Chapter 16: Multinomial processing trees

Shravan Vasishth (vasishth.github.io)

March 2026

Contents

Textbook

Introduction to Bayesian Data Analysis for Cognitive Science

Nicenboim, Schad, Vasishth

- Online version: https://bruno.nicenboim.me/bayescogsci/
- Source code: https://github.com/bnicenboim/bayescogsci
- Physical book: here

Be sure to read the textbook's chapter 16 in addition to watching this lecture.

Introduction

In this lecture, we introduce a widely-used class of cognitive models that can be implemented in Stan, the **multinomial processing tree**. This model is useful in situations where the behavioral response from the subject is one of several possible categorical outcomes. As an example, we will look into a word production task, where we ask individuals with aphasia [a language impairment that is usually due to a cerebrovascular accident or head trauma], to name the object shown in a picture, e.g., a picture of a cat. The participant (hereafter, subject) could produce

- the correct name ("cat"),
- a semantically and phonologically related but incorrect name ("rat"),
- a semantically unrelated but phonologically

related word ("hat"), or
- a non-word ("cag").

The researcher may have a theory about how each possible outcome ends up being probabilistically produced. Such a theoretical process model can be expressed as a multinomial processing tree. Before we dive into multinomial processing trees, we discuss the distributions that generalize the binomial and Bernoulli distribution for modeling more than two possible outcomes.

Modeling multiple categorical responses

One way to model categorical responses is using multinomial or categorical distributions. The categorical responses could be "yes" or "no"; "blue", "red" or "yellow"; "true", "false", or "I don't know"; or more complicated categories. Crucially, the response of each observation can be coded as belonging to only one of several possible categories. The multinomial and the categorical distribution represent two ways of characterizing the underlying **generative process** for such data.

The *multinomial distribution* is the generalization of the binomial distribution for more than two possible outcomes. Recall that the binomial works like this: in order to randomly generate the number of successes in an observation consisting of 10 trials, with the probability of success of 0.5, one can type:

```r
rbinom(1, size = 10, prob = 0.5)
```

```
## [1] 4
```

It is possible to repeatedly generate multiple observations as follows. Suppose five simulated observations are needed, each with 10 trials:

```r
rbinom(5, size = 10, prob = 0.5)
```

```
## [1] 6 5 7 7 2
```

Now, suppose that there are N=3 possible answers to a question (yes, no, don't know), and suppose that the probabilities of producing each answer are:

- $P(\text{yes}) = 0.1$
- $P(\text{no}) = 0.1$
- $P(\text{don't know}) = 0.8$

The probabilities must sum to 1, because those are the only three possible outcomes. Given such a situation, it is possible to simulate a single experiment with 10 trials, where each of the three possibilities appears a certain number of times. We do this with `rmultinom()`:

```r
(random_sample <- rmultinom(1, size = 10,
                            prob = c(0.1, 0.1, 0.8)))
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    7
```

The above call returns the result of the random sample: 1 cases of the first answer type, 2 cases

of the second; and 7 cases of the third.

Analogously to the binomial function shown above, five observations can be simulated, each having 10 trials:

```
rmultinom(5, size = 10, prob = c(0.1, 0.1, 0.8))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    3    1    0    0
## [2,]    1    1    1    2    0
## [3,]    8    6    8    8   10
```

The *categorical distribution* is the generalization of the Bernoulli distribution for more than two possible outcomes, and it is the special case of the multinomial distribution when we have only one trial. Recall that the Bernoulli distribution can be used as follows. If we carry out a coin toss (each coin toss counts as a single trial), we will either get a heads or a tails:

```
rbern(5, prob = 0.5)
```

```
## [1] 1 0 0 0 0
```

```
## equivalent rbinom command:
rbinom(5, size = 1, prob = 0.5)
```

```
## [1] 1 1 1 1 1
```

Thus, what the Bernoulli is to the binomial, the categorical is to the multinomial. For example, one can simulate five observations, each of which will give one of the three responses with the given probabilities. We do this with `rcat()` from the `extraDistr` package.

```
rcat(5, prob = c(0.1, 0.1, 0.8),
     labels = c("yes", "no", "dontknow"))
```

```
## [1] dontknow dontknow yes      yes      dontknow
## Levels: yes no dontknow
```

The above is analogous to using the multinomial with `size = 1` (a single trial in each experiment). In the output below, the `rmultinom()` function shows which of the three categories is produced.

```
rmultinom(5, size = 1,
          prob = c(0.1, 0.1, 0.8))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    1    1    1    1    1
```

With these distributions as background, consider now a simulated situation where multiple responses are possible.

A model for multiple responses using the multinomial likelihood

Impaired picture naming (anomia) is a common form of aphasia. It is assessed as part of most comprehensive aphasia test batteries, since picture naming accuracy is relatively easily obtained and is a reliable test score; in addition, the types of errors that are committed can provide useful information for diagnosis.

In this simulated experiment, the responses are categorized as shown below.

Table 1: Categorization of responses for the simulated experiment.

Type	Description	Example
Correct	The response matches the target.	cat
Neolog.	The response is not a word, but it has a phonological relation to the target.	cag
Formal	The response is a word with only a phonological relation to the target.	hat
Mixed	The response is a word with both a semantic and phonological relation the target.	rat
NR	All other responses, including omissions, descriptions, non-nouns, etc.	–

First, generate data assuming a multinomial distribution. The outcomes will be determined by a vector $\boldsymbol{\theta}$ (called `true_theta` below in the R code) that indicates the probability of each outcome:

```
(true_theta <- tibble(theta_NR = 0.2,
                      theta_Neologism = 0.1,
                      theta_Formal = 0.2,
                      theta_Mixed = 0.08,
                      theta_Correct =  1 -
 (theta_NR +
    theta_Neologism +
    theta_Formal +
     theta_Mixed)))
```

```
## # A tibble: 1 x 5
##   theta_NR theta_Neologism theta_Formal theta_Mixed theta_Correct
##      <dbl>           <dbl>        <dbl>       <dbl>         <dbl>
## 1      0.2             0.1          0.2        0.08          0.42
```

The probabilities must sum to 1:

```
sum(true_theta)
```

```
## [1] 1
```

Given this vector of probabilities $\boldsymbol{\theta}$, generate values assuming a multinomial distribution of responses in 100 trials:

```
N_trials <- 100
(ans_mn <- rmultinom(1, N_trials, true_theta))
```

```
##                  [,1]
## theta_NR           19
## theta_Neologism     7
## theta_Formal       17
## theta_Mixed         7
## theta_Correct      50
```

Now, we'll try to recover the probability of each answer with a model with the following likelihood:

$$ans \sim Multinomial(\boldsymbol{\theta}) \tag{1}$$

where $\boldsymbol{\theta} = \langle \theta_{nr}, \theta_{neol.}, \theta_{formal}, \theta_{mix}, \theta_{corr} \rangle$.

A common prior for the vector of probability parameters of a multinomial distribution is the Dirichlet distribution, which extends the Beta distribution to cases where more than two categories are available.

$$\boldsymbol{\theta} \sim Dirichlet(\boldsymbol{\alpha}) \tag{2}$$

The Dirichlet distribution has a parameter α, called the concentration parameter, and it is a

vector with the same length as $\boldsymbol{\theta}$. If we set $\boldsymbol{\alpha} = \langle 2, 2, 2, 2, 2 \rangle$, this is analogous to $\sim Beta(2, 2)$. The intuition behind this concentration parameter is that the prior probability distribution of the vector $\boldsymbol{\theta}$ corresponds to having seen each outcome twice.

A Stan model assuming a multinomial likelihood and Dirichlet prior is shown below. Since the elements of $\boldsymbol{\theta}$ should sum to one, we declare this vector, `theta`, is of type `simplex`. The simplex type ensures that its elements sum to one and also constrains them to have non-negative values. In order to generate the vector $\boldsymbol{\alpha}$ that contains five times the value two, we use `rep_vector(2, 5)` (which is similar to `rep(2, 5)` in R).

```
data {
  int<lower = 1> N_trials;
  array[5] int<lower = 0,upper = N_trials> ans;
}
parameters {
  simplex[5] theta;
}
model {
  target += dirichlet_lpdf(theta | rep_vector(2, 5));
  target += multinomial_lpmf(ans | theta);
}
generated quantities{
  array[5] int pred_ans = multinomial_rng(theta, N_trials);
}
```

Fit the model:

```
# Create a list:
# c(ans_mn) makes a vector out of the matrix ans_mn
data_mn <-  list(N_trials = N_trials,
                 ans = c(ans_mn))
str(data_mn)
```

```
multinom <- system.file("stan_models",
                        "multinom.stan",
                        package = "bcogsci")
fit_mn <- stan(multinom, data = data_mn)
```

Print the posteriors:

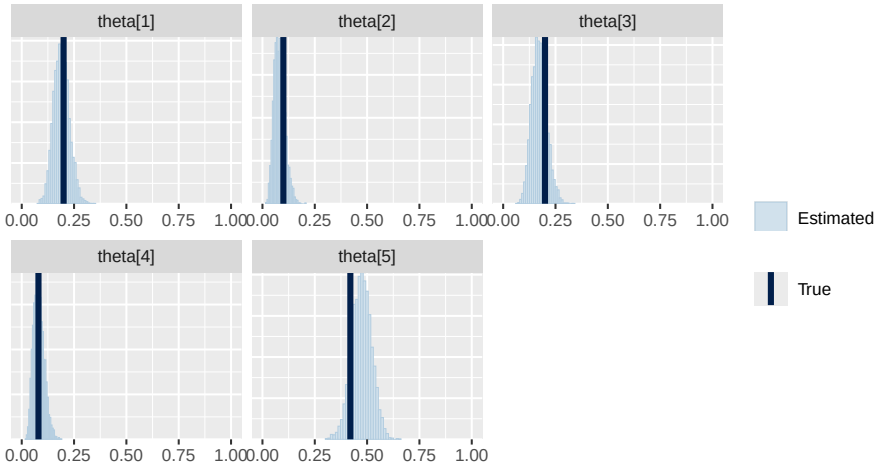
```
print(fit_mn, pars = c("theta"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## theta[1] 0.19       0 0.04 0.12 0.16 0.19 0.21  0.27  4462    1
## theta[2] 0.08       0 0.03 0.04 0.06 0.08 0.10  0.14  4055    1
## theta[3] 0.17       0 0.04 0.11 0.15 0.17 0.20  0.25  4520    1
## theta[4] 0.08       0 0.03 0.04 0.06 0.08 0.10  0.14  4156    1
## theta[5] 0.47       0 0.05 0.38 0.44 0.47 0.51  0.57  4393    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 15 10:35:16 2026.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

Next, use `mcmc_recover_hist()` in the code below to confirm that the posterior distributions of the elements of $\boldsymbol{\theta}$ are close to the true point values that were set up when simulating the data. See the figure below.

Posterior distributions and true means of `theta` for the multinomial model defined in `multinom.stan`:

```
as.data.frame(fit_mn) %>%
  select(starts_with("theta")) %>%
  mcmc_recover_hist(true = unlist(true_theta)) +
  coord_cartesian(xlim = c(0, 1))
```



Here, we evaluate whether our model is able to "recover" the true point values of its parameters. By "recover," we mean that the true point values are somewhere inside the posterior distribution of the model.

The frequentist properties of Bayesian models guarantee that if we simulate data several times, 95% of the true values should be inside of the 95% CrI intervals generated by a "well-calibrated" model. Furthermore, if the true values of some parameters are consistently well above or below their posterior distribution, it may mean that there is some problem with the model specification. For now we are going to verify that our model is roughly correct. A more principled (and computationally demanding) approach uses simulation based calibration (SBC); this is discussed in the textbook and in the online supplement provided with the textbook.

A model for multiple responses using the categorical distribution

Using the same information as above, we can model each response one at a time, instead of

aggregating them. Using the categorical distribution gives us more flexibility to define what happens at every trial. However, we are not using the additional flexibility yet. As a result, the next model and the previous one are equivalent.

```
data {
  int<lower = 1> N_obs;
  array[N_obs] int<lower = 1, upper = 5> w_ans;
}
parameters {
  simplex[5] theta;
}
model {
  target += dirichlet_lpdf(theta | rep_vector(2, 5));
  for(n in 1:N_obs)
    target += categorical_lpmf(w_ans[n] | theta);
}
generated quantities{
  array[N_obs] int pred_w_ans;
  for(n in 1:N_obs)
    pred_w_ans[n] = categorical_rng(theta);
}
```

Given the same set of probabilities $\boldsymbol{\theta}$ as above, generate 100 individual observations using `rcat()` from the `extraDistr` package:

```
N_obs <- 100
ans_cat <- rcat(N_obs, prob = as.matrix(true_theta))
```

The above output is how Stan expects to see the data. The data fed into the Stan model is defined as a list as usual:

```
data_cat <-  list(N_obs = N_obs,
                  w_ans = ans_cat)
str(data_cat)
```

```
## List of 2
##  $ N_obs: num 100
##  $ w_ans: num [1:100] 3 2 5 1 3 5 1 4 2 1 ...
```

Fitting the Stan model (`categorical.stan`)

should yield approximately the same $\boldsymbol{\theta}$ as with the multinomial likelihood defined in the model `multinom.stan`.

```
categorical <- system.file("stan_models",
                           "categorical.stan",
                           package = "bcogsci")
fit_cat <- stan(categorical, data = data_cat)
```

```
print(fit_cat, pars = c("theta"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##           mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## theta[1] 0.19       0 0.04 0.12 0.16 0.19 0.22  0.27  4548    1
## theta[2] 0.09       0 0.03 0.05 0.07 0.09 0.11  0.15  4895    1
## theta[3] 0.25       0 0.04 0.18 0.23 0.25 0.28  0.34  4580    1
## theta[4] 0.05       0 0.02 0.02 0.04 0.05 0.07  0.10  4237    1
## theta[5] 0.41       0 0.05 0.32 0.38 0.41 0.44  0.50  3967    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 15 10:36:56 2026.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

The above models estimate the posterior distribution for the probability for each possible response. The slightly different posteriors with the two methods are due to differences in the randomly generated data and the sampling process, not differences in grouping. If we had some experimental manipulation, we could even fit regressions to these parameters. This is called a multinomial logistic regression or categorical regression; see further readings in the textbook chapter for some examples.

Modeling picture naming abilities in aphasia with MPT models

Multinomial processing tree (MPT) modeling is a method that estimates latent variables that have a psychological interpretation given categorical data. In other words, an MPT model is just one way to model categorical responses following a multinomial or categorical distribution. MPT models assume that the observed response categories result from a sequences of underlying cognitive events which are represented as a binary branching tree. Each binary branching is associated with a parameter that represents the probability of going down either branch. Every successive node is assumed to be independent of the preceding node, allowing us to use the product rule from probability theory to compute the probability of going down a particular path. The leaves of the binary branching tree are the observed response in the data. The goal is to derive posterior distributions of the latent probability parameters specified for the binary branching in the model.

Here is an MPT model[1] that specifies a set of possible internal errors that lead to the various possible response types during a picture naming trial for persons with aphasia. Here we'll explore a simplification of the original model.

The model assumes that when an attempt is made to produce a word, errors in production

[1]The original work is reported in Walker, Grant M, Gregory Hickok, and Julius Fridriksson. 2018. "A Cognitive Psychometric Model for Assessment of Picture Naming Abilities in Aphasia." Psychological Assessment 6: 809–26.

can arise at the whole word level (lexical level) or the segmental level (phonological level). Semantic errors are assumed to arise from the lexical substitutions, and neologism errors from phonological substitutions. Real word responses that are phonologically related to the correct target word can arise from substitutions at the lexical or phonological level.

The task for the subject is to view a picture and name the object represented in the picture. When an attempt is made to retrieve the word from memory, the following possible steps can unfold (this is a simplified version of the original model):

- Either the subject will make some **lexical selection**, or fail to make a lexical selection, returning a non-response (NR). The probability of making some lexical selection is a, so the probability of a non-response is $1 - a$, as these are only two possibilities at this initial stage of the binary branching tree. Example: the subject sees the picture of a cat, and either produces the response "I don't know", or starts the process of producing a word.
- If a lexical selection is made, the target word is selected with probability t, or some other word is chosen with probability $1 - t$.
- Once a word is selected, either its phonological representation is selected with probability f, or some other (incorrect) phonological representation is selected with probability

$1 - f$.

- Once a word is selected, there can be a phonological change that leads to a real, formally related word with probability c, or a neologism with probability $1 - c$. Example: the subjects produces either a formally related word "hat," or a neologism like "cag."

The end-result of walking down this tree is that the subject either produces a non-response ("I don't know" or silence), a correct response, a related word, a mixed word, or a neologism. There is more than one way to produce a neologism or a related word, and the posterior probabilities of the various paths will determine the probability of each possible path.

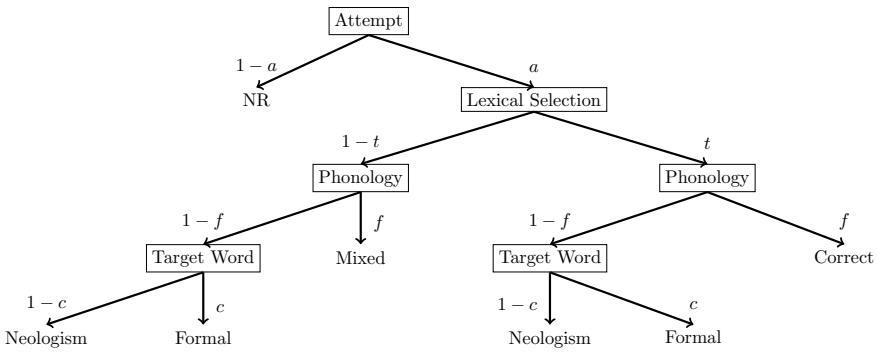


Figure 1: Representation of a simplification of the MPT used in Walker et al. 2018.

Table 2: Psychological interpretation of the parameters of the MPT model.

Par.	Meaning
a	Probability of initiating an attempt
t	Probability of selecting a target word over competitors
f	Probability of retrieving correct phonemes

Par.	Meaning
c	Probability of a phoneme change in the target creating a real word

Calculation of the probabilities in the MPT branches

By navigating through the branches of the MPT, we can calculate the probabilities of the five responses (the categorical outcomes), based on the four underlying parameters assumed in the MPT:

- $P(NR|a,t,f,c) = 1-a$
- $P(Neologism|a,t,f,c) = a\cdot(1-t)\cdot(1-f)\cdot(1-c) + a\cdot t\cdot(1-f)\cdot(1-c)$
- $P(Formal|a,t,f,c) = a\cdot(1-t)\cdot(1-f)\cdot c + a\cdot t\cdot(1-f)\cdot c$
- $P(Mixed|a,t,f,c) = a\cdot(1-t)\cdot f$
- $P(Correct|a,t,f,c) = a\cdot t\cdot f$

Given that

$$P(NR|a,t,f,c)+P(Neologism|a,t,f,c)+P(Formal|a,t,f,c)+P(Mixed|a,t,f,c)+P(Correct|a,t,f,c)=1 \quad (3)$$

there is no need to characterize every outcome: we can always calculate any one of the remaining responses as one minus the other responses.

A simple MPT model

First, simulate 200 trials assuming no variability between items and subjects. It is convenient to define functions to compute each outcome's probability, based on the previous MPT. One needs to assign "true values" to the underlying parameters of the MPT; these point values are only for illustration. Ideally, one should simulate data using parameter values that are realistic; that is, one should use values based on the literature, including the uncertainty of these parameters.

```
# The probabilities of the different answers:
Pr_NR <- function(a, t, f, c)
    1 - a
Pr_Neologism <- function(a, t, f, c)
    a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 - f) * (1 - c)
Pr_Formal <- function(a, t, f, c)
    a * (1 - t) * (1 - f) * c +  a * t * (1 - f) * c
Pr_Mixed <- function(a, t, f, c)
    a * (1 - t) * f
Pr_Correct <- function(a, t, f, c)
    a * t * f
# The true underlying values for simulated data:
a_true <- 0.75
t_true <- 0.9
f_true <- 0.8
c_true <- 0.1
# The probability of the different answers:
Theta <-
  tibble(NR = Pr_NR(a_true, t_true, f_true, c_true),
         Neologism = Pr_Neologism(a_true, t_true, f_true, c_true),
         Formal = Pr_Formal(a_true, t_true, f_true, c_true),
         Mixed = Pr_Mixed(a_true, t_true, f_true, c_true),
         Correct = Pr_Correct(a_true, t_true, f_true, c_true))
N_trials <- 200
(ans <- rmultinom(1, N_trials, c(Theta)))
```

```
##           [,1]
## NR          49
## Neologism   26
## Formal       5
## Mixed       17
## Correct    103
```

The above data can be modeled in Stan as

discussed below (see `mpt_mnm.stan`). The probabilities of the different categories go into the `transformed parameters` section because they are derived from the probability parameters in the model. The data are modeled as coming from a multinomial likelihood. If priors are not specified, then a Beta distribution with $a = 1$ and $b = 1$ (a Uniform(0,1) distribution) is assumed for the parameters a, t, f, and c. Unlike $\boldsymbol{\theta}$, the values of these parameters are independent of each other and they do not sum to one. For this reason, we should not use a Dirichlet prior here.

We define the following model:

$$
\begin{aligned}
\theta_{nr} &= 1 - a \\
\theta_{neol.} &= a \cdot (1 - t) \cdot (1 - f) \cdot (1 - c) + a \cdot t \cdot (1 - f) \cdot (1 - c) \\
\theta_{formal} &= a \cdot (1 - t) \cdot (1 - f) \cdot c + a \cdot t \cdot (1 - f) \cdot c \\
\theta_{mix} &= a \cdot (1 - t) \cdot f \\
\theta_{corr} &= a \cdot t \cdot f \\
\boldsymbol{\theta} &= \langle \theta_{nr}, \theta_{neol.}, \theta_{formal}, \theta_{mix}, \theta_{corr} \rangle \\
ans &\sim Multinomial(\theta) \\
a, t, f, c &\sim Beta(2, 2)
\end{aligned}
\tag{4}
$$

This translates to the following code:

```
data {
  int<lower = 1> N_trials;
  array[5] int<lower = 0, upper = N_trials> ans;
}
parameters {
  real<lower = 0, upper = 1> a;
  real<lower = 0, upper = 1> t;
  real<lower = 0, upper = 1> f;
  real<lower = 0, upper = 1> c;
}
```

```
transformed parameters {
  simplex[5] theta;
  theta[1] = 1 - a; //Pr_NR
  theta[2] = a * (1 - t) * (1 - f) * (1 - c) +
    a * t * (1 - f) * (1 - c); //Pr_Neologism
  theta[3] = a * (1 - t) * (1 - f) * c
    + a * t * (1 - f) * c;  //Pr_Formal
  theta[4] = a * (1 - t) * f; //Pr_Mixed
  theta[5] = a * t * f; //Pr_Correct
}
model {
  target += beta_lpdf(a | 2, 2);
  target += beta_lpdf(t | 2, 2);
  target += beta_lpdf(f | 2, 2);
  target += beta_lpdf(c | 2, 2);
  target += multinomial_lpmf(ans | theta);
}
generated quantities{
    array[5] int pred_ans;
  pred_ans = multinomial_rng(theta, N_trials);
}
```

Fit the model:

```
data_sMPT <-  list(N_trials = N_trials,
                   ans = c(ans))
```

```
mpt_mnm <- system.file("stan_models",
                       "mpt_mnm.stan",
                       package = "bcogsci")
fit_sMPT <- stan(mpt_mnm, data = data_sMPT)
```

Print out a summary of the posterior of the parameter of interest:

```
print(fit_sMPT, pars = c("a", "t", "f", "c"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##   mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## a 0.75       0 0.03 0.69 0.73 0.75 0.77  0.81  4687    1
## t 0.85       0 0.03 0.78 0.83 0.85 0.87  0.90  4699    1
## f 0.79       0 0.03 0.72 0.77 0.79 0.81  0.85  4514    1
## c 0.20       0 0.07 0.08 0.15 0.20 0.25  0.36  4069    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 15 10:37:19 2026.
## For each parameter, n_eff is a crude measure of effective sample size,
```

```
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

What the model gives us is posterior distributions of each of the parameters `a`, `t`, `f`, `c`. From these we can derive the probabilities of producing the different observed responses, and the posterior predictive distributions, which could be used for model evaluation.

An important sanity check in modeling is checking whether the model can in principle recover the true parameters that generated the data; see the figure below.

```
as.data.frame(fit_sMPT) %>%
  select(c("a","t","f","c")) %>%
  mcmc_recover_hist(true = c(a_true,
                             t_true,
                             f_true,
                             c_true)) +
  coord_cartesian(xlim = c(0, 1))
```

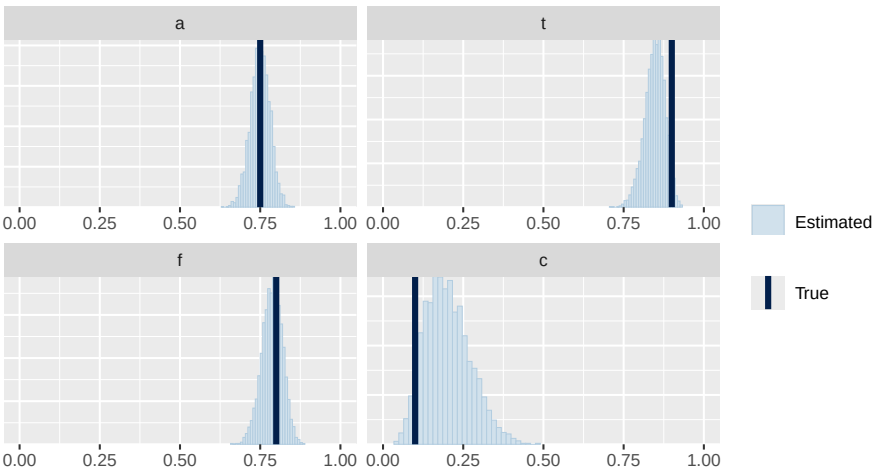


Figure 2: Posterior distributions and true point values of the parameters of the simple MPT model (`mpt_mnm.stan`).

The above figure shows that the model can indeed recover the true parameters fairly accurately.

The posterior distributions of the $\boldsymbol{\theta}$ parameters can also be summarized:

```
print(fit_sMPT, pars = c("theta"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd 2.5%  25%  50%  75% 97.5% n_eff Rhat
## theta[1] 0.25       0 0.03 0.19 0.23 0.25 0.27  0.31  4687    1
## theta[2] 0.13       0 0.02 0.08 0.11 0.13 0.14  0.17  4418    1
## theta[3] 0.03       0 0.01 0.01 0.02 0.03 0.04  0.06  3793    1
## theta[4] 0.09       0 0.02 0.06 0.08 0.09 0.10  0.13  4732    1
## theta[5] 0.50       0 0.03 0.43 0.48 0.50 0.52  0.57  4548    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 15 10:37:19 2026.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

These posteriors tell us the probability of producing each of the possible responses. This model might be useful for estimating the latent parameters, a, t, f, c, but without further constraints it is unfalsifiable.

Recall that for the multinomial likelihood, we had a simplex of size five, which means that we had four free parameters (since the fifth can be deduced based on the others). With five possible answers we can always estimate a vector of probabilities $\boldsymbol{\theta}$ that fits the data, in the same way that with two possible answers (e.g., zeros and ones), we can always estimate a single probability θ that fits the data (using a Bernoulli or binomial likelihood).

The MPT that we present here is just reparameterizing the vector $\boldsymbol{\theta}$ of the multinomial likelihood (with the same number of free parameters). This means that it will always achieve a perfect fit. This doesn't mean that this MPT model is

"useless": Under the assumption that the model is theoretically meaningful, one can estimate its latent parameters and this estimation can have theoretical implications. If we want to be able to falsify this model, we'll need to constrain it more, as we suggest below.

An MPT model assuming by-item variability

The use of aggregated data implies the assumption that the estimated parameters do not vary too much between subjects and items. If this assumption is incorrect, the analysis of aggregated data may lead to erroneous conclusions: reliance on aggregated data in the presence of parameter heterogeneity may lead to biased parameter estimates and the underestimation of credible intervals.

If it is known that f is affected by the phonological complexity of the individual word (e.g., *cat* is easier to produce than *umbrella*), the previous model does not have a way to include that information.

Simulated data can be generated taking into account the `complexity` of the items. Assume here for simplicity that the complexity of items is given as data and it is scaled and centered; i.e., mean complexity is represented by 0, and the standard deviation is assumed to be 1. We will assume a regression model that determines the parameter, f, as a function of the phonological complexity of each trial.

One important detail is that `f` is a probability

and needs to be bounded between 0 and 1. To make sure that this property is met, the computation of `f` for each item will be converted to probability space using the logistic function. This is achieved as follows.

Suppose that f' is a linear function of complexity. For example, two parameters α_f and β_f (intercept and slope respectively) could determine how f' is affected by complexity:

$f'_j = \alpha_f + complexity_j \cdot \beta_f.$

The parameters α_f and β_f are defined in an unconstrained log-odds space (they can be any real number). The model that is fit then yields an f'_j value for each item j in log-odds space. The log-odds value f'_j can be converted to a probability value f_{true} by applying the logistic function (or the inverse logit, $logit^{-1}$) to f'. Recall from the generalized linear model discussed earlier that if we have a model in log-odds space:

$$\log\left(\frac{p_j}{1-p_j}\right) = \alpha + \beta \cdot x_j = \mu_j \qquad (5)$$

Then we can recover the probability p_j by solving for p_j:

$$p_j = \frac{\exp(\mu_j)}{1+\exp(\mu_j)} = \frac{1}{1+\exp(-\mu_j)} \qquad (6)$$

The above is the logistic or inverse logit function: it takes as input μ_j and returns the corresponding probability p_j. The `plogis()` function in R carries out the calculation shown above.

```r
N_obs <- 50
complexity <- rnorm(N_obs) # by default mean = 0, sd = 1
## choose some hypothetical values:
alpha_f <- 0.3
# the negative sign indicates that
# increased complexity will lead to a reduced value of f:
beta_f <- -0.3
# f' as a linear function of complexity:
f_prime <- alpha_f + complexity * beta_f
head(f_prime,n=3)
```

```
## [1]  0.4681427  0.3690532 -0.1676125
```

```r
## probabilities f for each item:
f_true <- plogis(f_prime)
head(f_true,n=3)
```

```
## [1] 0.6149441 0.5912302 0.4581947
```

This change in our assumptions entails that the probability of each response changes depending on the item associated with each observation. The parameters `theta` now have to be a matrix. This is in R; in Stan, we will code it as an array of simplexes, i.e., an array of non-negative values that sums to 1.

We continue with the functions defined earlier, and the same values for `a_true`, `t_true`, and `c_true` as defined in that section. Since most of the equations depend on `f`, and `f` is a vector now, the outcomes are automatically vectors. But this is not the case for `theta_NR_v`, and thus we need to repeat the value.

```r
theta_NR_v <- rep(Pr_NR(a_true, t_true, f_true, c_true), N_obs)
theta_Neologism_v <- Pr_Neologism(a_true, t_true, f_true, c_true)
```

```
theta_Formal_v <- Pr_Formal(a_true, t_true, f_true, c_true)
theta_Mixed_v <- Pr_Mixed(a_true, t_true, f_true, c_true)
theta_Correct_v <- Pr_Correct(a_true, t_true, f_true, c_true)
theta_item <- matrix(c(theta_NR_v,
                       theta_Neologism_v,
                       theta_Formal_v,
                       theta_Mixed_v,
                       theta_Correct_v),
                     ncol = 5)
dim(theta_item)
```

```
## [1] 50  5
```

```
head(theta_item,n = 3)
```

```
##      [,1]      [,2]       [,3]       [,4]      [,5]
## [1,] 0.25 0.2599128 0.02887920 0.04612080 0.4150872
## [2,] 0.25 0.2759196 0.03065774 0.04434226 0.3990804
## [3,] 0.25 0.3657186 0.04063540 0.03436460 0.3092814
```

Store this in a data frame:

```
sim_data_cx <- tibble(item = 1:N_obs,
                      complexity = complexity,
                      w_ans = c(rcat(N_obs,theta_item)))
sim_data_cx
```

```
## # A tibble: 50 x 3
##     item complexity w_ans
##    <int>      <dbl> <dbl>
##  1     1    -0.560      5
##  2     2    -0.230      2
##  3     3     1.56       2
##  4     4     0.0705     5
##  5     5     0.129      2
##  6     6     1.72       5
##  7     7     0.461      5
##  8     8    -1.27       5
##  9     9    -0.687      2
## 10    10    -0.446      1
## # i 40 more rows
```

The following model (saved in `mpt_cat.stan`)

is essentially doing the same thing as the previous model but instead of fitting a multinomial to the summary of all the trials, it is fitting a categorical distribution to each individual observation. (This is analogous to the difference between the Bernoulli and binomial distributions).

This is still not an appropriate model for the generative process that we are assuming in this section, because it still ignores the effect of complexity. But it is a good start.

```
data {
  int<lower = 1> N_obs;
  array[N_obs] int<lower = 1, upper = 5> w_ans;
}
parameters {
  real<lower=0,upper=1> a;
  real<lower=0,upper=1> t;
  real<lower=0,upper=1> f;
  real<lower=0,upper=1> c;
}
transformed parameters {
  array[N_obs] simplex[5] theta;
  for(n in 1:N_obs){
    //Pr_NR:
    theta[n, 1] = 1 - a;
    //Pr_Neologism:
    theta[n, 2] = a * (1 - t) * (1 - f) * (1 - c) +
      a * t * (1 - f) * (1 - c);
    //Pr_Formal:
    theta[n, 3] = a * (1 - t) * (1 - f) * c +
      a * t * (1 - f) * c;
    //Pr_Mixed:
```

```
    theta[n, 4] = a * (1 - t) * f;
    //Pr_Correct:
    theta[n, 5] = a * t * f;
  }
}
model {
  target += beta_lpdf(a | 2, 2);
  target += beta_lpdf(t | 2, 2);
  target += beta_lpdf(f | 2, 2);
  target += beta_lpdf(c | 2, 2);
  for(n in 1:N_obs)
    target += categorical_lpmf(w_ans[n] | theta[n]);
}
generated quantities{
    array[N_obs] int pred_w_ans;
  for(n in 1:N_obs)
    pred_w_ans[n] = categorical_rng(theta[n]);
}
```

An important aspect of the previous model is that `theta` is declared as `array[N_obs] simplex[5] theta;`. This means that `theta` is an array of simplexes and thus has now two dimensions: each element of the array (of length `N_obs`) is a simplex and sums to one. That's why we iterate over the `N_obs`. However, one limitation of the previous model is that the latent parameters `a`, `t`, `f`, `c` are declared as `real` and they do not vary in each iteration of the loop.

A hierarchical MPT

The previous model doesn't take into account that subjects might vary. Let's focus on taking into account the differences between subjects.

Different subjects might not be equally motivated to do the task. This can be accounted for by adding hierarchical structure to the parameter `a`, the probability of initiating an attempt. Begin by simulating some data that incorporates by-subject variability.

First, define the number of items and subjects, and the number of observations:

```
N_item <- 20
N_subj <- 30
N_obs <- N_item * N_subj
```

Then, generate a vector for subjects and for items. Assume here that each subject sees each item.

```
subj <- rep(1:N_subj, each = N_item)
item <- rep(1:N_item, time = N_subj)
```

A vector representing complexity is created for the number of items we have, and this vector is repeated as many times as there are subjects:

```
complexity <- rep(rnorm(N_item), times = N_subj)
```

Next, create a data frame with all the above information:

```
(exp_sim <- tibble(subj = subj,
                   item = item,
```

```
                    complexity = complexity))
```

```
## # A tibble: 600 x 3
##     subj  item complexity
##    <int> <int>      <dbl>
##  1     1     1    -0.560
##  2     1     2    -0.230
##  3     1     3     1.56
##  4     1     4     0.0705
##  5     1     5     0.129
##  6     1     6     1.72
##  7     1     7     0.461
##  8     1     8    -1.27
##  9     1     9    -0.687
## 10     1    10    -0.446
## # i 590 more rows
```

To create subject-level variability in the data, a between-subject standard deviation needs to be defined. This standard deviation represents the deviations of subjects about the grand mean. We are defining this adjustment in log-odds space.

```
# New parameters, in log-odds space:
tau_u_a <- 1.1
## generate subject adjustments in log-odds space:
u_a <- rnorm(N_subj, 0, tau_u_a)
str(u_a)
```

```
##  num [1:30] -1.175 -0.24 -1.129 -0.802 -0.688 ...
```

Given the fixed `a_true` probability value of 0.75, the subject-level values for individual `a_true` can be derived by (a) first converting the overall `a_true` value to log-odds space, (b) adding the by-subject adjustment to this converted overall

value, and (c) then converting back to probability space using the logistic or inverse logit (`plogis()`) function. Essentially we generate data assuming the following:

$$\begin{aligned} a'_{h,n} &= \alpha_a + u_{a,subj[n]} \\ a_{h,n} &= \text{logit}^{-1}(a'_{h,n}) \end{aligned} \tag{7}$$

where $u_{a,subj[n]}$ is a vector with the same length as the total number of observations.

This is done in R as follows:

```
a_true <- 0.75 # as before
## convert the intercept to log-odds space:
alpha_a <- qlogis(a_true)
## a_h' in log-odds space:
a_h_prime <-  alpha_a + u_a[subj]
## convert back to probability space
a_true_h <- plogis(a_h_prime)
str(a_true_h)
```

```
##  num [1:600] 0.481 0.481 0.481 0.481 0.481 ...
```

What this achieves mathematically is adding varying intercepts by subjects to `alpha_a`, and then the values adjusted by subject are saved in probability space.

As before, `f_true` is computed as a function of complexity:

```
alpha_f <- 0.3; beta_f <- -0.3
f_true <- plogis(alpha_f + complexity * beta_f)
```

We continue with the same probability functions and the rest of the true point values remain the same as well.

```
t_true <- 0.9; c_true <- 0.1
Pr_NR <- function(a, t, f, c)
    1 - a
Pr_Neologism <- function(a, t, f, c)
    a * (1 - t) * (1 - f) * (1 - c) + a * t * (1 - f) * (1 - c)
Pr_Formal <- function(a, t, f, c)
    a * (1 - t) * (1 - f) * c +  a * t * (1 - f) * c
Pr_Mixed <- function(a, t, f, c)
    a * (1 - t) * f
Pr_Correct <- function(a, t, f, c)
    a * t * f
```

Now, we can define the probabilities of different outcomes:

```
# Aux. parameters that define the probabilities:
theta_NR_v_h <- Pr_NR(a_true_h, t_true, f_true, c_true)
theta_Neologism_v_h <- Pr_Neologism(a_true_h, t_true, f_true, c_true)
theta_Formal_v_h <- Pr_Formal(a_true_h, t_true, f_true, c_true)
theta_Mixed_v_h <- Pr_Mixed(a_true_h, t_true, f_true, c_true)
theta_Correct_v_h <- Pr_Correct(a_true_h, t_true, f_true, c_true)
theta_h <- matrix(c(theta_NR_v_h,
                    theta_Neologism_v_h,
                    theta_Formal_v_h,
                    theta_Mixed_v_h,
                    theta_Correct_v_h),
                  ncol = 5)
dim(theta_h)
```

```
## [1] 600   5
```

The probability specifications shown above can now generate the simulated data:

```
(sim_data_h <- mutate(exp_sim,
                      w_ans = rcat(N_obs,theta_h)))
```

```
## # A tibble: 600 x 4
##     subj  item complexity w_ans
##    <int> <int>      <dbl> <dbl>
##  1     1     1    -0.560      2
##  2     1     2    -0.230      1
##  3     1     3     1.56       1
##  4     1     4     0.0705     5
##  5     1     5     0.129      1
##  6     1     6     1.72       5
##  7     1     7     0.461      5
##  8     1     8    -1.27       2
##  9     1     9    -0.687      1
## 10     1    10    -0.446      1
## # i 590 more rows
```

Next, define the following model; we omit the steps with f' and a' and directly apply the logistic function to a regression. The parameters t, c do not vary by item or subject and therefore do not have the

subscript $_n$. We start by defining relatively weak priors for all the parameters in the following model.

$$
\begin{aligned}
\alpha_a, \alpha_f &\sim Normal(0, 1.5) \\
\beta_f &\sim Normal(0, 1) \\
t, c &\sim Beta(2, 2) \\
\tau_u &\sim Normal_+(0, 1) \\
u_a &\sim Normal(0, \tau_{u_a}) \\
a_n &= logit^{-1}(\alpha_a + u_{a,subj[n]}) \\
f_n &= logit^{-1}(\alpha_f + complexity_n \cdot \beta_f) \\
\theta_{n,nr} &= 1 - a_n \\
\theta_{n,neol.} &= a_n \cdot (1-t) \cdot (1-f_n) \cdot (1-c) + a_n \cdot t \cdot (1-f_n) \cdot (1-c) \\
\theta_{n,formal} &= a_n \cdot (1-t) \cdot (1-f_n) \cdot c + a_n \cdot t \cdot (1-f_n) \cdot c \\
\theta_{n,mix} &= a_n \cdot (1-t) \cdot f_n \\
\theta_{n,corr} &= a_n \cdot t \cdot f_n \\
\theta_n &= \langle \theta_{n,nr}, \theta_{n,neol.}, \theta_{n,formal}, \theta_{n,mix}, \theta_{n,corr} \rangle \\
ans_n &\sim Categorical(\theta_n)
\end{aligned}
\tag{8}
$$

The corresponding Stan model `mpt_h.stan` will look like this:

```
data {
  int<lower = 1> N_obs;
  array[N_obs] int<lower = 1, upper = 5> w_ans;
  array[N_obs] real complexity;
  int<lower = 1> N_subj;
  array[N_obs] int<lower = 1, upper = N_subj> subj;
}
parameters {
  real<lower = 0, upper = 1> t;
  real<lower = 0, upper = 1> c;
  real alpha_a;
  real<lower = 0> tau_u_a;
  vector[N_subj] u_a;
  real alpha_f;
  real beta_f;
}
transformed parameters {
  array[N_obs] simplex[5] theta;
  for (n in 1:N_obs){
    real a = inv_logit(alpha_a + u_a[subj[n]]);
    real f = inv_logit(alpha_f + complexity[n] * beta_f);
    //Pr_NR
    theta[n, 1] = 1 - a;
    //Pr_Neologism
    theta[n, 2] = a * (1 - t) * (1 - f) * (1 - c) +
      a * t * (1 - f) * (1 - c);
    //Pr_Formal
    theta[n, 3] = a * (1 - t) * (1 - f) * c
      + a * t * (1 - f) * c;
    //Pr_Mixed
```

```
    theta[n, 4] = a * (1 - t) * f;
    //Pr_Correct
    theta[n, 5] = a * t * f;
  }
}
model {
  target += beta_lpdf(t | 2, 2);
  target += beta_lpdf(c | 2, 2);
  target += normal_lpdf(alpha_a | 0, 1.5);
  target += normal_lpdf(alpha_f | 0, 1.5);
  target += normal_lpdf(beta_f | 0, 1);
  target += normal_lpdf(u_a | 0, tau_u_a);
  target += normal_lpdf(tau_u_a | 0, 1) - normal_lccdf(0 | 0, 1);
  for(n in 1:N_obs)
    target +=  categorical_lpmf(w_ans[n] | theta[n]);
}
generated quantities{
  array[N_obs] int<lower = 1, upper = 5> pred_w_ans;
  for(n in 1:N_obs)
    pred_w_ans[n] = categorical_rng(theta[n]);
}
```

For ease of exposition, we are not using the **non-centered parameterization** discussed in a previous chapter. We could also apply it here; that will speed up and improve the convergence of the model.

It would be a good idea to plot prior predictive distributions for this model, but we skip this step here. Next, fit the model to the simulated data, by first defining the data as a list:

```
sim_list_h <-  list(N_obs = nrow(sim_data_h),
                    w_ans = sim_data_h$w_ans,
                    N_subj = max(sim_data_h$subj),
                    subj = sim_data_h$subj,
                    complexity = sim_data_h$complexity)
```

```
mpt_h <- system.file("stan_models",
                     "mpt_h.stan",
                     package = "bcogsci")
fit_mpt_h <- stan(mpt_h, data = sim_list_h)
```

Print out a summary of the posterior:

```
print(fit_mpt_h,
      pars = c("t", "c", "tau_u_a", "alpha_a", "alpha_f", "beta_f"))
```

```
## Inference for Stan model: anon_model.
## 4 chains, each with iter=2000; warmup=1000; thin=1;
## post-warmup draws per chain=1000, total post-warmup draws=4000.
##
##          mean se_mean   sd  2.5%   25%   50%   75% 97.5% n_eff Rhat
## t        0.91    0.00 0.02  0.87  0.90  0.91  0.92  0.94  4968    1
## c        0.11    0.00 0.02  0.07  0.10  0.11  0.13  0.16  5009    1
## tau_u_a  0.99    0.00 0.18  0.68  0.86  0.97  1.09  1.40  2407    1
## alpha_a  1.00    0.01 0.21  0.61  0.87  1.00  1.14  1.40  1572    1
## alpha_f  0.25    0.00 0.10  0.06  0.18  0.25  0.32  0.45  5350    1
## beta_f  -0.22    0.00 0.10 -0.42 -0.29 -0.22 -0.15 -0.03  4837    1
##
## Samples were drawn using NUTS(diag_e) at Sun Mar 15 10:38:09 2026.
## For each parameter, n_eff is a crude measure of effective sample size,
## and Rhat is the potential scale reduction factor on split chains (at
## convergence, Rhat=1).
```

If we had fit this to real data, we would now conclude that:

(i) given the value of `beta_f`, complexity has an adverse effect on the probability of retrieving the correct phonemes, and

(ii) given the posterior distribution of `tau_u_a`, there is a great deal of variation in the subjects' probability of initiating an attempt at each trial. Furthermore, if we had some expectation about t and c based on previous research, we could conclude that our results are in line (or not) with previous findings.

One could inspect how one unit of complexity is affecting the probability of retrieving the correct phoneme (f). We first derive the value of `f` for an item of zero complexity (that is $\alpha_f + 0 \times \beta_f$) and then the value of `f` for an item with a complexity

of one ($\alpha_f + 1 \times \beta_f$). We are interested in summarizing the difference between the two:

```
as.data.frame(fit_mpt_h) %>%
  select(alpha_f, beta_f) %>%
  mutate(f_0 = plogis(alpha_f),
         f_1 = plogis(alpha_f + beta_f),
         diff_f = f_1 - f_0) %>%
  summarize(Estimate = mean(diff_f),
            `2.5%` = quantile(diff_f, 0.025),
            `97.5%` = quantile(diff_f, 0.975))
```

```
##      Estimate       2.5%        97.5%
## 1 -0.05524616 -0.1050317 -0.006829887
```

It is worth highlighting that MPT models (like other cognitive models) should be experimentally validated before interpreting the parameters as measures of cognitive processes. Particularly powerful are tests of selective influence, where one demonstrates that each of the parameters can be selectively influenced as theoretically predicted, while all other parameters remain unaffected.

One further interesting step could be to develop a competing model that assumes a different latent process, and then comparing the performance of the MPT with this competing model, using Bayes factors or K-fold-CV (or both).

Since we generated the data based on known latent parameters, we also plot the posteriors together with the true point values of the parameters in the figure below. This is something that we can only do with simulated data.

```
as.data.frame(fit_mpt_h) %>%
  select(tau_u_a, alpha_a, t, alpha_f, beta_f, c) %>%
  mcmc_recover_hist(true = c(tau_u_a,
                             qlogis(a_true),
                             t_true, alpha_f,
                             beta_f, c_true))
```

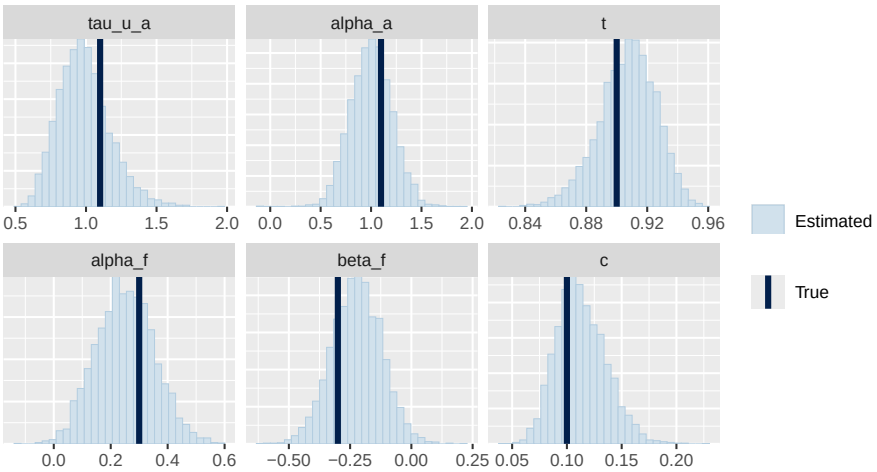


Figure 3: Posterior of the hierarchical MPT with true point values as vertical lines (model `mpt_h.stan`).

If everything is correctly defined in the model, we should be able to generate posterior predictive data based on our estimates that looks quite similar to the simulated data; see the figure below. The error bars in y_{rep} include 90% of the probability mass of the predictive distribution (this is a default of `ppc_bars()`). In a well-calibrated model, the data (y, here the proportion of answers of each type) should be inside the error bars in 90% of the cases.

```
gen_data <- rstan::extract(fit_mpt_h)$pred_w_ans
ppc_bars(sim_list_h$w_ans, gen_data) +
  ggtitle ("Hierarchical model")
```

It is also useful to look at the individual subjects' posteriors; these are shown in the figure below.

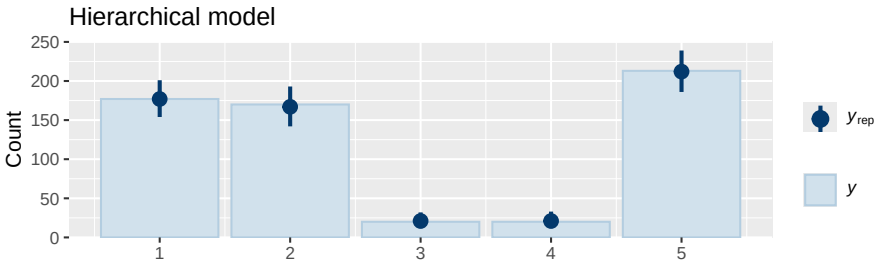


Figure 4: A posterior predictive check for aggregated data in the hierarchical MPT model.

```
ppc_bars_grouped(sim_list_h$w_ans,
                 gen_data, group = subj) +
  ggtitle ("By-subject plot for the hierarchical model")
```

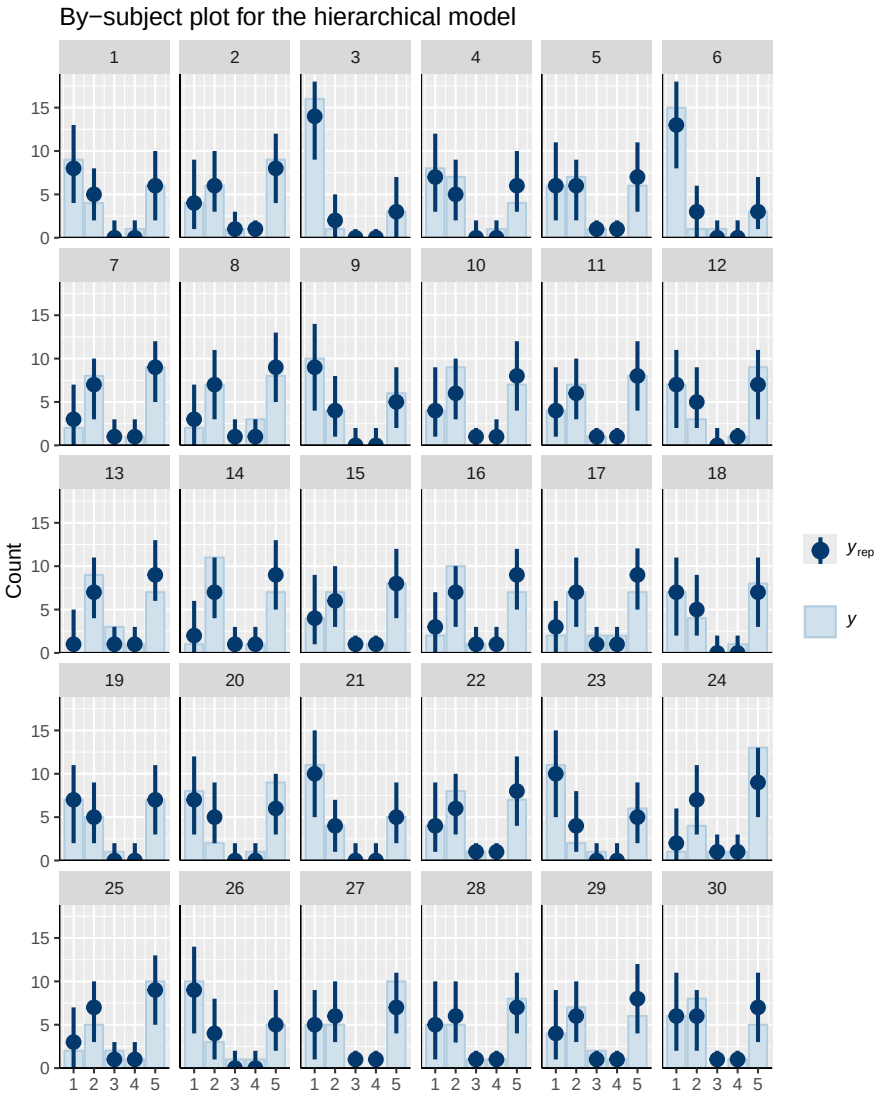


Figure 5: Individual subjects in the hierarchical MPT model.

But what about the first *non-hierarchical* MPT model (`mpt_cat.stan`)?

```
mpt_cat <- system.file("stan_models",
                       "mpt_cat.stan",
                       package = "bcogsci")
fit_sh <- stan(mpt_cat, data = sim_list_h)
```

The aggregated data looks great (see the figure below).

```
gen_data_sMPT <- rstan::extract(fit_sh)$pred_w_ans

ppc_bars(sim_list_h$w_ans, gen_data_sMPT) +
  ggtitle ("Non-hierarchical model")
```

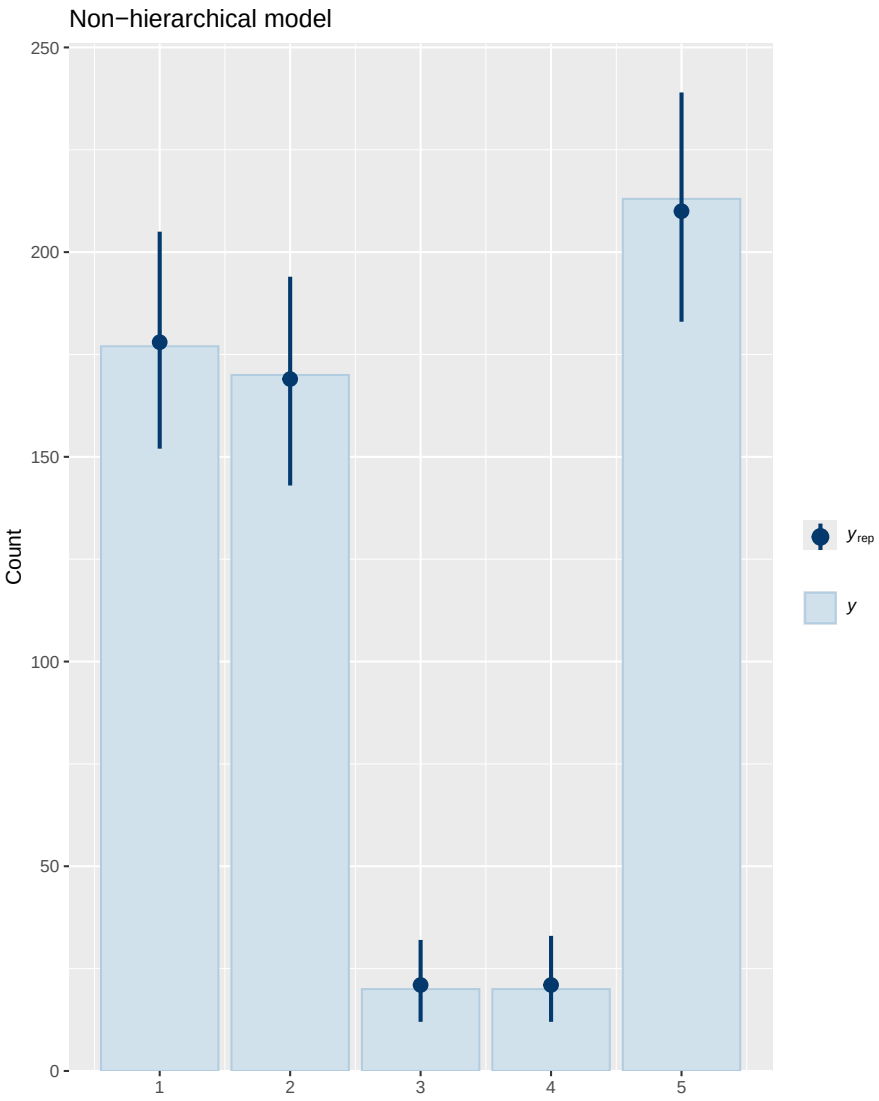


Figure 6: Posterior predictive check for aggregated data in a non-hierarchical MPT model (`mpt_cat.stan`).

However, in the non-hierarchical model, the fit to individual subjects is not as good (see the figure below): The error bars of the predicted distribution is of course identical for each subject (up to sampling variability) and do not include the observed proportion of answers for many of the subjects.

```
ppc_bars_grouped(sim_list_h$w_ans, gen_data_sMPT, group = subj) +
  ggtitle ("By-subject plot for the non-hierarchical model")
```

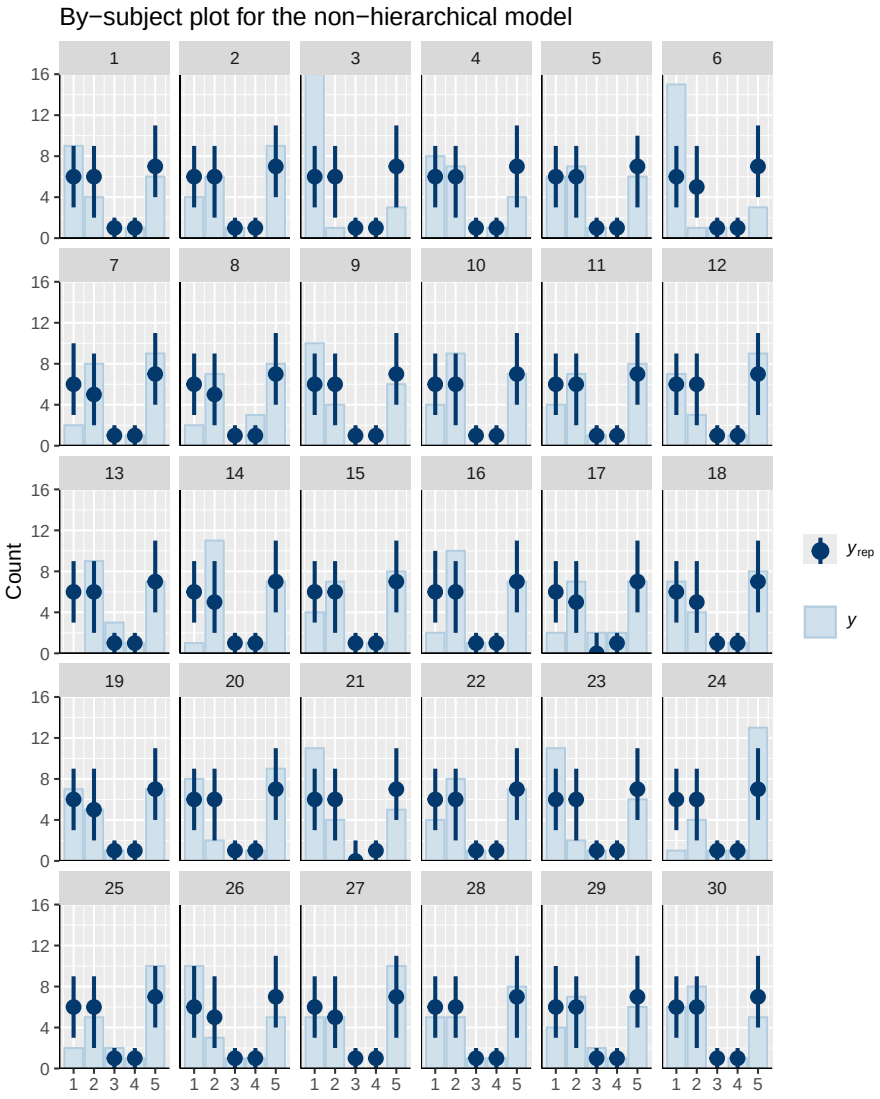


Figure 7: Individual subjects in the non-hierarchical MPT model (`mpt_cat.stan`).

The hierarchical model does a better job of modeling individual-level variability.

Some examples from my lab of papers using MPTs

Dario Paape (https://d-paape.github.io/) is the most prolific user in my lab of MPTs.

- Dario Paape and Shravan Vasishth. Context ameliorates but does not eliminate garden-pathing: Novel insights from latent-process modeling. Journal of Memory and Language, 2026.
- Dario Paape and Shravan Vasishth. Estimating the true cost of garden-pathing: A computational model of latent cognitive processes. Cognitive Science, 46:e13186, 2022.