CMBBHT: Binomial Tutorial

Contents

Introduction

This is a tutorial that shows an in-depth example of using the R package CMBBHT. The package name is an (excessive) acronym that expands to Cell Means Based Bayesian Hypothesis Tests. You can download the main package manual as a PDF.

CMBBHT provides a reasonably simple interface for performing Bayesian hypothesis tests in designs where cell means, differences between cells, or something similar are estimated in a factorial design. Factorial designs are very common in psychology. Many studies analyze their data directly with analysis of variance (ANOVA). I tend to use various kinds of psychometric model, so instead of having data to analyze, I instead have parameter estimates from the fitted model. It isn't good practice to use ANOVA on parameter estimates, so I needed some other way to perform tests of main effects and interactions in parameter values for my psychometric models.

I worked out some not-too-complex logic that allows for the estimation Bayes factors related to tests of main effects and interactions (generically, *ANOVA effects* or just *effects*). The logic is explained in the package manual along with code examples that perform the steps in the procedure.

This document includes an in-depth tutorial showing how to use CMBBHT with an example model. I will specify the model, fit it to simulated data, and show how to use CMBBHT to test hypotheses about model

parameters. Additional topics are discussed, including how to deal with different kinds of factorial design and how to analyze within-participants and between-participants data.

Model

This tutorial works with a binomial model for a factorial within-participants (repeated measures) design, although I will also show how to analyze between-participants designs without changing the model. Each participant will perform multiple trials of a task in each of several different experimental conditions. Each trial will result in a success or a failure, so the data are binomial, with the number of successes and the number of trials for each participant by experimental condition cell being the dependent variable. We are primarily interested in the effect of experimental condition, but we want to also estimate baseline participant ability to account for differences between participants and to better account for the differences between experimental conditions.

The data level of the model is

$$S_{ij} \sim \text{Binomial}(N_{ij}, P_{ij})$$

where S_{ij} are the number of successes for the ith participant in the jth experimental condition, N_{ij} is the number of trials, and P_{ij} is the probability of a success. Rather than directly estimating P_{ij}, it is much more parsimonious to estimate a participant parameter and a condition parameter and combine those two parameters to get P_{ij}. This is done as follows

$$P_{ij} = \text{logit}^{-1}(\rho_i + \delta_j)$$

where logit^{-1} is the CDF of the logistic distribution (i.e. `plogis` in R), ρ_i is the participant parameter and δ_j is the condition parameter. The use of the logit transform guarantees that P_{ij} is bounded between 0 and 1 while ρ_i and δ_j can take on any value.

To obtain hierarchical constraint on the estimation of the participant parameters, ρ_i, a hierarchical prior is used. Note that I parameterize Normal distributions in terms of variance, while stan and R parameterize them in terms of standard deviation.

$$\begin{aligned}
\rho_i &\sim \text{Normal}(\mu_\rho, \sigma_\rho^2) \\
\mu_\rho &\sim \text{Normal}(0, 2) \\
\sigma_\rho^2 &\sim \text{Inverse Gamma}(2, 2)
\end{aligned}$$

Notice that an additional advantage of using the logit transformation and allowing ρ_i to take on any value is that a Normal prior can be placed on ρ_i. If ρ_i was bounded between 0 and 1, a different prior would have to be used, probably with loss of conjugacy. In this case, the prior on ρ_i is conjugate.

The condition parameters, δ_j, have a moderately informative Cauchy prior placed on them.

$$\delta_j \sim \text{Cauchy}(0, 1)$$

Note that the priors on ρ_i and δ_j are not uninformative, instead being moderately informative. The use of moderately informative priors, or at least not uninformative priors, is very important to the calcuation of meaningful Bayes factors. See the Effect of Uninformative Priors section of the CMBBHT manual for more information on the effect of uninformative priors. See the Choosing Priors on δ section of this document for information on how this specific prior was chosen.

As the model is written so far, the mean of the δ_j parameters and the mean of the ρ_i parameters could trade off perfectly. To allow for identification of the parameters, it is neccessary to place an additional constraint

on those parameters. I will use my favorite constraint, which is a cornerstone parameterization in which one of the δ_j is set to the constant value 0. For this tutorial, I will set

$$\delta_1 = 0$$

to identify the parameters of the model.

Stan Model

The model was specified and fit using stan through the rstan package interface. The stan model specification is included here for completeness. You can probably skip it for now.

```
data {
  int<lower=0> I; // Number of participants
  int<lower=0> J; // Number of conditions

  int<lower=0> trials[I,J]; // Number of trials
  int<lower=0> success[I,J]; // Number of successes

  int<lower=0> nuc; // Number of unique conditions

  // A vector where a 0 means that condition is the cornerstone condition.
  // All matching nonzero values share the same delta parameter value.
  int deltaEq[J];
}
parameters {

  real rho[I];
  real delta_base[nuc]; // Does not include the cornerstone condition

  real mu_rho;
  real<lower=0.0001> var_rho;

}
transformed parameters {

  real delta[J]; // Includes the cornerstone condition

  real<lower=0.0001, upper=0.9999> P[I,J];

  for (j in 1:J) {

    // Copy from delta_base to delta, observing the cornerstone condition
    int deltaInd = deltaEq[j];
    if (deltaInd >= 1) {
      delta[j] = delta_base[deltaInd];
    } else {
      delta[j] = 0;
    }

    for (i in 1:I) {
      P[i,j] = inv_logit(rho[i] + delta[j]);
    }
  }
```

```
}
model {

  mu_rho ~ normal(0, sqrt(2));
  var_rho ~ inv_gamma(2, 2);

  // Prior on the non-cornerstone deltas
  for (k in 1:nuc) {
    delta_base[k] ~ cauchy(0, 1);
  }

  for (i in 1:I) {
    rho[i] ~ normal(mu_rho, sqrt(var_rho));

    for (j in 1:J) {
      success[i,j] ~ binomial(trials[i,j], P[i,j]);
    }
  }
}
```

Choosing Priors on δ

One of the difficulties in choosing reasonable moderately-informative priors on the δ_j is that fact that probabilities of success, P_{ij}, are related to δ_j with the following expression (restated from the model definition).

$$P_{ij} = \text{logit}^{-1}(\rho_i + \delta_j)$$

To help understand how P changes with δ, define

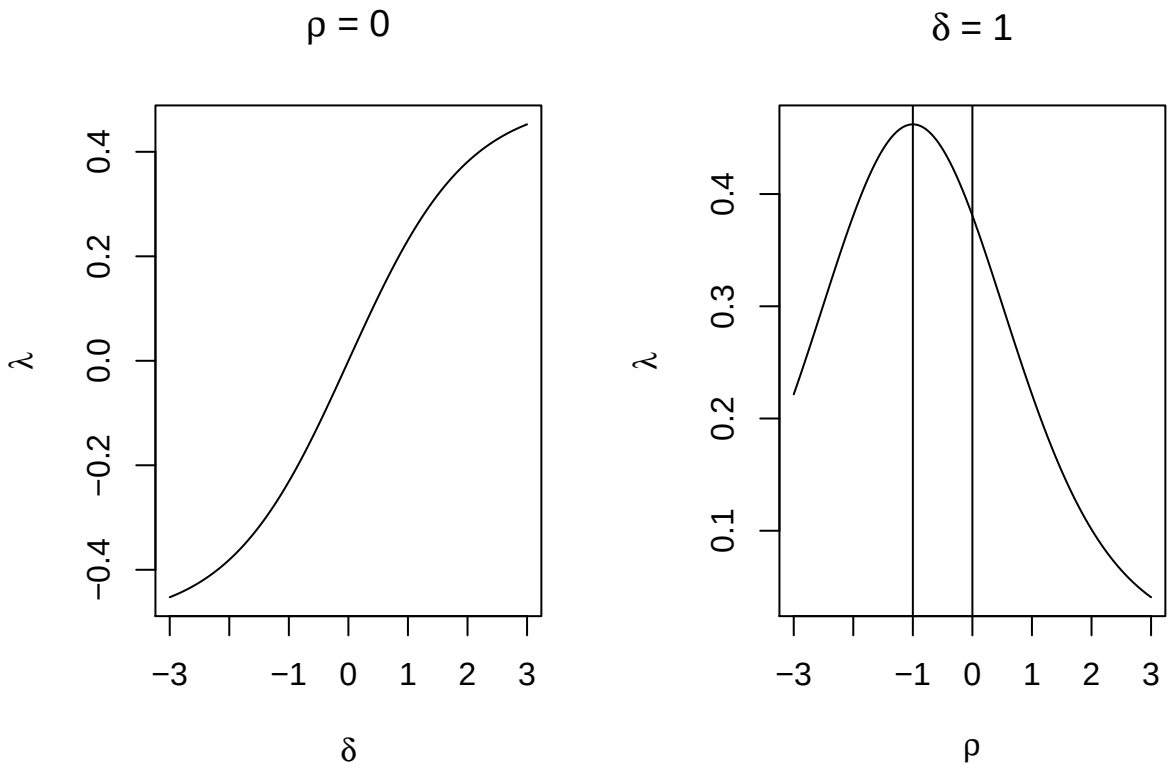
$$\lambda = \text{logit}^{-1}(\rho + \delta) - \text{logit}^{-1}(\rho + 0)$$

Thus, λ is the difference in the probability space between δ having some value versus δ being 0, i.e. how P changes as δ changes. Note that λ is defined for some ρ, as the effect that δ has on P depends on ρ.

```
lambda = function(rho, delta) {
    plogis(rho + delta) - plogis(rho)
}
```

Using λ, we can plot how P changes with δ for some value of ρ, which is shown in the left panel below for $\rho = 0$, which transforms to a P of 0.5 if $\delta = 0$. We can also plot λ with respect to ρ for for a constant value of δ, which is shown in the right panel.

```
par(mfrow=c(1,2))

deltas = seq(-3, 3, 0.01)
plot(deltas, lambda(rho=0, deltas), type='l', xlab=bquote(delta),
  ylab=bquote(lambda), main=bquote(rho*" = "*0))

rhos = seq(-3, 3, 0.01)
plot(rhos, lambda(rhos, delta=2), type='l', xlab=bquote(rho),
  ylab=bquote(lambda), main=bquote(delta*" = "*1))
abline(v=c(-1,0))
```



The left plot shows that λ is a nonlinear function of δ, with the shape indicating that extreme values of δ result in diminishing returns. The right plot shows the interesting result that a fixed value of δ affects P most strongly when $\rho + \delta/2 = 0$. Since the δ_j are the same for all participants, this indicates that participants with different ρ_i will experience different changes in P for the same δ.

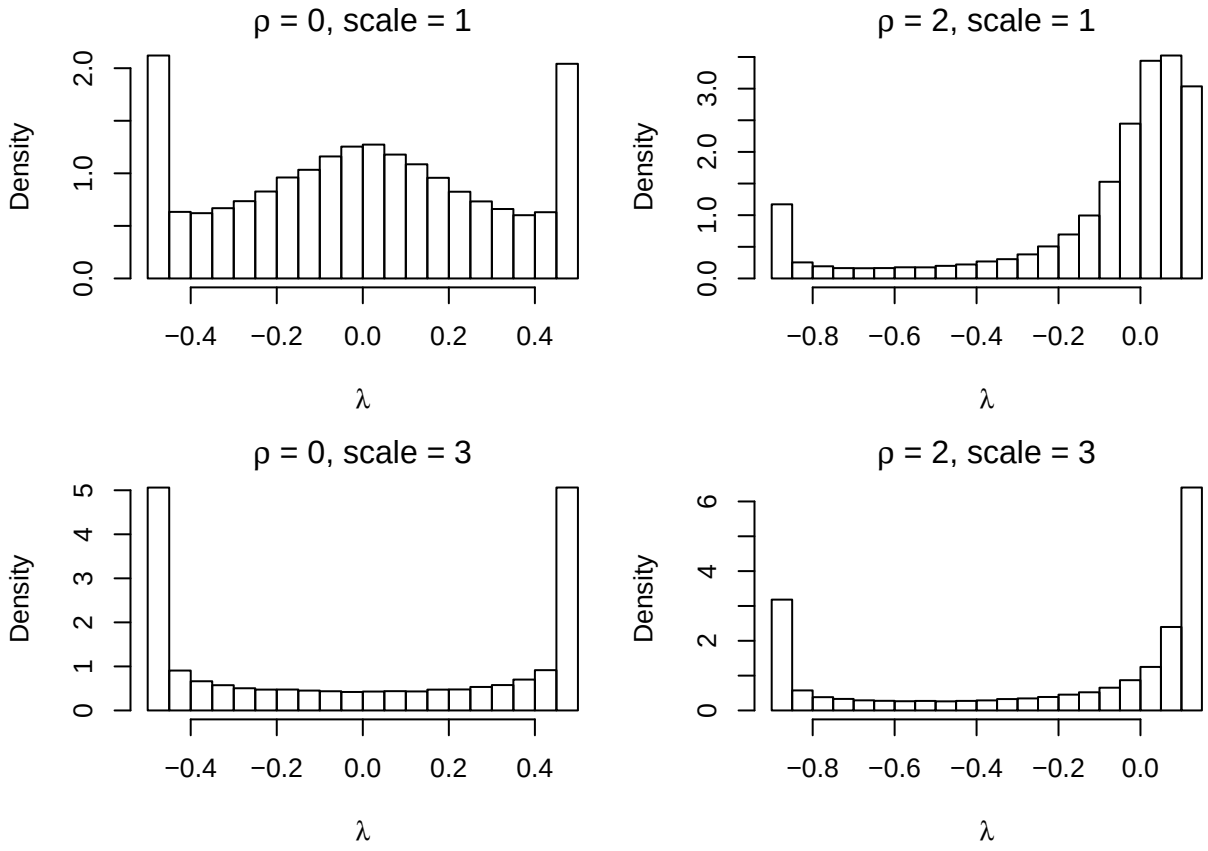
With the preceding information in mind, it should be clear why choosing a moderately informative prior on δ requires a little thought.

The value of the location parameter of the prior on δ can be easily chosen to be 0. The Cauchy distribution is symmetrical, so choosing a location of 0 indicates the belief that the most likely outcome is that there is no difference between any of the task conditions (the mode of the Cauchy is at its location). Thus, the focus is on the scale parameter of the Cauchy.

I will use the strategy of varying ρ and the prior scale, sampling from the prior on δ, and calculating λ. Plotted below are histograms of λ for some different values of ρ and the prior scale of δ.

```
sampleLambdaPrior = function(rho, loc0, scale0, n) {
  delta = rcauchy(n, loc0, scale0)

  lambda(rho, delta)
}

par(mfrow=c(2,2), mar=c(4.5,4,1.5,1))

ce = sampleLambdaPrior(0, 0, 1, n=1e5)
hist(ce, main=bquote(rho*" = "*0*", scale = "*1), prob=TRUE, xlab=bquote(lambda))

ce = sampleLambdaPrior(2, 0, 1, n=1e5)
```

```
hist(ce, main=bquote(rho*" = "*2*", scale = "*1), prob=TRUE, xlab=bquote(lambda))

ce = sampleLambdaPrior(0, 0, 3, n=1e5)
hist(ce, main=bquote(rho*" = "*0*", scale = "*3), prob=TRUE, xlab=bquote(lambda))

ce = sampleLambdaPrior(2, 0, 3, n=1e5)
hist(ce, main=bquote(rho*" = "*2*", scale = "*3), prob=TRUE, xlab=bquote(lambda))
```



A ρ of 0 corresponds to a P of 0.5 and a ρ of 2 corresponds to a P of 0.88. I find the distributions with a scale of 3 to put too much mass on extreme values of λ, while a scale of 1 places a more appropriate amount of mass of moderate values of λ. In addition, a scale of 1 is wide enough that when ρ is high, that the prior density is still reasonably high for large absolute magnitude values of λ.

Fitting the Model

The model is implemented in Stan, a Bayesian modeling language, using the `rstan` package. A link to the Stan model file can be found at the end of this post. In addition, I use a couple of helper functions, `sampleData`, `fitModel`, and `sampleDeltaPriors`, which I include here, but which you can probably skip without much loss.

```
library(rstan)

sampleData = function(trueRho, trueDelta, trialsPerCell = 50) {
  I = length(trueRho)
  J = length(trueDelta)
```

```
  success = trials = matrix(trialsPerCell, nrow=I, ncol=J)
  for (i in 1:I) {
    for (j in 1:J) {
      p = plogis(trueRho[i] + trueDelta[j])
      success[i,j] = rbinom(1, trials[i,j], p)
    }
  }
  list(success=success, trials=trials)
}

fitModel = function(success, trials, deltaEq = NULL) {
  I = nrow(success)
  J = ncol(success)

  if  (is.null(deltaEq)) {
    deltaEq = seq.int(0, J - 1)
  }

  mod_data = list(I=I, J=J, trials=trials, success=success,
                  deltaEq = deltaEq)
  mod_data$nuc = length(unique(mod_data$deltaEq)) - 1

  rval = list(mod_data = mod_data)

  rval$fit = stan(file = 'binomialModel.stan', data = mod_data,
              iter = 5000, chains = 2, control = list(adapt_delta = 0.8))

  rval$param = extract(rval$fit, permuted=TRUE)
  rval$iterations = nrow(rval$param$delta)

  rval
}

sampleDataAndFitModel = function(trueRho, trueDelta, trialsPerCell = 50, deltaEq = NULL) {
  d = sampleData(trueRho, trueDelta, trialsPerCell)
  fitModel(d$success, d$trials, deltaEq)
}
```

Choose true parameter values and some other basic configuration for the data simulation.

```
set.seed(123)

# Sample true rhor values
trueRho = rnorm(20, 0, 0.3)

# Specify true delta values
trueDelta = c(0, 0.5, 0.5, 0.5)

# Number of trials per participant by condition cell
trialsPerCell = 50
```

Given the true parameter values, sample data and then fit the model with that data.

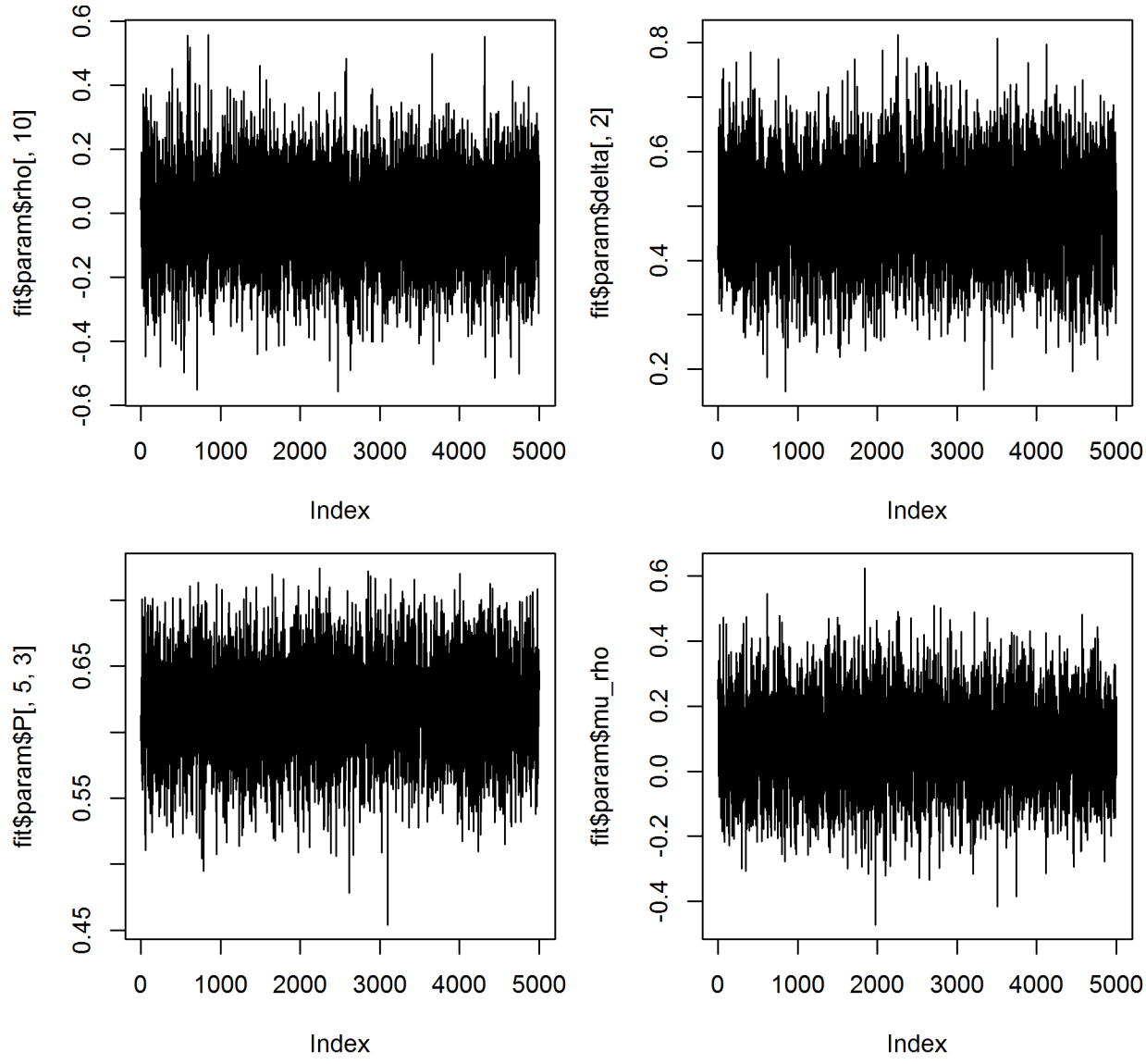
```
dat = sampleData(trueRho, trueDelta, trialsPerCell)
dat$success
```

```
##       [,1] [,2] [,3] [,4]
##  [1,]   19   30   30   30
##  [2,]   21   34   33   31
##  [3,]   33   33   41   37
##  [4,]   22   35   31   34
##  [5,]   30   29   27   33
##  [6,]   30   41   38   39
##  [7,]   24   33   30   30
##  [8,]   23   27   24   25
##  [9,]   24   40   29   31
## [10,]   22   29   31   34
## [11,]   32   34   36   33
## [12,]   31   33   25   28
## [13,]   22   36   36   31
## [14,]   27   30   33   35
## [15,]   26   34   29   29
## [16,]   31   38   37   32
## [17,]   27   29   28   32
## [18,]   17   20   29   22
## [19,]   33   28   32   37
## [20,]   24   24   29   30
```

```
fit = fitModel(dat$success, dat$trials)
```

Perform some basic diagnostics on the fitted model. First, visually check convergence of some of the parameters, which looks great.

```
par(mfrow=c(2,2), mar=c(4.5, 4.5, 0.5, 0.5))
plot(fit$param$rho[,10], type='l')
plot(fit$param$delta[,2], type='l')
plot(fit$param$P[,5,3], type='l')
plot(fit$param$mu_rho, type='l')
```

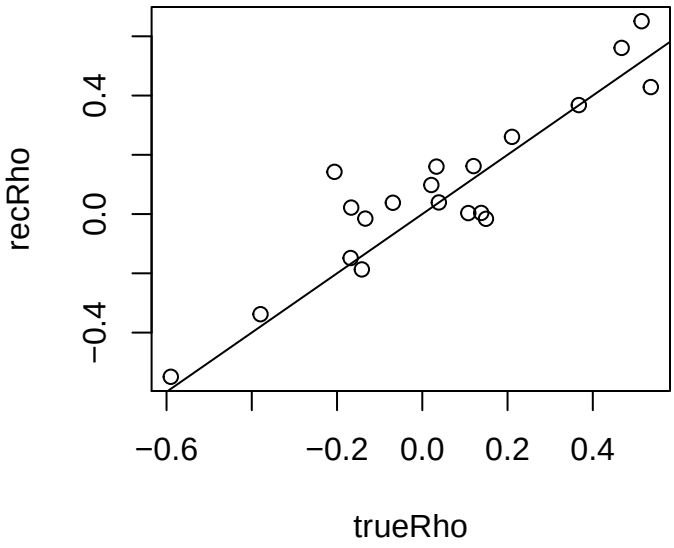


Second, are the ρ_i values properly recovered? We know the true parameter values used to generate the data, so compare those values to the posterior means from the fitted model.

```
par(mar=c(5, 4, 0, 0))
recRho = apply(fit$param$rho, 2, mean)
cor(trueRho, recRho)
```

```
## [1] 0.9132035
```

```
plot(trueRho, recRho)
abline(0,1)
```



Third, as inference will focus on the effects of task condition, check that the δ_j parameter values are properly recovered.

```
recDelta = apply(fit$param$delta, 2, mean)
trueDelta
```

```
## [1] 0.0 0.5 0.5 0.5
```

```
round(recDelta, 2)
```

```
## [1] 0.00 0.49 0.45 0.47
```

In all cases, the diagnostics suggest that the model fitting was successful within reasonable tolerances. Remeber that noise is added during the data sampling step, so the recovered parameter values are likely to vary somewhat from the true parameter values just due to data sampling noise.

Collecting the Pieces

I want to test the hypothesis that the conditions differed from one another. This is conceptually equivalent to a one-way ANOVA, just on model parameters rather than data. We are interested in the differences between task conditions. Those differences are accounted for in the model by the condition parameters, δ_j. Thus, inference will focus on those parameters. For this test, no information about the participant parameters is required as the participant parameters are the same in all task conditions.

As an outline, to test this hypothesis with CMBBHT, you need:

1. A matrix of samples from the prior distribution of the parameters of interest (e.g. δ_j). This is usually just a mechanical process that requires a few lines of code (10 or less for simple cases and perhaps 50 lines for a complex case with hierarchical priors).
2. A matrix of samples from the posterior distribution of the parameters of interest. You already have to sample these parameters to do most Bayesian analyses, so the only work you have to do here is to put the parameters into a matrix if they are not already in a matrix.
3. A data frame with a number of rows equal to the number of parameters of interest and a column for each factor of the design. Each row provides the levels of the factors that correspond to the parameters of interest. Examples of this will be given and, as you will see, this is typically very easy to create.

Piece 1: Prior Samples

In this case, sampling from the priors is very simple. The prior on δ_j is a Cauchy with location 0 and scale 1, except for δ_1, which is set to 0. To sample from the prior, we will use the `sampleDeltaPriors` function defined below.

```
sampleDeltaPriors = function(nCond, iterations) {
  # Set matrix to 0 and skip over the cornerstone condition
  pr = matrix(0, nrow=iterations, ncol=nCond)
  for (j in 2:nCond) {
    pr[,j] = rcauchy(iterations, 0, 1)
  }
  pr
}

prior = sampleDeltaPriors(ncol(fit$param$delta), fit$iterations)

dim(prior)
```

```
## [1] 5000    4
```

```
head(prior)
```

```
##      [,1]        [,2]       [,3]        [,4]
## [1,]    0  -0.3898615 -0.1026647  -0.5934911
## [2,]    0  -0.4188521 13.1088383   1.2610299
## [3,]    0   1.8005084  1.4693157  -0.8646756
## [4,]    0 -23.8958263 -0.5687443 -17.7704449
## [5,]    0  52.4804025 -3.2195704  -2.8649549
## [6,]    0  -0.7585241 -2.7939867  -6.7815133
```

As you can see, the first column of `prior` is all 0s because that is the cornerstone condition.

Piece 2: Posterior Samples

The samples from the posterior of the δ_j were taken during model fitting and are stored in `fit$param$delta`.

```
post = fit$param$delta

dim(post)
```

```
## [1] 5000    4
```

```
head(post)
```

```
##
## iterations [,1]      [,2]      [,3]      [,4]
##       [1,]    0 0.4269326 0.3942929 0.4208992
##       [2,]    0 0.3782313 0.3426868 0.3615241
##       [3,]    0 0.5587148 0.5235968 0.4784190
##       [4,]    0 0.4845315 0.3089328 0.3378329
##       [5,]    0 0.6440234 0.5422284 0.4982239
##       [6,]    0 0.5507246 0.5368799 0.5900135
```

For the default method of estimating the Bayes factor, the dimensions of the prior and posterior matrices should be the same. It is possible to use different numbers of prior and posterior samples, but for the defaults of CMBBHT, the number of iterations should be the same. You will get a warning if they are not equal.

```r
all(dim(prior) == dim(post))
```

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

Piece 3: Factors

We also need to create a data frame that tells CMBBHT about what factor levels each condition is related to. We are assuming a one-factor design, so we will make a data frame with one factor, called `f`.

```r
factors = data.frame(f = 1:ncol(prior))
factors
```

```
##   f
## 1 1
## 2 2
## 3 3
## 4 4
```

Even though the design is just a one-factor design, it is still neccessary to specify the design. I will give examples of specifying `factors` for multi-factor designs later.

Testing the Hypothesis

With the three pieces `prior`, `post`, and `factors` set up, we can test a hypothesis about whether there is a main effect of the single factor. This test uses the function `testHypothesis` from CMBBHT.

```r
library(CMBBHT)

ht = testHypothesis(prior, post, factors, "f")
ht
```

```
## $success
## [1] TRUE
##
## $bf01
## [1] 0.01261597
##
## $bf10
## [1] 79.26461
##
## $pKept
## [1] 0.9
```

The result of `testHypothesis` is (by default) a list with some elements. `bf01` and `bf10` are Bayes factors related to the hypothesis being tested. For `bf01`, "01" means that the Bayes factor is for the null hypothesis (0) over the alternative hypothesis (1), thus it is a Bayes factor in favor of the null hypothesis. For `bf10`, it is for the alternative hypothesis (1) over the null (0), so `bf10` is the Bayes factor in favor of the hypothesis that there is an effect.

The Bayes factor in favor of there being a main effect, `bf10`, is 79.3, which is clear evidence in favor of there being a main effect, which is plausible as an effect was built into the true parameter values.

Notice that this is the first place we have used anything from the CMBBHT package. Sampling from the priors and posteriors and specifying the factorial design are done independently of CMBBHT. You are free to specify and fit your model with few restrictions. The two most important restrictions are that 1) your model

includes cell means or something like cell means that account for differences between conditions and 2) that the priors on the parameters of interest are not uninformative.

See the Effect Parameters section of the package manual for information on performing follow up tests, such as pairwise comparisons of conditions.

The Cornerstone Condition

In the test performed above, the cornerstone parameter values were included in the test. The δ parameter in the cornerstone condition is the constant value 0. A parameter with constant value 0 doesn't seem like it should be important, which leads to a temptation to exclude the cornerstone condition. However, it is wrong to exclude the cornerstone condition, for reasons that will be demonstrated in an example here.

The cornerstone condition is at index 1, so exclude the first δ parameter.

```
prior = prior[ , -1]
post = post[ , -1]
factors = data.frame(f = 1:ncol(prior))

ht = testHypothesis(prior, post, factors, "f")
ht$bf01
```

```
## [1] 498.8274
```

When we perform this test, we find strong evidence in favor of the null hypothesis of no effect. The reason for differing results depending on the inclusion or exclusion of the cornerstone condition is that there is no difference between the non-cornerstone conditions in this example. The true condition effects are in `trueDelta`. As we can see, the non-cornerstone conditions all have the same difference from the cornerstone condition.

```
trueDelta
```

```
## [1] 0.0 0.5 0.5 0.5
```

This means that the non-cornerstone conditions do not differ from one another. When a test is performed on only the non-cornerstone conditions, all of those conditions have the same value of δ. Thus, there is no main effect when only the non-cornerstone conditions are considered. When the cornerstone condition is included, however, it is possible to detect that there is a difference between the cornerstone condition and the non-cornerstone conditions. True effects can be missed if the cornerstone condition is not included, so be sure to include it.

If you use a different constraint than a cornerstone parameterization, such as a sums-to-zero constraint, make sure that you include all of the condition effect parameters, not just the estimated parameters. In general, you should include one parameter per cell of the experimental design.

Factorial Design

Let us imagine that our design with four levels is actually a 2x2 factorial design. We will get the priors and posteriors as before.

```
prior = sampleDeltaPriors(ncol(fit$param$delta), fit$iterations)
post = fit$param$delta
```

When we create the `factors` data frame, we create two factors, `let` and `num`.

```
factors = data.frame(
  let = c('a', 'a', 'b', 'b'),
```

```
  num = c( 1,   2,   1,   2)
)

factors
```

```
##   let num
## 1   a   1
## 2   a   2
## 3   b   1
## 4   b   2
```

The critical thing whenever you make `factors` is that the rows of `factors` correspond to the columns of the `prior` and `post`. The `j`th column of prior and post corresponds to the `j`th row of `factors`. Thus `j = 3` corresponds to the `b` level of the `let` factor and the `1` level of the `num` factor.

Since there are two factors, we can test 1) the main effect of `let`, 2) the main effect of `num`, and 3) the interaction between `let` and `num`. We can do those tests as follows.

```
testHypothesis(prior, post, factors, "let")$bf10     # Main of let
testHypothesis(prior, post, factors, "num")$bf10     # Main of num
testHypothesis(prior, post, factors, "let:num")$bf10 # Interaction
```

Note that interactions are specified by putting a colon between the factor names. The order of the factors is irrelevant.

Alternately, you can use the `testHypotheses` (plural) convenience function to test multiple hypotheses at once.

```
testHypotheses(prior, post, factors, c("let", "num", "let:num"))
```

```
##   testName success       bf01      bf10 pKept
## 1      let    TRUE 0.11174380  8.949042   0.9
## 2      num    TRUE 0.02089819 47.851024   0.9
## 3  let:num    TRUE 0.34268182  2.918159   0.9
```

If you have many effects to test, you may want to leave the specific tests unspecified, in which case all possible effects are tested. Sometimes, however, you might want to perform only a subset of all possible tests.

Unbalanced Factorial Design

This seems like a bad example because it is too simple of a design.

Let us imagine that our design with four cells is a 2x3 factorial design, but with two missing cells. We will get the priors and posteriors as before.

```
prior = sampleDeltaPriors(ncol(fit$param$delta), fit$iterations)
post = fit$param$delta

factors = data.frame(
  let = c('a', 'b', 'b', 'b'),
  num = c( 1,   1,   2,   3)
)
```

`factors` still has the same two factors, but the factors have different levels. Note that the design is such that an interaction cannot be estimated, only the two main effects.

Below is a table of the true δ values and how they map onto the cells of the design.

/	1	2	3
a	0	.	.
b	0.5	0.5	0.5

This table implies that we should find a main effect of `let` because the δ is 0 for `let = a` and 0.5 for `let = b`. We may find a main effect of `num` because the marginal mean δs for `num` 1, 2, and 3 are 0.25, 0.5, and 0.5, respectively, but it should be a weak effect if present. We find:

```
testHypothesis(prior, post, factors, "let")$bf10
```

```
## [1] 56498.76
```

```
testHypothesis(prior, post, factors, "num")$bf10
```

```
## [1] 0.353742
```

If you were totry to test the interaction, you would get a helpful error message telling you that the design lacks the appropriate cells to be able to estimate interaction terms.

Given that the design is unbalanced, the result of the tests depends on what design matrix you choose to use. For more information about unbalanced designs, see the Non-Fully Crossed/Unbalanced Designs section of the CMBBHT manual. For example, you can use a design matrix with both main effects.

```
testHypothesis(prior, post, factors, "let", dmFactors = "let")$bf10
```

```
## [1] 56498.76
```

```
testHypothesis(prior, post, factors, "let", dmFactors = " ~ let + num")$bf10
```

```
## [1] 3020.122
```

Naturally, changing the design matrix affects the results when using non-orthogonal contrasts.

Between-Participants Design

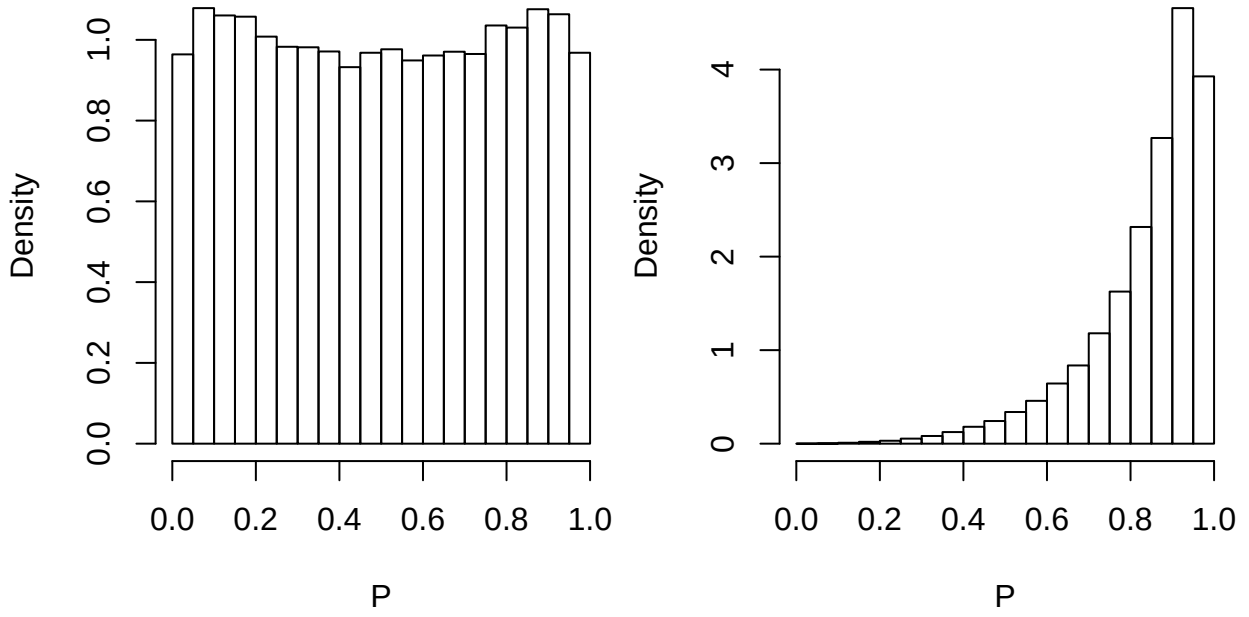
Assume that we now have a mixed design with a between-participants factor with two levels and a within-participants factor with three levels. In this case, the analysis becomes more complex. It is possible, however, to analyze this design without modifying the model, which is convenient, but requires a minor fudge as will be explained.

Choosing the ρ Prior

One important difference between fully within-participants designs and between-participants or mixed between/within designs is that the prior on ρ, and neccessarily also μ_ρ, becomes relevant. With a fully within-participants design, the only information needed to test the main effect of the within-participants factor are the δ_j. With a between-participants or mixed design, the mean of ρ is also important to determine which groups differ from one another. It is not useful to use uninformative priors on ρ (or μ_ρ), so I will choose moderately-informative priors.

The following code allows you to play around with the values of the priors on μ_ρ and σ^2_ρ, both of which contribute to the prior on ρ. Since ρ is transformed to the probability space to become the manifest quantity P, ρ is harder to interpret than P, so I will plot P rather than ρ. The priors I have used result in an approximately uniform prior on P, which can be seen in the left panel below.

```r
sampleRhoPrior = function(mu0, var0, a0, b0, n = 1e5) {
  mu_rho = rnorm(n, mu0, sqrt(var0))
    var_rho = MCMCpack::rinvgamma(n, a0, b0)

    rnorm(n, mu_rho, sqrt(var_rho))
}

par(mfrow=c(1,2), mar=c(5, 4, 0.25, 0.5) )
rhoPrior = sampleRhoPrior(mu0=0, var0=sqrt(2), a0=2, b0=2)
hist(plogis(rhoPrior), main="", prob=TRUE, xlab="P")

rhoPrior = sampleRhoPrior(mu0=2, var0=1^2, a0=3, b0=0.5)
hist(plogis(rhoPrior), main="", prob=TRUE, xlab="P")
```



If you don't like the shape of the prior I used, I encourage you to play around with the prior parameter values until you find a shape that you like. For example, if chance performance in your task results in 50% accuracy, you might want there to be only limited prior mass below 0.5, such as shown in the right panel above.

Fitting the Model

Once priors have been chosen, the next step is to fit the model separately for each group of participants. Note that the two groups' ρ parameters are sampled with a different mean which should result in there being a main effect of group. In addition, within each group, there are three levels of some within-participants factor. Thus, this is a 2 (group) X 3 (condition) design.

```r
set.seed(123)

# Group A
trueRho = rnorm(20, 0, 0.3)
trueDelta = c(0, -0.5, 0.5)

trialsPerCell = 50
```

```
grpA = sampleDataAndFitModel(trueRho, trueDelta, trialsPerCell)


# Group B
trueRho = rnorm(20, 0.5, 0.3)
trueDelta = c(0, 0.5, -0.5)

grpB = sampleDataAndFitModel(trueRho, trueDelta, trialsPerCell)

fitList = list(A = grpA, B = grpB)
```

Combining Group Means and Condition Effects

Once the model has been fitted, the next step is to collect prior and posterior parameter matrices. This is more complex than the within-participants designs because the analysis depnds on not only the condition parameters, δ_j, but also the value of ρ in each group.

We are interested in whether P depends on group. In order to calculate P in each group by condition cell of the design, we must combine the value of ρ of that group with the δ_j for that condition. But what value of ρ should we use: The individual ρ_i or the mean of the ρ parameters, μ_ρ?

As I will show later, using the individual ρ_i is equivalent to testing whether the *sample* means of ρ differ while using μ_ρ is equivalent to testing whether the *population* means of ρ differ. We obviously want to test whether the population means of ρ differ, so we will use μ_ρ.

The function `betweenParticipantsPieces` collects the neccessary information to test hypotheses using CMBBHT for this model. The process is fairly mechanical, so I will not go into much detail about it here. The key point is that we are using μ_ρ rather than the individual participant ρ to calculate P. Thus, the equation for P becomes

$$\bar{P}_{jk} = logit^{-1}(\mu_{\rho k} + \delta_{jk})$$

where $\bar{P}_{jk}$ is the mean P in the jth within-participants condition and kth between-participants group.

```
betweenParticipantsPieces = function(fitList) {

    # Create variables to be added to
    factors = prior = post = NULL

    for (grp in names(fitList)) {

        fit = fitList[[grp]]

        J = ncol(fit$param$delta)

        ## 3: Factors
        fact = data.frame(grp=grp, cond=1:J)
        factors = rbind(factors, fact)

        # Name the matrix columns for clarity
        cellNames = paste(grp, fact$cond, sep=":")

        ## 2: Posterior samples
        postP = matrix(NA, nrow=fit$iterations, ncol=J)
```

```
        for (j in 1:J) {
            # Note that the plogis transfomration is not strictly neccessary here
            # or below for the prior. Normally, I would omit unneccessary
            # transformations, but include it here for clarity purposes.
            postP[,j] = plogis(fit$param$mu_rho + fit$param$delta[,j])
        }

        colnames(postP) = cellNames

        post = cbind(post, postP)

        ## 1: Prior samples
        prior_mu_rho = rnorm(fit$iterations, 0, sqrt(2))
        priorDelta = sampleDeltaPriors(J, fit$iterations)
        priorP = priorDelta # Same dimensions

        for (j in 1:J) {
            priorP[,j] = plogis(prior_mu_rho + priorDelta[,j])
        }

        colnames(priorP) = cellNames

        prior = cbind(prior, priorP)
    }

    list(prior=prior, post=post, factors=factors)
}

# Get the pieces and copy them out of the list
pieces = betweenParticipantsPieces(fitList)

prior = pieces$prior
post = pieces$post
factors = pieces$factors
```

The completed prior and posterior matrices have been collected, the resulting matrices have one parameter per cell of this 2 X 3 design.

```
head(post)
```

```
##            A:1       A:2       A:3       B:1       B:2       B:3
## [1,] 0.5376251 0.3600798 0.6414104 0.6535019 0.7389250 0.5279756
## [2,] 0.4790691 0.3480230 0.6527458 0.6168422 0.7276099 0.5256906
## [3,] 0.5539470 0.3966356 0.6762792 0.7021535 0.7275307 0.5300302
## [4,] 0.4923122 0.3220939 0.6105332 0.6710803 0.7675352 0.5339942
## [5,] 0.4792395 0.3218828 0.5698502 0.6640706 0.7405649 0.5426193
## [6,] 0.4890836 0.3875872 0.6368493 0.6735276 0.7424994 0.5063129
```

In addition, we can verify that the columns of `prior` and `post` correspond to the rows of `factors`.

```
factors
```

```
##   grp cond
## 1   A    1
## 2   A    2
```

```
## 3   A    3
## 4   B    1
## 5   B    2
## 6   B    3
```

Once the three pieces have been collected, testing hypotheses is as easy as usual. Note that when the effects to be tested are not specified, all possible effects are tested.

```
testHypotheses(prior, post, factors)
```

```
##   testName success         bf01         bf10  pKept
## 1      grp    TRUE 8.379104e-02 1.193445e+01 1.0000
## 2     cond    TRUE 1.370797e+01 7.295026e-02 0.9000
## 3 grp:cond    TRUE 3.535645e-11 2.828338e+10 0.9994
```

As usual, you can do follow-up tests, etc., with `post` and `factors`. Here I examine the grouping of the levels of the within-participants `cond` factor.

```
ep = getEffectParameters(post, factors, "cond")
groupEffectParameters(ep)
```

```
##
## cond.2 A
## cond.3 A B
## cond.1   B
```

Whether to use ρ_i or μ_ρ

Earlier, I claimed that using the individual ρ_i parameters rather than μ_ρ was equivalent to testing whether the sample means of ρ differed between groups. To be more precise, using the ρ_i is equivalent to testing that the sample means differ in the asymptote as the amount of data per participant increases. I will now argue that point. I will present an example that makes the point for me and then explain why the example generalizes.

I will do two things in this example:

1. Assume a large, near infinite amout of data per participant.
2. Sample true ρ values for the two groups from the same population.

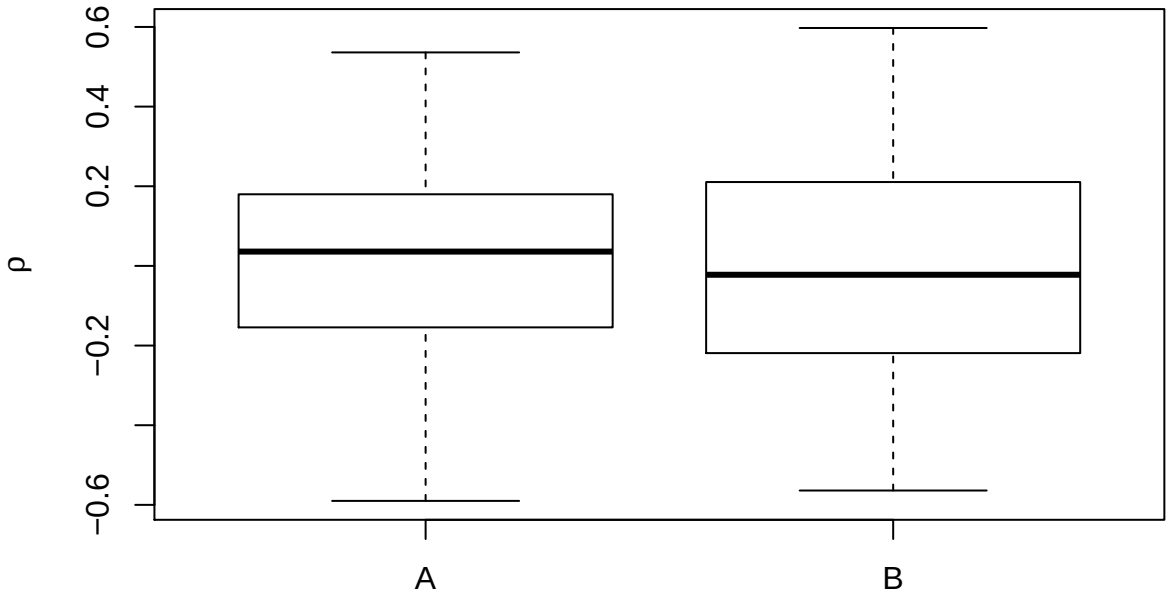
In addition, I will make the δ_j equal between the two groups to prevent any difference in δ being picked up as a difference between the groups, for example if the δ_j do not sum to the same value in both groups.

```
set.seed(123)

# Infinite data
trialsPerCell = 1e5

# Group A
trueRho_A = rnorm(20, 0, 0.3)
trueDelta_A = c(0, 0.5, -0.5)

grpA = sampleDataAndFitModel(trueRho_A, trueDelta_A, trialsPerCell)


# Group B
trueRho_B = rnorm(20, 0, 0.3)
trueDelta_B = c(0, 0.5, -0.5)
```

```
grpB = sampleDataAndFitModel(trueRho_B, trueDelta_B, trialsPerCell)

fitList = list(A = grpA, B = grpB)
```

As the boxplot below indicicates, the two groups' ρ parameters have very similar distributions and differ slightly in terms of the median value.

```
boxplot(trueRho_A, trueRho_B, names = c("A", "B"), ylab=expression(rho) )
```




```
mean(trueRho_A)
```

```
## [1] 0.04248714
```

```
mean(trueRho_B)
```

```
## [1] 0.007129911
```

The mean values of ρ are also very close. No trained statistician would look at the boxplot and think that the population means are probably different from one another, with their logic being that the individual variability is substantially larger than the difference between the means.

The function below is performs the same operations as `betweenParticipantsPieces` except that it uses ρ_i rather than μ_ρ.

```
# This function uses indivudal rho parameters
betweenParticipantsPieces_participant = function(fitList) {

    prior = post = factors = NULL
```

```
for (fn in names(fitList)) {

    fit = fitList[[fn]]

    I = ncol(fit$param$rho)
    J = ncol(fit$param$delta)

    ## 3: Factors

    # Cross participant by condition
    fact = expand.grid(part=1:I, cond=1:J)
    fact$grp = fn

    factors = rbind(factors, fact)

    cellNames = paste(fn, fact$part, fact$cond, sep=":")

    ## 2: Posterior samples
    postP = matrix(NA, nrow=fit$iterations, ncol=nrow(fact))

    for (rr in 1:nrow(fact)) {
        i = fact$part[rr]
        j = fact$cond[rr]
        postP[,rr] = plogis(fit$param$rho[,i] + fit$param$delta[,j])
    }

    colnames(postP) = cellNames

    post = cbind(post, postP)

    ## 1: Prior samples

    # Sample prior rho
    # First sample from priors on mu_rho and var_rho
    mu_rho = rnorm(fit$iterations, 0, sqrt(2))
    var_rho = MCMCpack::rinvgamma(fit$iterations, 2, 2)

    # Then given samples from mu_rho and var_rho, sample from prior on rho
    priorRho = matrix(NA, nrow=fit$iterations, ncol=I)
    for (i in 1:I) {
        priorRho[,i] = rnorm(fit$iterations, mu_rho, sqrt(var_rho))
    }

    # Sample prior delta
    priorDelta = sampleDeltaPriors(J, fit$iterations)

    # Combine rho and delta
    priorP = matrix(NA, nrow=fit$iterations, ncol=nrow(fact))
    for (rr in 1:nrow(fact)) {
        i = fact$part[rr]
        j = fact$cond[rr]

        priorP[,rr] = plogis(priorRho[,i] + priorDelta[,j])
```

```
        }
        colnames(priorP) = cellNames

        prior = cbind(prior, priorP)
    }

    list(factors = factors, prior = prior, post = post)

}
```

Using the same data and fitted models, use the ρ_i approach and the μ_ρ approach.

```
bp_rho = betweenParticipantsPieces_participant(fitList)

bp_mu = betweenParticipantsPieces(fitList)
```

One difference between the approaches is that when using ρ_i, you end up with $I * J * K$ (`20 * 3 * 2 = 120`) P parameters versus the $J * K$ (`3 * 2 = 6`) mean P parameters when using μ_ρ. Correspondingly, when using ρ_i, `factors` has three columns, including a participant column (`part`).

```
dim(bp_rho$post)
```

```
## [1] 5000  120
```

```
dim(bp_mu$post)
```

```
## [1] 5000    6
```

```
bp_rho$post[1:5, 1:6]
```

```
##          A:1:1     A:2:1     A:3:1     A:4:1     A:5:1     A:6:1
## [1,] 0.4563319 0.4851040 0.6153142 0.5038579 0.5121217 0.6246215
## [2,] 0.4573581 0.4821227 0.6146024 0.5059636 0.5130461 0.6246294
## [3,] 0.4584233 0.4816291 0.6140732 0.5025642 0.5123754 0.6245598
## [4,] 0.4577095 0.4832275 0.6155299 0.5036245 0.5128561 0.6249182
## [5,] 0.4579964 0.4841479 0.6144522 0.5046210 0.5123991 0.6242278
```

```
bp_rho$factors[1:5,]
```

```
##   part cond grp
## 1    1    1   A
## 2    2    1   A
## 3    3    1   A
## 4    4    1   A
## 5    5    1   A
```

Using both approaches, let's test a main effect of the `grp` factor. In both cases, I will just print `bf10`.

```
testHypothesis(bp_rho$prior, bp_rho$post, bp_rho$factors, "grp")$bf10
```

```
## Warning in polspline::logspline(x, lbound = lbound, ubound = ubound): too
## much data close together
```

```
## Warning in polspline::logspline(x, lbound = lbound, ubound = ubound): re-
## ran with oldlogspline
```

```
## [1] 1.706966e+27
```

```
testHypothesis(bp_mu$prior, bp_mu$post, bp_mu$factors, "grp")$bf10
```

```
## [1] 0.1115926
```

When using ρ_i, we find a very substantial Bayes factor in favor of there being a difference between the groups. This goes against statistical common sense based on the boxplot of true ρ parameters. When using μ_ρ, we instead find a Bayes factor in favor of there being no difference between the groups, which is more sensible.

To examine why these results differ so dramatically, I will examine the effect parameters related to the `grp` main effect.

```
# rho
ep_rho = getEffectParameters(bp_rho$post, bp_rho$factors, "grp")
eps_rho = summarizeEffectParameters(ep_rho)
eps_rho
```

```
##   effect         mean         2.5%          50%        97.5%
## 1  grp.A  0.004126142  0.003845378  0.004128266  0.004396831
## 2  grp.B -0.004126142 -0.004396831 -0.004128266 -0.003845378
```

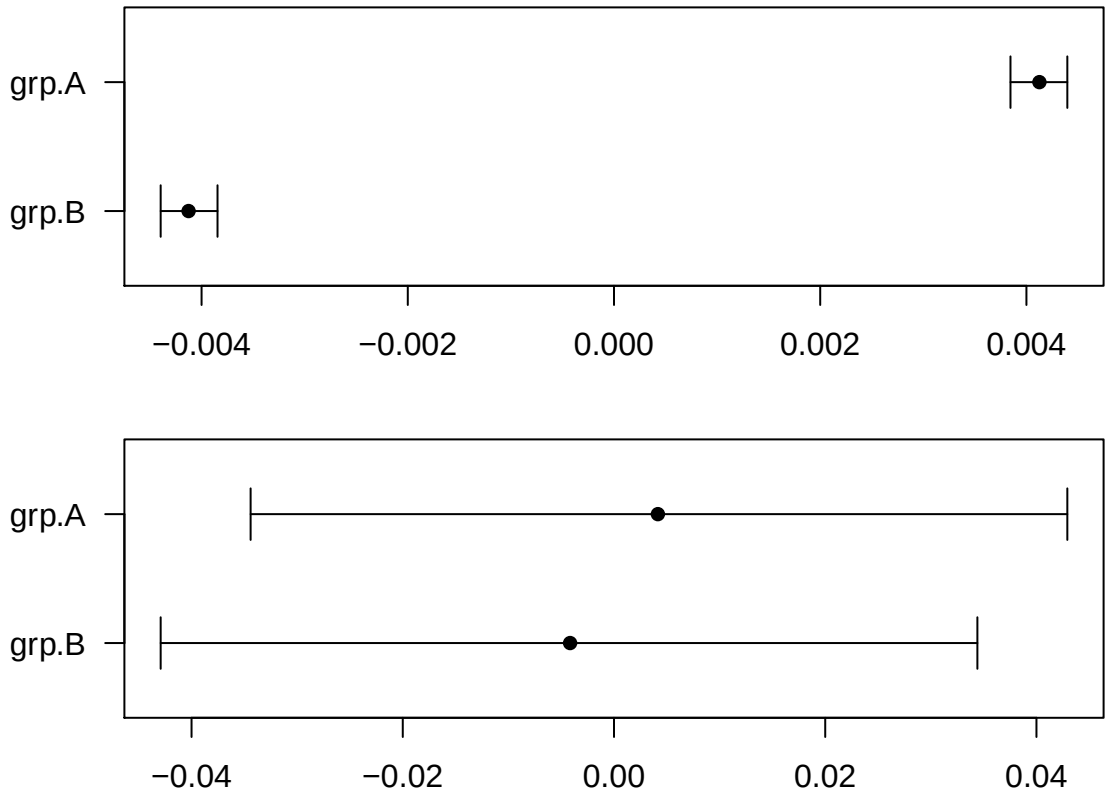
```
# mu_rho
ep_mu = getEffectParameters(bp_mu$post, bp_mu$factors, "grp")
eps_mu = summarizeEffectParameters(ep_mu)
eps_mu
```

```
##   effect        mean        2.5%         50%      97.5%
## 1  grp.A  0.00415496 -0.03440798  0.00406272 0.04292497
## 2  grp.B -0.00415496 -0.04292497 -0.00406272 0.03440798
```

Notice that the posterior mean of the effect parameters is very similar regardless of whether ρ_i or μ_ρ is used. This makes sense because μ_ρ estimates the mean of the ρ_i.

As shown by the following plots, however, the credible intervals for the effect parameters are strikingly different.

```
par(mfrow=c(2,1))
plotEffectParameterSummary(eps_rho)
plotEffectParameterSummary(eps_mu)
```



The top panel uses ρ_i and the bottom panel uses μ_ρ. When using μ_ρ, the credible intervals for the `grp` main effect parameters are much wider than when ρ_i is used. This difference in credible interval widths is directly comparable to the difference in the Bayes factors. Again, using μ_ρ produces plausible results while using ρ_i produces implausible results. In particular, using ρ_i produces results that seem to be an answer to the question of whether the sample means differ.

The logic of why using ρ_i tests whether the sample means differ but using μ_ρ tests whether the population means differ goes as follows.

1. By using a near infinite amount of data per participant, we are able to know each ρ_i with arbitrary precision. As such, the posterior distributions of each ρ_i will be arbitrarily narrow, approximately a point mass.
2. When calculating the condition effects for the `grp` main effect, the individual ρ_i are used. If, from point 1, all of the ρ_i are essentially point masses, the arithmetic mean of the ρ_i will also be essentially constant.
3. The `grp` effect parameters are calculated as a weighted sum of the ρ_i, which is essentially taking the arithmetic mean of the ρ_i. Because the ρ_i are essentially constant, the `grp` effect parameters will also be essentially constant.
4. If the `grp` effect parameters are essentially constant, then any difference in the effect parameters between the groups will be very large with respect to the variability of the effect parameters. As a result, there will be very strong evidence in favor of there being a main effect of `grp`.
5. Given the preceding logic, using the ρ_i when testing for a main effect of `grp` is a test of whether the sample means of ρ_i differ between the groups.

Now I turn to the argument that using μ_ρ allows for a test about differences in the population.

1. Regardless of how precisely each individual ρ_i is known, there is still some degree of uncertainty about the means of the populations of ρ.

2. Imagine for a moment that you have a single observation, y_i of a real-valued variable for each of some number of participants. The typical statistical approach is to treat y_i as known and fixed.
3. Even if all of the y_i are known and fixed, there is still uncertainty about the mean of the population that the y_i were sampled from.
4. Having arbitrarily precise posterior knowledge about ρ_i is analogous to having known, fixed y_i.
5. Thus, even if the ρ_i are all known perfectly, you still don't have complete certainty about the population mean of ρ_i.
6. Uncertainty about the population mean of ρ is captured by the posterior uncertainty of μ_ρ. This (loosely) follows from the fact that μ_ρ is not calculated from the ρ_i but instead sampled based on the values of the ρ_i.
7. Given the preceding logic, using μ_ρ allows inferences about differences between groups to be made while accounting for uncertainty about the population mean of ρ. In other words, using μ_ρ allows for inferences about the populations that participants were sampled from.

Problems with using μ_ρ

Although using the individual ρ_i is very problematic, that does not prove that using μ_ρ is correct. In fact, there is an issue with using μ_ρ that I would like to address. That issue is that the model nowhere states

$$\bar{P}_j = logit^{-1}(\mu_\rho + \delta_j)$$

where $\bar{P}_j$ is the mean P in a cell. The model states that

$$P_{ij} = logit^{-1}(\rho_i + \delta_j)$$

and given that

$$\bar{P}_j = \frac{1}{I}\sum_i P_{ij}$$

it follows, via substitution, that the model states that

$$\bar{P}_j = \frac{1}{I}\sum_i logit^{-1}(\rho_i + \delta_j)$$

It would be nice if the value of $\bar{P}_j$ would have the same value regardless of whether it is calculated in the exact way that the model states or in terms of μ_ρ.

The logit transformation is nonlinear, however, which means that there is no reason to believe that the following equality holds (even if μ_ρ is equal to the arithmetic mean of ρ_i, which it is not).

$$logit^{-1}(\mu_\rho + \delta_j) = \frac{1}{I}\sum_i logit^{-1}(\rho_i + \delta_j)$$

Thus, the value of $\bar{P}_j$ according to the model is not the same as the value of $\bar{P}_j$ that I used to test hypotheses about differences between groups.

There are at least three ways to try to address this problem.

1. Approximate Value: Show that $\bar{P}_j$ has approximately the same value, to sufficient tolerance, regardless of how it is calculated.
2. Dominance: Show that $\bar{P}_j$ is greater for one group than for another group regardless of how it is calculated.

3. Omitting Transformations: Do not use the logit transformation when performing tests, to work with latent parameters rather than manifest parameters.

These potention solutions are discussed in the following sections.

Approximate Value

The first option is to show that $\bar{P}_j$ has approximately the same value regardless of how it is calculated. The comparison is between two options: Taking the mean of the transformed values, as you do when the ρ_i are used, or transforming the mean, which is what you do when μ_ρ is used. For this approach, I will begin by taking many samples of latent parameter values (i.e. ρ_i) for a group.

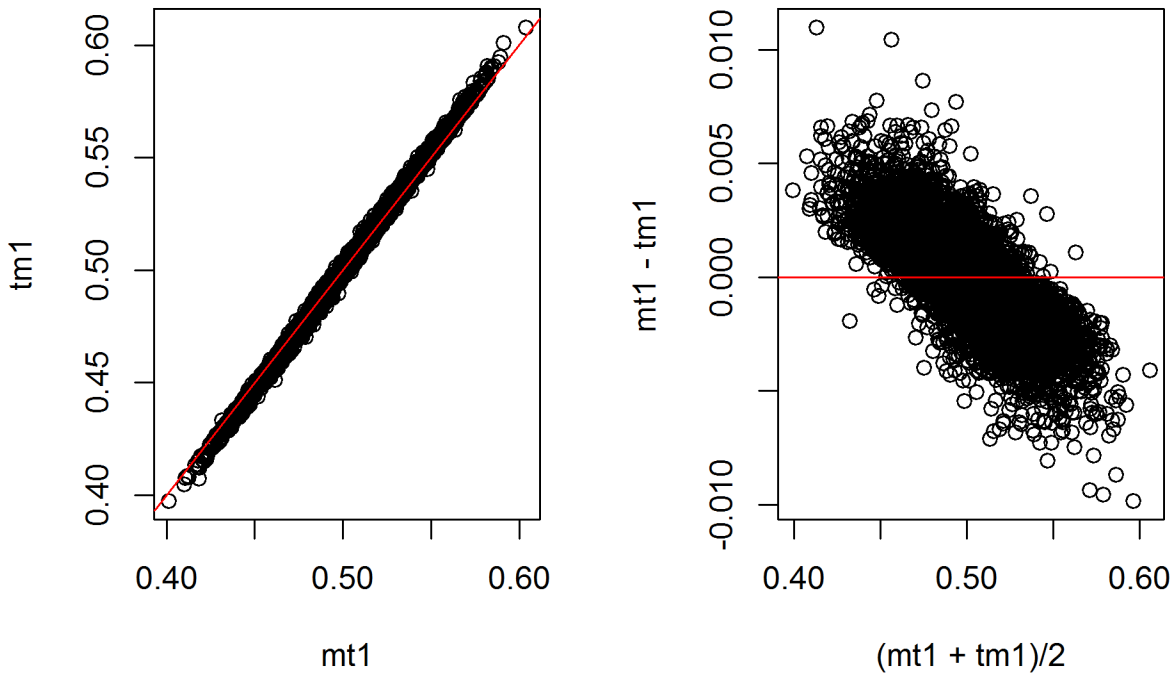
```
set.seed(123)

n = 1e4 # Number of samples
nPart = 20 # Participants per group

lat1 = matrix(0, nrow=n, ncol=nPart)
for (i in 1:nPart) {
    lat1[,i] = rnorm(n, 0, 0.5)
}
```

I will calculate the mean of the transformed parameters (`mt`; what we are ideally interested in) and the transformation of the mean value (`tm`; not actually stated by the model).

```
trans = plogis

# Like using rho_i
meanTrans = function(x) {
    mean(trans(x))
}

# Like using mu_rho
transMean = function(x) {
    trans(mean(x))
}

mt1 = apply(lat1, 1, meanTrans)
tm1 = apply(lat1, 1, transMean)
```

I will plot `tm` against `mt` (left panel) to show how similar the overall values are. In addition, I plot the difference between `tm` and `mt` against the mean of `tm` and `mt` (right panel) to clearly show how the difference clearly depends on the magnitude.

```
par(mfrow=c(1,2))

plot(mt1, tm1)
abline(0, 1, col="red")

plot((mt1 + tm1) / 2, mt1 - tm1)
abline(0, 0, col="red")
```



Whether or not you consider `tm` and `mt` to have approximately the same value is up to you and depends to some extent on the magnitude of differences between groups that you are expecting. In addition, you might want to try different means and standard deviations of the populations that are sampled from to better approximate what you expect your observed parameter values to look like.

Dominance

Moving on to dominance, we want to answer the following question: If `mt` for group 1 is greater than `mt` for group 2, is `tm` for group 1 greater than `tm` for group 2? If the answer to this question is "yes" nearly all of the time, then it may be the case that inferences about differences between group do not depend very much on whether `mt` or `tm` is used.

I will begin by sampling true latent parameter values (i.e. ρ_i) from another group. The first group had a population mean of 0, while this second group has a population mean of 0.3.

```
lat2 = matrix(0, nrow=n, ncol=nPart)
for (i in 1:nPart) {
    lat2[,i] = rnorm(n, 0.3, 0.5)
}

mt2 = apply(lat2, 1, meanTrans)
tm2 = apply(lat2, 1, transMean)
```

With samples from the second group, we can determine which cases in which `mt` is greater for group 1 than for group 2, and the same for `tm`.

```
mt_gt = mt1 > mt2
tm_gt = tm1 > tm2

mean(mt_gt == tm_gt)
```

```
## [1] 0.9976
```

At least for the settings used to sample the true parameter values that I used, it appears that both `mt` and `tm` go the same direction nearly all of the time.

When there is disagreement, the difference between the groups is very small compared to when there is agreement, as shown below.

```
whichDisagree = which(mt_gt != tm_gt)

sd((mt1 - mt2)[-whichDisagree])
```

```
## [1] 0.03725513
```

```
sd((mt1 - mt2)[whichDisagree])
```

```
## [1] 0.001217672
```

```
sd((tm1 - tm2)[-whichDisagree])
```

```
## [1] 0.03923604
```

```
sd((tm1 - tm2)[whichDisagree])
```

```
## [1] 0.002214189
```

This suggests that even when there is disagreement about which of `tm` or `mt` is greater for the two groups, the difference between the groups is small enough that a test of group differences is likely to support the hypothesis that the groups do not differ.

Omitting Transformations

Another way to be able to use μ_ρ is not using the logit transformation when performing tests. This approach works on the logic that we don't really care about whether P *per se* differs between groups, but just whether some quantity that combines information about ρ and δ differs between groups.

If we let

$$\eta_{ij} = \rho_i + \delta_j$$

then

$$P_{ij} = logit^{-1}(\eta_{ij})$$

without changing the model in any way. Rather than focus on P, the inferential focus can be on η. Focusing on η could be a loss in some cases, given that $\eta = 10$ and $\eta = 100$ both result in $P \approx 1$. On the other hand, the equality

$$\mu_\rho + \delta_j = \frac{1}{I}\sum_i (\rho_i + \delta_j)$$

is more likely to hold, or hold more accurately, than an equality in which the logit transformation is used (it would hold exactly if μ_ρ were equal to the arithmetic mean of the ρ_i, but that is typically not exactly the case). By working with η rather than P, it is possible for μ_ρ to fairly accurately substitute for the ρ_i.