2htm            7      106.3814    120 0.808191            0
## 2 br.2htm.ineq            7      139.1176    120 0.111898            1
##   G.Squared.aggregated df.aggregated p.aggregated FIA.penalty.sum
## 1             2.835736             3     0.417651        478.3210
## 2             2.835736             3     0.417651        289.6886
##   delta.FIA.sum FIA.best  FIA.sum FIA.penalty.aggregated
## 1      172.2643        0 531.5117               24.86910
## 2        0.0000       40 359.2473               20.15329
##   delta.FIA.aggregated FIA.aggregated delta.AIC.sum wAIC.sum AIC.best
## 1              4.71581       26.28697       0.00000        1       39
## 2              0.00000       21.57116      32.73613        0       15
##    AIC.sum delta.AIC.aggregated wAIC.aggregated AIC.aggregated
## 1 666.3814                    0             0.5       16.83574
## 2 699.1176                    0             0.5       16.83574
##   delta.BIC.sum wBIC.sum BIC.best  BIC.sum delta.BIC.aggregated
## 1       0.00000        1       39 2736.327                    0
## 2      32.73613        0       15 2769.063                    0
##   wBIC.aggregated BIC.aggregated
## 1             0.5       68.58437
## 2             0.5       68.58437
```

The returned table compares one model per row (here split across multiple rows) using the information criteria FIA, AIC, and BIC. For each criterion, delta values (i.e., in reference to the smallest value) and absolute values are presented. The columns labeled `.best` indicate how often each model provided the best account when comparing the individual datasets. As can be seen, for FIA the model with the order restriction always provided the best account for the individual datasets (40 out of 40 individuals). In contrast, for AIC and BIC the unrestricted 2HTM only provided the best account for 26 individuals. For 14 individuals AIC and BIC were identical for both models. As the number of parameters is identical for both models and the penalty factor of AIC and BIC only includes the number of parameters as a proxy of model complexity, the difference in AIC and BIC between those two models merely reflects their differences in model fit. Furthermore, the table presents AIC and BIC weights (`wAIC` and `wBIC`; **?**).

Overall the results indicate the utility of FIA as a measure for model selection. As expected, when the order restriction holds (as is the case for the aggregated data) FIA prefers the less complex model (i.e., the one in which the possible parameter space is reduced due to inequality restrictions). This preference for the less complex models is even evident for the cases where the order restriction might not completely hold as FIA prefers the order-restricted model for all individuals even though the order-restricted model provides a worse fit for 26 individuals. For the datasets obtained by **?** the more complex (i.e., unrestricted) 2HTM model, albeit providing the better fit, seems unjustifiably flexible when using FIA as the model selection criterion.

Note that the calculation of FIA is computationally demanding, especially when inequality restrictions are applied (see the Appendix on how the model is reparametrized for inequality restrictions). Therefore, **MPTinR** now uses the **RcppEigen** package (**?**) for performing this step. This dramatically reduced the calculation time compared to previous versions of **MPTinR** using pure R. Furthermore, the calculation can also be split across different cores for further speed improvements. But this should only be necessary for comparatively large models.

4.4 Extending the Session

4.4.1 Model Selection on Specific Datasets

Given that for two participants (2 & 7) parameters are not uniquely identified we might want to repeat the model selection excluding those participants. To this end, `select.mpt` contains the argument `dataset` which lets us specify for which dataset we want to perform the model selection (this can be a scalar value or vector). Here we decide to perform it on all participants minus the two problematic ones:

```
select.mpt(list(br.2htm, br.2htm.ineq), output = "full", dataset = (1:40)[-c(2, 7)])

##          model n.parameters G.Squared.sum df.sum    p.sum p.smaller.05
## 1      br.2htm            7      102.1259    114 0.779620            0
## 2 br.2htm.ineq            7      129.0831    114 0.158253            1
##   FIA.penalty.sum delta.FIA.sum FIA.best  FIA.sum delta.AIC.sum wAIC.sum
## 1        454.4049      165.7222        0 505.4679       0.00000 0.999999
## 2        275.2041        0.0000       38 339.7457      26.95719 0.000001
##   AIC.best  AIC.sum delta.BIC.sum wBIC.sum BIC.best  BIC.sum
## 1       37 634.1259       0.00000 0.999999       37 2586.930
## 2       15 661.0831      26.95719 0.000001       15 2613.887
```

As can be seen, this does not change the conclusions (i.e., the restricted model performs better). Furthermore, reporting of the aggregated data is suppressed now. Given that the aggregated data included all data sets (i.e., also the two now excluded ones), this makes a lot of sense as it avoids reporting of misleading results.

4.4.2 Considering Additional Models

As mentioned in the beginning, a popular version of the 2HTM restricts the memory parameters to be equal across item types (i.e., $D_o = D_n$). We also might be interested in whether or not this restriction is justified for our data.

```
br.2htmr <- fit.mpt(d.broeder, textConnection(mod_2htm_2), list("Do = Dn"), fia = 25000)

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:26"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:26"
## Time difference of 0.367135 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:26"

## Optimization routine for dataset(s) 6 7
##  did not converge succesfully.  Tried again with use.gradient == FALSE.
## Optimization for dataset(s) 6 7
##  using numerically estimated gradients produced better results.  Using those results.
##  Old results saved in output == 'full' [['optim.runs']].

## [1] "Model fitting stopped at 2015-07-26 21:35:29"
## Time difference of 2.289819 secs
```

```r
br.2htmr.ineq <- fit.mpt(d.broeder, textConnection(mod_2htm_2),
                         list("g1 < g2 < g3 < g4 < g5", "Do = Dn"), fia = 25000)

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:29"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:35"
## Time difference of 6.04716 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:35"

## Optimization routine for dataset(s) 6 7
##  did not converge succesfully.  Tried again with use.gradient == FALSE.
## Optimization for dataset(s) 6 7
##  using numerically estimated gradients produced better results.  Using those results.
##  Old results saved in output == 'full' [['optim.runs']].

## [1] "Model fitting stopped at 2015-07-26 21:35:45"
## Time difference of 10.3367 secs
```

Given we now have four models, we again use `select.mpt` for a model selection table.

```r
select.mpt(list(br.2htm, br.2htm.ineq, br.2htmr, br.2htmr.ineq), output = "full")

##           model n.parameters G.Squared.sum df.sum    p.sum p.smaller.05
## 1       br.2htm            7      106.3814    120 0.808191            0
## 2  br.2htm.ineq            7      139.1176    120 0.111898            1
## 3      br.2htmr            6      155.2634    160 0.590982            1
## 4 br.2htmr.ineq            6      182.5636    160 0.106868            2
##   G.Squared.aggregated df.aggregated p.aggregated FIA.penalty.sum
## 1             2.835736             3     0.417651        478.3210
## 2             2.835736             3     0.417651        289.6886
## 3             6.834057             4     0.144922        457.8010
## 4             6.834057             4     0.144922        276.6893
##   delta.FIA.sum FIA.best  FIA.sum FIA.penalty.aggregated
## 1    172.264340        0 531.5117               24.86910
## 2      0.000000       17 359.2473               20.15329
## 3    176.185353        0 535.4327               22.51166
## 4      8.723801       23 367.9711               17.98387
##   delta.FIA.aggregated FIA.aggregated delta.AIC.sum wAIC.sum AIC.best
## 1             4.886069       26.28697      31.11806 0.000000       10
## 2             0.170259       21.57116      63.85419 0.000000        1
## 3             4.527791       25.92869       0.00000 0.999999       30
## 4             0.000000       21.40090      27.30021 0.000001       14
##    AIC.sum delta.AIC.aggregated wAIC.aggregated AIC.aggregated
## 1 666.3814             0.000000        0.365447       16.83574
## 2 699.1176             0.000000        0.365447       16.83574
## 3 635.2634             1.998322        0.134553       18.83406
## 4 662.5636             1.998322        0.134553       18.83406
##   delta.BIC.sum wBIC.sum BIC.best  BIC.sum delta.BIC.aggregated
## 1     326.82454 0.000000        1 2736.327              5.39434
## 2     359.56067 0.000000        1 2769.063              5.39434
## 3       0.00000 0.999999       39 2409.502              0.00000
## 4      27.30021 0.000001       14 2436.802              0.00000
##   wBIC.aggregated BIC.aggregated
## 1         0.03157       68.58437
## 2          0.03157       68.58437
## 3          0.46843       63.19003
## 4          0.46843       63.19003
```

Overall the FIA results are not completely conclusive. The clear pattern is that the two models including the inequality restrictions on g are preferred compared to the ones without the inequality restriction. At the level of the individual data, the restricted 2HTM provides the best account for more participants (23/22 versus 17/18 for the unrestricted 2HTM) but its summed FIA value is somewhat smaller by around 8. On the level of the aggregated data the pattern is inverted although the difference between restricted and unrestricted 2HTM is minimal (0.17).

As a final results let us now consider the individual parameter estimates of the two best models. The parameter values of the individual data are contained in a three-dimensional array in which the first dimension corresponds to the parameter, with the second dimension we can either obatin the parameter estimates (`"estimates"`), the confidence intervals, or an indicator variable whether or not a given parameters is restricted or free to vary, and the third dimension corresponds to the data set.

```
str(br.2htmr$parameters$individual)

##  num [1:7, 1:4, 1:40] 0.4422 0.4422 0.0294 0.1011 0.5571 ...
##  - attr(*, "dimnames")=List of 3
##   ..$ : chr [1:7] "Dn" "Do" "g1" "g2" ...
##   ..$ : chr [1:4] "estimates" "lower.conf" "upper.conf" "restricted.parameter"
##   ..$ : chr [1:40] "dataset: 1" "dataset: 2" "dataset: 3" "dataset: 4" ...
```

We can use this array to obtain mean or median estimates from the two models (i.e., selecting all parameters and all individuals, but only the parameter estimate):

```
apply(br.2htm$parameters$individual[,1,], 1, mean)

##        Dn        Do        g1        g2        g3        g4        g5
## 0.4383640 0.4768138 0.1989059 0.2830743 0.4477036 0.6325479 0.6950021

apply(br.2htm$parameters$individual[,1,], 1, median)

##        Dn        Do        g1        g2        g3        g4        g5
## 0.4639994 0.4970286 0.1303557 0.2239789 0.4653007 0.6527825 0.7618422

apply(br.2htmr$parameters$individual[,1,], 1, mean)

##        Dn        Do        g1        g2        g3        g4        g5
## 0.5085945 0.5085945 0.1592395 0.2456915 0.4226900 0.6331431 0.6919976

apply(br.2htmr$parameters$individual[,1,], 1, median)

##        Dn        Do        g1        g2        g3        g4        g5
## 0.5225297 0.5225297 0.1268384 0.2348676 0.4567806 0.6305706 0.6969520
```

These results show that D_o and D_n are quite near to each other in the unrestricted model, reinforcing the finding that the restricted model seemed to provide a better account. Furthermore, when restricting D_o and D_n to be equal, the other parameters do not dramatically change suggesting that this is not achieved by some sort of parameter trade-off.

Note that the mean parameter estimates are also part of the returned list:

```
br.2htm$parameters$mean

##    estimates lower.conf upper.conf
## Dn 0.4383640          NA          NA
## Do 0.4768138          NA          NA
## g1 0.1989059          NA          NA
## g2 0.2830743          NA          NA
## g3 0.4477036          NA          NA
## g4 0.6325479          NA          NA
## g5 0.6950021          NA          NA
```

4.4.3 Potential Problems with FIA

FIA provides an asymptotic measure of model complexity approximating normalized maximum likelihood only in the limit when the number of data points (i.e., trials for a given participant) goes to infinity. A consequence of this is that for finite sample sizes FIA may provide inacurate or even inconsistent penalties (**??**). This problem is specifically prevalent when only having small numbers of trials for a given data set and cannot be overcome by collecting more participants, but only by more trials per participant. For example, imagine that in addition to the models dicussed above we would also want to include the 1HT model into our set of candidate models for the data of **?** for which $D_n = 0$.

```
br.1htm <- fit.mpt(d.broeder, textConnection(mod_2htm_2), list("Dn = 0"), fia = 25000)

## Restriction starting with Dn:  Constant is either equal to 0 or 1 or outside the interval
from 0 to 1.  This may lead to problems.
## Restriction starting with Dn:  Constant is either equal to 0 or 1 or outside the interval
from 0 to 1.  This may lead to problems.

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:46"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:47"
## Time difference of 0.329118 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:47"
## [1] "Model fitting stopped at 2015-07-26 21:35:49"
## Time difference of 1.757632 secs
```

```
br.1htm.ineq <- fit.mpt(d.broeder, textConnection(mod_2htm_2),
                        list("g1 < g2 < g3 < g4 < g5", "Dn = 0"), fia = 25000)

## Restriction starting with Dn:  Constant is either equal to 0 or 1 or outside the interval
from 0 to 1.  This may lead to problems.
## Restriction starting with Dn:  Constant is either equal to 0 or 1 or outside the interval
from 0 to 1.  This may lead to problems.

## [1] "Computing FIA: Iteration begins at 2015-07-26 21:35:49"
## [1] "Computing FIA: Iteration stopped at 2015-07-26 21:35:55"
## Time difference of 6.131032 secs

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:35:55"

## Optimization routine for dataset(s) 6 7
##  did not converge succesfully.  Tried again with use.gradient == FALSE.
## Optimization for dataset(s) 6 7
##  using numerically estimated gradients produced better results.  Using those results.
##  Old results saved in output == 'full' [['optim.runs']].
## [1] "Model fitting stopped at 2015-07-26 21:36:04"
## Time difference of 8.921769 secs
```

```
select.mpt(list(br.2htm, br.2htm.ineq, br.2htmr, br.2htmr.ineq, br.1htm, br.1htm.ineq),
           output = "full")[,1:16]

##             model n.parameters G.Squared.sum df.sum    p.sum p.smaller.05
## 1       br.2htm            7      106.3814    120 0.808191            0
## 2  br.2htm.ineq            7      139.1176    120 0.111898            1
## 3      br.2htmr            6      155.2634    160 0.590982            1
## 4 br.2htmr.ineq            6      182.5636    160 0.106868            2
## 5       br.1htm            6      199.1704    160 0.019253            6
## 6  br.1htm.ineq            6      225.5127    160 0.000497            7
##   G.Squared.aggregated df.aggregated p.aggregated FIA.penalty.sum
## 1             2.835736             3     0.417651        478.3210
## 2             2.835736             3     0.417651        289.6886
## 3             6.834057             4     0.144922        457.8010
## 4             6.834057             4     0.144922        276.6893
## 5            57.389632             4     0.000000        480.3194
## 6            57.389632             4     0.000000        289.0994
##   delta.FIA.sum FIA.best  FIA.sum FIA.penalty.aggregated
## 1    172.264340        0 531.5117               24.86910
## 2      0.000000       11 359.2473               20.15329
## 3    176.185353        0 535.4327               22.51166
## 4      8.723801       22 367.9711               17.98387
## 5    220.657230        0 579.9046               23.07462
## 6     42.608379        7 401.8557               18.29412
##   delta.FIA.aggregated FIA.aggregated
## 1             4.886069       26.28697
## 2             0.170259       21.57116
## 3             4.527791       25.92869
## 4             0.000000       21.40090
## 5            30.368538       51.76944
## 6            25.588038       46.98894
```

The results reveal an interesting picture. Although the 1HTM is nested in the (unrestricted) 2HTM the FIA penalty of the summed data for the former is slightly larger than for the latter. In other words, FIA is inconsistent for those two models as the 1HTM *must* be less complex then the 2HTM. Given the number of trials per individual in the data (300 trials per individual) we cannot use FIA to compare the 2HTM and the 1HTM on the individual level. For the aggregated data this is different as we here observe the expected ordering for FIA penalties: 1HTM ¡ 2HTM. In line with **?** the restricted 2HTM is somewhat less complex than the 1HTM.

To sum this up, when using FIA it is important to check if the FIA penalties are logically consistent; nested models must have lower penalties than the superordinate models. If this is not the case it may be preferrable to use the aggregated data instead of the individual level data (despite the problem this may have), as long as the parameters estimates of the aggregated data are not grossly diverging from the individual levels ones. When doing so for the current data it is clear that the 2HTM versions are clearly preferred to the 1HTM with a slight preference for the restricted 2HTM (while in all cases models with inequaltiy restrictions on g are preferred).

4.5 Bootstrapping

Besides model fitting and model selection, the next major functionality of MPTinR concerns *bootstrap simulation* (**?**). In the previous example using the individual data sets obtained by **?**, the response frequencies were low due to the small number of trials. In such cases, asymptotic statistics such as the sampling distribution of the G^2 statistic or the asymptotic confidence intervals for the parameter estimates can be severely compromised (e.g., **?**). Another situation that compromises the assumptions underlying those asymptotic statistics is when parameter estimates are close to the boundaries of the parameter space (i.e., near to 0 or 1; **?**). In such cases, the use of bootstrap simulations may overcome these problems.

According to the *bootstrap principle* (**?**), if one assumes that an observed data sample $\hat{F}$, randomly drawn from a probability distribution F, provides a good characterization of the latter, then one can evaluate F by generating many random samples (with replacement) from $\hat{F}$ and treating them as "replications" of $\hat{F}$. These bootstrap samples can then be used to draw inferences regarding the model used to fit the data, such as obtaining standard errors for the model's parameter estimates ($\hat{\Theta}$). When the data samples are generated on the basis of the observed data, and no assumption is made regarding the adequacy of the model to be fitted, the bootstrap is referred to as *nonparametric*. Alternatively, bootstrap samples can be based on the model's parameters estimates that were obtained with the original data. In this case, the model is assumed to correspond to the true data-generating process and the bootstrap is designated as *parametric*. The parametric bootstraps can be used to evaluate the sampling distribution of several statistics such as the G^2 and the p-values under distinct hypotheses or models (**?**) (see also **?**). The use of the parametric and nonparametric bootstraps not only provides a way to overcome the limitations of asymptotic statistics, but also to evaluate parameter estimates and statistics under distinct assumptions.

MPTinR contains two higher level functions, `gen.data` and `sample.data`, that can be used for bootstrap simulations. The function `gen.data` produces bootstrap samples based on a given model and a set of parameter values. The function `sample.data` produces bootstrap samples based on a given dataset. These functions can be used separately or jointly in order to obtain parametric and nonparametric bootstrap samples. These are general-purpose functions that can be used for a wide variety of goals, such as a) obtaining confidence intervals for the estimated parameters, b) sampling distributions of the G^2-statistic and p-values under several types of null-hypotheses (**?**), and c) model-mimicry analysis (**?**). Also, bootstrap simulations assuming individual differences, as implemented by **?** and **?**, can also be obtained using these functions. Both functions are calling R's `rmultinom` function to obtain multinomially distributed random data.

Given the variety of bootstrap methods and their goals (**?**), we only provide a simple example in which 200 parametric bootstrap samples are used to estimate the 95% confidence intervals for the parameter estimates obtained with the aggregated data from Br"oder and Sch"utz (2009) (note we now only use the aggregated data but the same model again). Note, 200 is also just used for illustration purposes, in real applications values such as 1,000 or 10,000 are more appropriate.

```r
br.2htm.2 <- fit.mpt(colSums(d.broeder), textConnection(mod_2htm_2))

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:36:05"
## [1] "Model fitting stopped at 2015-07-26 21:36:05"
## Time difference of 0.06001997 secs
```

```r
t(br.2htm.2[["parameters"]])

##                   Dn        Do        g1        g2        g3        g4
## estimates  0.4450308 0.5561326 0.1411012 0.2152983 0.3998479 0.6045703
## lower.conf 0.3716383 0.5102111 0.1129961 0.1736149 0.3388262 0.5446874
## upper.conf 0.5184232 0.6020540 0.1692063 0.2569817 0.4608696 0.6644531
##                   g5
## estimates  0.6895514
## lower.conf 0.6424590
## upper.conf 0.7366437

bs.data <- gen.data(br.2htm.2[["parameters"]][,1], 200,
                     textConnection(mod_2htm_2), data = colSums(d.broeder))

br.2htm.bs <- fit.mpt(bs.data, textConnection(mod_2htm_2), fit.aggregated = FALSE)

## Presenting the best result out of 5 minimization runs.

## [1] "Model fitting begins at 2015-07-26 21:36:05"
## [1] "Model fitting stopped at 2015-07-26 21:36:16"
## Time difference of 11.48756 secs
```

```
apply(br.2htm.bs[["parameters"]][["individual"]][,1,],
      1, quantile, probs = c(0.025, 0.975))

##              Dn        Do        g1        g2        g3        g4
## 2.5%  0.3516992 0.5100496 0.1145862 0.1741473 0.3359785 0.5411508
## 97.5% 0.5062910 0.6012799 0.1706003 0.2548288 0.4575785 0.6533729
##              g5
## 2.5%  0.6395261
## 97.5% 0.7279860
```

In this example, we first fit the original data to the (unrestricted) 2HTM to obtain parameter estimates. These estimates are displayed (along with the asymptotic confidence intervals based on the Hessian matrix) and then used as an argument to the `gen.data` function, requesting the bootstrap samples. In the next step the bootstrap samples are fitted using `fit.mpt` setting the `fit.aggregated` argument to `FALSE` to prevent MPTinR from trying to fit the (meaningless) aggregated dataset. Finally, the 95% confidence intervals are calculated by obtaining the 2.5% and 97.5% quantile from the resulting distribution of estimates for each parameter (conveniently done using R's `apply` function). As can be seen, the Hessian based confidence intervals and bootstrapped confidence intervals strongly agree indicating that the variance-covariance matrix obtained via the Hessian matrix is a good approximation of the true variance-covariance matrix (see **?**).

Fitting the bootstrap samples takes only a few seconds. If one requests more samples or the model becomes more complicated, the multi-core functionality of MPTinR (which is more thoroughly described below and in the documentation of `fit.mpt`) can be used to reduce the fitting time by distributing the fitting across cores. For obtaining nonparametric confidence intervals the call to `gen.data` should be replaced with a call to `sample.data`. For example: `bs.data <- sample.data(d.broeder.agg, 10000, "2htm.model")`

4.6 Additional Functionality

This section gives an overview over the additional functions in MPTinR besides the main functions described above (see also Table 1). As already sketched above, MPTinR can fit many types of cognitive models. `fit.model` is a copy of `fit.mpt` (i.e., a model needs to be defined as strings either in a model file or via `textConnection`) with the additional possibility to specify upper and lower bounds for the parameters (as for example needed to fit SDT models). Its documentation contains examples of fitting SDT models, for example to the data of **?**. `fit.mptinr` is a function that allows for even more flexibility in representing a model. Instead of a model as string, a model needs to be specified as a R function returning the log-likelihood of the model (known as an *objective* function) which will be minimized. This allows one to fit models to categorical data that can not be specified in model files, for example models containing integrals. The documentation of `fit.mptinr` contains an example of how to fit a SDT model to a recognition memory experiment in which the memory performance is measured via a ranking task (**?**). Actually, `fit.mptinr` is called by `fit.mpt` and `fit.model` with objective functions created for the models in the model file. `fit.mpt.old` is the old version of MPTinR's main function containing a different fitting algorithm (see it's documentation for more information). Note that `select.mpt` accepts results from any of the fitting functions in MPTinR (as all output is produced by `fit.mptinr`). To reduce computational time for large datasets or models, MPTinR contains the possibility to use multiple processors or a computer cluster by parallelizing the fitting algorithm using the snowfall package (**?**). Furthermore, fitting of the aggregated dataset can be disabled by setting `fit.aggregated = FALSE`.

Table 1: Overview of functions in MPTinR

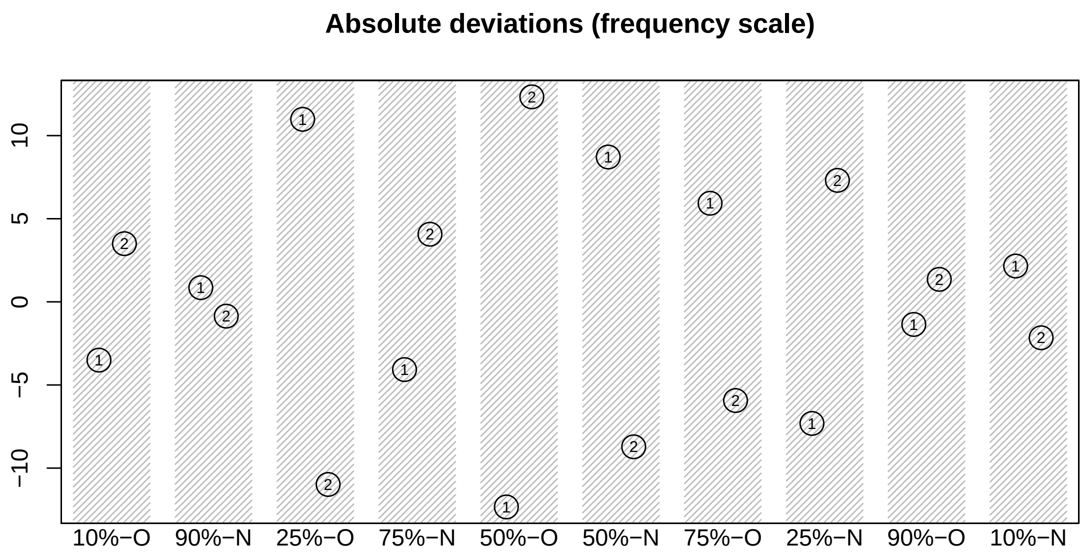
Function name	Description
`fit.mpt`	fit MPT models from model files (allows computation of FIA)
`fit.model`	fit models from model files (specify parameter bounds)
`fit.mptinr`	fit models from objective function (called by `fit.mpt`)
`fit.mpt.old`	fit MPT models from model files using old fitting algorithm
`prediction.plot`	plot observed versus predicted values for fitted model
`select.mpt`	make model selection table from fitted results
`gen.data`	generate data from model and parameter values (i.e., parametric bootstrap)
`sample.data`	generate data from a given dataset (i.e., nonparametric bootstrap)
`gen.predictions`	generate response proportions from given model and parameter values
`make.mpt.cf`	returns word in $\mathrm{L}_{\mathrm{BMPT}}$ for MPT model from model file
`lbmpt.to.mpt`	takes word in $\mathrm{L}_{\mathrm{BMPT}}$ and returns MPT model file
`get.mpt.fia`	conveniently obtain FIA for an MPT model file
`bmpt.fia`	R port of `BMPTFIA` (Wu et al., 2010a)
`prepare.mpt.fia`	make string to obtain FIA in MATLAB using `BMPTFIA`
`make.eqn`	make `eqn` model file from model in `easy` format
`make.mdt`	make `mdt` data file from data `vector`, `matrix`, or `data.frame`

Note. Documentation containing all the arguments for each function can be obtained by typing the function name at the R prompt preceded by a `?`, e.g., `?fit.mpt`

To visualize the model fit, the results from any of the fitting functions can be passed to `prediction.plot` to visualize the model misfit. As an example we again use the fit to the aggregated data.

```r
br.2htm.2 <- fit.mpt(colSums(d.broeder), textConnection(mod_2htm_2), show.messages = FALSE)

axis.labels <- c("10%-O", "90%-N", "25%-O", "75%-N", "50%-O", "50%-N",
                 "75%-O", "25%-N", "90%-O", "10%-N")
prediction.plot(br.2htm.2, textConnection(mod_2htm_2), axis.labels = axis.labels,
                args.plot = list(main = "Absolute deviations (frequency scale)"))
```



This plot shows several things. First, as the trees have only two leafs (in other words, the data is binomially distributed) the misfits in one tree always cancel each other out perfectly. Second, overall, there seems to be more misfit for the old then for the new items. Third, the misfit is particularly strong for the middle category. These observations could provide inside into how to extend the model. It shows us where the model fails.

In addition to the data generating functions `gen.data` and `sample.data` described in the previous section, the function `gen.predictions` returns predicted response proportions or predicted data from a vector of parameter values for a given model. This function can be used to check if a model recovers certain parameter values (i.e., by fitting the predicted responses) and simulated identifiability (i.e., repeating this steps multiple times with random parameter values; ?). The function `gen.data` internally calls `gen.predictions`. Note that the data generating functions that take a model as an argument also work with any model that can be specified in a model file, such as signal-detection models.

`bmpt.fia` is our R port of the original `BMPTFIA` function from ?. As `bmpt.fia` requires a model to be entered as a word in L_{BMPT} we provide the convenience function `get.mpt.fia` which takes similar arguments as `fit.mpt` and will call `bmpt.fia` with the correct arguments. In some cases researchers might prefer to obtain the FIA from MATLAB. To this end, MPTinR contains the convenience function `prepare.mpt.fia`. It takes the same arguments as `get.mpt.fia` but will return a string that is the call to the original `BMPTFIA` function in MATLAB (i.e., the string just needs to be copied and pasted into MATLAB). Note however, that since using **RcppEigen** for calculating FIA the **MPTinR** implementation is faster than the corresponding Matlab code. As noted above, FIA can be directly computed in a call to `fit.mpt` (if one wants to use `select.mpt` it is necessary to use `fit.mpt` and not the other just described functions).

Finally, MPTinR contains several helper functions. `make.mpt.cf` will take a model file as an argument and will produce a word in L_{BMPT} using the algorithm described above. The converse can be achieved by function `lbmpt.to.mpt`[8] which takes a word in L_{BMPT} and produces a model file. `make.eqn` will take a model file in `easy` format and will produce a file in the EQN format. Similarly, `make.mdt` will take data (either a single `vector` or a `matrix` or `data.frame`) and will produce a single file in the MDT format containing all the datasets. EQN and MDT files are used by other programs for fitting MPT models such as MultiTree (?) or HMMTree (?).

[8]This function was contributed by Quentin Gronau and Franz Dietrich.

5 Appendix: MPTinR Algorithms

The purpose of this section is to give an overview of the algorithms used by MPTinR as they diverge from the usual employed fitting algorithm for MPT models (the EM algorithm, Hu & Batchelder, 1994). Readers mainly interested in using MPTinR for fitting MPT models may skip this section. The main task of MPTinR is model fitting, that is iteratively finding the maximum likelihood parameter estimates $\hat{\Theta}$. Instead of the EM algorithm, MPTinR uses the general purpose optimization algorithm implemented in R's `nlminb` function. This algorithm is a variation of Newton's method that can use the analytical or approximated (i.e., quasi-Newton) gradient or Hessian to obtain the optimal parameters within parameter bounds and is part of the PORT library (?). Previous versions of MPTinR used a different optimization algorithm (L-BFGS-B; ?), which is still available in the function `fit.mpt.old`. However, as L-BFGS-B cannot use an analytical gradient we changed the algorithm to `nlminb`.

The advantages of using a general purpose optimization routine instead of a specialized one are twofold. First, the PORT routines are implemented reasonably fast in the FORTRAN programming language (compared to optimization algorithms implemented in pure R code). Second, MPTinR does not require a model to strictly follow an MPT form as described in (?) or (?). Instead, MPTinR literally uses the right hand sides of the model equations (e.g., Equations 1-4) and evaluates them using the current parameter values at each iteration of the optimization process. In other words, the model is not transformed into any matrix notation (see ?). Consequently, MPTinR can fit any model that can be described in a model file using (inbuilt or self-written) R functions.

When calling `fit.mpt` or `fit.model` the following steps are performed for obtaining $\hat{\Theta}$: The equations in the model file are parsed line by line into R `expressions` (i.e., code that can be executed). These `expression`s are concatenated to obtain the likelihood function for a given model (Equation 6). To avoid numerical underflows, the negative log of the likelihood function gives the *objective* function (i.e., the function that will be minimized). From this function the functions to calculate the gradient and the Hessian matrix of the model are derived using symbolical derivation implemented in the `D` function. The objective, gradient and Hessian function are then passed as arguments to `fit.mptinr`, the workhorse of MPTinR. `fit.mptinr` can also be called directly with an objective function if it cannot be specified in a model file.

Both the objective and the gradient function are per default passed to `nlminb` for obtaining $\hat{\Theta}$. In cases where `nlminb` does not converge successfully, fitting is restarted using a numerically approximated gradient (with warning). Furthermore, `fit.model` and `fit.mptinr` allows one to specify whether or not the gradient function or even the Hessian function should be passed to `nlminb` (using the Hessian function for fitting did neither improve the speed nor the quality of the fitting of MPT models and is therefore deactivated per default). To allow an assessment of the quality of the fitting algorithm, MPTinR reports the `summary` statistic of a vector containing the values of the objective function at the obtained minima for each fitting run (if `n.optim > 1`) in output element `fitting.runs`. Our experience is that the dispersion of the minima is usually 0, unless the data contains many zero cells or the model is not identified for that dataset.

When model restrictions are specified they will be applied before creating the objective function in that the expressions representing the model will be altered. In case of equality restrictions (i.e., either setting two parameters equal, e.g., $x = y$, or fixing a parameter to a value e.g., $x = 0$) the model equations are altered so that the to be restricted parameter is replaced with the restriction. To apply inequality restrictions the model is altered using a variant of method A described by (?). More specifically, each instance of the to be restricted parameter is replaced by the product of a dummy parameter and the restriction (e.g., for the restriction $x1 < x2$, each instance of $x1$ would be replaced with $x2 * hank.y1$). Note that all dummy parameters in MPTinR start with *hank.* and it should therefore be avoided to use parameter names starting with these characters. Whereas this reparametrization is equivalent to method A of Knapp and Batchelder, it does not preserve the $\mathrm{L}_{\mathrm{BMPT}}$ structure of the model. In general it holds that for order restrictions only the rightmost element remains and all other parameters will replaced. Note that confidence intervals around inequality restricted parameters are based on variance bounds of the parameter estimates (i.e., they represent a "worst case scenario"; ?, Equation 19).

For obtaining the FIA, MPTinR first transforms a model into a word in $\mathrm{L}_{\mathrm{BMPT}}$ (if a model consists of multiple trees these are concatenated by joining parameters) which is then passed to our R port of the algorithm by Wu et al. (2010a). If a model is not a member of $\mathrm{L}_{\mathrm{BMPT}}$ calculation of the FIA will fail. MPTinR tries to minimize computational time for the FIA by only calculating the penalty factor of the FIA (i.e., the integral in Equation 10) as many time as needed (i.e., as many times as the ratio of the N between tree differs not as many times as N differs).

As our method of reparametrizing inequality restrictions does not preserve the MPT structure of a given model (i.e., even if it is a member of $\mathrm{L}_{\mathrm{BMPT}}$ before applying the restrictions it is not thereafter), MPTinR obtains the FIA for inequality restricted models by passing the *unrestricted* model to the Wu et al. (2010a) algorithm but specifying the restrictions in the corresponding arguments. Whether or not one wants to enforce the restrictions when fitting the model is controlled by the `reparam.ineq` argument to `fit.mpt`. The default behavior is to enforce the inequality restriction by reparametrizing the model (i.e., `reparam.ineq = TRUE` is the default). This contrasts with the example given by Wu et al. in which "parameter estimates do not violate" (p. 282) the inequality restrictions and therefore the inequality restrictions are not enforced when fitting the model but the restrictions are nevertheless passed to `BMPTFIA`. This behavior can be emulated by setting `reparam.ineq = FALSE` in which case the inequality restrictions are not enforced when fitting the model, but will be taken into account when obtaining the FIA.