Non-hierarhical vs hierarhical estimation of the two parameter mixture model: Parameter recovery simulations

Table of Contents

[Introduction 1](#_Toc114047768)

[Generate synthetic data 1](#_Toc114047769)

[Visualize synthetic data 3](#_Toc114047770)

[Fit non-hierarhical MLE version of the mixture model 6](#_Toc114047771)

[Fit hierarhical version with brms 8](#_Toc114047772)

### Introduction

In this simulation, we will demonstrate how with few observations per participant (i.e. N=50), the non-hierarchical maximum likelihood estimates (MLE) of the two parameter mixture model are not recovered well. At the same time, the hierarchical version of the model, implemented in the brms package does a much better job at recovering the parameters for each participant.

To start, let’s load the relevant packages, as well as the saved output of the brms model. This will save us time, as we don’t have to refit the model. If you want, you can replicate the analyses without preloading the model fit, by uncommenting all relevant lines and running the notebook. Note that the model fitting will take a few hours.

suppressPackageStartupMessages({  
 library(tidyverse)  
 library(here)  
 library(brms)  
 library(patchwork)  
})  
options(mc.cores = parallel::detectCores())  
load(here('output/par\_rec\_multi\_level\_simulation.RData'))

### Generate synthetic data

First, we will generate synthetic continuous reproduction data. Each of N=20 participants contributes Nobs=50 observations in two conditions A and B. We selected 50 observations, because from our experience, with so few observations the MLE non-hierarchical method often fails to recover the correct individual parameters, so it will serve as a good showcase. We assume that participants vary in their memory precision (i.e. the parameter of the von Mises distribution) and in the likelihood that their response comes from memory (; here we specify it as , which is the logit transformed , because that’s how brms estimates it). Therefore, the observations for each participant in conditions A and B are generated as follows, where j indexes participants, and are the differences in the parameters between conditions B and A, is the von Mises distribution and is the uniform distribution:

In simple terms, participants on average have a precision of in Condition A, but they vary around that estimate (SD=2.5). In condition B, participants’ precision is increased on average by relative to Condition A, but this also varies (SD=5). Similarly for the probability in memory () parameter. The code below accomplishes this (parts are commented out, because we load these values at the beginning of the script:

#### generate synthetic data  
  
#### first, set population parameters  
N = 20  
Nobs = 50  
kappa1a\_mu = 10  
kappa1a\_sd = 2.5  
kappa1delta\_mu = 20  
kappa1delta\_sd = 5  
theta1a\_mu = 1  
theta1a\_sd = 0.5  
theta1delta\_mu = -1  
theta1delta\_sd = 0.25  
  
#### generate participant parameters (for theta, logit units)  
# kappa1a\_i = rnorm(N, kappa1a\_mu, kappa1a\_sd)  
# kappa1delta\_i = rnorm(N, kappa1delta\_mu, kappa1delta\_sd)  
# theta1a\_i = rnorm(N, theta1a\_mu, theta1a\_sd)  
# theta1delta\_i = rnorm(N, theta1delta\_mu, theta1delta\_sd)  
  
#### put parameters together  
# true\_pars <- data.frame(id = rep(1:N, 2), condition = rep(c('A','B'), each=N),   
# kappa = c(kappa1a\_i, kappa1a\_i+kappa1delta\_i),  
# pmem = gtools::inv.logit(c(theta1a\_i, theta1a\_i+theta1delta\_i)))  
  
  
#### simulate data for each trial  
# dat <- data.frame()  
# for (i in 1:N) {  
# A\_n = floor(Nobs\*gtools::inv.logit(theta1a\_i[i]))  
# B\_n = floor(Nobs\*gtools::inv.logit(theta1a\_i[i]+theta1delta\_i[i]))  
# datA <- data.frame(y=c(rvon\_mises(A\_n, 0, kappa1a\_i[i]), runif(Nobs-A\_n,-pi,pi)), condition = "A", id = i)  
# datB <- data.frame(y=c(rvon\_mises(B\_n, 0, kappa1a\_i[i]+kappa1delta\_i[i]), runif(Nobs-B\_n,-pi,pi)), condition = "B", id = i)  
# DAT <- bind\_rows(datA,datB)  
# dat <- bind\_rows(dat,DAT)  
# }

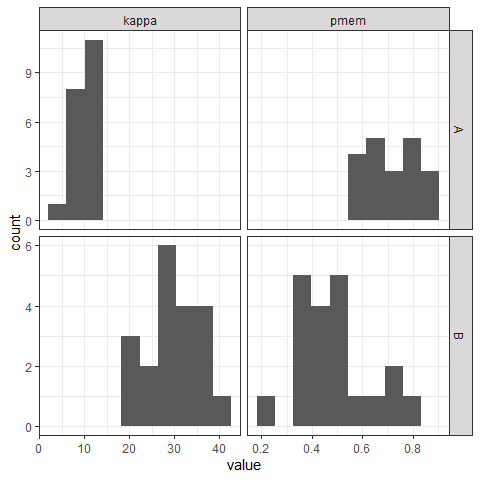
### Visualize synthetic data

Here are the parameters for each participant and condition:

true\_pars

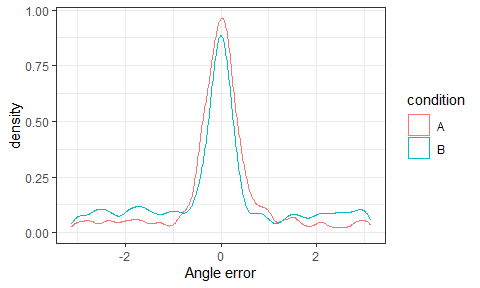
## id condition kappa pmem  
## 1 1 A 9.478360 0.8627272  
## 2 2 A 11.715649 0.6834658  
## 3 3 A 13.740791 0.7820907  
## 4 4 A 4.692635 0.7039524  
## 5 5 A 9.656958 0.7651808  
## 6 6 A 10.968378 0.7203603  
## 7 7 A 13.598416 0.6795157  
## 8 8 A 10.010820 0.5860563  
## 9 9 A 13.366425 0.5915341  
## 10 10 A 12.204754 0.6769404  
## 11 11 A 10.568784 0.6493821  
## 12 12 A 8.888721 0.6566303  
## 13 13 A 12.475556 0.6142899  
## 14 14 A 11.251142 0.7795837  
## 15 15 A 8.391865 0.7743183  
## 16 16 A 10.207182 0.7079415  
## 17 17 A 9.736036 0.5572051  
## 18 18 A 7.835275 0.8916764  
## 19 19 A 8.278295 0.8525931  
## 20 20 A 13.322670 0.7690013  
## 21 1 B 28.923805 0.7063868  
## 22 2 B 31.202578 0.3832080  
## 23 3 B 32.503713 0.5396529  
## 24 4 B 20.241197 0.4716653  
## 25 5 B 29.987560 0.4270660  
## 26 6 B 30.961803 0.5047793  
## 27 7 B 35.266937 0.4891554  
## 28 8 B 19.996771 0.3925293  
## 29 9 B 37.457545 0.2399274  
## 30 10 B 30.225545 0.4020229  
## 31 11 B 33.576452 0.3754082  
## 32 12 B 24.988404 0.4132181  
## 33 13 B 41.206685 0.4409797  
## 34 14 B 36.394617 0.5327623  
## 35 15 B 27.157000 0.6347218  
## 36 16 B 21.471546 0.3948013  
## 37 17 B 30.043178 0.3331486  
## 38 18 B 28.469415 0.8281906  
## 39 19 B 25.249893 0.6915995  
## 40 20 B 35.677790 0.5827495

true\_pars %>%   
 gather(par, value, kappa, pmem) %>%   
 ggplot(aes(value)) +  
 geom\_histogram(bins=10) +  
 facet\_grid(condition ~ par, scales="free") +  
 theme\_bw()



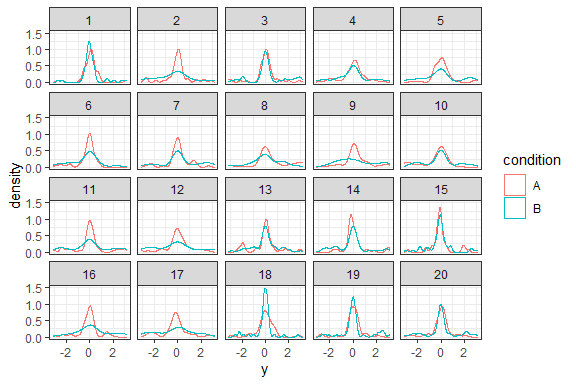
And here is the distribution of errors, characterized by higher precision in condition B, but with more guessing:

dat %>%   
 ggplot(aes(y)) +  
 geom\_density(aes(color=condition)) +  
 theme\_bw() +  
 xlab('Angle error')



And as you can see, there is quite some variability by participant:

ggplot(dat, aes(y, color=condition)) +  
 geom\_density() +  
 facet\_wrap(~id) +  
 theme\_bw()



### Fit non-hierarhical MLE version of the mixture model

As is standard in the literature, we will first fit the mixture model using MLE separately to each participant and condition. We accomplish this using a couple of custom functions, but the same can be achieved with some existing R packages.

# 2-p likelihood function  
LL <- function(dat) {  
 LL\_resp <- function(par) {  
 y = dat$y  
 kappa = exp(par[1])  
 pmem = gtools::inv.logit(par[2])  
 lik\_vm <- brms::dvon\_mises(y,mean(y),kappa)  
 lik\_un <- brms::dvon\_mises(y,mean(y),0)  
 lik <- pmem\*lik\_vm+(1-pmem)\*lik\_un  
 LL <- -sum(log(lik))  
 }  
}  
  
# function for fit and return parameter estimates  
fit\_mixture <- function(dat) {  
 require(stats4)  
 LL\_resp <- LL(dat)   
 fit <- optim(c(logkappa=2,theta=1), LL\_resp)  
 coef = as.data.frame(t(fit$par))[1,]  
 coef$convergence <- fit$convergence  
 coef$kappa = exp(coef$logkappa)  
 coef$pmem = gtools::inv.logit(coef$theta)  
 return(coef)  
}  
  
  
# estimate parameters separately for each participant and condition  
mle\_est <- dat %>%   
 group\_by(id, condition) %>%   
 do({fit\_mixture(.)}) %>%   
 arrange(condition, id)

## Loading required package: stats4

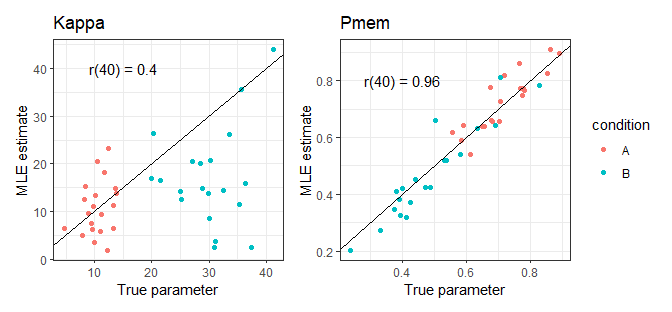
First, we check that all fits have converged:

mean(mle\_est$convergence == 0)

## [1] 1

And now, we can check how well the individual participant paramters have been recovered. We plot the estimated parameters (y-axis) vs the true generating parameters (x-axis):

r\_kappa <- round(cor.test(true\_pars$kappa, mle\_est$kappa)$est,2)  
r\_pmem <- round(cor.test(true\_pars$pmem, mle\_est$pmem)$est,2)  
  
p1 <- left\_join(true\_pars, mle\_est, by=c('id','condition')) %>%   
 ggplot(aes(kappa.x, kappa.y, color=condition)) +  
 geom\_point() +  
 ggtitle('Kappa') +  
 xlab('True parameter') +  
 ylab('MLE estimate') +  
 theme\_bw() +  
 annotate('text', x=15, y=40, label=paste0('r(40) = ', r\_kappa)) +  
 geom\_abline(intercept=0, slope=1) +  
 theme(legend.position="")  
  
p2 <- left\_join(true\_pars, mle\_est, by=c('id','condition')) %>%   
 ggplot(aes(pmem.x, pmem.y, color=condition)) +  
 geom\_point() +  
 ggtitle('Pmem') +  
 xlab('True parameter') +  
 ylab('MLE estimate') +  
 theme\_bw() +  
 annotate('text', x=0.4, y=0.8, label=paste0('r(40) = ', r\_pmem)) +  
 geom\_abline(intercept=0, slope=1)  
  
p1+p2



ggsave(here('figures/par\_rec\_sim\_mle.png'), width=7, height=3.25, units='in')

As we can, the non-hierarhical MLE estimates of Pmem are pretty good; however, the model fails to accurately estimate the individual kappa parameters. With only 50 observations per participant and condition, that’s normal in our experience. Notably, from these estimates, a researcher might erroneously conclude that our manipulation affects only pmem, but not kappa (which, interestingly, happens often in published research on VWM). Let’s see if the hierarhical model can do better.

### Fit hierarhical version with brms

The code is commented out, because it takes several hours to fit. The results are preloaded at the begining of the script, so they can be accessed by the relevant object names.

# # create mixture of von Mises distributions  
# mix\_vonMises <- mixture(von\_mises,von\_mises,order = "none")  
#   
# # set up mixture model. allow kappa and theta to vary by condition  
# bf\_mixture <- bf(y ~ 1,  
# kappa1 ~ condition + (condition||id),  
# kappa2 ~ 1,  
# theta1 ~ condition + (condition||id))  
#   
#   
# # check default priors  
# get\_prior(bf\_mixture, dat, mix\_vonMises)  
#   
# # contrain priors. Set mean of von mises distributions to be 0; set kappa of the  
# # second von mises distribution to be very low, approximating a uniform distribution  
# mix\_priors <- prior(constant(0), class = Intercept, dpar = "mu1") +  
# prior(constant(0), class = Intercept, dpar = "mu2") +  
# prior(constant(-100), class = Intercept, dpar = "kappa2")  
#   
# brms\_fit <- brm(bf\_mixture, dat, mix\_vonMises, mix\_priors)

Let’s examine the model. Convergence of all parameters looks good (Rhat < 1.01, and hight Bulk\_ESS; no warnings).

brms\_fit

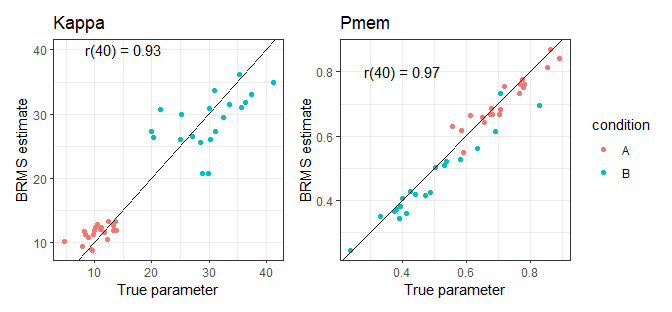
## Family: mixture(von\_mises, von\_mises)   
## Links: mu1 = tan\_half; kappa1 = log; mu2 = tan\_half; kappa2 = log; theta1 = identity; theta2 = identity   
## Formula: y ~ 1   
## kappa1 ~ condition + (condition || id)  
## kappa2 ~ 1  
## theta1 ~ condition + (condition || id)  
## Data: dat4 (Number of observations: 2000)   
## Samples: 4 chains, each with iter = 2000; warmup = 1000; thin = 1;  
## total post-warmup samples = 4000  
##   
## Group-Level Effects:   
## ~id (Number of levels: 20)   
## Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS  
## sd(kappa1\_Intercept) 0.22 0.10 0.03 0.42 1.00 975  
## sd(kappa1\_conditionB) 0.17 0.12 0.01 0.46 1.00 1590  
## sd(theta1\_Intercept) 0.53 0.14 0.28 0.83 1.00 956  
## sd(theta1\_conditionB) 0.26 0.17 0.01 0.64 1.00 722  
## Tail\_ESS  
## sd(kappa1\_Intercept) 728  
## sd(kappa1\_conditionB) 2029  
## sd(theta1\_Intercept) 1853  
## sd(theta1\_conditionB) 1087  
##   
## Population-Level Effects:   
## Estimate Est.Error l-95% CI u-95% CI Rhat Bulk\_ESS Tail\_ESS  
## mu1\_Intercept 0.00 0.00 0.00 0.00 1.00 4000 4000  
## kappa1\_Intercept 2.44 0.09 2.27 2.63 1.00 3279 2446  
## mu2\_Intercept 0.00 0.00 0.00 0.00 1.00 4000 4000  
## kappa2\_Intercept -100.00 0.00 -100.00 -100.00 1.00 4000 4000  
## theta1\_Intercept 0.92 0.15 0.63 1.22 1.00 1882 2779  
## kappa1\_conditionB 0.93 0.13 0.68 1.18 1.00 4278 3091  
## theta1\_conditionB -1.10 0.14 -1.38 -0.82 1.00 4474 3181  
##   
## Samples were drawn using sampling(NUTS). For each parameter, Bulk\_ESS  
## and Tail\_ESS are effective sample size measures, and Rhat is the potential  
## scale reduction factor on split chains (at convergence, Rhat = 1).

Now, the values above don’t tell us much, because in brms, kappa is log transformed and theta is the logit transformation of pmem. Therefore, we first need to extract the random effects, which are the parameter estimates for each participant, and then transform the parameters into the relevant scale:

ranefs <- ranef(brms\_fit)$id  
logkappa <- c(fixef(brms\_fit)['kappa1\_Intercept','Estimate']+ranefs[,'Estimate','kappa1\_Intercept'],   
 fixef(brms\_fit)['kappa1\_Intercept','Estimate']+ranefs[,'Estimate','kappa1\_Intercept']+  
 fixef(brms\_fit)['kappa1\_conditionB','Estimate']+ranefs[,'Estimate','kappa1\_conditionB'])  
  
theta = c(fixef(brms\_fit)['theta1\_Intercept','Estimate']+ranefs[,'Estimate','theta1\_Intercept'],  
 fixef(brms\_fit)['theta1\_Intercept','Estimate']+ranefs[,'Estimate','theta1\_Intercept']+  
 fixef(brms\_fit)['theta1\_conditionB','Estimate']+ranefs[,'Estimate','theta1\_conditionB'])  
  
brms\_est <- data.frame(id = rep(1:N, 2),   
 condition = rep(c('A','B'), each=N),   
 kappa = exp(logkappa),  
 pmem = c(gtools::inv.logit(theta)))

As with the MLE model before, we can now check how well the individual participant parameters have been recovered. We plot the estimated parameters (y-axis) vs the true generating parameters (x-axis):

r\_kappa <- round(cor.test(true\_pars$kappa, brms\_est$kappa)$est,2)  
r\_pmem <- round(cor.test(true\_pars$pmem, brms\_est$pmem)$est,2)  
  
p3 <- left\_join(true\_pars, brms\_est, by=c('id','condition')) %>%   
 ggplot(aes(kappa.x, kappa.y, color=condition)) +  
 geom\_point() +  
 ggtitle('Kappa') +  
 xlab('True parameter') +  
 ylab('BRMS estimate') +  
 theme\_bw() +  
 annotate('text', x=15, y=40, label=paste0('r(40) = ', r\_kappa)) +  
 geom\_abline(intercept=0, slope=1) +  
 theme(legend.position="")  
  
p4 <- left\_join(true\_pars, brms\_est, by=c('id','condition')) %>%   
 ggplot(aes(pmem.x, pmem.y, color=condition)) +  
 geom\_point() +  
 ggtitle('Pmem') +  
 xlab('True parameter') +  
 ylab('BRMS estimate') +  
 theme\_bw() +  
 annotate('text', x=0.4, y=0.8, label=paste0('r(40) = ', r\_pmem)) +  
 geom\_abline(intercept=0, slope=1)  
  
p3+p4



ggsave(here('figures/par\_rec\_sim\_brms.png'), width=7, height=3.25, units='in')

And voila - we can see that the BRMS model does a drastically better job at recovering the kappa parameter, relative to the MLE non-hierarhical version. This is because in the hierarhical model, the data from all participants partially informs parameter estimates for each individual participant, which is particularly helpful when we only have a few observations per participant.