Supplement to 'A Tutorial on Cognitive Modeling for Cognitive Aging Research'

Nathaniel R. Greene[1] & Stephen Rhodes[2]

[1] University of Missouri
[2] Rotman Research Institute

Contents

Materials available at https://github.com/stephenrho/cognitive-aging-modeling

Correspondence concerning this article should be addressed to Nathaniel R. Greene, 9J McAlester Hall, Department of Psychological Sciences, University of Missouri, Columbia, MO 65211. E-mail: ngreene@mail.missouri.edu

Getting and running the example code

The example code is available on github at https://github.com/stephenrho/cognitive-aging-modeling. You can download the entire repository as a zip folder by clicking the green "code" button and then "Download ZIP". Alternatively you can "fork" the project.[1]

- `models/` - contains the `stan` implementations of the signal detection theory models discussed in the tutorial
- `data/` - contains the data set used in the example analysis
- `simulate-data.R` - contains the code used to simulate the data set
- `fit_SDT.R` - uses the `rstan` package to fit the models
- `MPT` - contains the files needed to fit the multinomial processing tree model

`fit_SDT.R` is the main script of interest, although we also suggest opening each relevant `.stan` file when working through the examples.

To run the code you will need to download `R` (https://www.r-project.org/) and we strongly recommend `R studio` (https://rstudio.com/). `R` has packages that extend its base functionality. For the examples covered here you will need to run the following code to install the required packages (this will require an internet connection and may take some time to run):

```r
install.packages(pkgs = c("rstan",
                          "bridgesampling",
                          "loo",
                          "bayesplot",
                          "HDInterval")
                 )
```

We also recommend visiting https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started for more information on installing `rstan`. Mac users may need to install command line tools (Xcode) to get things working.

Once the relevant packages are installed you will need to set the "working directory" to let `R` know where to find the models and data prior to running `fit_SDT.R`. This will depend on where you downloaded the github repository to. In `R studio` you can click "Session" –> "Set Working Directory" –> "Choose Directory. . . " to select the folder/directory containing the code. Alternatively, (also assuming you are using `R studio`) you can add the following code at the beginning of `fit_SDT.R`:

[1] https://docs.github.com/en/github/getting-started-with-github/fork-a-repo

```
setwd(dirname(rstudioapi::getActiveDocumentContext()$path))
```

Note that before running each model `fit_SDT.R` checks if a model has been previously run and saved. Users are also able to set `LOAD` at the beginning of the script to `FALSE` (or `F`) if they would like to override this and always re-run models. The example code below shows how this works:

```
if (!LOAD | !file.exists("models/SDT_m1_fit.rds")){
  SDT_m1_fit <- stan(
    file = "models/SDT_m1.stan",
    data = data_list,
    chains = nchains,
    warmup = nwarm,
    iter = niter,
    # the lines below exclude the stan parameters listed from
    # being saved. Otherwise the model object is v large
    pars=c("d", "s", "a", "b", "c", "theta"),
    include=F
  )
  # save the model object for later
  saveRDS(SDT_m1_fit, file = "models/SDT_m1_fit.rds")
} else {
  # load
  SDT_m1_fit = readRDS("models/SDT_m1_fit.rds")
}
```

If `LOAD` is set to `FALSE` (and therefore `!LOAD` is `TRUE`) *or* a previously saved file cannot be found, `stan` is used to fit the model and the results are saved in an `.rds` file (`|` is the "OR" operator in `R`; see, https://www.tutorialspoint.com/r/r_operators.htm). Otherwise (else), the previously saved results are loaded into `R`. As the models take some time to run, we have made fitted models available for download at https://drive.google.com/drive/folders/14gmtoYXKHMtZL7yjIzdmjKEhjIrmsGaq. The `.rds` files at that link should be placed in the `models/` folder.

Priors for the signal detection model

In the `model` block of `SDT_m1.stan` we start by specifying the priors on the hierarchical parameters:

```
  // priors
  B_d[1] ~ normal(0, 1);
  B_d[2] ~ normal(0, 0.5);
  B_a[1] ~ normal(0, 1);
  B_a[2] ~ normal(0, 0.5);
  B_b[1] ~ normal(0, 2);
  B_b[2] ~ normal(0, 1);
  B_s[1] ~ normal(0, 0.5);
  B_s[2] ~ normal(0, 0.25);

  tau_d ~ cauchy(0, 1);
  tau_a ~ cauchy(0, 1);
  tau_b ~ cauchy(0, 2);
  tau_s ~ cauchy(0, 0.5);
```

The prior distribution reflects the degree of belief in particular parameter values before seeing new data. Here we have tried to specify *weakly informative* priors that cover reasonable parameters values but are not too constraining as to "let the data speak" (for more detail on selecting priors, see https://github.com/stan-dev/stan/wiki/Prior-Choice-Recommendations). It is important to remember that for most of the signal detection parameters (d, s, and a) we are working on a transformed scale (specifically log scale) in order to map these constrained parameters onto all real numbers (to allow linear modeling). So the line `B_d[1] ~ normal(0, 1);` places most of the prior probability on an intercept $d = 1$ (as exp(0) = 1) and most of the prior mass between d = exp(-1) = 0.37 and d = exp(1) = 2.7. These are reasonable values for d and a (the scale factor for the response criteria), therefore we use the same Normal(0, 1) prior for both. For the intercept for s there is a smaller range of reasonable values. Empirically this parameter tends to fall in the range of 0.7–1.3 (Swets, 1986). The prior of `B_s[1] ~ normal(0, 0.5);` is a little broader than this but places most of the prior mass between s = exp(-.5) = 0.61 and s = exp(.5) = 1.65. Finally, the b parameter is the only one that is not transformed as technically it can take any number. However, reasonable values for this fall in the -2 to +2 range, hence the `B_b[1] ~ normal(0, 2);`.

For the hierarchical parameters that control the age difference we have chosen normal prior distributions with half the variability of the corresponding intercept parameter. This reflects a prior belief that age differences are likely smaller than the reasonable range of parameter values. The selection of these priors is particularly important when one wants to calculate a Bayes factor in favor or against a particular age effect or interaction. The prior should reflect a reasonable alternative and not be overly diffuse. For example, in most cognitive aging applications there may be a strong reason to expect an age difference in a particular direction (i.e., a directional hypothesis). This could be encoded in a prior that places all mass on positive or negative values (depending on how things are coded in the design matrix, `X`).

Finally, for the `tau` parameters, which control between participant variability in the SDT parameters, we use a half-Cauchy distribution (Stan takes into account that the `taus` are constrained to be positive in the `parameters` block). The Cauchy distribution is often used for its heavy tails, which means than most of the prior mass falls within the scale parameter (the second number in `cauchy(0, 1)`) but much larger values are not ruled out.

When dealing with putting priors on transformed variables it can be useful to simulate parameters on their natural scale using `R` to see if the values are truly reasonable and not overly restrictive:

```
# simulate some individual level parameters
nrep = 1e+05 # some large number

mu = rnorm(n = nrep, mean = 0, sd = 1) # population level mean
tau = abs(rcauchy(n = nrep, location = 0, scale = 1)) # population level SD

# sample individuals and transform to natural scale
mu_i = exp( rnorm(n = nrep, mean = mu, sd = tau) )

round(quantile(mu_i, probs = c(.05, .25, .5, .75, .95)), 2)
```

```
##     5%    25%    50%    75%    95%
##   0.01   0.33   1.00   3.04 165.81
```

Generated quantities

An important part of assessing how the model predictions relate to the observed data is to calculate the log likelihood of each observation under the predicted model probabilities and to simulate data from the fitted model (y_{rep}), which are known as posterior predictions. This is done in the `generated quantities` block, which comes after the `model` block (see `SDT_m1.stan`). The Stan function `categorical_lpmf` gives the log probability mass of the observation given the predicted probabilities according to SDT, whereas the `categorical_rng` generates a random number from 1 to K with the probabilities given by `theta` (which was created in the `transformed parameters` block in the main manuscript).

```
generated quantities {
  vector[N] log_lik;      // log likelihood matrix
  vector[N] y_rep;        // posterior predictions
  for (i in 1:N){
    log_lik[i] = categorical_lpmf(y[i] | theta[i]);
    y_rep[i] = categorical_rng(theta[i]);
  }
}
```

The `log_lik` matrix is particularly important when comparing different models fit to the data and is used to calculate metrics like WAIC and leave one out cross validation (LOO-CV), which are beyond the scope of the current tutorial (see Vehtari, Gelman, & Gabry, 2017).

As mentioned in the main manuscript, one graphical approach to assessing the adequacy of the model fit is to plot posterior predictions alongside the observed data. These predictions are generated in the `generated quantities` block of the Stan code. Using each sample of the different model parameters is used to generate a simulated response for each observation. This results in a predicted distribution for *each observation* in the data frame that propagates the uncertainty in the model parameters. The `R` code below extracts the posterior predictions (`yrep`) and the observed data (`y`) and uses the `bayesplot` package (Gabry & Mahr, 2018) to plot the two (see Figure 1). Specifically, the `ppc_bars_grouped` function allows us to produce separate plots for different groupings in the data (e.g., certain conditions, individual participants) and, in this case, we have chosen to plot the two age groups and different trial types (signal/noise).

```r
yrep <- extract(SDT_m1_fit, pars = "y_rep")[[1]] # posterior predictions
# the [[1]] produces a vector instead of list
y <- rdat$rating # data
```

```r
library(bayesplot)
gr = paste0("group = ", rdat$group,
            "; trial = ", rdat$signal) # combine group and trial type
ppc_bars_grouped(y, yrep, group = gr, prob = .95, freq = F)
```

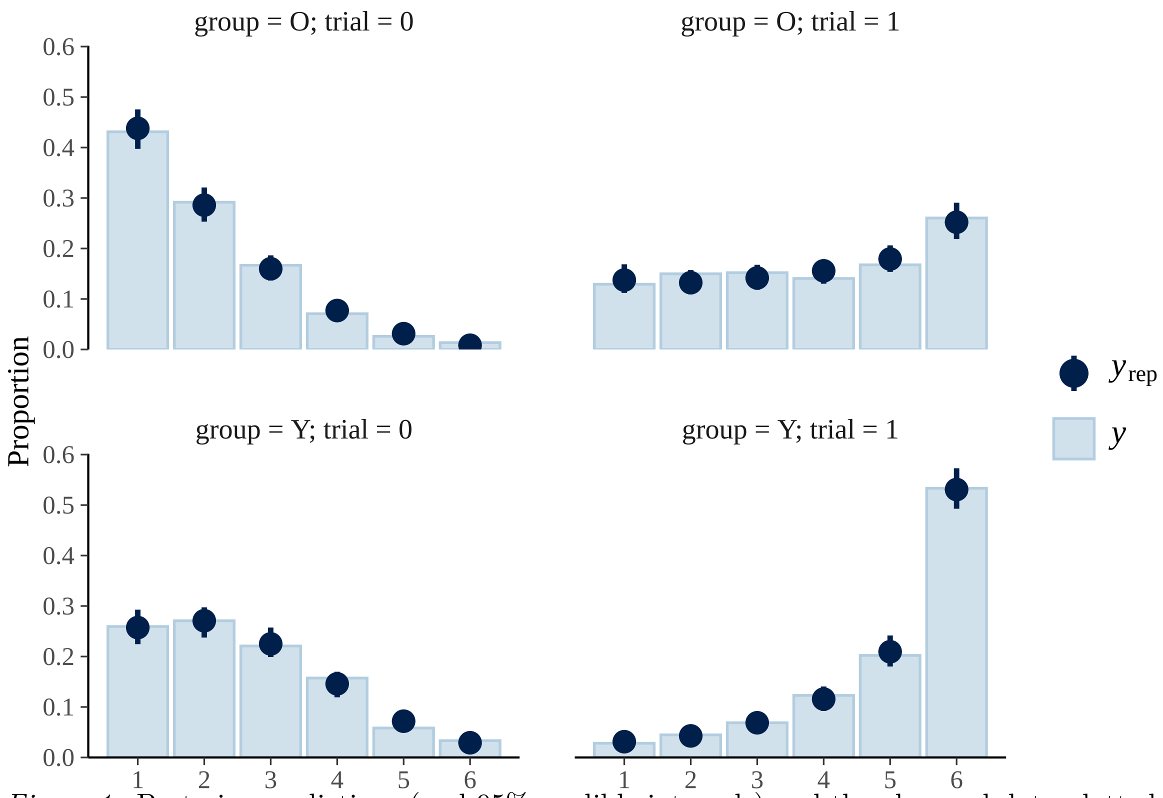


Figure 1. Posterior predictions (and 95% credible intervals) and the observed data plotted by group (Y = younger, O = older) and trial type (0 = noise, 1 = signal).

Disparities between the observed data and posterior predictions may suggest modifications to improve model fit (https://mc-stan.org/bayesplot/articles/graphical-ppcs.html).

However, there is a trade-off between fitting one data set really well and the ability of a model to generalize to new data sets (see section on model comparison below).

Comparing models 1 and 2 via Bayes' factors

In the main manuscript the primary example (`SDT_m1.stan`) is that of a model in which all parameters were allowed to differ by age group. To test the weight of evidence for an age difference in d in particular we could restrict this initial model and remove the age effect. In the `parameters` block this involves changing the hierarchical parameters for d as follows:

```
  // d
  real B_d;
  real b_d[J];
  real<lower=0> tau_d;
```

Now `B_d` is a single real valued number reflecting average $\log(d)$ across both age groups instead of a vector of two values as in the initial model (recall that we model most of the parameters on a transformed scale to allow linear predictors).

In the `transformed parameters` block we also change the line determining `d[i]`:

```
  d[i] = exp( B_d + b_d[id[i]] );
```

Otherwise the model save in `SDT_m2.stan` is the same as that in `SDT_m1.stan`. We can fit this model using `rjags` using the same call to the `stan` function and saving to an object `SDT_m2_fit`:

```
SDT_m2_fit <- stan(
  file = "models/SDT_m2.stan",
  data = data_list,
  chains = 4,
  warmup = 1000,
  iter = 2000,
  cores = 4
)
```

We can now compare `SDT_m1_fit` and `SDT_m2_fit` to see which better accounts for the observed data accounting for the fact that `SDT_m1_fit` is more flexible (as it includes the age difference in d, whereas `SDT_m2_fit` does not). One way to do this is to calculate a Bayes' factor using the `bridgesampling` package (Gronau & Singmann, 2017). The `bridge_sampler` function estimates the marginal likelihood for a particular model, $p(\text{data} \mid \text{model})$ (see Gronau et al., 2017).

```r
library(bridgesampling)

SDT_m1_ll = bridge_sampler(SDT_m1_fit)
SDT_m2_ll = bridge_sampler(SDT_m2_fit)
```

The Bayes' factor is the ratio of marginal likelihoods for the two models: $BF_{12} = p(\text{data} \mid \text{model 1})/p(\text{data} \mid \text{model 2})$. This is computed via the `bayes_factor` function:

```r
bayes_factor(SDT_m1_ll, SDT_m2_ll)
```

```
## Estimated Bayes factor in favor of x1 over x2: 56.97056
```

Model 1, the model that allowed for age group differences in d, is favored over model 2, the model without the age difference, by a factor of around 56-to-1. Therefore, in this case there is strong evidence for an age difference (although the posterior probability of the model/hypothesis also depends on our prior belief in one model/hypothesis over another).

Bayes' factors naturally account for model flexibility [ref] but when calculating them it is important to remember that the model is defined not only by the structure of the model but the priors placed on the parameters. Therefore, the priors should cover reasonable possible parameter values and should not be overly broad. For instance, in the example above where we are comparing a "null" model (model 2 with no age difference) to an alternative model (model 1 with an age difference) choosing a more diffuse prior for the age difference ($\beta_1^{(d)}$) would result in greater support for the null hypothesis for fixed data (Lindley, 1957).

There are other approaches to comparing models that have different conceptual foundations. In particular the widely applicable (or Watanabe–Akaike) information criterion (WAIC; Watanabe, 2013) and leave-one out cross validation (LOO-CV; Vehtari et al., 2017) are implemented in the `loo` package (Vehtari, Gabry, Yao, & Gelman, 2019).

Other possible extensions to model 1

Item effects

In performing analysis of variance researchers typically average over items (or stimuli) used in the experimental paradigm. This aggregation can be problematic (Clark, 1973), especially when dealing with non-linear models like the signal detection theory model we use in our example. Accurate estimation of age differences in performance requires that variability due to item effects be taken into account in a hierarchical model (see Morey, Pratte, & Rouder, 2008; Rouder & Lu, 2005). In the example data set there is a column `item` that codes the stimulus seen on a particular trial. Here we will extend model 1 to allow for item effects on discriminability, d. The extension involves adding an "item-level" effect $\alpha_m^{(d)}$ for items $1, ..., M$ ($M = 20$ in the example data set). Thus the expression for d for observation i is:

$$d_i = \exp\left(\beta^{(d)}_{0j[i]} + \beta^{(d)}_{1j[i]} x_i + b^{(d)}_{j[i]} + \alpha^{(d)}_{m[i]}\right).$$

Note that we are now using the notation $j[i]$ and $m[i]$; this can be read as "the individual or item that observation i relates to" and is similar to the way in which the model is written in Stan as we will see below. The item effect is then assumed to be normally distributed with an estimated standard deviation:

$$\alpha_m^{(d)} \sim \text{Normal}(0,\ \tau^{(\alpha)}).$$

To modify model 1 (`SDT_m1.stan`) to include item effects (see `SDT_m1.2.stan`) we first need to add item information to the `data` block:

```
  int<lower=0> M;                // number of items
  int<lower=1,upper=M> item[N]; // item ids
```

We then add to the `paramaters` block:

```
  real alpha_d[M];
  real<lower=0> tau_alpha_d;
```

and then change the expression for observation level d to add the item specific deviation:

```
d[i] = exp( dot_product(X[i,], B_d) + b_d[id[i]] + alpha_d[item[i]] );
```

Finally, we just need to modify the `model` block to put priors on the item level parameters:

```
  tau_alpha_d ~ cauchy(0, 1);
  alpha_d ~ normal(0, tau_alpha_d);
```

We are now ready to fit this model in `R` with `rstan`:

```
# add item information to the data list
data_list$item = rdat$item
data_list$M = length(unique(rdat$item))

SDT_m1.2_fit <- stan( # do the sampling
  file = "models/SDT_m1.2.stan",
  data = data_list,
  chains = 4,
  warmup = 1000,
  iter = 2000,
  cores = 4
)
```

The following code then extracts the posterior distribution of the item effects and plots them (using `bayesplot`; Figure 2) from most positive to most negative:

```
item_d = as.array(SDT_m1.2_fit, pars="alpha_d") # extract item samples as an array

ord=order(apply(item_d, 3, median), decreasing = T) # order of the posterior medians

bayesplot::mcmc_areas(item_d[,,ord], prob = .8)
```

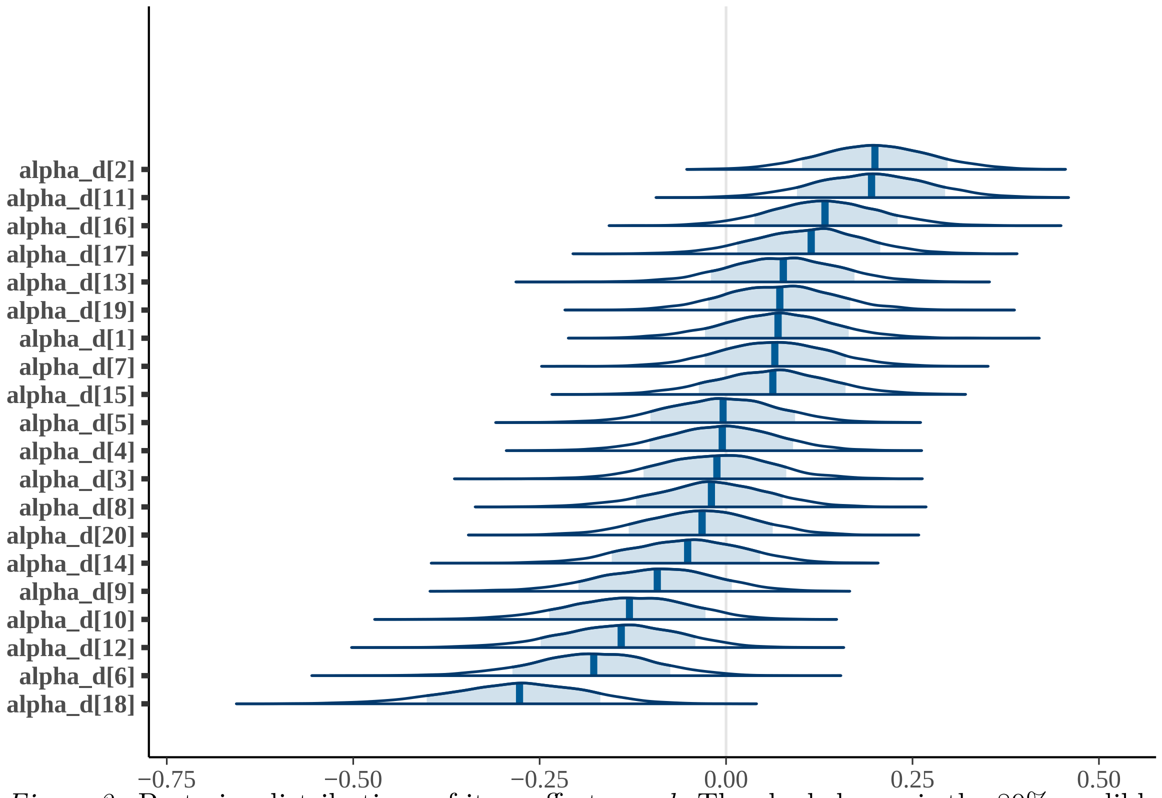


Figure 2. Posterior distributions of item effects on d. The shaded area is the 80% credible interval.

Figure 3 shows the effect of modeling item differences on our estimate of age differences in discriminabilty.

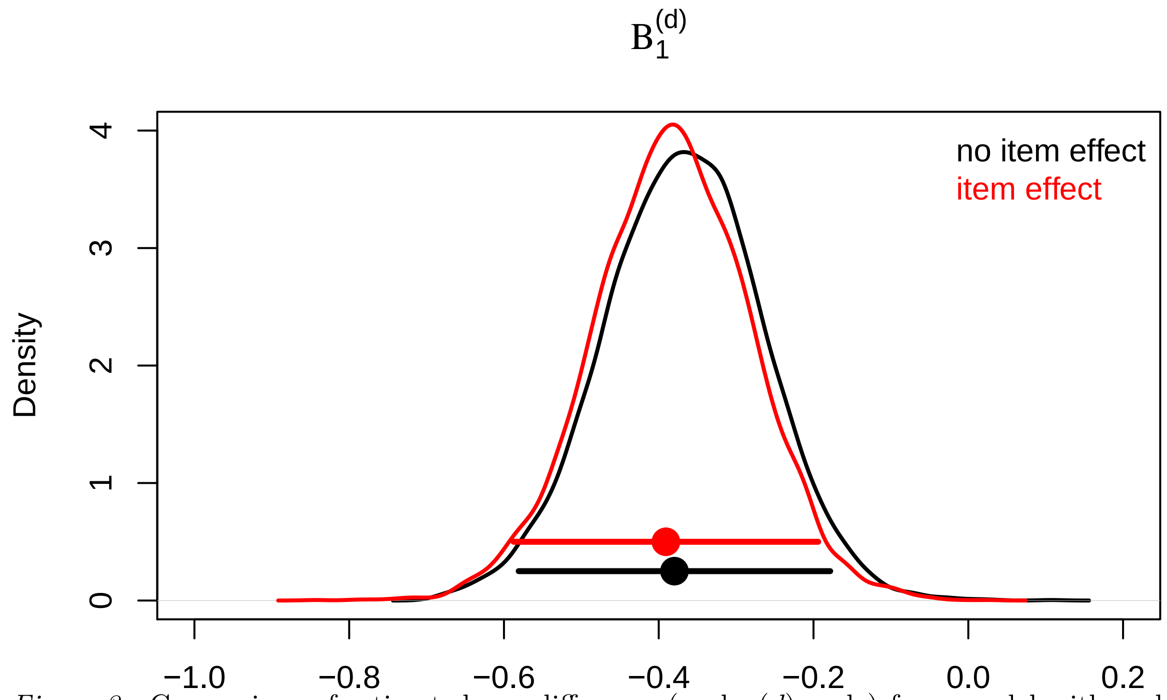


Figure 3. Comparison of estimated age difference (on log(d) scale) from model with and without item effects. Points are medians and horizontal lines are 95% highest density intervals.

Allow groups to differ in variability (Heteroskedasticity)

A common finding in cognitive aging research is that, in addition to mean differences in task performance, age groups also differ in their variability (i.e., between participant variability often increases with age; e.g., Shammi, Bosman, & Stuss, 1998). In model 1 parameter values for participants from different age groups are drawn from the same normal distribution (i.e., one shared standard deviation). We can allow the groups to differ in their variability by estimating separate individual level standard deviation parameters (again using d as the example, $\tau_1^{(d)}$ for younger adults and $\tau_2^{(d)}$ for older). Therefore, the expression for individual level effects on d becomes:

$$b_j^{(d)} \sim \text{Normal}(0,\ \tau_{g[j]}^{(d)}).$$

To implement this in Stan by modifying `SDT_m1.stan` we first add to the `data` block an indicator that codes group membership (0=younger, 1=older):

```
int group[J];
```

Then in `parameters` we change `tau_d` to a vector of two real-valued numbers with a lower bound of zero:

```
real<lower=0> tau_d[2];
```

The final change to produce the `SDT_m1.3.stan` model is to change the `model` block to sample individual level effects according to group membership. The `group[j]+1` means that `tau_d[1]` will be selected when group=0 (i.e., younger) and `tau_d[2]` will be selected when group=1 (i.e., older):

```
  for (j in 1:J){
    b_d[j] ~ normal(0, tau_d[group[j]+1]);
  }
```

In `R` we then create a vector that codes whether each participant is younger or older and add this to the data list. We then use `stan` to fit the model:

```r
# get the group of each participant
grps = rdat$group[!duplicated(rdat$id)]
# add to the data list
data_list$group = as.integer(grps == "O")

SDT_m1.3_fit <- stan(
  file = "models/SDT_m1.3.stan",
  data = data_list,
  chains = 4,
  warmup = 500,
  iter = 1000,
  cores = 4
)
```

The following `R` code then extracts the `tau_d` samples and plots them in Figure 4. The left panel shows the density of each group separately whereas the right panel subtracts the two to plot the difference.

```r
# extract the samples
tau_d = extract(SDT_m1.3_fit, pars=c("tau_d"))[[1]]

par(mfrow=c(1,2)) # make a two panel plot
# plot density for each group
plot(density(tau_d[,1]), lwd=2,
     main=bquote(tau^"(d)"), xlab="", xlim=c(0, .9))
lines(density(tau_d[,2]), lwd=2, col="blue")
legend("topright", legend=c("younger", "older"),
       text.col=c("black", "blue"), bty="n")
# plot difference
tau_diff = apply(tau_d, 1, diff) # or tau_d[,2] - tau_d[,1]
```

```
plot(density(tau_diff), lwd=2,
     main=bquote(tau[older]^"(d)" - tau[younger]^"(d)"), xlab="")
```

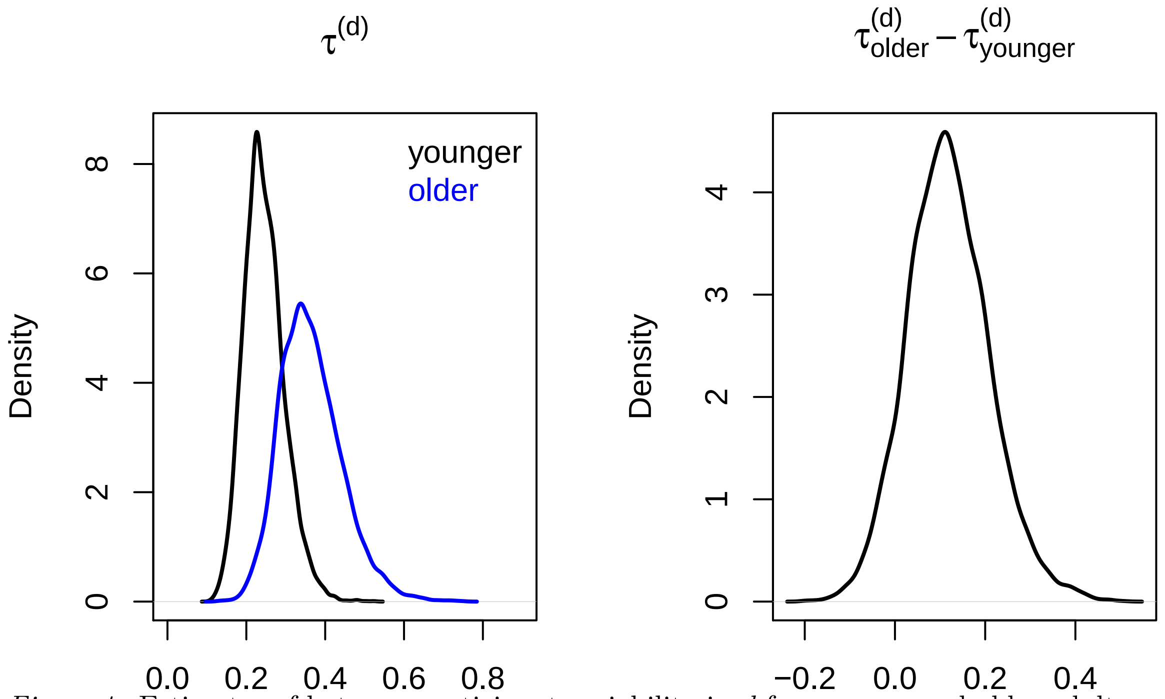


Figure 4. Estimates of between participant variability in d for younger and older adults.

The next code chunk then calculates the HDI of the group difference in `tau_d` and the proportion of samples that are above zero difference.

```
# HDI
HDInterval::hdi(tau_diff)
```

```
##       lower       upper
## -0.05247102  0.30794142
## attr(,"credMass")
## [1] 0.95
```

```
# proportion of posterion above zero
mean(tau_diff > 0)
```

```
## [1] 0.9118333
```

Correlate random effects with other measures

Researchers may wish to relate latent cognitive processes to other participant characteristics (e.g., neuropsychological test scores, personality measures, years of education).

One way to estimate the correlation between individual differences in model parameters and another measure is to expand the random effects section of the model to include the extra measure. The file `SDT_m1.4.stan` extends `SDT_m1.stan` to estimate the correlation between individual differences in d (the individual level deviations $b_j^{(d)}$) and scores from another measure found in `data/cor-scores.csv`.

To modify model 1 we first change the `data` block to include the score variable. There is one score for each of the `J` participants:

```
real score[J];
```

Next we modify the `parameters` block to include parameters for estimating the mean (`mu_score`) and variability (`tau_score`) of the score as well as its correlation with the d random effects (`Sigma`).

```
  // correlation between individual differences in d
  // and score from another measure
  real mu_score; // average score
  real<lower=0> tau_score; // score SD

  corr_matrix[2] Sigma; // correlation of d random effect and score
```

Finally the `model` block is modified to include the priors for the new score associated parameters (remember to consider the scale of variables when setting priors) and to express the assumption that individual deviations in d and the scores on the other measure are sampled from a multivariate normal distribution with a correlation determined by `Sigma`.

```
  // priors for mean and SD of score
  mu_score ~ normal(0, 1);
  tau_score ~ cauchy(0, 1);

  // prior for correlation between score and individual b_ds
  Sigma ~ lkj_corr(1.0);

  // individual level deviations
  for (j in 1:J){
    [b_d[j],score[j]] ~ multi_normal([0,mu_score], quad_form_diag(Sigma, [tau_d,tau_score])
  }
```

In `fit_SDT.R` we add the scores to the data list and then fit the model.

```
scores = read.csv("data/cor-scores.csv") # one score for each person

data_list$score = scores$score
```

```r
SDT_m1.4_fit <- stan(
  file = "models/SDT_m1.4.stan",
  data = data_list,
  chains = nchains,
  warmup = nwarm,
  iter = niter,
  pars=c("d", "s", "a", "b", "c", "theta"),
  include=F
)
```

The code below produces Figure 5 which plots the correlation matrix (`Sigma`). `Sigma[1,2]` and `Sigma[2,1]` refers to the correlation between the score and individual differences in d.

```r
plot(SDT_m1.4_fit, pars="Sigma")
```

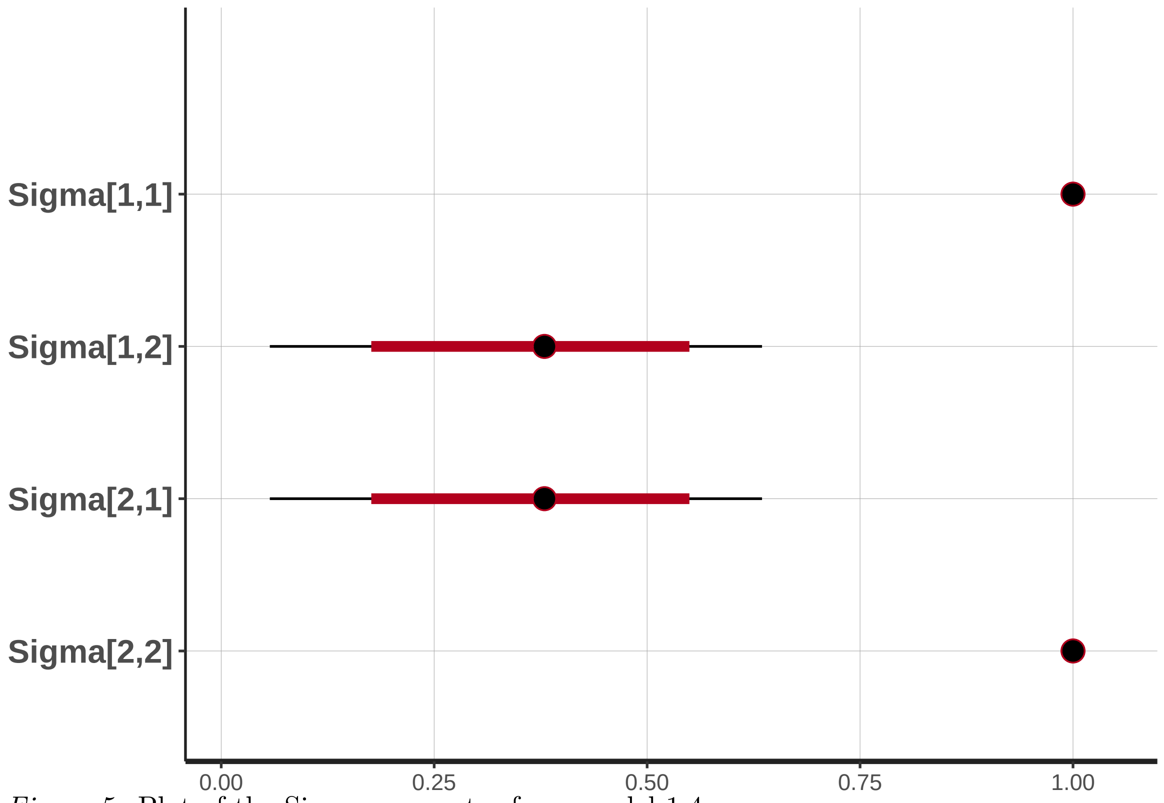


Figure 5. Plot of the Sigma parameter from model 1.4

The code below then extracts the samples associated with the correlation and calculate some quantities of interest.

```r
# extract the correlation samples
rho = extract(SDT_m1.4_fit, pars="Sigma[1,2]")[[1]]
# posterior mean and median
mean(rho)
```

```
## [1] 0.3693048
```

```r
median(rho)
```

```
## [1] 0.3793044
```

```r
# and 95% credible intervals
quantile(rho, probs = c(0.025, .975))
```

```
##       2.5%      97.5%
## 0.05707102 0.63484355
```

Exploring age by condition interactions (model 3)

The main manuscript describes how to implement a model that takes condition into account when estimating group differences in discriminability, *d*. The Stan model is contained in `SDT_m3.stan` and samples from this model are saved in the `R` object `SDT_m3_fit`. The following `R` code shows how to extract the relevant samples from this model, convert them into posterior samples of *d* for each group in each condition:

```r
# extract the group level effects for d
B_d = extract(SDT_m3_fit, pars="B_d")[[1]]

# convert back to d scale for both groups and conditions
# condition A was coded 0 and B coded 1
# group Y was coded 0 and group O coded 1
# therefore, we can multiply the matrix of B_d samples
# by particular vectors that code for group and condition

d_youngA = exp( B_d %*% c(1,0,0,0) ) # use exp to transform to d
d_youngB = exp( B_d %*% c(1,0,1,0) )
d_oldA = exp( B_d %*% c(1,1,0,0) )
d_oldB = exp( B_d %*% c(1,1,1,1) )
```

Figure 6 presents density plots of these posterior samples (the `fit_SDT.R` contains the code to create these plots). In addition to plotting we can also calculate quantities of interest, like highest density intervals:

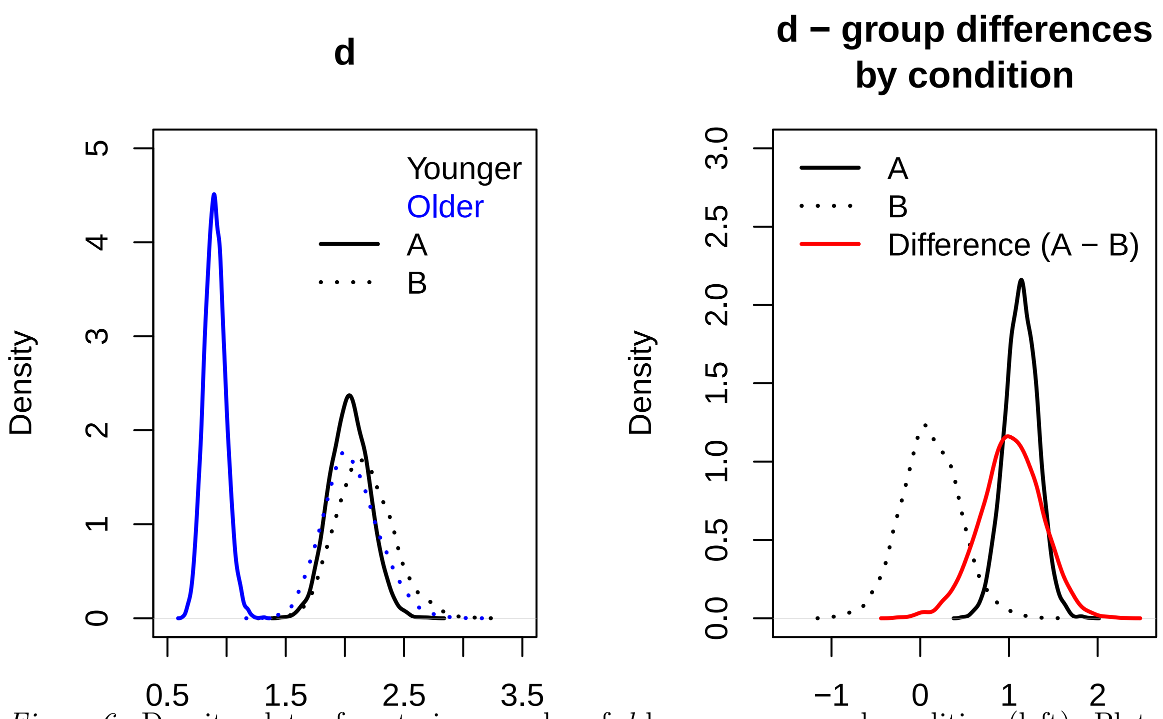


Figure 6. Density plots of posterior samples of *d* by age group and condition (left). Plots of age group differences (younger - older) by condition and the condition difference in these differences (right; i.e., how much larger was the age difference in condition A relative to condition B?).

```r
library(HDInterval)
# group difference in condition A
hdi(d_youngA - d_oldA)
```

```
##            [,1]
## lower 0.7525997
## upper 1.4899763
## attr(,"credMass")
## [1] 0.95
```

```r
# group difference in condition B
hdi(d_youngB - d_oldB)
```

```
##             [,1]
## lower -0.5560693
## upper  0.7623439
## attr(,"credMass")
## [1] 0.95
```

```r
# difference of differences (i.e., interaction)
hdi((d_youngA - d_oldA) - (d_youngB - d_oldB))
```

```
##              [,1]
## lower 0.3542292
## upper 1.7204612
## attr(,"credMass")
## [1] 0.95
```

The age difference in *d* is approximately 1.02 larger in condition A relative to condition B, with a 95% HDI of [0.35, 1.72].

Multinomial Processing Tree Modeling Tutorial

The signal detection theory (SDT) models described in detail in the main text assume that the representation of information underlying discrimination and response selection is continuous or graded. In contrast to this assumption, a competing class of models – multinomial processing tree (MPT) models – assumes that this representation is mediated by discrete mental states. The most common MPT used in studies of recognition memory is the Two-High-Threshold model (2HTM; Snodgrass & Corwin, 1988), which assumes three discrete states created by the partitioning of the decision process by two latent thresholds, one corresponding to a detect-old state and the other to a detect-new state. If the signal passes the upper or lower threshold, then with 100% certainty the participant will recognize the item as "old" or "new," respectively. Otherwise, if the signal falls between these two thresholds, the participant is in a state of uncertainty and then must guess whether the signal is old or new. The 2HTM can be conceived as a variant of a SDT model with rectangular as opposed to Gaussian distributions (Swets, 1986), but there is much debate as to which model is more appropriate for modeling recognition memory (and other choice discrimination) data (e.g., Dube & Rotello, 2012; Klauer & Kellen, 2011; Pazzaglia, Dube, & Rotello, 2013; Province & Rouder, 2012).

Nevertheless, MPTs are useful measurement models as they provide a parametric way to link psychological theory to observed responses (Batchelder & Riefer, 1999). Given their popularity in many domains of psychology, we offer a brief overview of how to fit MPTs to derive age comparisons on parameters corresponding to different cognitive processes. We use an example of fitting Bröder, Kellen, Schütz, and Rohrmeier (2013)'s ratings-extension of the 2HTM to the ratings data reported in the main text, but this approach can be applied more broadly to other types of MPTs.

Although MPTs can be implemented in Stan, like our example of SDT models in the main text, there are many useful `R` packages for fitting MPTs, and our preference is the `TreeBUGS` package (Heck, Arnold, & Denis, 2018), as it incorporates the most up-to-date advances in MPT modeling. Specifically, the `TreeBUGS` package allows for estimation of hierarchical Bayesian MPTs, which enable researchers to account for the heterogeneity in responses across participants in a principled way, with individual parameters drawn from a parent distribution. Response heterogeneity across participants is not accounted for in fits of MPTs to the aggregate, in which observations are assumed to be independent and identically distributed (i.i.d.) for all participants. Violations of the i.i.d. assumption can result in biased parameter estimates and erroneous confidence intervals (Klauer, 2006; Smith & Batchelder, 2008, 2010). The `TreeBUGS` package makes it easy to test for participant heterogeneity before model fitting; if heterogeneity is present, it is recommended that the researcher proceed with a hierarchical Bayesian model. The `TreeBUGS` package interfaces with the Markov chain Monte Carlo sampler JAGS (Plummer, M, 2003) via the `R` package **`runjags`** (Denwood, 2016) and allows users to specify either the latent-trait (Klauer, 2010) or beta MPT (Smith & Batchelder, 2010) model, two principled models for dealing with participant heterogeneity.

The model

Bröder et al. (2013) introduced a ratings-extension of the 2HTM, which includes the original model's parameters of Old and New detection (p_o and p_n) and proclivity to guess "old" in the uncertain state (b), as well as response mapping parameters to relate responses to the different ratings (1 through 6, in our ratings task). If a participant enters a detect state (with probability p_0 for Old items or p_n for New items), he or she traverses a response parameter r_i, denoting the probability of responding with rating i. For the detect-old state, r_i can take on a rating of 4, 5, or 6 (as these ratings correspond to "maybe old", "probably old," and "sure old," respectively); for the detect-new state, these values are 3, 2, or 1. Thus, if a participant is in one of the detection states, he or she will not give a rating from the opposite range of the scale (e.g., assigning a "maybe new" rating to an Old item in the detect-old state). However, if the participant is in the uncertain state, recognition judgments are based solely on guessing, such that all ratings are possible. The probability of assigning rating i in the uncertain state is given by q_i.

Full model equations describing the probability of providing rating i to Old and New items are given in Table 1 of Bröder et al. (2013) and are reproduced in a text editor, converted to a .eqn file, to be read into `R` for analysis with the `TreeBUGS` package. Specifically, we use the model equations of the binary reparameterisation of Bröder et al. (2013)'s extended 2HTM.

```r
readEQN("2htmratings.eqn")
```

```
##      Tree Category                                   Equation
## 1   T_Old    Old_6                            po*r6+(1-po)*b*q6
## 2   T_Old    Old_5               po*(1-r6)*r5+(1-po)*b*(1-q6)*q5
## 3   T_Old    Old_4     po*(1-r6)*(1-r5)+(1-po)*b*(1-q6)*(1-q5)
## 4   T_Old    Old_3                 (1-po)*(1-b)*(1-q1)*(1-q2)
## 5   T_Old    Old_2                     (1-po)*(1-b)*(1-q1)*q2
## 6   T_Old    Old_1                            (1-po)*(1-b)*q1
## 7   T_New    New_6                                (1-pn)*b*q6
## 8   T_New    New_5                         (1-pn)*b*(1-q6)*q5
## 9  T_New    New_4                         (1-pn)*b*(1-q6)*(1-q5)
## 10 T_New    New_3 pn*(1-r1)*(1-r2)+(1-pn)*(1-b)*(1-q1)*(1-q2)
## 11 T_New    New_2           pn*(1-r1)*r2+(1-pn)*(1-b)*(1-q1)*q2
## 12 T_New    New_1                          pn*r1+(1-pn)*(1-b)*q1
```

The column `Tree` in the output indicates that there are two separate trees, one for Old items and one for New items, and each tree is defined by its own unique multinomial distribution. The `Category` column gives the observed rating category. The `Equation` column specifies the model equations linking the detection parameters (p_o and p_n), the guessing parameter (b), and the response mapping parameters (r_i for detect-states, and q_i for uncertain state) to the observed ratings in the `Category` column.

There are 11 free parameters (p_o, p_n, b, r_6, r_5, r_1, r_2, q_6, q_5, q_1, and q_2) but only ten degrees of freedom, so the model is oversaturated. (Note also that there are no ratings parameters for the (4) "maybe old" or (3) "maybe new" ratings, as these values take the remainder of the conditional probability at the node of the tree branching into their adjacent ratings, e.g., r_4 = (1-r_6)*(1-r_5).) To address this non-identifiability issue, parameter restrictions are required. We have chosen to impose equality restraints on the response mapping parameters in the uncertain state, assuming participants use the scale symmetrically in this state (i.e., $q_1 = q_6$ and $q_2 = q_5$, such that also $q_3 = q_4$; see Bröder et al. (2013)). Alternative possibilities can be entertained, resulting in different models that represent unique subclasses of the full parameter space, and thereby motivating a model comparison approach (see main text for details about model comparison). However, for this brief tutorial, we will only focus on this specific model to demonstrate how to compare parameters of the same MPT model across age groups.

Because we will be fitting this model to the simulated data reported in the main text, which included a within-subject condition variable, we have to extend the MPT model to include separate trees for each condition (A and B). We can use the `withinSubjectEQN` command to expand our tree structure.

```r
withinSubjectEQN("2htmratings.eqn", labels=c("A","B"), save="2htmratings2.eqn")
```

```
##      Tree Category                                              Equation
## 1  A_Old  A_Old_6                              po_A*r6+(1-po_A)*b_A*q6_A
## 2  A_Old  A_Old_5                po_A*(1-r6)*r5+(1-po_A)*b_A*(1-q6_A)*q5_A
## 3  A_Old  A_Old_4        po_A*(1-r6)*(1-r5)+(1-po_A)*b_A*(1-q6_A)*(1-q5_A)
## 4  A_Old  A_Old_3                        (1-po_A)*(1-b_A)*(1-q1_A)*(1-q2_A)
## 5  A_Old  A_Old_2                            (1-po_A)*(1-b_A)*(1-q1_A)*q2_A
## 6  A_Old  A_Old_1                                     (1-po_A)*(1-b_A)*q1_A
## 7  A_New  A_New_6                                         (1-pn_A)*b_A*q6_A
## 8  A_New  A_New_5                                (1-pn_A)*b_A*(1-q6_A)*q5_A
## 9  A_New  A_New_4                            (1-pn_A)*b_A*(1-q6_A)*(1-q5_A)
## 10 A_New  A_New_3 pn_A*(1-r1)*(1-r2)+(1-pn_A)*(1-b_A)*(1-q1_A)*(1-q2_A)
## 11 A_New  A_New_2          pn_A*(1-r1)*r2+(1-pn_A)*(1-b_A)*(1-q1_A)*q2_A
## 12 A_New  A_New_1                         pn_A*r1+(1-pn_A)*(1-b_A)*q1_A
## 13 B_Old  B_Old_6                             po_B*r6+(1-po_B)*b_B*q6_B
## 14 B_Old  B_Old_5              po_B*(1-r6)*r5+(1-po_B)*b_B*(1-q6_B)*q5_B
## 15 B_Old  B_Old_4     po_B*(1-r6)*(1-r5)+(1-po_B)*b_B*(1-q6_B)*(1-q5_B)
## 16 B_Old  B_Old_3                  (1-po_B)*(1-b_B)*(1-q1_B)*(1-q2_B)
## 17 B_Old  B_Old_2                      (1-po_B)*(1-b_B)*(1-q1_B)*q2_B
## 18 B_Old  B_Old_1                               (1-po_B)*(1-b_B)*q1_B
## 19 B_New  B_New_6                                   (1-pn_B)*b_B*q6_B
## 20 B_New  B_New_5                          (1-pn_B)*b_B*(1-q6_B)*q5_B
## 21 B_New  B_New_4                      (1-pn_B)*b_B*(1-q6_B)*(1-q5_B)
## 22 B_New  B_New_3 pn_B*(1-r1)*(1-r2)+(1-pn_B)*(1-b_B)*(1-q1_B)*(1-q2_B)
## 23 B_New  B_New_2         pn_B*(1-r1)*r2+(1-pn_B)*(1-b_B)*(1-q1_B)*q2_B
## 24 B_New  B_New_1                         pn_B*r1+(1-pn_B)*(1-b_B)*q1_B
```

The output looks much more complicated, but it is simply the same set of equations repeated separately for each within-subject condition.

Reading the Data

The data must be a table (either as a matrix or data frame in `R` or read into `R` as a comma-separated file) of individual frequencies. Each row corresponds to a participant, and each column gives the observed categories.

The following R code reads in the data then splits it into young and old adult groups. Note that we will be fitting the ratings MPT separately to each age group, which is analogous to assuming the variability in parameter estimates differs between young and older adults.

```r
d <- read.csv("examplempt.csv")
do <- subset(d, Group=="O", select=-(1:2)) #Old adults
dy <- subset(d, Group=="Y", select = -(1:2)) #Young adults

head(dy) #First 6 rows of the YA data
```

```
##    A_Old_1 A_Old_2 A_Old_3 A_Old_4 A_Old_5 A_Old_6 A_New_1 A_New_2 A_New_3
## 25       0       0       0       3       9       8       2       9       7
## 26       0       1       0       3       7       9       8       7       2
## 27       0       0       6       3       7       4       1      10       2
## 28       0       0       1       0       5      14       3       6       4
## 29       0       0       0       3       2      15       9       3       0
## 30       0       0       1       2       4      13      10       6       2
##    A_New_4 A_New_5 A_New_6 B_Old_1 B_Old_2 B_Old_3 B_Old_4 B_Old_5 B_Old_6
## 25       2       0       0       0       0       0       0       0      20
## 26       1       2       0       2       2       2       4       2       8
## 27       4       3       0       0       0       1       4       8       7
## 28       2       4       1       0       1       1       4       5       9
## 29       4       0       4       0       1       2       0       2      15
## 30       1       1       0       0       0       0       0       0      20
##    B_New_1 B_New_2 B_New_3 B_New_4 B_New_5 B_New_6
## 25       4       8       6       2       0       0
## 26       3       3       8       4       2       0
## 27       5       5       5       2       3       0
## 28       8       6       3       2       1       0
## 29       9       0       2       6       1       2
## 30       9       3       2       4       1       1
```

The output of the `head` command shows the categorical ratings in the first row. The next six rows are the data from first six young adult participants.

Testing for Participant Heterogeneity

Next, we test for homogeneity of responses across participants (in each age group). In the `R` code below, the `tree` argument indicates which columns in the frequency table pertain to Old or New items. In the MPT framework, each item type belongs to a separate multinomial distribution.

```r
testHetChi(freq=dy, tree=c("A_Old", "A_Old", "A_Old", "A_Old", "A_Old", "A_Old",
                           "A_New", "A_New", "A_New", "A_New", "A_New", "A_New",
                           "B_Old", "B_Old", "B_Old", "B_Old", "B_Old", "B_Old",
                           "B_New", "B_New", "B_New", "B_New", "B_New", "B_New"))

testHetChi(freq=do, tree=c("A_Old", "A_Old", "A_Old", "A_Old", "A_Old", "A_Old",
                           "A_New", "A_New", "A_New", "A_New", "A_New", "A_New",
                           "B_Old", "B_Old", "B_Old", "B_Old", "B_Old", "B_Old",
                           "B_New", "B_New", "B_New", "B_New", "B_New", "B_New"))
```

The resulting test statistics, $\chi^2(460) = 841.25, p < .001$ for the young adults, and $\chi^2(460) = 828.72, p < .001$ for the old adults, indicate there is substantial heterogeneity between participants in both age groups. Thus, fitting an MPT to the aggregate (i.e., via maximum likelihood) would be inappropriate in this context (notably, this also illustrates the suitability of fitting hierarchical SDT models, as reported in the main text).

Fitting a hierarchical Bayesian MPT to each data set

The next step is to fit the model (whose equations are defined above in the .eqn file) to the data for each group, separately. `TreeBUGS` (Heck et al., 2018) enables users to specify either a latent-trait (Klauer, 2010) or beta-MPT (Smith & Batchelder, 2010), which define the hyperpriors placed on the population-level distributions of MPT parameters. More information about these model specifications are given in the respective sources, but here we will fit the latent-trait model as it accounts for the correlations among MPT parameters.

In the latent-trait model, each individual participant p_i has a unique parameter vector, θ_{p_i}, which is probit-transformed to ensure only positive values (as all parameters in MPT models are probabilities and constricted to fall within the unit interval). In turn, each $\Phi^{(-1)}(\theta_{p_i})$ is drawn from a multivariate normal distribution with group mean $\boldsymbol{\mu}$ and variance-covariance matrix $\boldsymbol{\Sigma}$. Hyperpriors can be placed on $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ and can be defined separately for each parameter in the MPT model. By default, `TreeBUGS` (Heck et al., 2018) uses weakly informative priors based on work by Klauer (2010) and Matzke, Dolan, Batchelder, and Wagenmakers (2015). Each μ_s, where s denotes each parameter in the model, follows a standard normal distribution. A scaled inverse Wishart prior is used for $\boldsymbol{\Sigma}$, where the Wishart prior is scaled by a scaling parameter for each parameter, ξ_s. `TreeBUGS` assumes a uniform [0,10] prior on the scaling parameters ξ_s. Although these priors can be changed, and the `TreeBUGS` tutorial (Heck et al., 2018) discusses how this can be done, we will use the program's default priors in our current implementation to the ratings data.

The following `R` code is used to fit the ratings-2HTM (Bröder et al., 2013) to the young and old adult data sets. The hashtags describe what each argument is doing.

```
##Young Adult Model

young1 <- traitMPT(eqnfile="2htmratings2.eqn", #fit a latent-trait MPT
                  #using the ratings-2HTM equations
                  data=dy, #frequencies for young adults
                  restrictions=list("q1_A=q6_A","q2_A=q5_A",
                                    "q1_B=q6_B", "q2_B=q5_B"), #parameter restrictions
                  modelfilename="young.jags", #saves the JAGS script
                  parEstFile="youngratings.txt", #saves the output to a text file
                  posteriorFile = "youngratingsmod.RData",
                  n.iter=75000, #number of MCMC iterations to run
                  n.chain=4, #number of chains
                  n.adapt=15000, #Adaptation period (to obtain convergence)
                  n.burnin = 10000, #burn-in period (these chains are discarded)
                  n.thin=10) #thinning rate (retains every 10 chains)

#Adaptation successful after 15,000 iterations
#Sampling time took 18.71 minutes

##Old Adult Model
old1 <- traitMPT(eqnfile="2htmratings2.eqn",
                   data=do, #frequencies for old adults
                   restrictions=list("q1_A=q6_A","q2_A=q5_A",
                                     "q1_B=q6_B", "q2_B=q5_B"),
                   modelfilename="old2.jags",
                   parEstFile="oldratings2.txt",
                   posteriorFile = "oldratingsmod2.RData",
                   n.iter=75000,
                   n.chain=4,
                   n.adapt=15000,
                   n.burnin = 10000,
                   n.thin=10)

#Adaptation successful
#Sampling time of 18.11 minutes
```

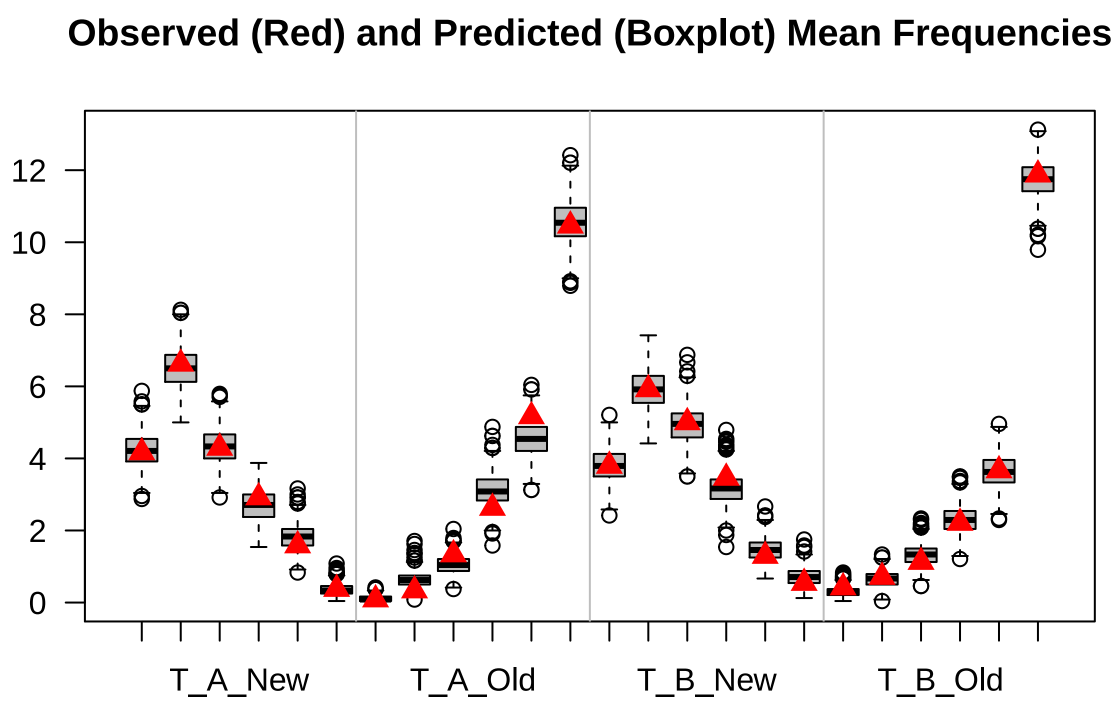
The adaptation was successful in fitting the model to both data sets, indicating the four posterior chains converged within the adaptation period. If the adaptation period is not successful, it is recommended that the researcher increases the number of adaptation chains, as otherwise the posterior samples will not be optimal and may not be interpretable.

In the below code, we load in the fitted model summaries for further analysis.

```
load("youngratingsmod.RData")
fittedModelY <- fittedModel
load("oldratingsmod2.RData")
fittedModelO <- fittedModel
```

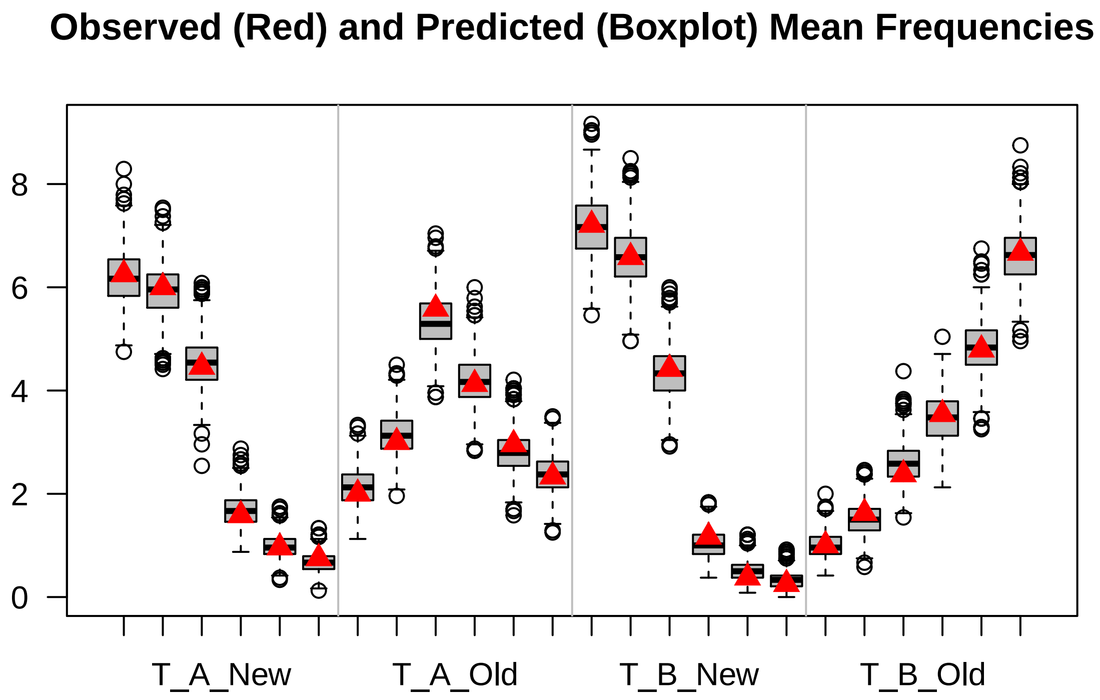
First, we can assess model fit to the observed data by plotting the model predictions against the observed frequencies in the data.

```
plotFit(fittedModelY)
```



The model is capturing the frequencies in the young adult data well, although there are some slight discrepancies.

```r
plotFit(fittedModelO)
```



The model also captures the frequencies in the old adult data well. Overall, the model has apparently good fit to the two data sets, even as it is undersaturated relative to the degrees of freedom in the data. Nevertheless, a good fit to one data set is not sufficient if the model's predictive accuracy to unobserved data is poor.

Therefore, the next crucial step in assessing model fit is in generating posterior predictive fits of the model to unobserved data. We can take the fitted model applied to each data set (from the young and old adults) and use it to compute posterior samples. This is easy to implement in the `TreeBUGS` package (Heck et al., 2018), which computes two statistics first proposed by Klauer (2010). The T_1 test statistic compares the observed means (for each participant, as well as the group-level mean) to those of the posterior-predicted data, while the T_2 statistic does the same for the covariances. The proportion of samples is computed for which $T_{observed} < T_{predicted}$ for each test statistic, yielding a posterior predictive p value (PPP). Although not to be confused with a frequentist p value, PPP values that are small (near 0) indicate insufficient model fit to posterior-predicted data.

The `PPP` function below resamples 1,000 posterior samples from the fitted model and computes the T_1 and T_2 statistics at both the group and individual-level.

```r
PPP(fittedModelY)
```

```
##  ## Mean structure (T1):
##  Observed =  1.160774 ; Predicted =  0.7959642 ; p-value =  0.16
##
```

```
## ## Covariance structure (T2):
##  Observed =  95.15489 ; Predicted =  84.89726 ; p-value =  0.308
##
## ## Individual fit (T1):
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 0.419 0.449 0.147 0.550 0.167 0.785 0.237 0.480 0.408 0.423 0.714 0.364 0.569
##    14    15    16    17    18    19    20    21    22    23    24
## 0.709 0.666 0.703 0.545 0.074 0.245 0.764 0.760 0.745 0.362 0.659
```

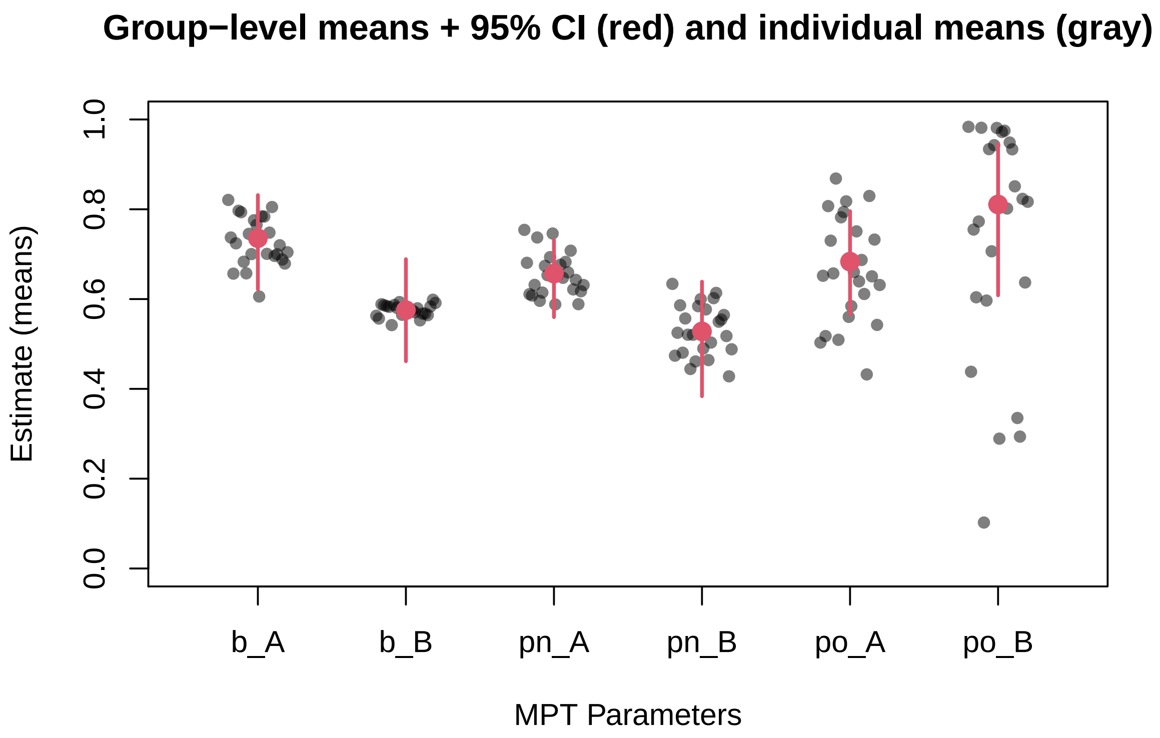
The output displays the mean and covariance test statistics at the group-level, as well as individual-level fits of the T_1 statistic. The resulting PPP values are in an acceptable range (>.05) for all fit statistics in the young adult group, indicating satisfactory model fit. The same is true of the old adult model (see output below).

```
PPP(fittedModelO)
```

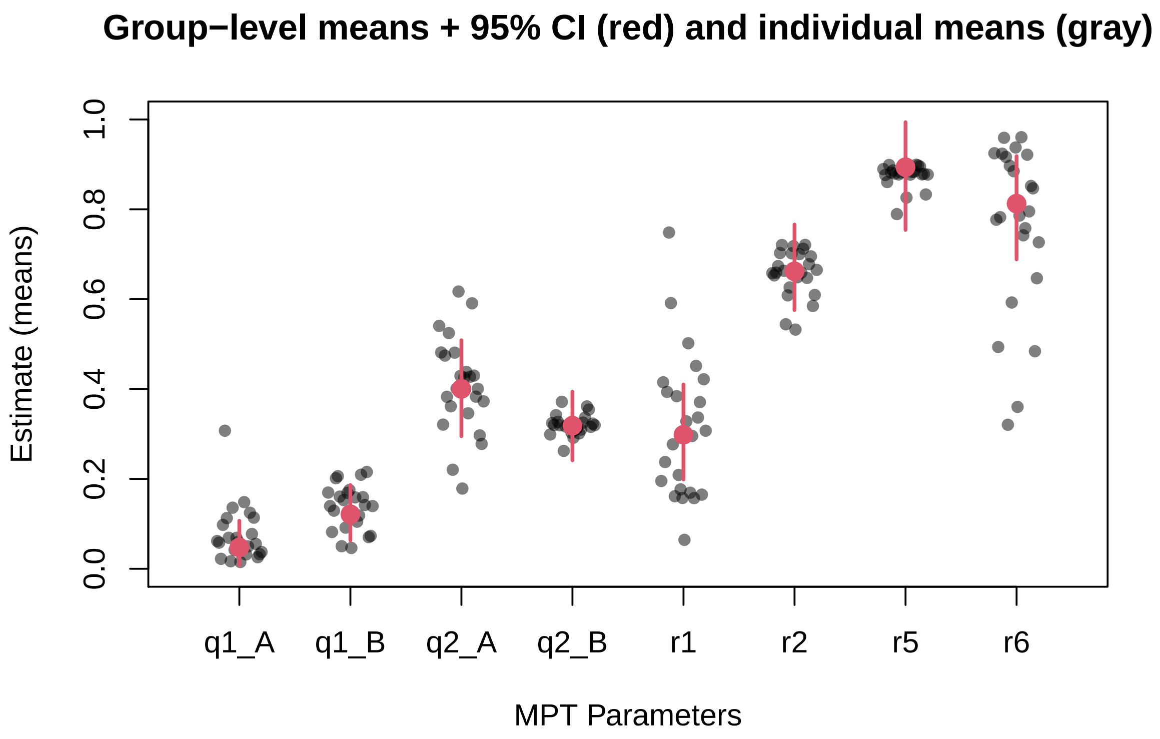
```
##  ## Mean structure (T1):
##  Observed =  0.6700155 ; Predicted =  0.8058267 ; p-value =  0.662
##
## ## Covariance structure (T2):
##  Observed =  71.30605 ; Predicted =  86.74114 ; p-value =  0.781
##
## ## Individual fit (T1):
##     1     2     3     4     5     6     7     8     9    10    11    12    13
## 0.900 0.862 0.746 0.461 0.326 0.826 0.579 0.607 0.749 0.813 0.779 0.863 0.951
##    14    15    16    17    18    19    20    21    22    23    24
## 0.973 0.874 0.767 0.369 0.868 0.637 0.732 0.744 0.643 0.776 0.974
```

Now that we have demonstrated satisfactory model fit, we can summarize the results from each model, and compare parameters across the models to derive age differences. The function below extracts the posterior group-level mean parameter estimates and 95% Bayesian credible intervals, while individual means appear in gray.

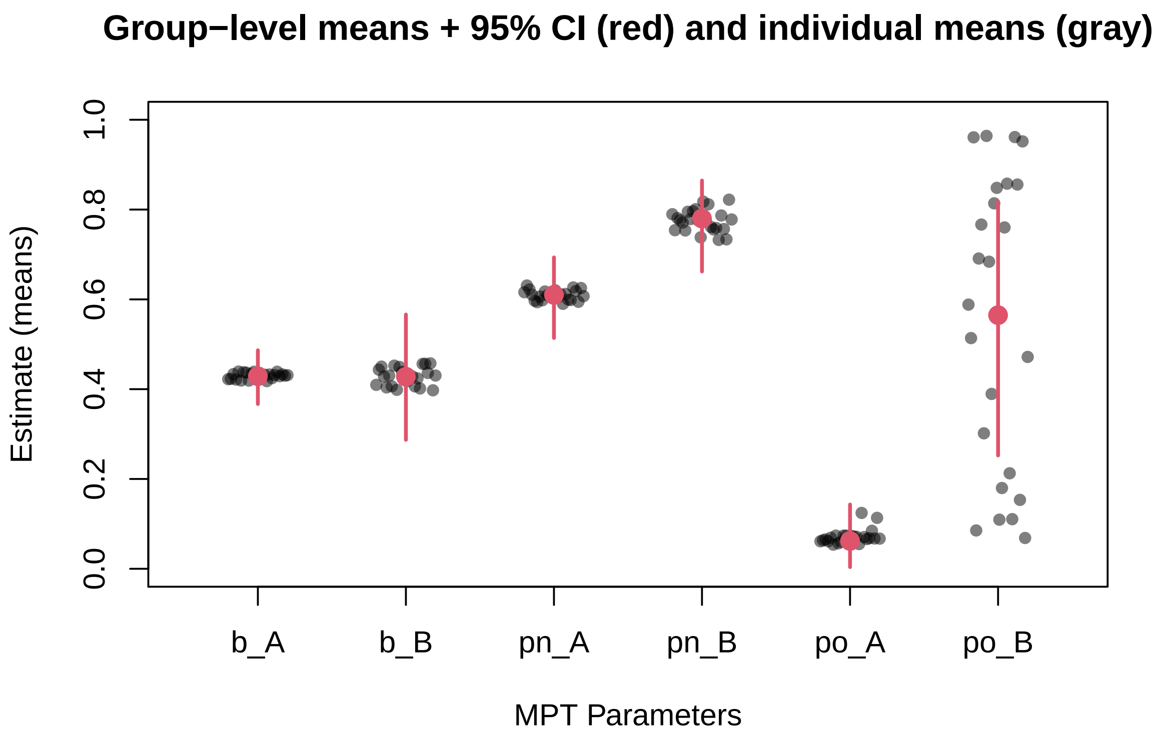
```
#Young adults' memory and guessing parameters
plotParam(fittedModelY, select=c("b_A", "b_B", "pn_A", "pn_B", "po_A", "po_B"))
```



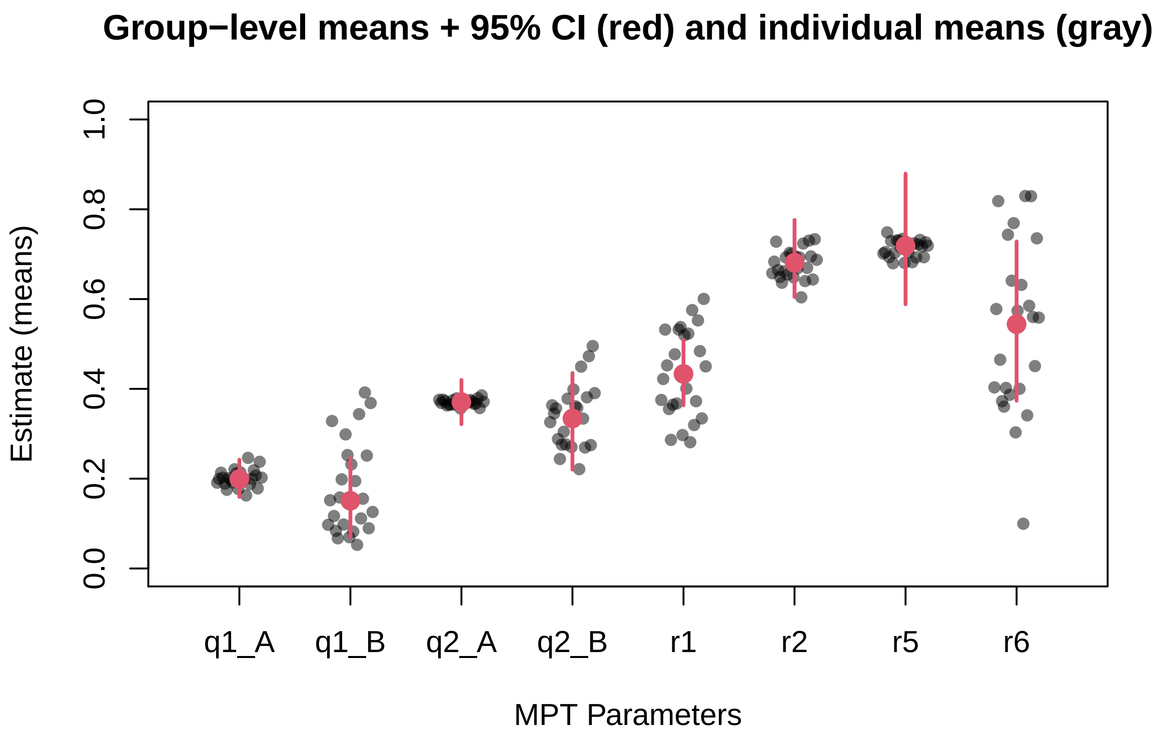
```r
#Young adults' response mapping parameters
plotParam(fittedModelY, select=c("q1_A", "q1_B", "q2_A", "q2_B", "r1", "r2",
                                 "r5", "r6"))
```



```
#Old adults' memory and guessing parameters
plotParam(fittedModelO, select=c("b_A", "b_B", "pn_A", "pn_B", "po_A", "po_B"))
```



```
#Old adults' response mapping parameters
plotParam(fittedModelO, select=c("q1_A", "q1_B", "q2_A", "q2_B", "r1", "r2",
                                   "r5", "r6"))
```



We will focus now on deriving between-group comparisons on the memory parameters (p_o and p_n) across condition A and B. A visual inspection of these estimates in the posterior plots above suggests that older and younger adults likely have very similar values of the detect-new parameter, but that for detect-old, there may be an interaction with age, given that older adults' estimates of p_o in Condition A is much smaller than the corresponding estimate of p_o in this condition among younger adults, whereas the 95% CI of the estimates in p_o for Condition B overlap across age groups.

The below function computes a comparison on the detection parameters across the two fitted models. The function computes the difference in mean parameters (young adults minus old adults) and the proportion of samples for which the mean in the young adult group is less than the mean in the old adult group.

```
#Are there age differences in detect-old probability in Condition A?
betweenSubjectMPT(fittedModelY, fittedModelO, par1="po_A")
```

```
##                      Mean   SD  2.5%   50% 97.5% Time-series SE n.eff  Rhat R_95%
## po_A.m1-po_A.m2 0.621 0.07 0.478 0.623 0.754          0.002  1395 1.001 1.001
## po_A.m1<po_A.m2 0.000 0.00 0.000 0.000 0.000            NaN     0   NaN   NaN
```

```
#Are there age differences in detect-old probability in Condition B?
betweenSubjectMPT(fittedModelY, fittedModelO, par1="po_B")
```

```
##                      Mean    SD   2.5%   50% 97.5% Time-series SE n.eff  Rhat R_95%
## po_B.m1-po_B.m2 0.246 0.169 -0.078 0.24 0.586          0.007   578 1.003 1.009
## po_B.m1<po_B.m2 0.068 0.252  0.000 0.00 1.000          0.007  1326 1.014 1.022
```

The results show that the difference in detect-old probability in condition A between young and old adults is about 0.62 (with a 95% CI ranging from 0.48 to 0.75), and that there are 0% of samples for which young adults had a smaller value of p_o than older adults. However, in Condition B, the mean difference of 0.25, with a 95% CI from -0.08 to 0.59, does include 0 within the interval, meaning that we cannot rule out the possibility that there is no age difference in this parameter.

References

Batchelder, W. H., & Riefer, D. M. (1999). Theoretical and empirical review of multinomial process tree modeling. *Psychonomic Bulletin & Review*, *6*, 57–86.

Bröder, A., Kellen, D., Schütz, J., & Rohrmeier, C. (2013). Validating a two-high-threshold measurement model for confidence rating data in recognition. *Memory*, *21*(8), 916–944.

Clark, H. H. (1973). The language-as-fixed-effect fallacy: A critique of language statistics in psychological research. *Journal of Verbal Learning and Verbal Behavior*, *12*(4), 335–359.

Denwood, M. J. (2016). Runjags: An R package providing interface utilities, model templates, parallel computing methods and additional distributions for mcmc models in JAGS. *Journal of Statistical Software*, *71*(9).

Dube, C., & Rotello, C. M. (2012). Binary rocs in perception and recognition memory are curved. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *38*(1), 130–151.

Gabry, J., & Mahr, T. (2018). *Bayesplot: Plotting for bayesian models.* Retrieved from https://CRAN.R-project.org/package=bayesplot

Gronau, Q. F., Sarafoglou, A., Matzke, D., Ly, A., Boehm, U., Marsman, M., … Steingroever, H. (2017). A tutorial on bridge sampling. *Journal of Mathematical Psychology*, *81*, 80–97.

Gronau, Q. F., & Singmann, H. (2017). *Bridgesampling: Bridge sampling for marginal likelihoods and bayes factors.* Retrieved from https://CRAN.R-project.org/package=bridgesampling

Heck, D. W., Arnold, N. R., & Denis, A. (2018). TreeBUGS: An R package for hierarchical multinomial-processing-tree modeling. *Behavior Research Methods*, *50*(1), 264–284.

Klauer, K. C. (2006). Hierarchical multinomial processing tree models: A latent-class approach. *Psychometrika*, *71*(1), 7–31.

Klauer, K. C. (2010). Hierarchical multinomial processing tree models: A latent-trait approach. *Psychometrika*, *75*(1), 70–98.

Klauer, K. C., & Kellen, D. (2011). The flexibility of models of recognition memory: An analysis by the minimum-description length principle. *Journal of Mathematical Psychology*, *55*(6), 430–450.

Lindley, D. V. (1957). A statistical paradox. *Biometrika*, *44*(1/2), 187–192.

Matzke, D., Dolan, C. V., Batchelder, W. H., & Wagenmakers, E.-J. (2015). Bayesian estimation of multinomial processing tree models with heterogeneity in participants and items. *Psychometrika*, *80*(1), 205–235.

Morey, R. D., Pratte, M. S., & Rouder, J. N. (2008). Problematic effects of aggregation in z roc analysis and a hierarchical modeling solution. *Journal of Mathematical Psychology*, *52*(6), 376–388.

Pazzaglia, A. M., Dube, C., & Rotello, C. M. (2013). A critical comparison of discrete-state and continuous models of recognition memory: Implications for recognition and beyond. *Psychological Bulletin*, *139*(6), 1173–1203.

Plummer, M. (2003). *JAGS: A program for analysis of bayesian graphical models using gibbs sampling.* Vienna, Austria. Retrieved from https://www.r-project.org/conferences/DSC-2003/Drafts/Plummer.pdf

Province, J. M., & Rouder, J. N. (2012). Evidence for discrete-state processing in recognition memory. *Proceedings of the National Academy of Sciences of the United States of America*, *109*(36), 14357–14362.

Rouder, J. N., & Lu, J. (2005). An introduction to bayesian hierarchical models with an application in the theory of signal detection. *Psychonomic Bulletin & Review*, *12*(4), 573–604.

Shammi, P., Bosman, E., & Stuss, D. T. (1998). Aging and variability in performance. *Aging, Neuropsychology, and Cognition*, *5*(1), 1–13.

Smith, J. B., & Batchelder, W. H. (2008). Assessing individual differences in categorical data. *Psychonomic Bulletin & Review*, *15*(4), 713–731.

Smith, J. B., & Batchelder, W. H. (2010). Beta-MPT: Multinomial processing tree models for addressing individual differences. *Journal of Mathematical Psychology*, *54*(1), 167–187.

Snodgrass, J. G., & Corwin, J. (1988). Pragmatics of measuring recognition memory: Applications to dementia and amnesia. *Journal of Experimental Psychology: General*, *117*(1), 34–50.

Swets, J. A. (1986). Form of empirical ROCs in discrimination and diagnostic tasks: Implications for theory and measurement of performance. *Psychological Bulletin*, *99*(2), 181–198.

Vehtari, A., Gabry, J., Yao, Y., & Gelman, A. (2019). Loo: Efficient leave-one-out cross-validation and waic for bayesian models. Retrieved from https://CRAN.R-project.org/package=loo

Vehtari, A., Gelman, A., & Gabry, J. (2017). Practical Bayesian model evaluation using leave-one-out cross-validation and WAIC. *Statistics and Computing*, *27*(5), 1413–1432.

Watanabe, S. (2013). A widely applicable bayesian information criterion. *Journal of Machine Learning Research*, *14*(Mar), 867–897.