Hierarchical Bayesian Estimation of metacognitive efficiency with hBayesDM

Heesun Park

data link:
https://drive.google.com/drive/folders/1-pREh7Nv_k1mXIbE79J3DAAJm_To5O0Z?usp=sharing

Goal of the project

- ***Write <u>Stan</u> code for estimating <u>metacognitive efficiency</u>***
  - ***meta-d'***
  - ***hierarchical meta-d'***
- ***Check the results and compare them with MLE results***
- ***Put into and Run with hBayesDM***

Not completed...

Content

1. Introduction
2. Data
3. Model: Type 2 Signal Detection Model
4. Codes
5. Results
6. Conclusion

Introduction

- ## Metacognition
  - Metacognition is defined as ***knowledge of one's cognitive process***. We can reflect on whether I know the answer or not and whether I was correct or not. **Metacognitive sensitivity** can be operationally assessed by ***the extent to which a subject's confidence ratings are predictive of their actual task performance***. If a subject has higher confidence ratings after correct responses and lower confidence ratings after incorrect responses, we can say that the subject has high metacognitive sensitivity. So in metacognition research, it is common to use a simple decision task followed by a confidence rating in the response, and to analyze them. In this manner, we can quantify metacognition for the research.

- ## Measure metacognitive sensitivity (and efficiency)
  - **Correlation**
    - To measure metacognitive sensitivity, simple correlation analysis could be used but the correlations are affected by ***decision task*** and ***response bias*** of confidence in a subject.
  - **Model-based approach (within the framework of Signal Detection Theory)**
    - Unlike correlation analysis, model-based approach is free of these confoundings within the framework of signal detection theory.

Data

Data

Paper

PROCEEDINGS B

rspb.royalsocietypublishing.org

Research

Check for updates

Correlated individual differences suggest a common mechanism underlying metacognition in visual perception and visual short-term memory

Jason Samaha[1] and Bradley R. Postle[1,2]

Open Science Framework

OSF

Data

Experiment (exp 3)

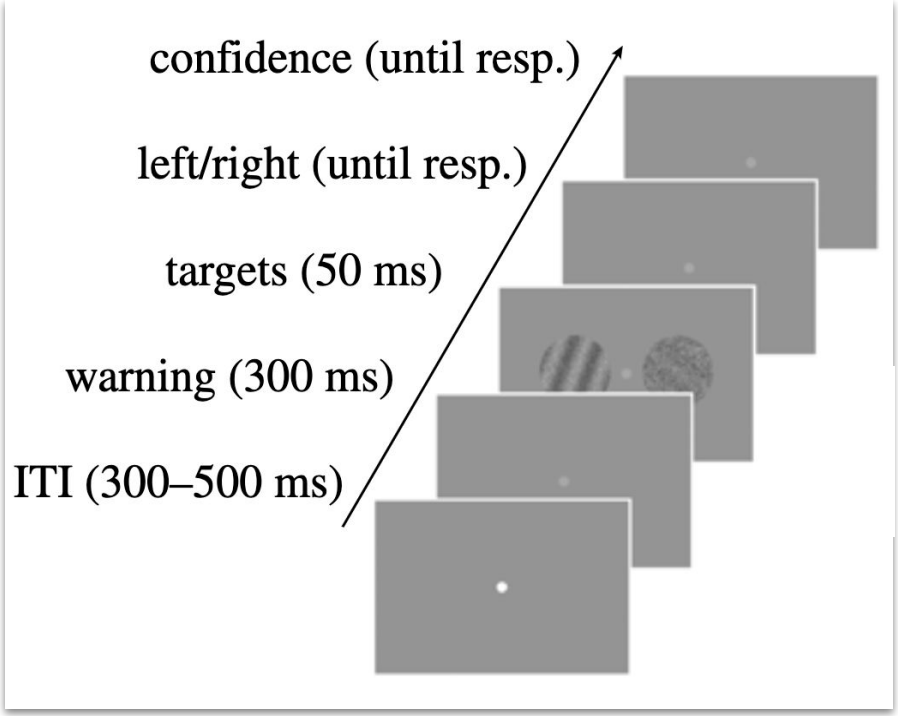
Among three experiments in the paper, exp 3 is designed and suitable for applying the type 2 signal detection model.

- **Subject = 20**
- **Type 1 task**
  - Contrast Perception Task (300 trials)
  - Orientation Perception Task (300 trials)
  - Orientation Visual Short-Term Memory Task (300 trials)
- **Type 2 task**: Confidence rating
- **With these 3 tasks,**
  - The paper showed a strong positive correlation between orientation perception and orientation visual short term memory metacognition and no correlation between other tasks.

Experiment

Contrast Perception Task

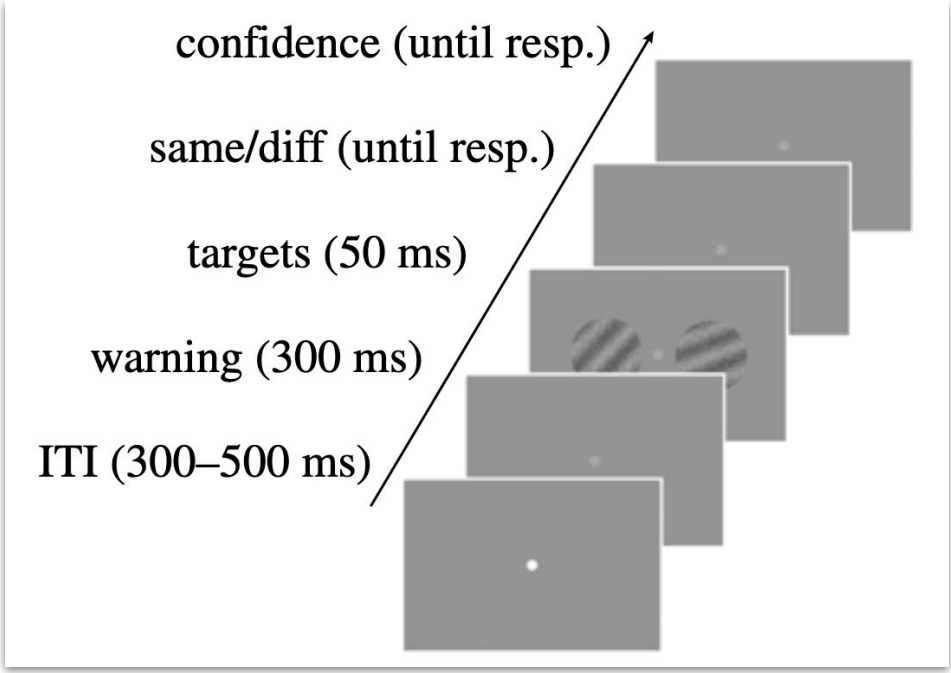
- Type 1: Choose a stimulus with a higher contrast grating
- Type 2: Confidence rating (1-4)
  - 1: Complete guess
  - 4: Very confident



Experiment

Orientation Perception Task

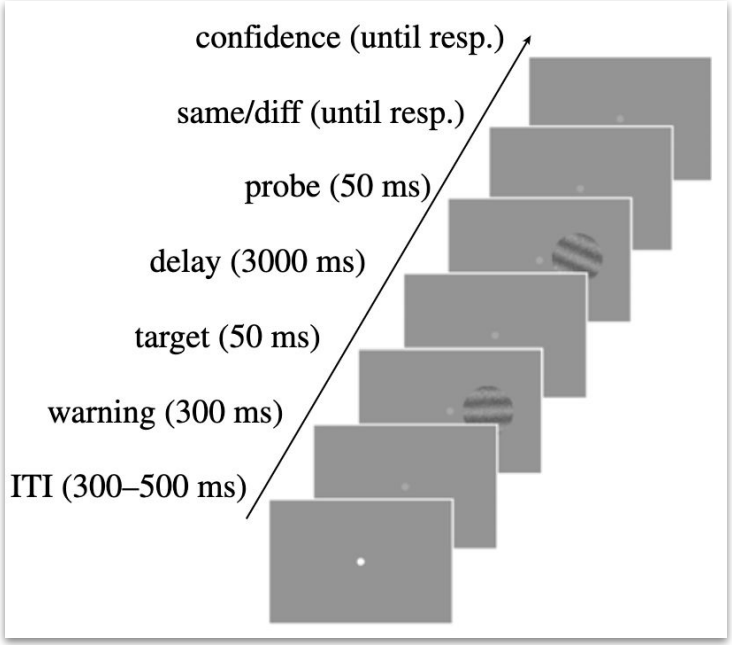
- Type 1: Decide whether two stimuli have the same or different orientation
- Type 2: Confidence rating (1-4)
  - 1: Complete guess
  - 4: Very confident



Experiment

Orientation Visual Short-Term Memory Task

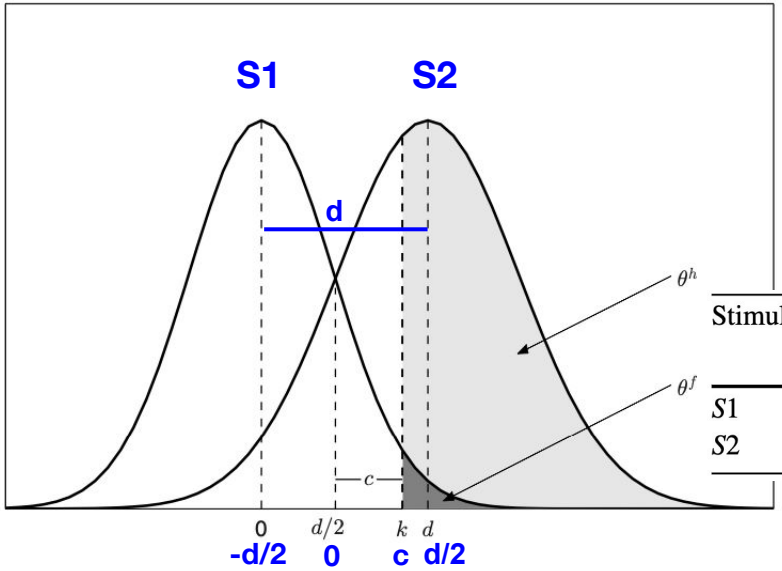
- Type 1: Decide whether two stimuli have the same or different orientation + temporal delay of 3 seconds between two stimuli → the orientation of the first stimulus should be remembered.
- Type 2: Confidence rating (1-4)
  - 1: Complete guess
  - 4: Very confident



Model

Model

Signal Detection Theory



S1: signal absent
S2: signal present
d: discriminability
c: decision criterion

Type 1

Two kinds of stimulus & two kinds of response
→ 4 possible combinations

Stimulus	Response	
	"$S1$"	"$S2$"
$S1$	Correct rejection (CR)	False alarm (FA)
$S2$	Miss	Hit

Assumption: the strengths of two stimuli are Gaussian distributions with equal variance.

Lee, M. D. & Wagenmakers, E.-J. (2014)
Maniscalco, B., & Lau, H. (2014)

Model

Meta-SDT

In addition to type 1 results, there could be response-specific type 2 hit, false alarm, and so on, depending on response and confidence rating.

Type 1

Stimulus	Response	
	"*S*1"	"*S*2"
*S*1	Correct rejection (CR)	False alarm (FA)
*S*2	Miss	Hit

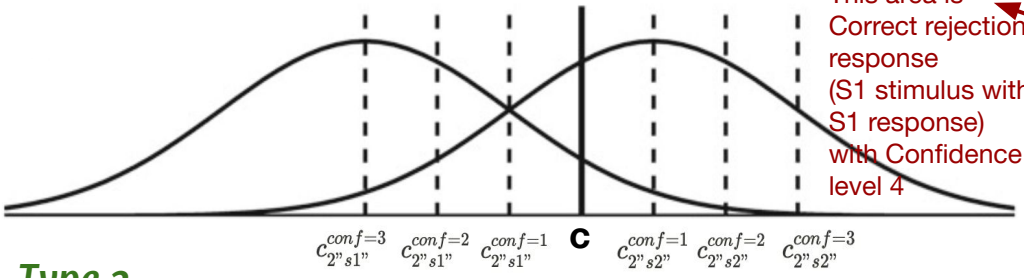
Type 2

Response			Confidence	
			Low	High
"*S*1"	Accuracy	Incorrect (Type 1 miss)	$CR_{2,\text{"S1"}}$	$FA_{2,\text{"S1"}}$
		Correct (Type 1 correct rejection)	$Miss_{2,\text{"S1"}}$	$Hit_{2,\text{"S1"}}$
"*S*2"	Accuracy	Incorrect (Type 1 false alarm)	$CR_{2,\text{"S2"}}$	$FA_{2,\text{"S2"}}$
		Correct (Type 1 hit)	$Miss_{2,\text{"S2"}}$	$Hit_{2,\text{"S2"}}$

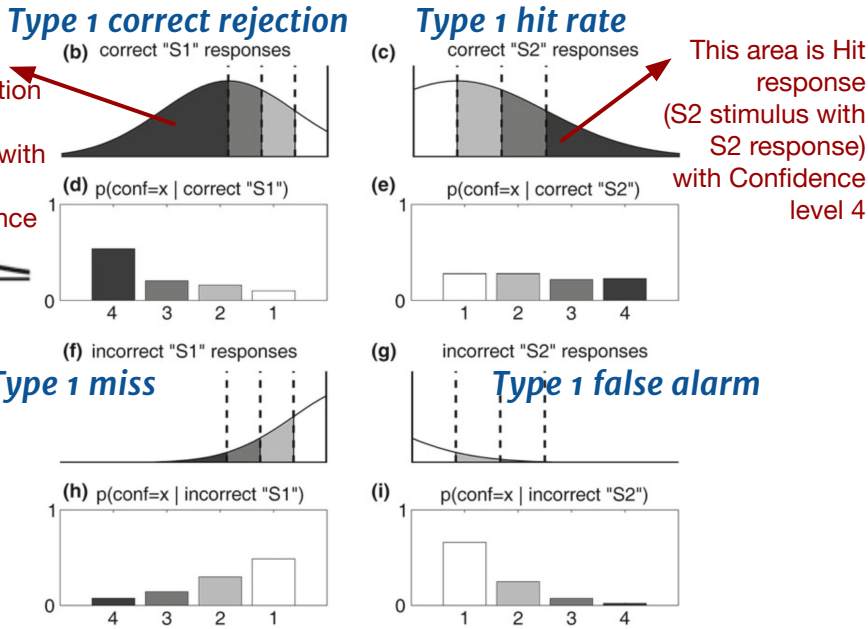
Maniscalco, B., & Lau, H. (2014)

Model
Meta-SDT

With type 2 data, an equal-variance Gaussian SDT model can be estimated by adding mechanism to the usual SDT model, by which confidence ratings can be generated. The model suppose that the observer simply uses additional decision criteria, analogous to the type 1 criteria c to generate confidence ratings.



This area is Correct rejection response (S1 stimulus with S1 response) with Confidence level 4



This area is Hit response (S2 stimulus with S2 response) with Confidence level 4

Type 2

Response			Confidence	
			Low	High
"*S1*"	Accuracy	Incorrect (Type 1 miss)	$CR_{2,"S1"}$	$FA_{2,"S1"}$
		Correct (Type 1 correct rejection)	$Miss_{2,"S1"}$	$Hit_{2,"S1"}$
"*S2*"	Accuracy	Incorrect (Type 1 false alarm)	$CR_{2,"S2"}$	$FA_{2,"S2"}$
		Correct (Type 1 hit)	$Miss_{2,"S2"}$	$Hit_{2,"S2"}$

Like (d), (e), (h), (i), each probability can be calculated. Detailed calculation is on the next slide.

Maniscalco, B., & Lau, H. (2014)

Model

Meta-SDT

The likelihood of the type 2 data can be characterized using the multinomial model.
Probability of confidence ***c*** given stimulus ***s*** and response ***r*** could be calculated like below.

The likelihood of the type 2 data as **the multinomial model**

$$L(\theta|data) \propto \prod_{c,s,r} Prob_\theta(Conf_c \mid Stim_s,\ Resp_r)^{n_{data}(Conf_c \mid Stim_s,\ Resp_r)}$$

Parameter set is like this. Meta-d' and type 2 decision criteria. As shown here, the number of parameters depends on the level of confidence ratings

$$\theta = (meta - d', c_{2S1}^{conf=1}, c_{2S1}^{conf=2}, \ldots, c_{2S1}^{conf=nratings}, c_{2S2}^{conf=1}, c_{2S2}^{conf=2}, \ldots, c_{2S2}^{conf=nratings})$$

For example,

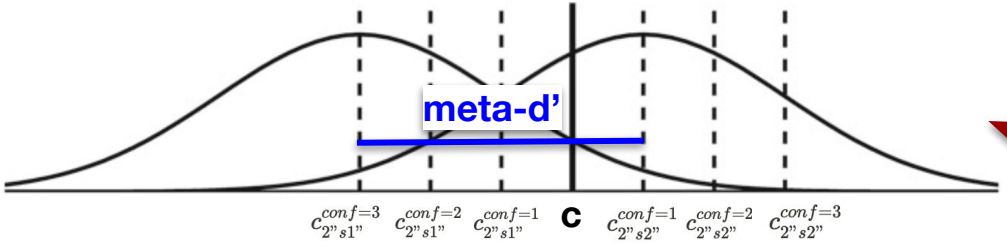
The probability of each confidence rating conditional on a given stimulus and response in a calculation of the area below a specific stimulus distribution and between two specific criteria, just as shown in the previous slide.

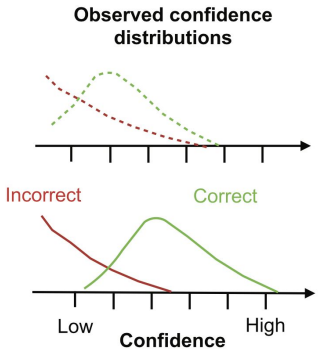
$$Prob_\theta(Conf = 2 \mid Stim = S1,\ Resp = S1) = \frac{\Phi(c_{2"S1"}^{conf=2}, -\frac{d'}{2}) - \Phi(c_{2"S1"}^{conf=3}, -\frac{d'}{2})}{\Phi(c, -\frac{d'}{2})}$$

Model

Meta-SDT

In summary, with confidence rating data, we can determine d' that best fits the data. And this d' is referred to as meta-d here





Maniscalco, B., & Lau, H. (2014)
Fleming, S. M. (2017)

Model *One more step left!*

Meta-SDT

Metacognitive efficiency

Metacognitive sensitivity

$$\log\left(\frac{\text{meta-}d'}{d'}\right)$$

Type 1 sensitivity

The ability to discriminate their own correct and incorrect responses, i.e. how well the confidence ratings track the task accuracy.
It is affected by type 1 task performance

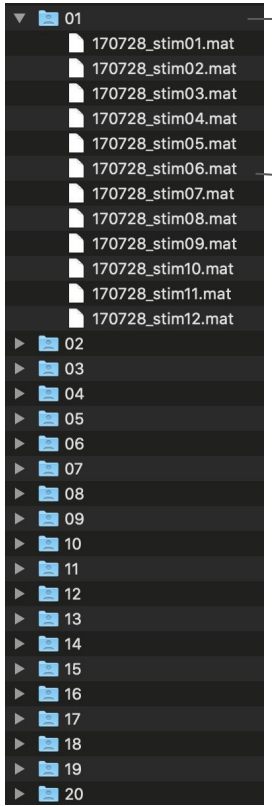
By being divided by Type 1 sensitivity, the relative measure of metacognitive sensitivity can be calculated. This is called metacognitive efficiency which can measure how efficient the individual's metacognition given a particular level of task performance.

meta-*d* ' = *d* ' : ideal observers
meta-*d* ' < *d* ' : hypo-metacognitive sensitivity
meta-*d* ' > *d* ' : hyper-metacognitive sensitivity

Maniscalco, B., & Lau, H. (2014)

Codes

Data processing



Subject (Each folder for the data of each subject, total 20 subjects)

numTrials	trialnum	taskID	taskFilename
60	61	1	ori_percept_stair.m
100	101	1	ori_percept.m
100	101	1	ori_percept.m
100	101	1	ori_percept.m
70	71	1	contrast_percept_stair.m
100	101	1	contrast_percept.m
100	101	1	contrast_percept.m
100	101	1	contrast_percept.m
60	61	1	ori_wm_stair.m
100	101	1	ori_wm.m
100	101	1	ori_wm.m
100	101	1	ori_wm.m

3 files for each task
1 file contains 100 trials, total 300 trials for each task

The structure of the raw data

Codes

Data processing

taskFilename
ori_percept_stair.m
100 trials ori_percept.m
100 trials ori_percept.m
100 trials ori_percept.m
contrast_percept_stair.m
contrast_percept.m
contrast_percept.m
contrast_percept.m
ori_wm_stair.m
ori_wm.m
ori_wm.m
ori_wm.m

Stimulus X Response X Confidence matrix can be represented in one vector!

COUNT																
The number of trials	nR_S1								nR_S1							
Data	6	22	80	10	7	21	4	1	0	3	15	9	9	76	34	3
Stim	S1	S1	S1	S1	S1	S1	S1	S1	S2	S2	S2	S2	S2	S2	S2	S2
Resp	S1	S1	S1	S1	S2	S2	S2	S2	S1	S1	S1	S1	S2	S2	S2	S2
Conf	4	3	2	1	1	2	3	4	4	3	2	1	1	2	3	4

final form! (whole information of one subject for a task), sum(count) = 300 trial

count = [6, 22, 80, 10, 7, 21, 4, 1, 0, 3, 15, 9, 9, 76, 34, 3]

for each task type

for each subject

Each number in a vector represent the number of trials for specific response data.
Half of the vector includes response when stimulus is S1.
In turn, half of them is response S1 given stimulus S1.
In order, they represent specific confidence rating.
This number **34** the number of trials when the stimulus was S2 and the subject responded with S2 & confidence 3

Data processing

Processed data

data from contrast perception task

→

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13	V14	V15	V16
1	6	22	80	10	7	21	4	1	0	3	15	9	9	76	34	3
2	4	17	73	27	17	15	2	0	0	2	10	17	34	62	15	5
3	13	66	35	5	8	11	22	2	0	11	2	6	6	12	59	42
4	19	15	4	3	0	0	5	4	3	1	0	0	1	1	22	22
5	2	54	75	5	3	1	3	0	0	23	32	5	2	28	66	1
6	87	39	15	0	1	5	15	6	14	12	4	0	1	15	38	48
7	31	45	32	8	6	8	16	6	2	7	11	4	5	15	46	58
8	29	87	36	2	4	8	2	0	3	23	21	2	2	61	19	1
9	13	11	20	57	11	5	15	9	2	2	4	13	30	22	42	44
10	63	47	10	0	0	10	7	3	18	14	7	1	1	32	49	35
11	0	52	57	3	2	20	5	0	0	9	19	4	4	65	56	4
12	16	45	20	6	5	11	30	7	0	3	1	1	7	16	89	42
13	1	46	46	17	4	22	14	1	0	1	10	5	12	58	58	5
14	5	16	31	7	3	22	22	2	0	8	21	7	3	15	30	8
15	28	70	16	0	0	2	14	10	16	22	6	0	0	10	39	67
16	7	23	40	11	2	28	19	11	1	3	5	6	1	12	34	97
17	0	13	65	36	8	16	6	0	0	2	18	8	40	61	27	0
18	0	39	67	4	0	7	20	11	1	7	14	1	0	14	77	38
19	7	70	31	1	1	4	28	9	2	10	6	0	0	7	72	52
20	31	39	26	18	16	8	7	5	3	4	4	15	30	23	47	24

subject Num

Count vector

Stan codes

Individual Bayesian Estimation (Non-hierarchical)

- **Version 1:** set boundary of meta-d
- **Version 2:** Additional Normalization after calculating all probabilities

why 2 versions?:

A previous stan code a problem that sampling from multinomial distribution cannot be done, because some of the probabilities in multinomial function were NaN. In turn, NaN was because of some denominators being zero after being sampled and transformed. In the model, p(Conf = c | Stimulus = s, Response = r) should be calculated and this is

p(Conf = c & Response = r | Stimulus = s) / p(Response = r | Stimulus = s)

And the denominator is calculated by cumulative normal density function so it occasionally became zero depending on the value of **the mean of distribution**.
To solve this problem, I made several attempts and two of them survived.
In Version 1, I set plausible boundary of meta-d, which is the mean of distribution.
In Version 2, all of the probabilities are just normalized after calculated and before put into multinomial function. Manipulating the denominators and other tricks didn't work but making a function of dividing each probability by sum of the probabilities worked and multiplying it by 0.999 and adding 0.001.

Stan codes

Hierarchical Bayesian Estimation

- Used the way of version 2 of Individual estimation code when facing the same problem with individual Bayesian estimation.

  Because it was difficult to set boundaries of meta-d in hierarchical structure, specifically in parameter expansion version
- **Version 1:** with Parameter expansion (similar with Fleming 2017)
- **Version 2:** without parameter expansion

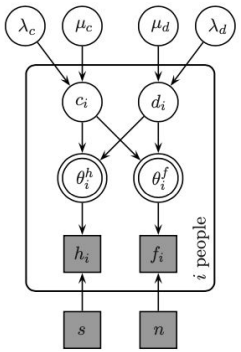
why 2 versions?:
Fleming (2017), which provides JAGS codes for hierarchical Bayesian estimation of meta-d, used parameter expansion to solve a problem that the hierarchical variance parameter (sigma of log M ratio) sometimes are trapped near zero during sampling. So I tried Fleming's version first, and a normal version without parameter expansion next.

Ironically, the same problem was detected in the model *with parameter expansion* when I ran my Stan codes.

Stan codes

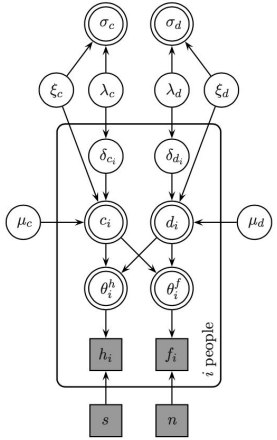
Hierarchical Bayesian Estimation

- What is parameter expansion?
- To augment the original hierarchical model with redundant multiplicative parameters. (Lee and Wagenmakers 2014)
- An example of parameter expansion of SDT model below.



$$
\begin{aligned}
\mu_d, \mu_c &\sim \text{Gaussian}(0, .001) \\
\lambda_d, \lambda_c &\sim \text{Gamma}(.001, .001) \\
d_i &\sim \text{Gaussian}(\mu_d, \lambda_d) \\
c_i &\sim \text{Gaussian}(\mu_c, \lambda_c) \\
\theta_i^h &\leftarrow \Phi(\tfrac{1}{2}d_i - c_i) \\
\theta_i^f &\leftarrow \Phi(-\tfrac{1}{2}d_i - c_i) \\
h_i &\sim \text{Binomial}(\theta_i^h, s) \\
f_i &\sim \text{Binomial}(\theta_i^f, n)
\end{aligned}
$$

Fig. 11.4 Graphical model for hierarchical signal detection theory.



$$
\begin{aligned}
\mu_d, \mu_c &\sim \text{Gaussian}(0, .001) \\
\lambda_d, \lambda_c &\sim \text{Gamma}(.1, .1) \\
\xi_d, \xi_c &\sim \text{Beta}(1, 1) \\
\delta_{d_i} &\sim \text{Gaussian}(0, \lambda_d) \\
\delta_{c_i} &\sim \text{Gaussian}(0, \lambda_c) \\
\sigma_d &\leftarrow |\xi_d| / \sqrt{\lambda_d} \\
\sigma_c &\leftarrow |\xi_c| / \sqrt{\lambda_c} \\
d_i &\leftarrow \mu_d + \xi_d \delta_{d_i} \\
c_i &\leftarrow \mu_c + \xi_c \delta_{c_i} \\
\theta_i^h &\leftarrow \Phi(\tfrac{1}{2}d_i - c_i) \\
\theta_i^f &\leftarrow \Phi(-\tfrac{1}{2}d_i - c_i) \\
h_i &\sim \text{Binomial}(\theta_i^h, s) \\
f_i &\sim \text{Binomial}(\theta_i^f, n)
\end{aligned}
$$

Lee & Wagenmakers (2014)

Fig. 11.8 Graphical model for the parameter-expanded hierarchical signal detection theory.

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Data Block

```
data {
  int<lower=1> K;              //Number of Subjects

  // Type 1 counts
  int<lower=0> N[K];           // Noise (S1 stimulus trials)
  int<lower=0> S[K];           // Signal (S2 stimulus trials)
  int<lower=0> H[K];           // Hit trials (resp=S2, stim=S2)
  int<lower=0> FA[K];          // False Alarm (resp=S2, stim=S1)

  //Type 2 data
  int<lower=1> nratings;       //the number of ratings
  int counts[4*nratings, K];   //count vectors for all subjects, counts = [nR_S1, nR_S2]

  // Type 1 SDT parameters
  vector[K] d1;                // Type 1 discriminability
  vector[K] c1;                // Type 1 bias
}
```

Type 1 d and c are not estimated but calculated here (Fleming 2017)

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Parameter Block

Ver 1

```
parameters {
  // Type 2 SDT parameters
  vector<lower=-10, upper = 10>[K] meta_d;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS1_p;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS2_p;
}
```

In version 1, I set boundary of meta-d like this.

Parameters are more than 3 because there are multiple type 2 c depending on the level of confidence rating

Ver 2

```
parameters {
  // Type 2 SDT parameters
  vector[K] meta_d;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS1_p;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS2_p;
}
```

In version 2, NO boundary

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Transformed Parameter Block

```
transformed parameters{
  vector[K] S2mu;                      //mean of S2 (Signal Distribution)
  vector[K] S1mu;                      //mean of S1 (Noise Distribution)
  matrix[K, nratings-1] cS1_raw;       //c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2_raw;       //c2 vectors for S2 (cS2_p -> cS2_raw -> cS2)
  matrix[K, nratings-1] cS1;           //c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2;           //c2 vectors for S1 (cS2_p -> cS2_raw -> cS2)
  matrix[K, 4] log_lik;

  for (i in 1:K){
    for (j in 1:(nratings-1)) {
      cS1_raw[i, j] = sample_from_tnormal_ub(c1[i], 0., 2., cS1_p[i, j]); //sampling from truncated normal distribution (upper bound)
      cS2_raw[i, j] = sample_from_tnormal_lb(c1[i], 0., 2., cS2_p[i, j]); //sampling from truncated normal distribution (lower bound)
      } //end of j loop

    // specify ordered prior on criteria
    cS1[i, 1:nratings-1] = to_row_vector(sort_asc(cS1_raw[i,]));
    cS2[i, 1:nratings-1] = to_row_vector(sort_asc(cS2_raw[i,]));
  } //end of i loop

  // Means of SDT distributions
  S2mu = meta_d/2;
  S1mu = -meta_d/2;

  for (i in 1:K){
    real C_area_rS1;              // Correct rejection
    real I_area_rS1;              // Miss
    real C_area_rS2;              // Hit
    real I_area_rS2;              // False Alarm
    vector[nratings*4] pr;        //vectors for probabilities
    vector[nratings*4] prT;       //transformed probabilities

    // Calculate normalization constants
    C_area_rS1 = Phi_approx(c1[i] - S1mu[i]);
    I_area_rS1 = Phi_approx(c1[i] - S2mu[i]);
    I_area_rS2 = 1-C_area_rS1;C_area_rS2 = 1-I_area_rS1;
```

Long transformed parameter block, because complex model calculations are done here. There are only samplings in model block.
Detailed explanations for this block will be in next slides.

```
    // Get nC_rS1 probs (Correct Rejection)
    pr[1] = Phi_approx(cS1[i,1] - S1mu[i])/C_area_rS1;
    for (k in 1:(nratings-2)) {pr[k+1] = (Phi_approx(cS1[i, k+1] - S1mu[i])-Phi_approx(cS1[i, k] - S1mu[i]))/C_area_rS1;}
    pr[nratings] = (Phi_approx(c1[i] - S1mu[i])-Phi_approx(cS1[i, nratings-1] - S1mu[i]))/C_area_rS1;
    // Get nI_rS2 probs (False alarm)
    pr[nratings+1] = ((1-Phi_approx(c1[i] - S1mu[i]))-(1-Phi_approx(cS2[i, 1]-S1mu[i])))/I_area_rS2;
    for (k in 1:(nratings-2)) {pr[nratings+1+k] = ((1-Phi_approx(cS2[i, k] - S1mu[i]))-(1-Phi_approx(cS2[i, k+1] - S1mu[i])))/I_area_rS2;}
    pr[nratings*2] = (1-Phi_approx(cS2[i, nratings-1] - S1mu[i]))/I_area_rS2;
    // Get nI_rS1 probs (Miss)
    pr[nratings*2+1] = Phi_approx(cS1[i, 1]-S2mu[i])/I_area_rS1;
    for (k in 1:(nratings-2)){pr[(nratings*2)+1+k] = (Phi_approx(cS1[i, k+1]-S2mu[i])-Phi_approx(cS1[i, k]-S2mu[i]))/I_area_rS1;}
    pr[nratings*3] = (Phi_approx(c1[i]-S2mu[i])-Phi_approx(cS1[i, nratings-1]-S2mu[i]))/I_area_rS1;
    // Get nC_rS2 probs (Hit)
    pr[(nratings*3)+1] = ((1-Phi_approx(c1[i]-S2mu[i]))-(1-Phi_approx(cS2[i, 1]-S2mu[i])))/C_area_rS2;
    for (k in 1:(nratings-2)){pr[(nratings*3)+1+k] = ((1-Phi_approx(cS2[i, k]-S2mu[i]))-(1-Phi_approx(cS2[i, k+1]-S2mu[i])))/C_area_rS2;}
    pr[nratings*4] = (1-Phi_approx(cS2[i, nratings-1]-S2mu[i]))/C_area_rS2;

    //Avoid underflow of probabilities
    for (k in 1:nratings*4){prT[k] = pr[k];} //end for k loop

    // Multinomial likelihood for response counts ordered as c(nR_S1,nR_S2)
    log_lik[i, 1] = multinomial_lpmf(counts[i, 1:nratings] | prT[1:nratings]); //CR
    log_lik[i, 2] = multinomial_lpmf(counts[i, (nratings+1):(nratings*2)] | prT[(nratings+1):(nratings*2)]); //False alarm
    log_lik[i, 3] = multinomial_lpmf(counts[i, ((nratings*2)+1):(nratings*3)] | prT[((nratings*2)+1):(nratings*3)]); //Miss
    log_lik[i, 4] = multinomial_lpmf(counts[i, ((nratings*3)+1):(nratings*4)] | prT[((nratings*3)+1):(nratings*4)]); //Hit
  } // end for i loop
}// end of transformed parameter block
```

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Transformed Parameter Block 1/4

```
transformed parameters{
  vector[K] S2mu;                      //mean of S2 (Signal Distribution)
  vector[K] S1mu;                      //mean of S1 (Noise Distribution)
  matrix[K, nratings-1] cS1_raw;       //c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2_raw;       //c2 vectors for S2 (cS2_p -> cS2_raw -> cS2)
  matrix[K, nratings-1] cS1;           //c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2;           //c2 vectors for S1 (cS2_p -> cS2_raw -> cS2)
  matrix[K, 4] log_lik;
```

c2 parameters for S1, S2 are transformed twice like this.
cS1_p and cS2_p are declared in parameter block.

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

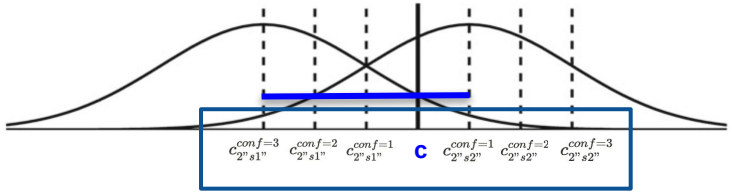
Transformed Parameter Block 2/4

```
for (i in 1:K){
  for (j in 1:(nratings-1)) {
    //sampling from truncated normal distribution (upper bound)
    cS1_raw[i, j] = sample_from_tnormal_ub(c1[i], 0., 2., cS1_p[i, j]);
    //sampling from truncated normal distribution (lower bound)
    cS2_raw[i, j] = sample_from_tnormal_lb(c1[i], 0., 2., cS2_p[i, j]);
    } //end of j loop

  // specify ordered prior on criteria
  cS1[i, 1:nratings-1] = to_row_vector(sort_asc(cS1_raw[i,]));
  cS2[i, 1:nratings-1] = to_row_vector(sort_asc(cS2_raw[i,]));
} //end of i loop
```

***[1] cS1_p & cS2_p** are sampled in model block. **cS1_raw & cS2_raw** are transformed with functions below, as if they were sampled from truncated normal distribution. I used this function because they were not sampled from truncated normal with T[,] function in Stan. Maybe that's because the upper bound and lower bound of them were different for each subject, so couldn't declared.*

***[2] cS1_raw & cS2_raw** are sorted because the order matters (related to confidence ratings!) Like blue box below.*




```
// Sampling from Truncated Normal
   //(https://sites.google.com/site/michaelchughessite/research/sampling-from-truncated-normal)
functions {
    real sample_from_tnormal_lb(real a, real mu, real sigma, real u_pr) {
        real phi_a_bar = Phi_approx((a - mu) / sigma);
        real u = phi_a_bar + (1 - phi_a_bar) * u_pr;
        real x = sigma * inv_Phi(u) + mu;
        return x;
    }
    real sample_from_tnormal_ub(real a, real mu, real sigma, real u_pr) {
        real phi_a_bar = Phi_approx((a - mu) / sigma);
        real u = phi_a_bar * u_pr;
        real x = sigma * inv_Phi(u) + mu;
        return x;
    }
}
```

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Transformed Parameter Block 3/4

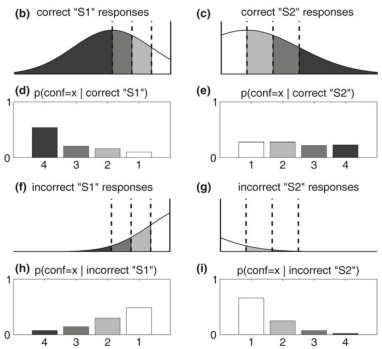
```
// Means of SDT distributions
S2mu = meta_d/2;
S1mu = -meta_d/2;

for (i in 1:K){
  real C_area_rS1;              // Correct rejection
  real I_area_rS1;              // Miss
  real C_area_rS2;              // Hit
  real I_area_rS2;              // False Alarm
  vector[nratings*4] pr;        //vectors for probabilities
  vector[nratings*4] prT;       //transformed probabilities

  // Calculate normalization constants
  C_area_rS1 = Phi_approx(c1[i] - S1mu[i]);
  I_area_rS1 = Phi_approx(c1[i] - S2mu[i]);
  I_area_rS2 = 1-C_area_rS1;C_area_rS2 = 1-I_area_rS1;
```

This whole code is to calculate probabilities for possible responses.
(p(Conf = c & Response = r \| Stimulus = s) / p(Response = r \| Stimulus = s))

$$Prob_\theta(Conf = 2 \mid Stim = S1,\ Resp = S1) = \frac{\Phi(c_{2"S1"}^{conf=2}, -\frac{d'}{2}) - \Phi(c_{2"S1"}^{conf=3}, -\frac{d'}{2})}{\Phi(c, -\frac{d'}{2})}$$




```
  // Get nC_rS1 probs (Correct Rejection)
  pr[1] = Phi_approx(cS1[i,1] - S1mu[i])/C_area_rS1;
  for (k in 1:(nratings-2)) {pr[k+1] = (Phi_approx(cS1[i, k+1] - S1mu[i])-Phi_approx(cS1[i, k] - S1mu[i]))/C_area_rS1;}
  pr[nratings] = (Phi_approx(c1[i] - S1mu[i])-Phi_approx(cS1[i, nratings-1] - S1mu[i]))/C_area_rS1;
  // Get nI_rS2 probs (False alarm)
  pr[nratings+1] = ((1-Phi_approx(c1[i] - S1mu[i]))-(1-Phi_approx(cS2[i, 1]-S1mu[i])))/I_area_rS2;
  for (k in 1:(nratings-2)) {pr[nratings+1+k] = ((1-Phi_approx(cS2[i, k] - S1mu[i]))-(1-Phi_approx(cS2[i, k+1] - S1mu[i])))/I_area_rS2;}
  pr[nratings*2] = (1-Phi_approx(cS2[i, nratings-1] - S1mu[i]))/I_area_rS2;
  // Get nI_rS1 probs (Miss)
  pr[nratings*2+1] = Phi_approx(cS1[i, 1]-S2mu[i])/I_area_rS1;
  for (k in 1:(nratings-2)){pr[(nratings*2)+1+k] = (Phi_approx(cS1[i, k+1]-S2mu[i])-Phi_approx(cS1[i, k]-S2mu[i]))/I_area_rS1;}
  pr[nratings*3] = (Phi_approx(c1[i]-S2mu[i])-Phi_approx(cS1[i, nratings-1]-S2mu[i]))/I_area_rS1;
  // Get nC_rS2 probs (Hit)
  pr[(nratings*3)+1] = ((1-Phi_approx(c1[i]-S2mu[i]))-(1-Phi_approx(cS2[i, 1]-S2mu[i])))/C_area_rS2;
  for (k in 1:(nratings-2)){pr[(nratings*3)+1+k] = ((1-Phi_approx(cS2[i, k]-S2mu[i]))-(1-Phi_approx(cS2[i, k+1]-S2mu[i])))/C_area_rS2;}
  pr[nratings*4] = (1-Phi_approx(cS2[i, nratings-1]-S2mu[i]))/C_area_rS2;
```

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Transformed Parameter Block 4/4

Ver 1

```
    //Avoid underflow of probabilities
    for (k in 1:nratings*4){prT[k] = pr[k];} //end for k loop

    // Multinomial likelihood for response counts ordered as c(nR_S1,nR_S2)
    log_lik[i, 1] = multinomial_lpmf(counts[i, 1:nratings] | prT[1:nratings]); //CR
    log_lik[i, 2] = multinomial_lpmf(counts[i, (nratings+1):(nratings*2)] | prT[(nratings+1):(nratings*2)]); //False alarm
    log_lik[i, 3] = multinomial_lpmf(counts[i, ((nratings*2)+1):(nratings*3)] | prT[((nratings*2)+1):(nratings*3)]); //Miss
    log_lik[i, 4] = multinomial_lpmf(counts[i, ((nratings*3)+1):(nratings*4)] | prT[((nratings*3)+1):(nratings*4)]); //Hit
  } // end for i loop
}// end of transformed parameter block
```

In version 1, there's no manipulation of probabilities because other manipulation is already done in parameter block by setting boundary of meta-d

Used multinomial functions like this.

Ver 2

```
//Avoid underflow of probabilities
for (j in 1:4) {
prT[(j-1)*nratings+1:j*nratings] = normalize(0.999 * pr[(j-1)*nratings+1:j*nratings] + 0.001);
}
```

```
functions {
  vector normalize(vector x) {
    return x / sum(x);
  }
}
```

In version 2, boundary of meta-d is not set and manipulation of probabilities is done. with normalize function and setting them to 1.

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Model Block

In model block, samplings of meta-d, cS1_p, and cS2_p are done.

```
model {
///// Type 2 SDT MODEL (META-d) /////

  //Type 2 priors
  meta_d ~ normal(d1,0.5);        //sampling meta-d

  for (i in 1:K){                 //sampling c2 vectors for S1 and S2
    for (j in 1:(nratings-1)) {
      cS1_p[i, j] ~ uniform(0,1); //c2 vectors (cS1_p -> cS1_raw -> cS1)
      cS2_p[i, j] ~ uniform(0,1); //c2 vectors (cS2_p -> cS2_raw -> cS2)
      }
  }

  target += sum(log_lik);
}
```

Version 1: set boundary of meta-d
Version 2: Normalize after calculating all probabilities

Codes

Stan code for individual estimation

Generated Quantities block

```
generated quantities{
  real counts_pred[K, 4*nratings];
  vector[K] Mratio;

  for (i in 1:K){
    Mratio[i] = meta_d[i]/d1[i];
  }


  ////For posterior predictive check

  //Set all posterior predictions to 0 (avoids NULL values)

  for (i in 1:K){
    for (r in 1:(4*nratings)){
      counts_pred[i,r] = -1;
    } //end of r loop
  } //end of i loop
```

Mratio is calculated for further exploration

Posterior predictive check.

Codes for calculating counts_pred will be skipped here because important parts for the code is almost same as those in transformed parameters block

Version 1: with Parameter expansion (Fleming 2017)
Version 2: without parameter expansion

Codes

Stan code for Hierarchical estimation

Data Block

```
data {
  int<lower=1> K;              //Number of Subjects

  // Type 1 counts
  int<lower=0> N[K];           // Noise (S1 stimulus trials)
  int<lower=0> S[K];           // Signal (S2 stimulus trials)
  int<lower=0> H[K];           // Hit trials (resp=S2, stim=S2)
  int<lower=0> FA[K];          // False Alarm (resp=S2, stim=S1)

  //Type 2 data
  int<lower=1> nratings;       //the number of ratings
  int counts[4*nratings, K];   //count vectors for all subjects, counts = [nR_S1, nR_S2]

  // Type 1 SDT parameters
  vector[K] d1;                // Type 1 discriminability
  vector[K] c1;                // Type 1 bias
}
```

Type 1 d and c are not estimated but calculated here (Fleming 2017)

Version 1: with Parameter expansion (Fleming 2017)
Version 2: without parameter expansion

Codes

Stan code for hierarchical estimation

Parameter Block

Ver 1

```
parameters {
  // Type 2 SDT parameters
  matrix<lower = 0, upper = 1>[K, nratings-1] cS1_p;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS2_p;


  // Type 2 SDT hyperparameters
  real mu_c2;                        // hyperparameter for c2
  real<lower=0> sigma_c2;            // hyperparameter for c2
  real mu_logMratio;                 // hyperparameter for logMratio
  real<lower=0> epsilon_logMratio;   // hyperparameter for logMratio
  vector[K] delta;                   // hyperparameter for logMratio
  real<lower=0> sigma_delta;
}
```

Ver 2

```
parameters {
  // Type 2 SDT hyperparameters
  real mu_c2;                        // hyperparameter for c2
  real<lower=0> lambda_c2;            // hyperparameter for c2

  real mu_logMratio;                 // hyperparameter for logMratio
  real<lower=0> lambda_logMratio;     // hyperparameter for logMratio

  // Type 2 SDT parameters
  vector[K] logMratio;             // log(meta-d/d)
  matrix<lower = 0, upper = 1>[K, nratings-1] cS1_p;
  matrix<lower = 0, upper = 1>[K, nratings-1] cS2_p;
}
```

Additional hyperparameters for parameter expansion

why logMratio?:

According to Fleming (2017), the group-level parameter is the ratio of meta-d'/d' rather than meta-d', because the parameter of interest is metacognitive efficiency at the group level. Meta-d' is influenced by subject- or group-level variability in type 1 performance (d'), while the ratio of meta-d'/d' ruled out type 1 d'

Version 1: with Parameter expansion (Fleming 2017)
Version 2: without parameter expansion

Codes

Stan code for hierarchical estimation

Transformed Parameter Block

Ver 1

```
transformed parameters{
  real lambda_c2;
  real lambda_delta;
  real sigma_logMratio;

  vector[K] Mratio;                // meta-d/d
  vector[K] logMratio;             // log(meta-d/d)
  vector[K] meta_d;                // meta-d' actually..
  vector[K] S2mu;                  // mean of S2 (Signal Distribution)
  vector[K] S1mu;                  // mean of S1 (Noise Distribution)
  matrix[K, nratings-1] cS1_raw;   // c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2_raw;   // c2 vectors for S2 (cS2_p -> cS2_raw -> cS2)
  matrix[K, nratings-1] cS1;       // c2 vectors (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2;       // c2 vectors (cS2_p -> cS2_raw -> cS2)
  matrix[K, 4] log_lik;

  lambda_c2 = pow(sigma_c2, -2);
  lambda_delta = pow(sigma_delta, -2);
  sigma_logMratio = fabs(epsilon_logMratio)*sigma_delta;

for (i in 1:K){
  real C_area_rS1;              // Correct rejection
  real I_area_rS1;              // Miss
  real C_area_rS2;              // Hit
  real I_area_rS2;              // False Alarm
  vector[nratings * 4] pr;        //vectors for probabilities
  vector[nratings * 4] prT;       //transformed probabilities

  for (j in 1:(nratings-1)) {
    //sampling from truncated normal distribution (upper bound)
    cS1_raw[i, j] = sample_from_tnormal_ub(c1[i], -mu_c2, lambda_c2, cS1_p[i, j]);
    //sampling from truncated normal distribution (lower bound)
    cS2_raw[i, j] = sample_from_tnormal_lb(c1[i], mu_c2, lambda_c2, cS2_p[i, j]);
  }

  // specify ordered prior on criteria
  cS1[i, 1:nratings-1] = to_row_vector(sort_asc(cS1_raw[i,]));
  cS2[i, 1:nratings-1] = to_row_vector(sort_asc(cS2_raw[i,]));

  // Means of SDT distributions
  logMratio[i] = mu_logMratio + epsilon_logMratio * delta[i];
  Mratio[i] = exp(logMratio[i]);
  meta_d[i] = Mratio[i] * d1[i];

  S2mu[i] = meta_d[i] / 2;
  S1mu[i] = -meta_d[i] / 2;
```

The remaining part of this block is same with that of individual estimation code.

Red boxes indicate different parts from those of individual estimation codes and from each other (ver 1 and ver 2)

Ver 2

```
transformed parameters{

  vector[K] Mratio;                // meta-d/d
  real sigma_c2;
  real sigma_logMratio;
  vector[K] meta_d;                // meta-d' actually..
  vector[K] S2mu;                  // mean of S2 (Signal Distribution)
  vector[K] S1mu;                  // mean of S1 (Noise Distribution)
  matrix[K, nratings-1] cS1_raw;   // c2 vectors for S1 (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2_raw;   // c2 vectors for S2 (cS2_p -> cS2_raw -> cS2)
  matrix[K, nratings-1] cS1;       // c2 vectors (cS1_p -> cS1_raw -> cS1)
  matrix[K, nratings-1] cS2;       // c2 vectors (cS2_p -> cS2_raw -> cS2)
  matrix[K, 4] log_lik;

  sigma_c2 = inv_sqrt(lambda_c2);
  sigma_logMratio = inv_sqrt(lambda_logMratio);

  for (i in 1:K){
    real C_area_rS1;              // Correct rejection
    real I_area_rS1;              // Miss
    real C_area_rS2;              // Hit
    real I_area_rS2;              // False Alarm
    vector[nratings * 4] pr;        //vectors for probabilities
    vector[nratings * 4] prT;       //transformed probabilities

    for (j in 1:(nratings-1)) {
      cS1_raw[i, j] = sample_from_tnormal_ub(c1[i], -mu_c2, lambda_c2, cS1_p[i, j]);
      cS2_raw[i, j] = sample_from_tnormal_lb(c1[i], mu_c2, lambda_c2, cS2_p[i, j]);
    }

    // specify ordered prior on criteria
    cS1[i, 1:nratings-1] = to_row_vector(sort_asc(cS1_raw[i,]));
    cS2[i, 1:nratings-1] = to_row_vector(sort_asc(cS2_raw[i,]));

    // Means of SDT distributions
    Mratio[i] = exp(logMratio[i]);
    meta_d[i] = Mratio[i] * d1[i];
```

Version 1: with Parameter expansion (Fleming 2017)
Version 2: without parameter expansion

Codes

Stan code for hierarchical estimation

Model Block

Ver 1

```
model {
  ///// Type 2 SDT MODEL (META-d) /////

  // Type 2 priors
  mu_c2 ~ normal(0,0.01);        //
  sigma_c2 ~ normal(0, 0.01)T[0,];
  mu_logMratio ~ normal(0,1);
  sigma_delta ~ normal(0,1)T[0, ];
  epsilon_logMratio ~ beta(1,1);

  for (i in 1:K){
    delta[i] ~ normal(0, lambda_delta);
    for (j in 1:(nratings - 1)) {
      cS1_p[i, j] ~ uniform(0, 1);
      cS2_p[i, j] ~ uniform(0, 1);
    }
  }

  target += sum(log_lik);
}
```

Ver 2

```
model {
  ///// Type 2 SDT MODEL (META-d) /////

  // Type 2 priors
  mu_c2 ~ normal(0,0.01);        //
  lambda_c2 ~ gamma(0.001, 0.001);
  mu_logMratio ~ normal(0,1);
  lambda_logMratio ~ gamma(0.001, 0.001);

  logMratio ~ normal(mu_logMratio, lambda_logMratio);

  for (i in 1:K){
    for (j in 1:(nratings - 1)) {
      cS1_p[i, j] ~ uniform(0, 1);
      cS2_p[i, j] ~ uniform(0, 1);
    }
  }

  target += sum(log_lik);
}
```

Generated quantities block is skipped

Results

Results_Indiv_version 1 & 2

Settings of Individual Bayesian Estimation

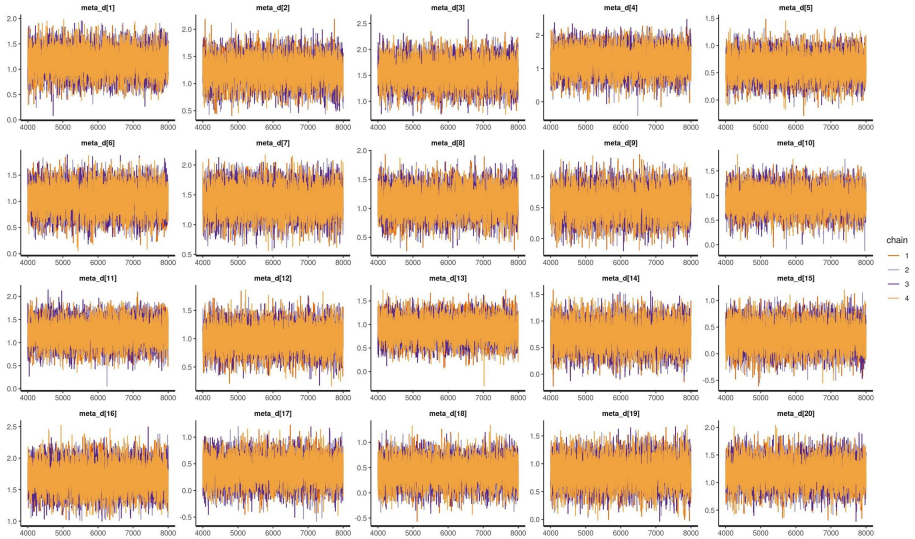
- Iteration = 8000
- Warmup = 4000
- Chains = 4
- Cores = 20
- Tested the model with Contrast perception task data first.
- I didn’t include plots of orientation perception and orientation memory tasks, but ran codes with them and included results and figures in the folder.

Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

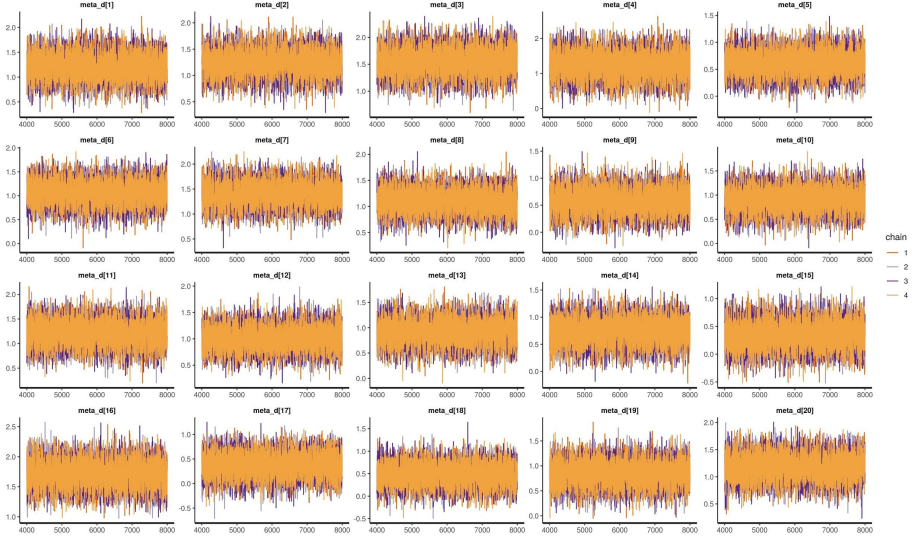
Results_Indiv_version 1 & 2

Individual_trace plot (meta_d)

Ver 1



Ver 2

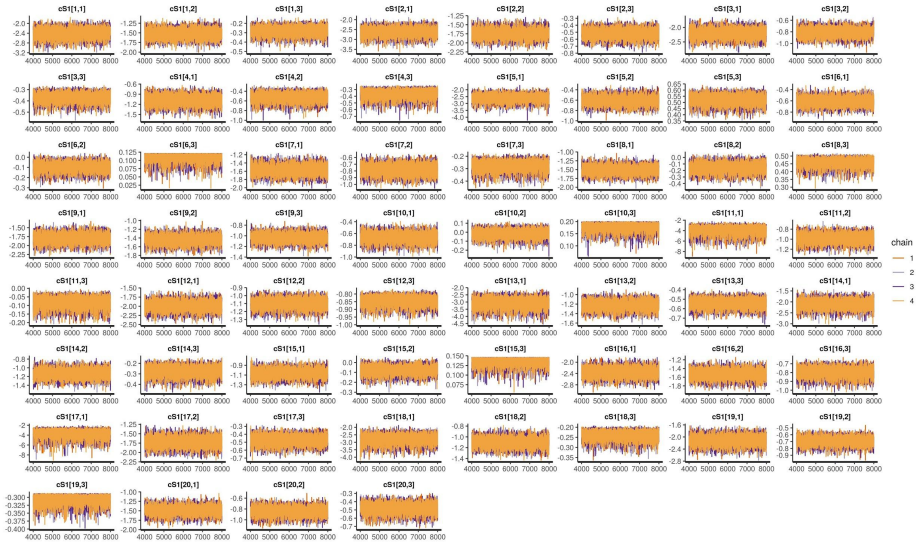


Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

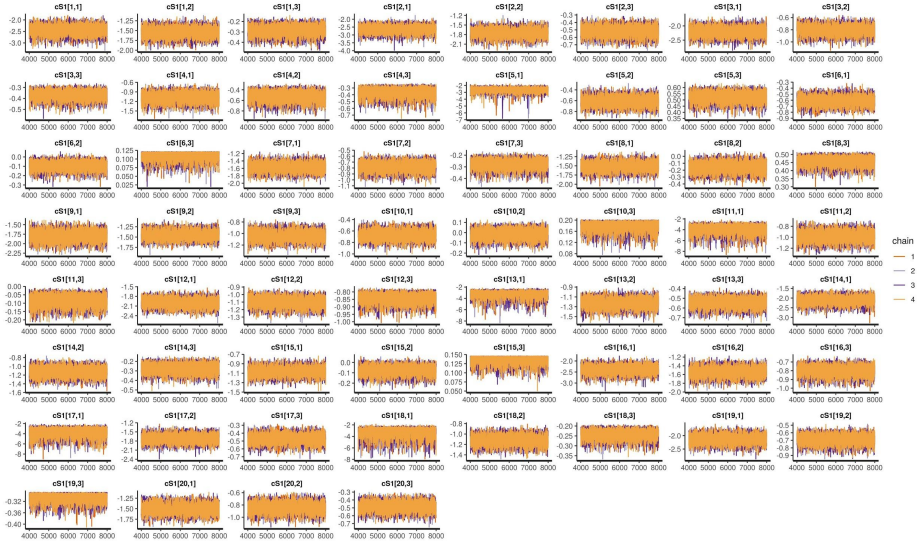
Results_Indiv_version 1 & 2

Individual_trace plot (cS1)

Ver 1



Ver 2



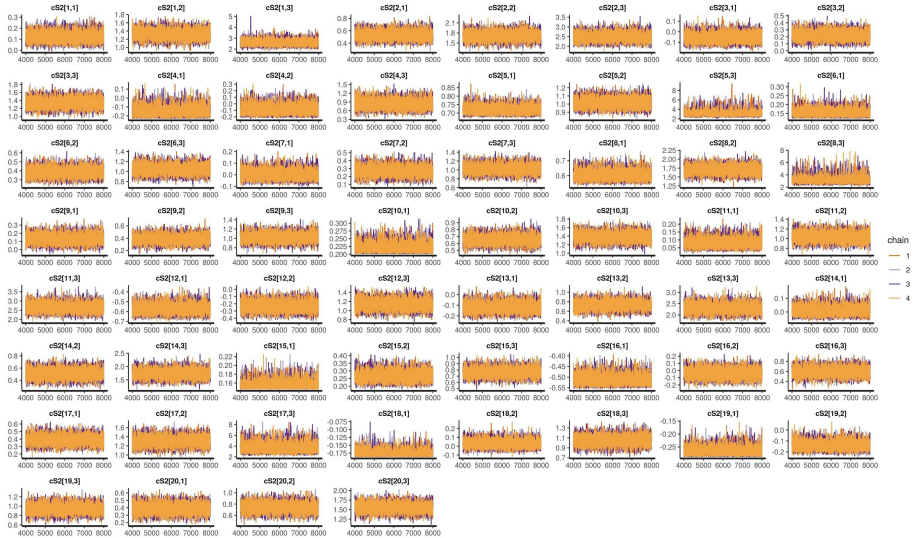
Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

Results_Indiv_version 1 & 2

Individual_trace plot (cS2)

Ver 1

Ver 2

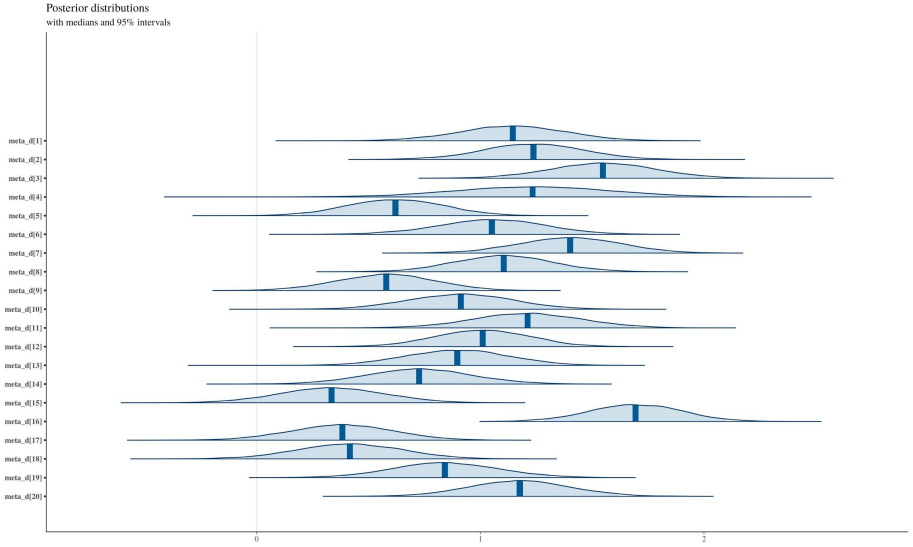


Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

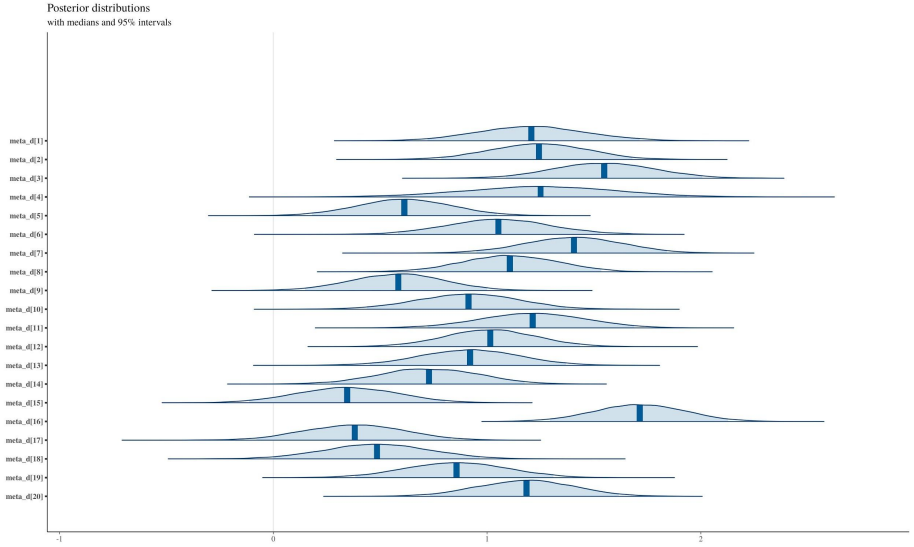
Results_Indiv_version 1 & 2

Individual_posterior distribution_meta_d

Ver 1



Ver 2

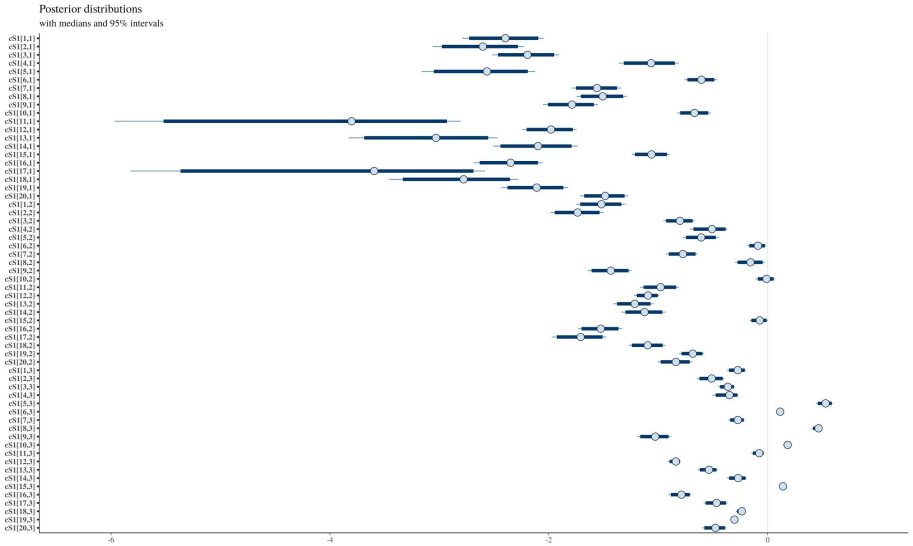


Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

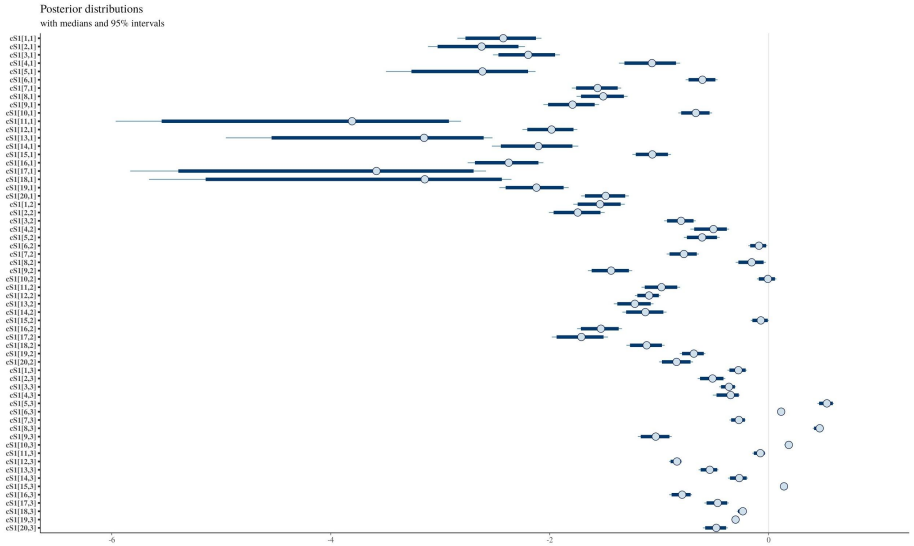
Results_Indiv_version 1 & 2

Individual_posterior distribution_cS1

Ver 1



Ver 2

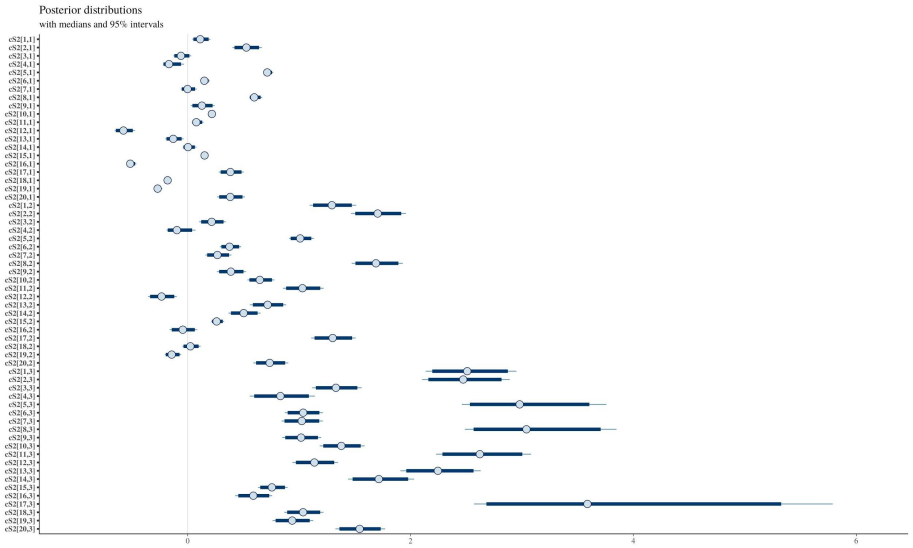


Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

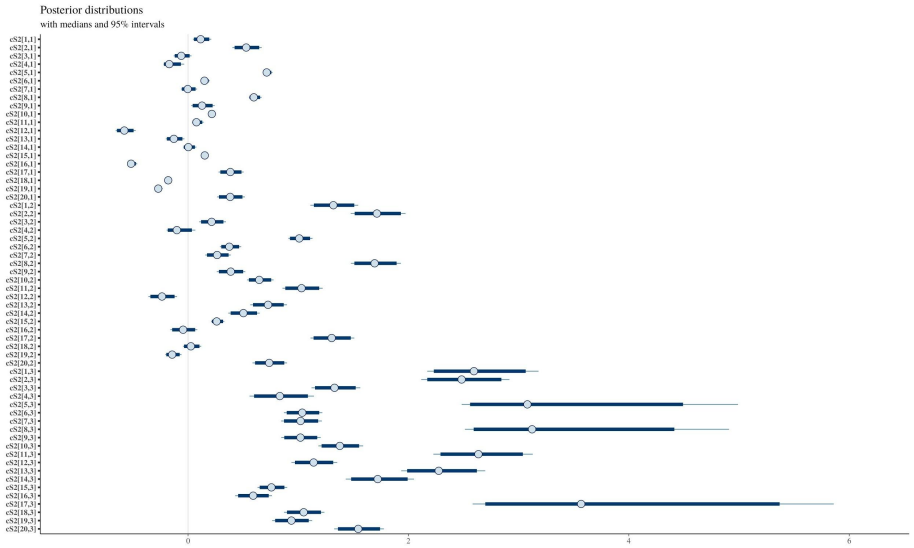
Results_Indiv_version 1 & 2

Individual_posterior distribution_cS2

Ver 1



Ver 2

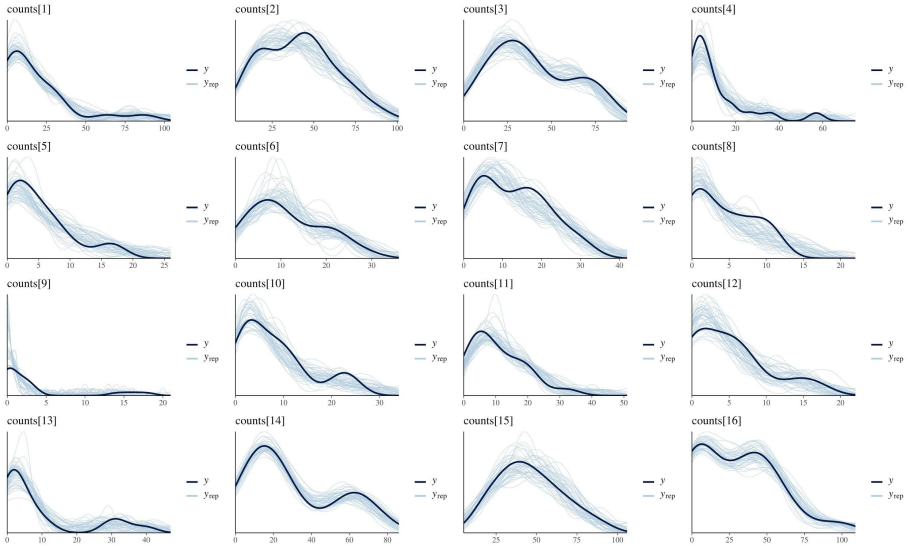


Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

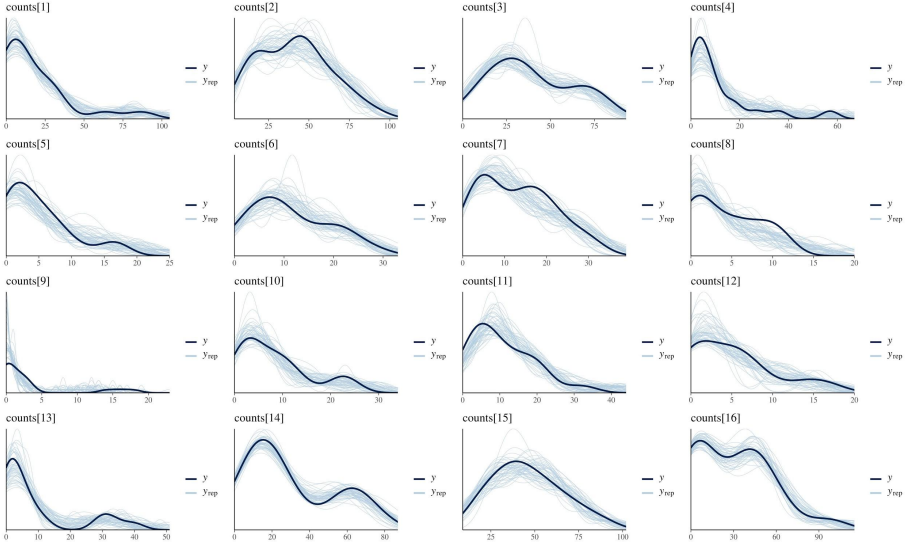
Results_Indiv_version 1 & 2

Posterior Predictive Check

Ver 1



Ver 2

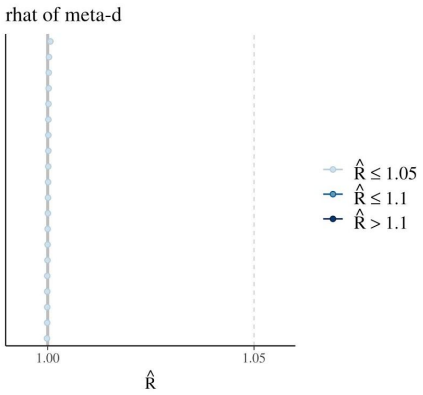


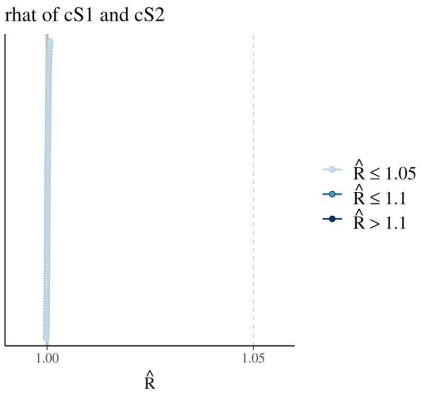
Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

Results_Indiv_version 1 & 2

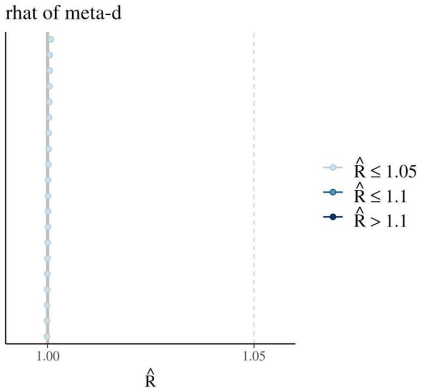
Rhat

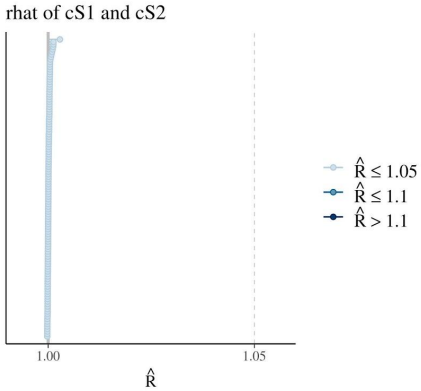
Ver 1





Ver 2



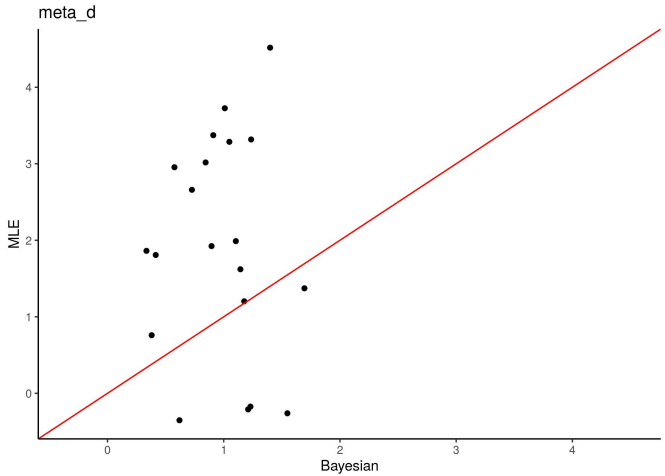


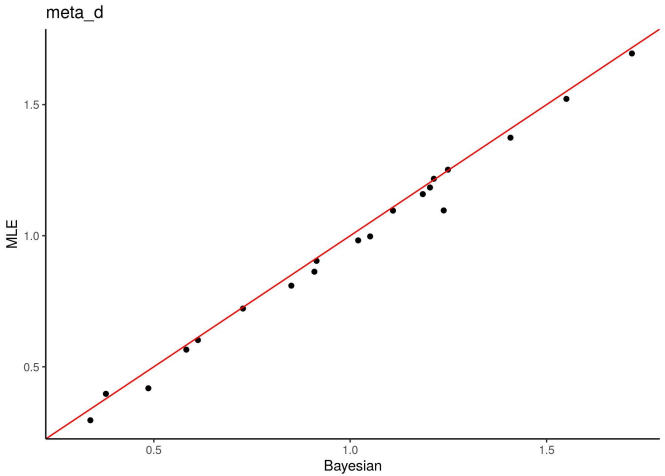
Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

Results_Indiv_version 1 & 2

Comparison with MLE_meta_d

Scales of x-axes and y-axes are different in both figures





Ver 1

```
optimizing(stan_model("1_1_Indiv.stan"), data = dataList,
  seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
  iter = 4000)
```

Ver 2

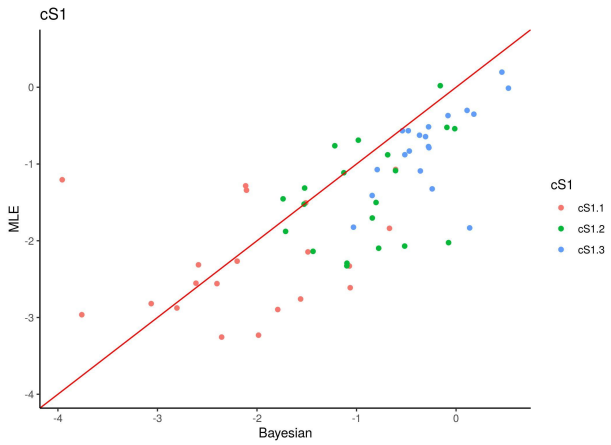
```
optimizing(stan_model("1_2_Indiv_v2.stan"), data = dataList,
          seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
          iter = 4000)
```

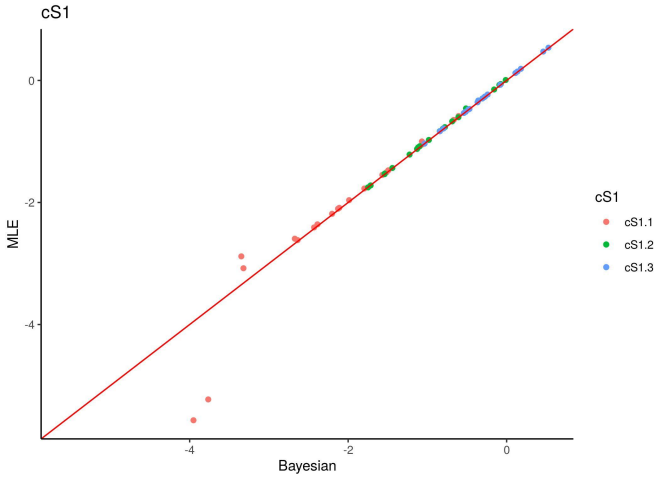
Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**
The results from Bayesian estimation is similar to each other, but MLE results are different.
In version 1, MLE estimates are much bigger than those in version 2.

Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

Results_Indiv_version 1 & 2

Comparison with MLE_cS1





Ver 1

```
optimizing(stan_model("1_1_Indiv.stan"), data = dataList,
  seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
  iter = 4000)
```

Ver 2

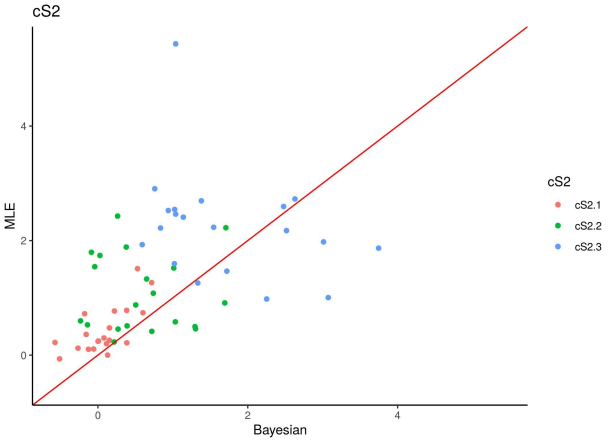
```
optimizing(stan_model("1_2_Indiv_v2.stan"), data = dataList,
          seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
          iter = 4000)
```

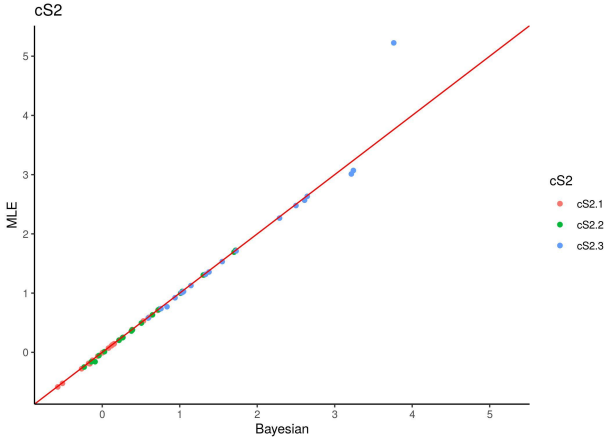
Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**
The results from Bayesian estimation is similar to each other, but MLE results are different.
Version 2 shows more similarity in MLE and Bayesian estimation results.

Version 1: set boundary of meta-d
Version 2: Additional normalization after calculating all probabilities

Results_Indiv_version 1 & 2

Comparison with MLE_cS2





Ver 1

```
optimizing(stan_model("1_1_Indiv.stan"), data = dataList,
  seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
  iter = 4000)
```

Ver 2

```
optimizing(stan_model("1_2_Indiv_v2.stan"), data = dataList,
          seed =2020, algorithm = c("LBFGS"),verbose = TRUE, importance_resampling=TRUE,
          iter = 4000)
```

Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**
The results from Bayesian estimation is similar to each other, but MLE results are different.
Version 2 shows more similarity in MLE and Bayesian estimation results.

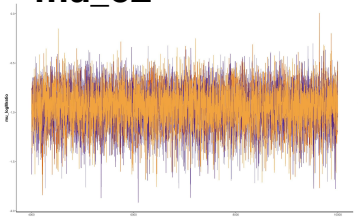
Settings of Hierarchical Bayesian Estimation

- Iteration = 10000
- Warmup = 4000
- Chains = 4
- Cores = 20
- Control= list (adapt_delta = 0.99, max_treedepth = 15)
- Tested the model with Contrast perception task data first.

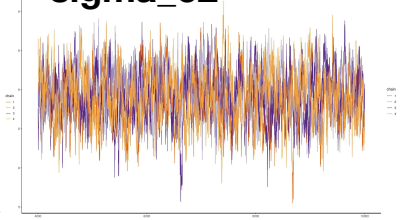
Trace plot (hyperparameters)

Normal

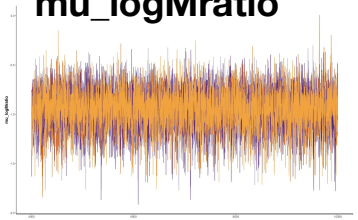
mu_c2



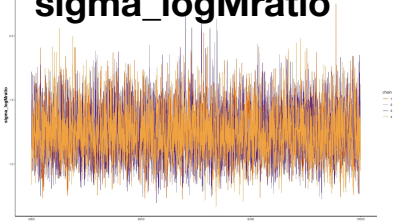
sigma_c2



mu_logMratio

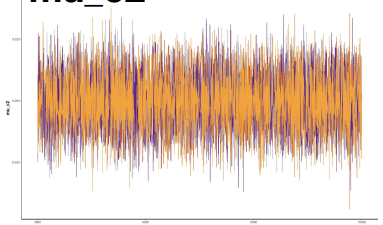


sigma_logMratio

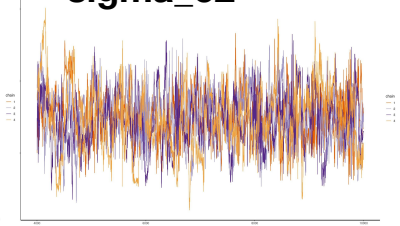


Parameter expansion

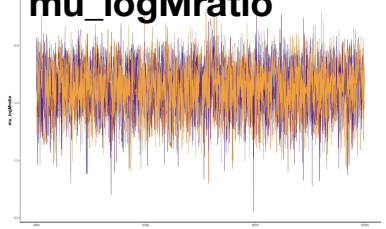
mu_c2



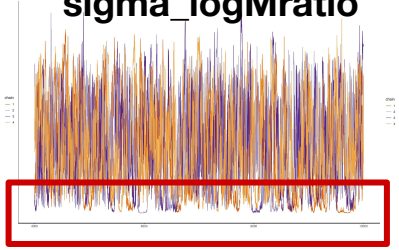
sigma_c2



mu_logMratio



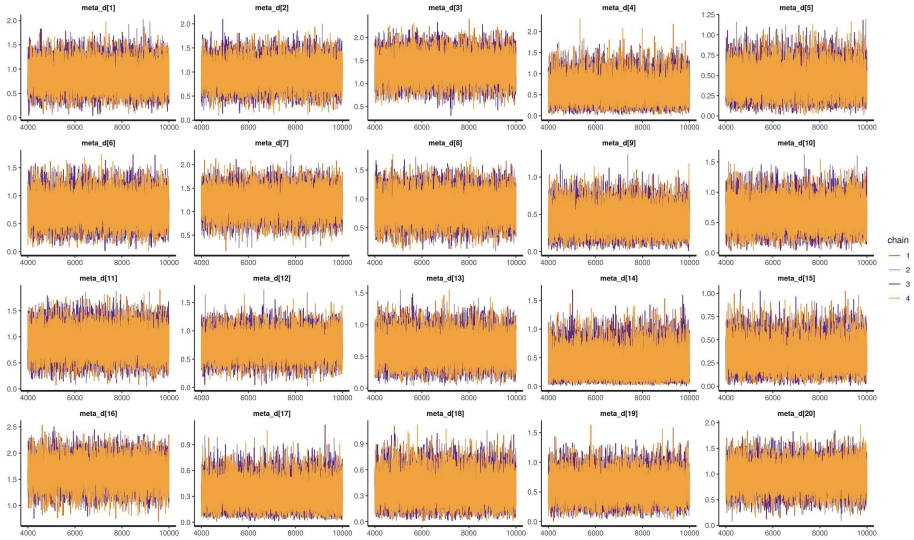
sigma_logMratio



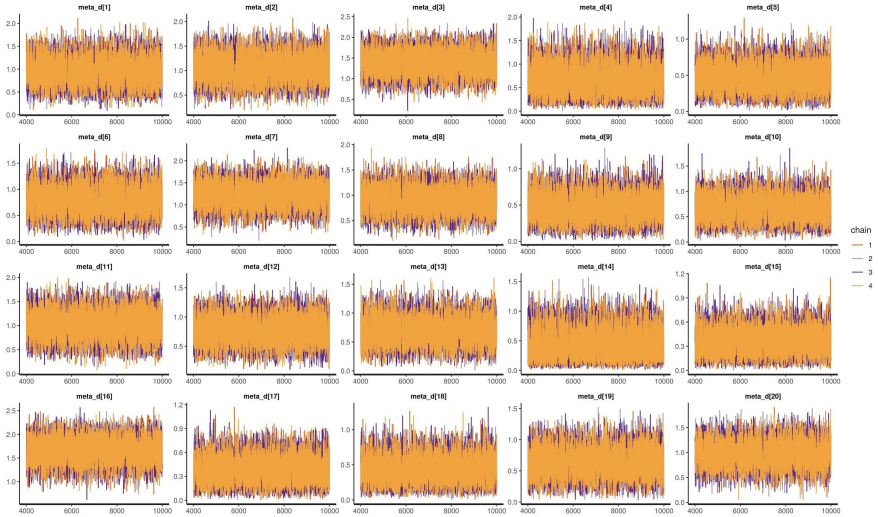
Ironically, *in the model **with** parameter expansion,* the hierarchical variance parameter (sigma logMratio) was trapped near zero during sampling.

Trace plot (meta_d)

Normal



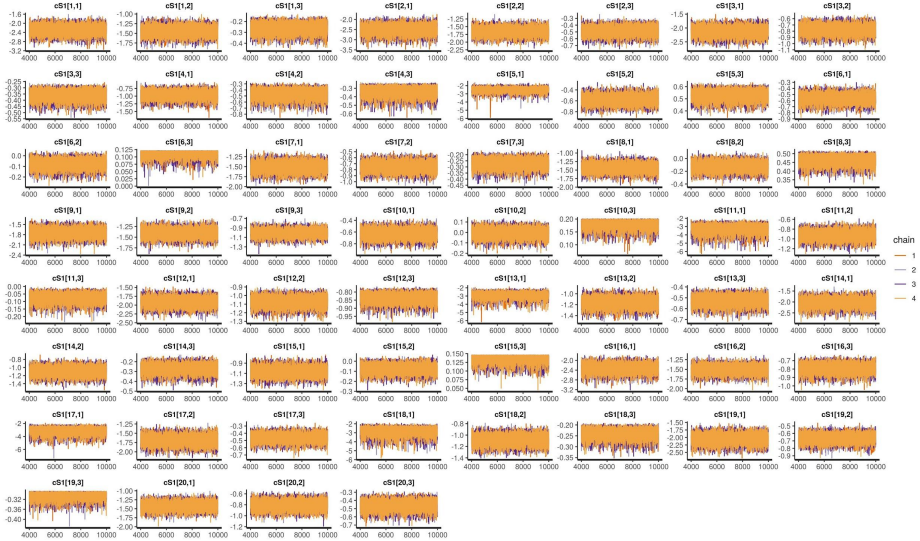
Parameter expansion



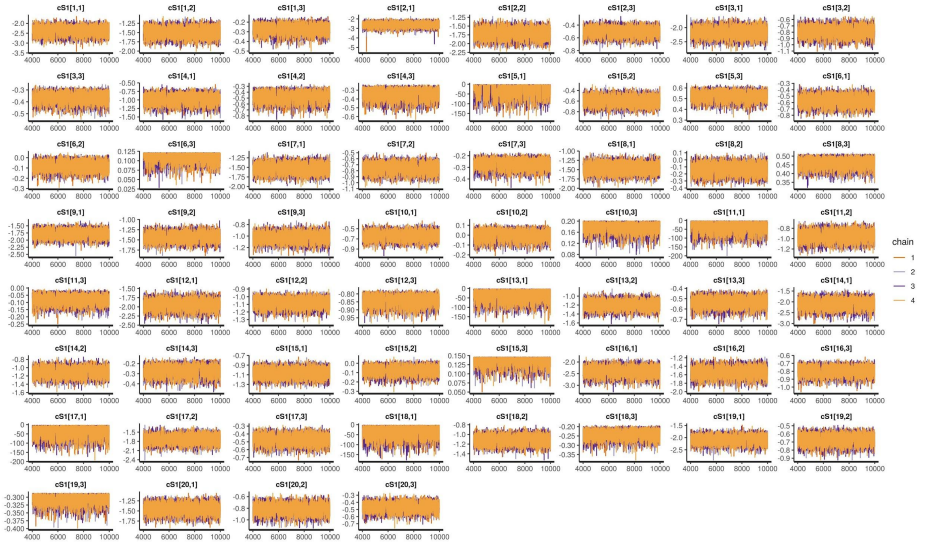
Results_Hierarchical_version 1 & 2

Trace plot (cS1)

Normal



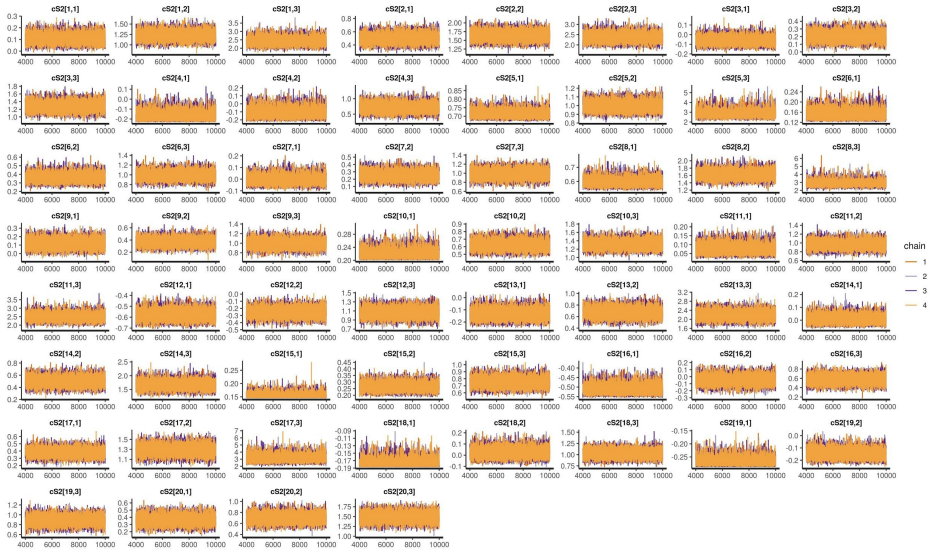
Parameter expansion



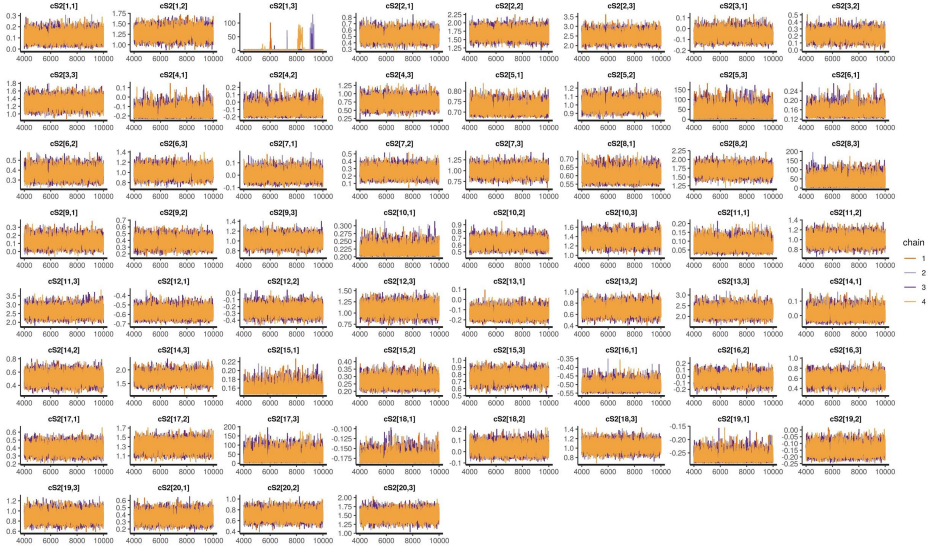
Results_Hierarchical_version 1 & 2

Trace plot (cS2)

Normal



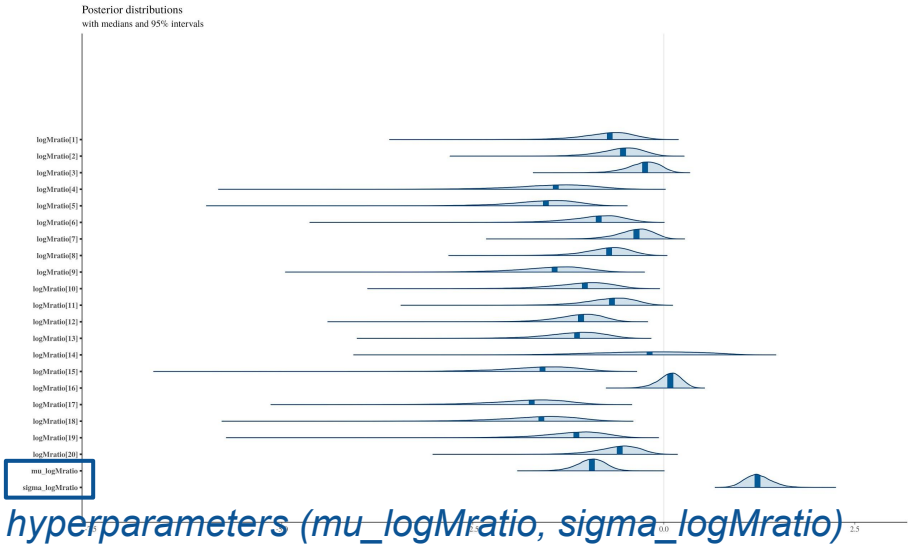
Parameter expansion



Results_Hierarchical_version 1 & 2

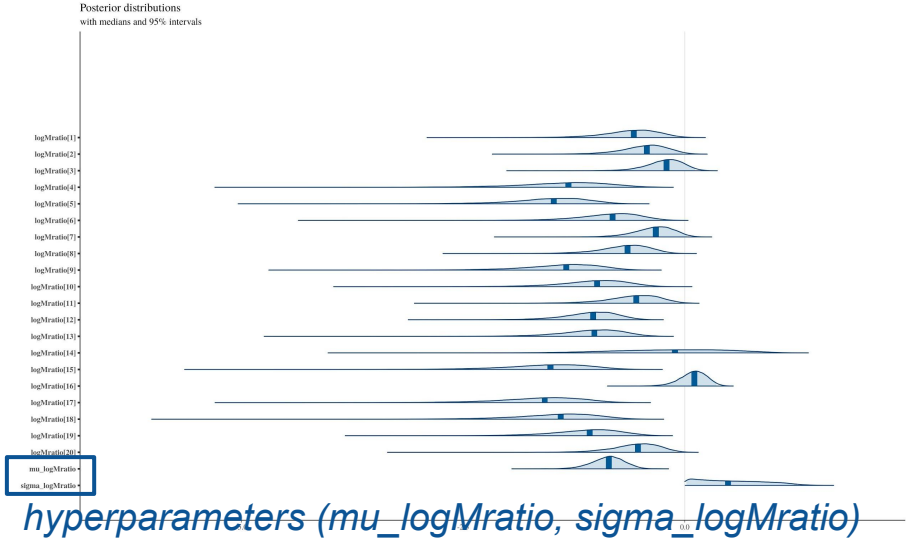
Hierarchical_posterior distribution_logMratio

Normal



hyperparameters (mu_logMratio, sigma_logMratio)

Parameter expansion

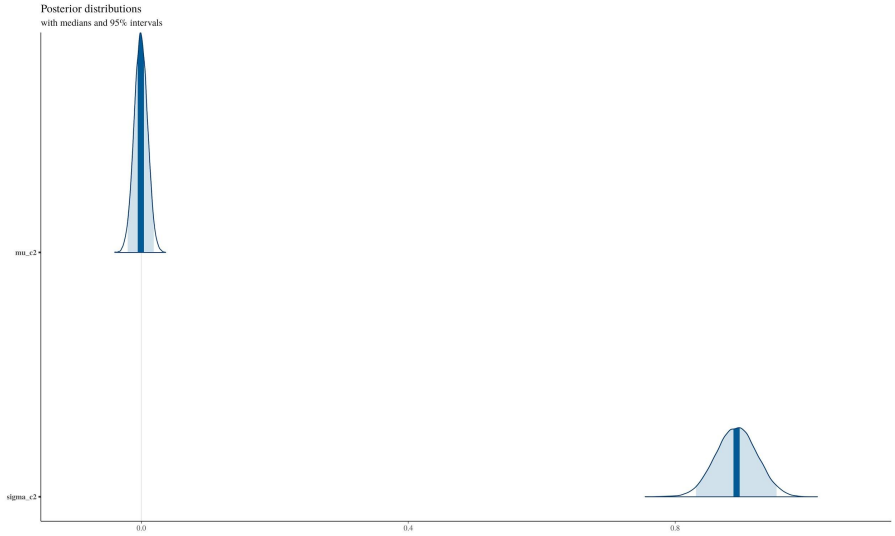


hyperparameters (mu_logMratio, sigma_logMratio)

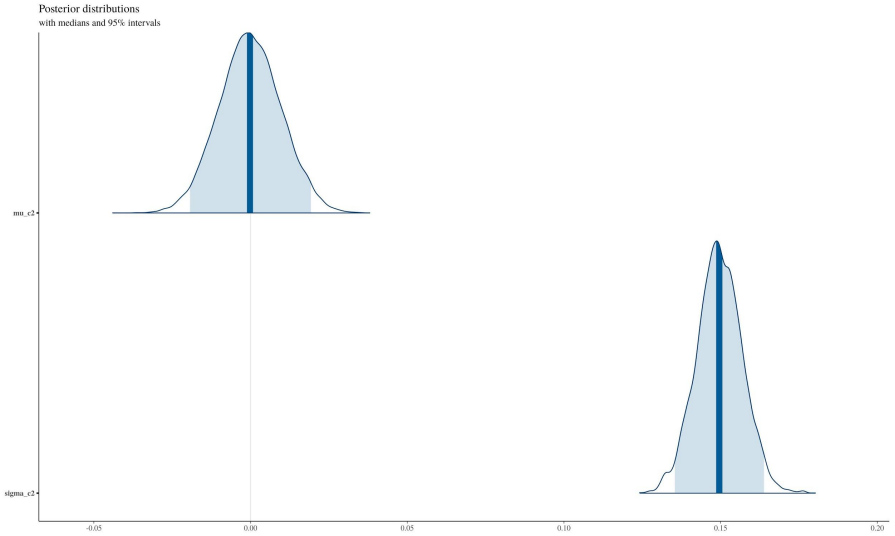
Results_Hierarchical_version 1 & 2

Hierarchical_posterior distribution_mu&sigma_c2

Normal



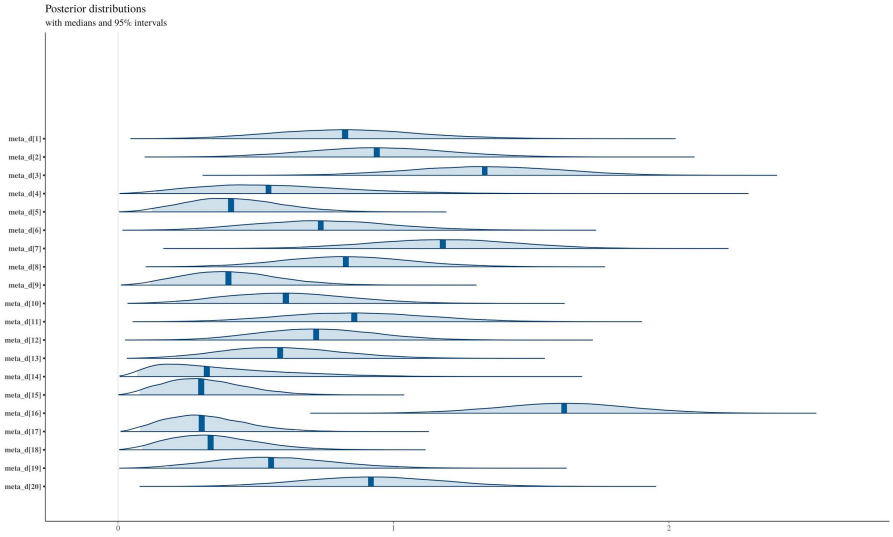
Parameter expansion



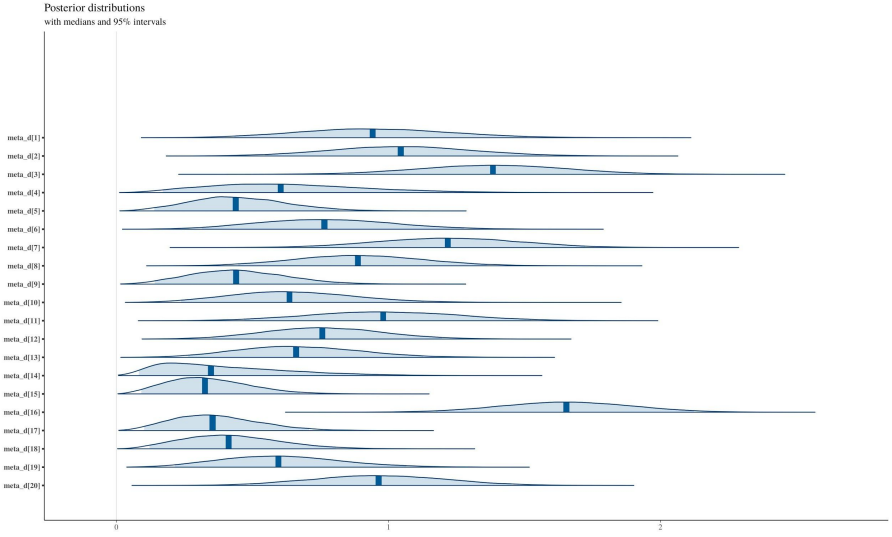
Results_Hierarchical_version 1 & 2

Hierarchical_posterior distribution_meta_d

Normal

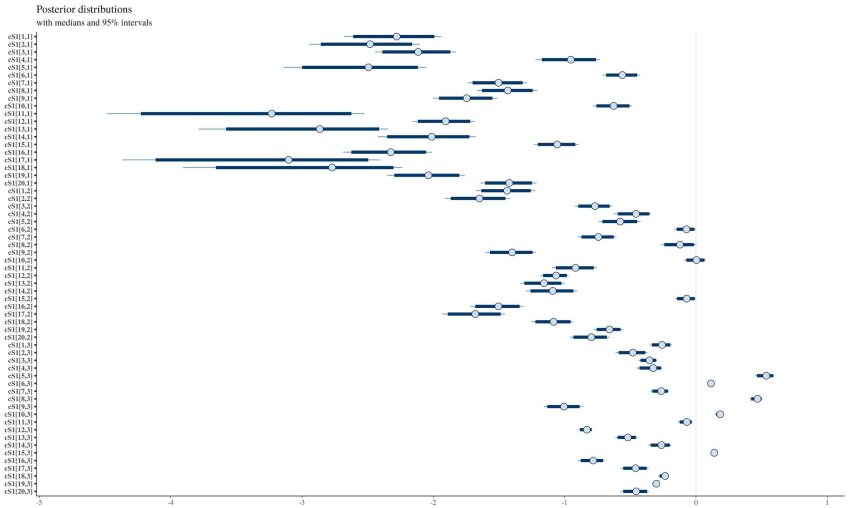


Parameter expansion

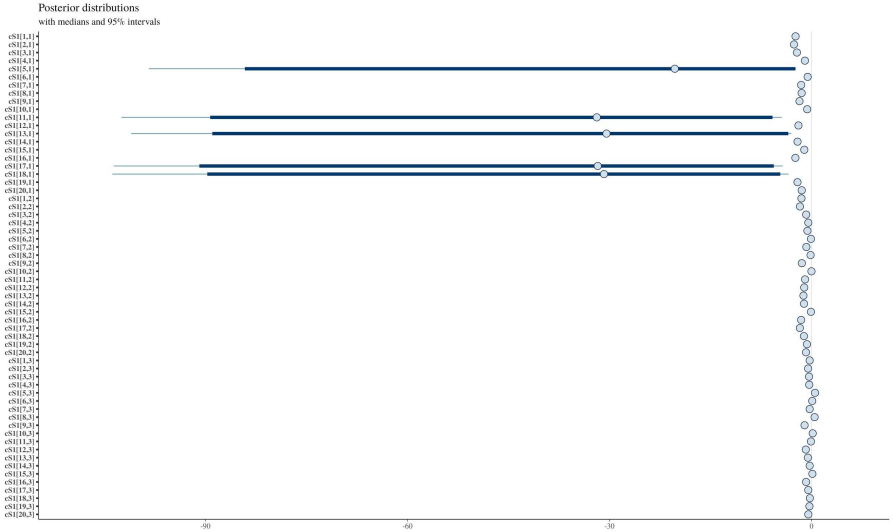


Hierarchical_posterior distribution_cS1

Normal



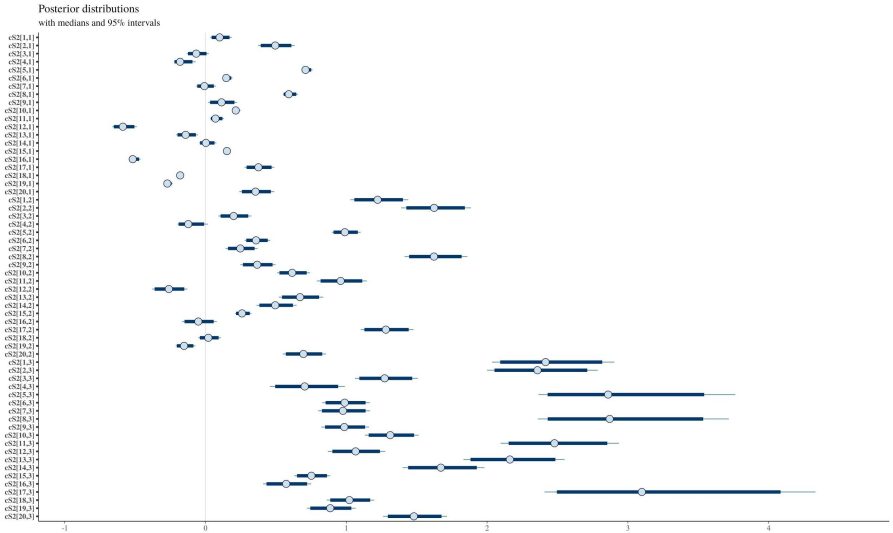
Parameter expansion



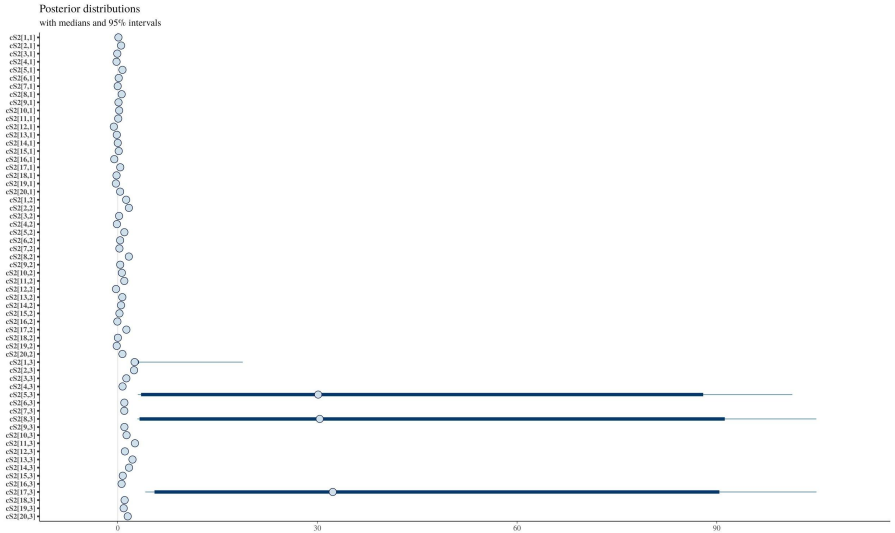
Estimation of c2S1 in parameter expanded model is poor. Most values are near 0.
I tried to figure out why, but even after multiple-check on my code, I couldn't solve the problem.

Hierarchical_posterior distribution_cS2

Normal



Parameter expansion

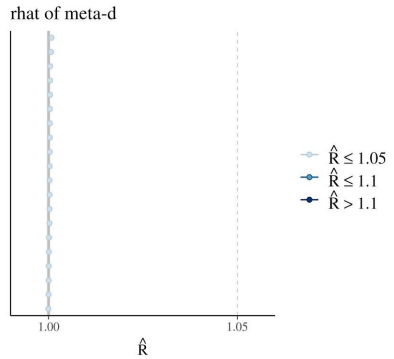


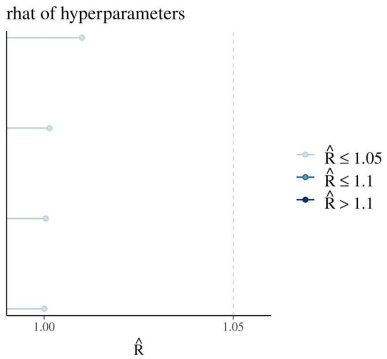
Estimation of c2S2 in parameter expanded model is poor as well. Most values are near 0.
I tried to figure out why, but even after multiple-check on my code, I couldn’t solve the problem.

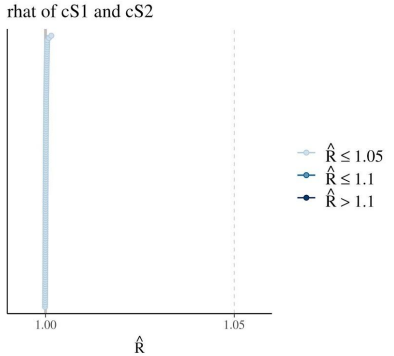
Results_Hierarchical_version 1 & 2

Rhat

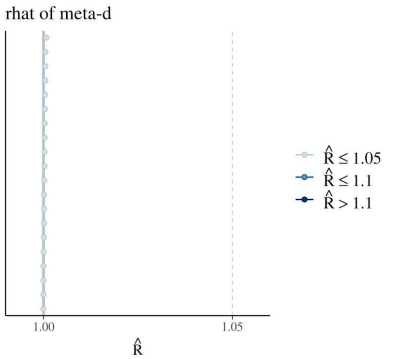
Normal

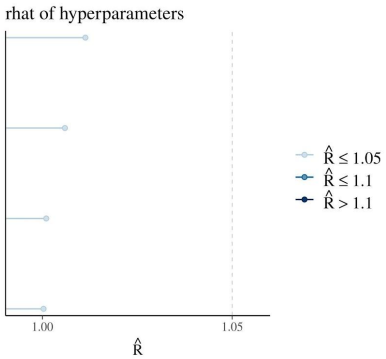


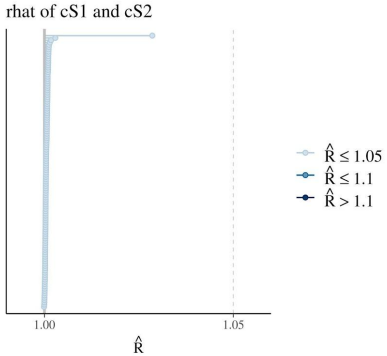




Parameter expansion



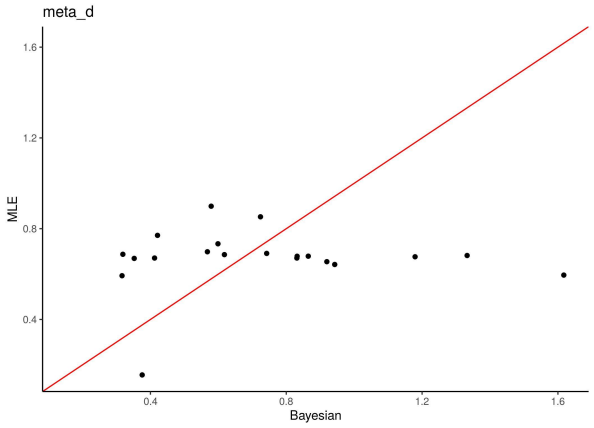


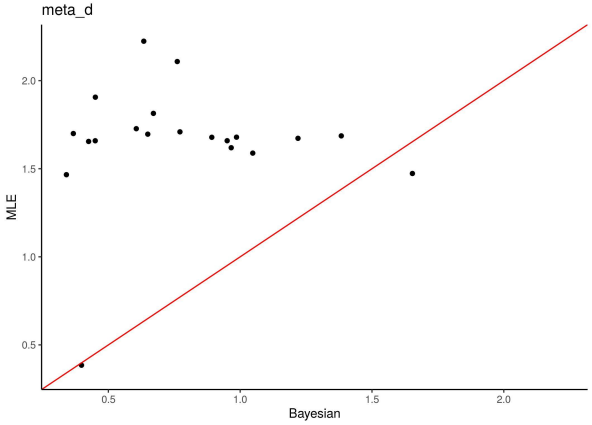


Results_Hierarchical_version 1 & 2

Comparison with MLE (meta-d)

Scales of x-axes and y-axes are different in both figures





Normal

Parameter expansion

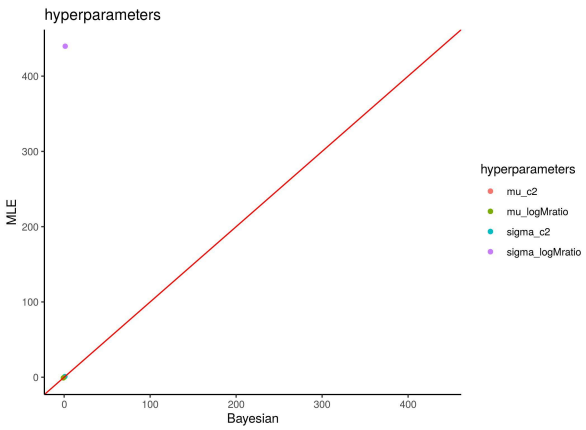
Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**

Overall, MLE results of meta-d is poor. Estimates of meta-d for all subjects are almost same.

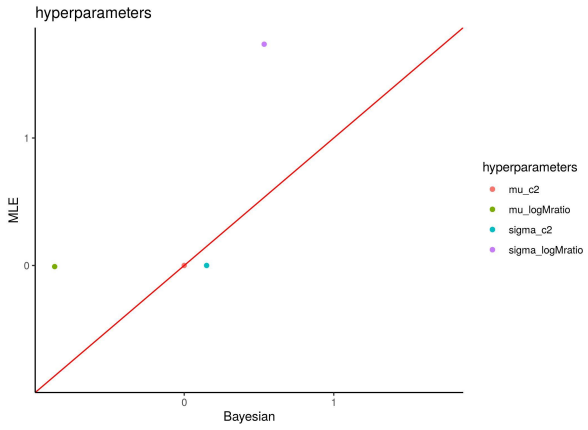
Results_Hierarchical_version 1 & 2

Comparison with MLE (hyperparameters)

Scales of x-axes and y-axes are different in both figures



Normal



Parameter expansion

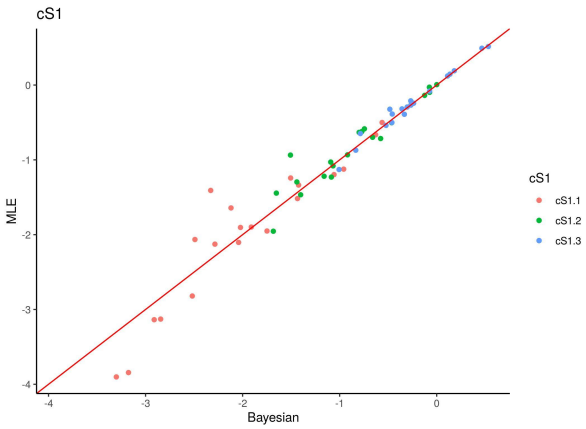
Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**

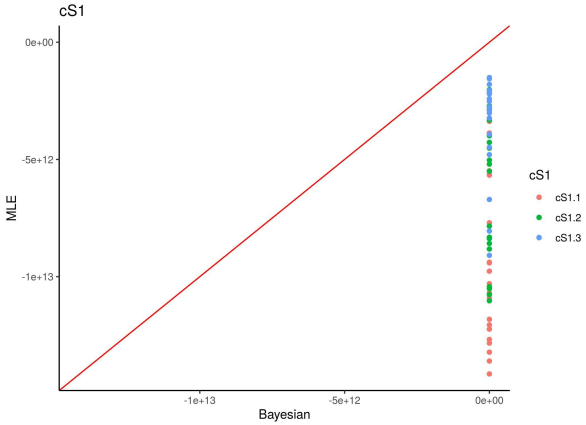
MLE results of hyperparameter is poor as well. Some parameters are abnormally big.

Results_Hierarchical_version 1 & 2

Comparison with MLE (cS1)

Scales of x-axes and y-axes are different in both figures





Normal

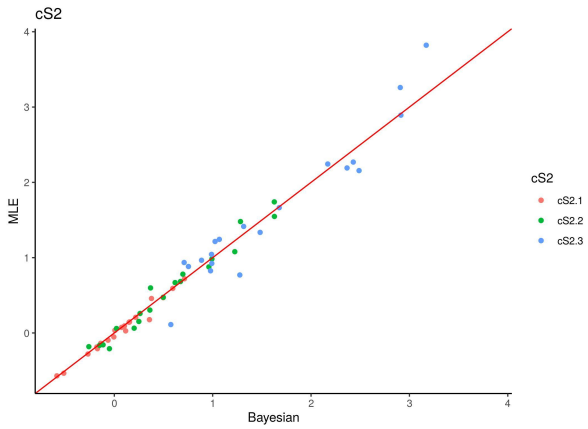
Parameter expansion

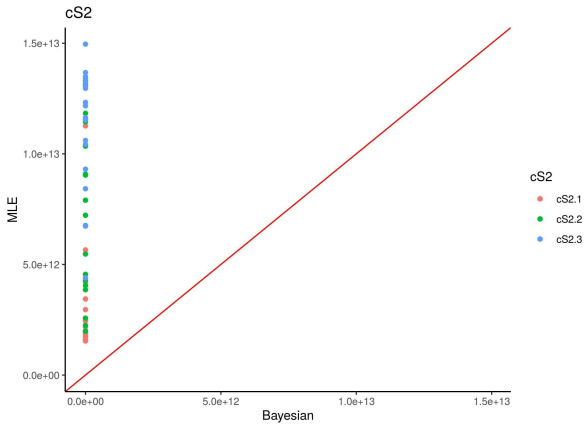
Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**
In normal model, MLE results and Bayesian estimates are similar, while Bayesian estimates for c2S1 were not good. Most values of Bayesian estimation are near zero.

Results_Hierarchical_version 1 & 2

Comparison with MLE (cS2)

Scales of x-axes and y-axes are different in both figures



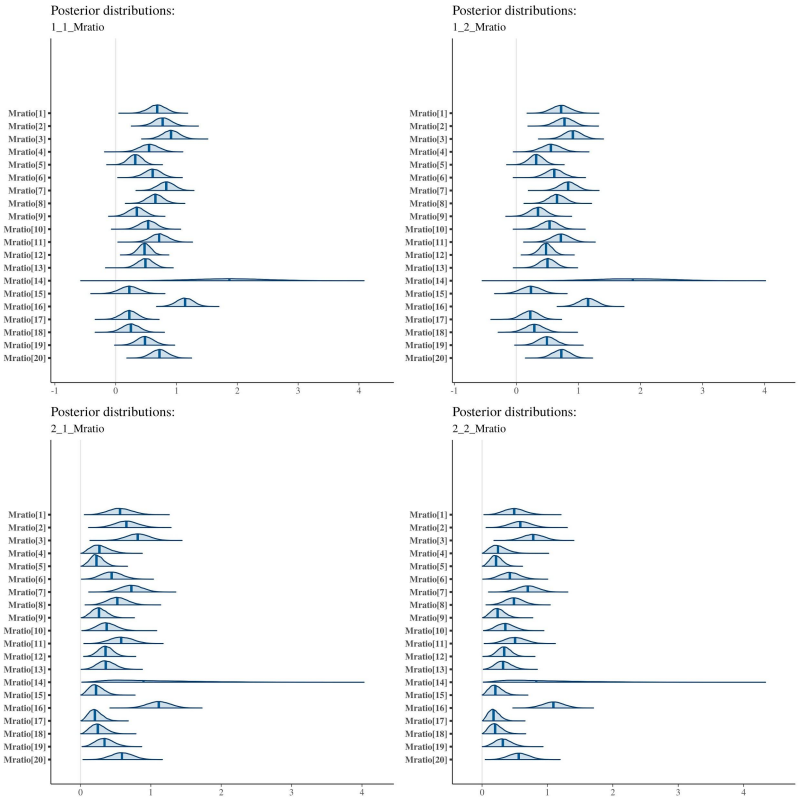


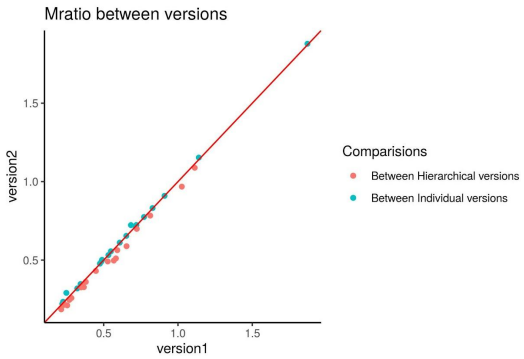
Normal

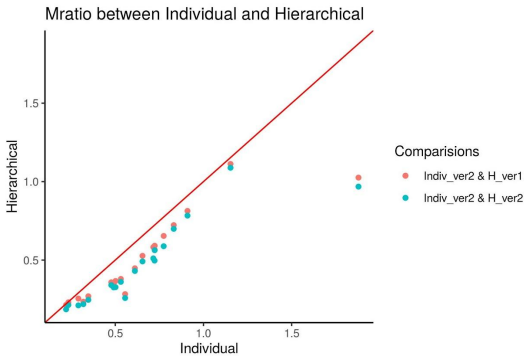
Parameter expansion

Used ‘optimizing()’ function to run MLE with stan code. Applied **same settings.**
In normal model, MLE results and Bayesian estimates are similar, while Bayesian estimates for c2S2 were not good. Most values of Bayesian estimation are near zero.

Additional Results (Individual vs hierarchical)





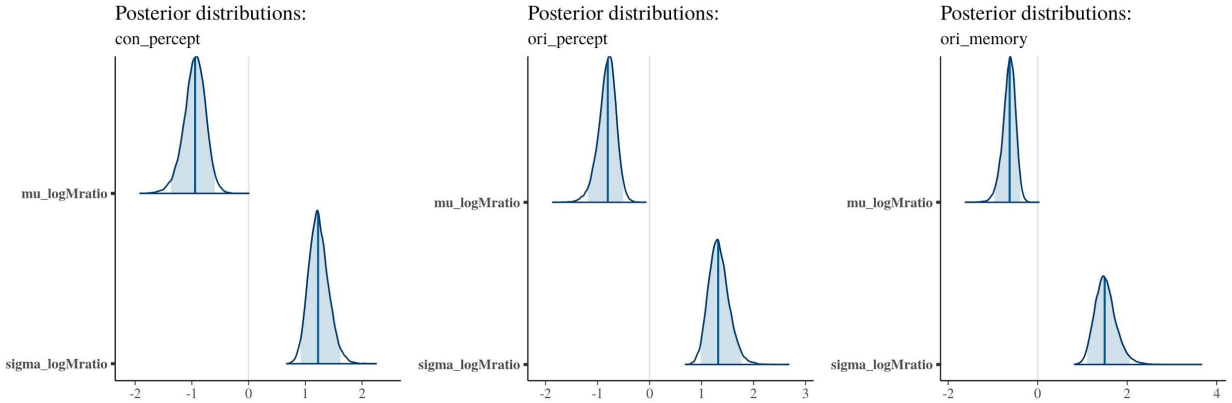


Comparison of posterior distributions (left) and posterior means (right) of Mratio estimates among different versions of codes.

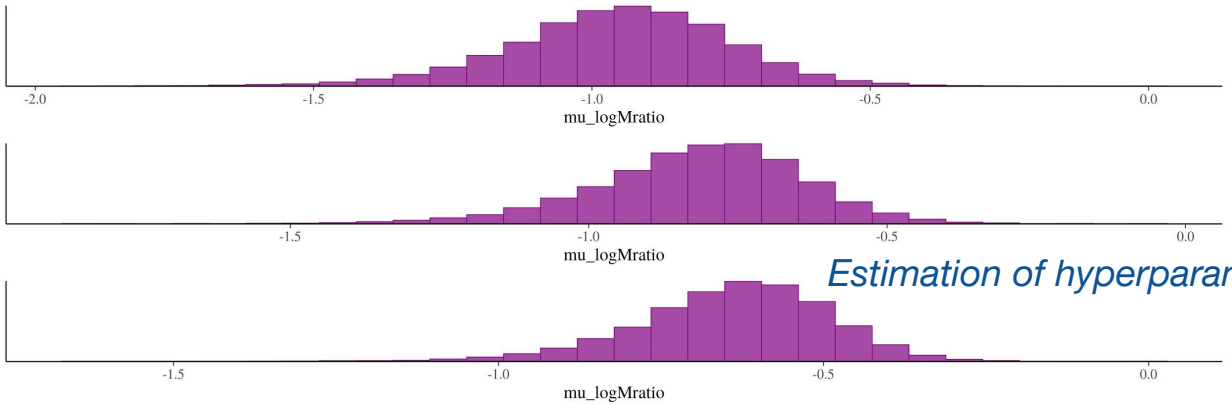
- Posterior means of Mratio are similar between two versions.

-Posterior means of Mratio are different between individual and hierarchical estimation.

Additional Results (with 3 tasks)

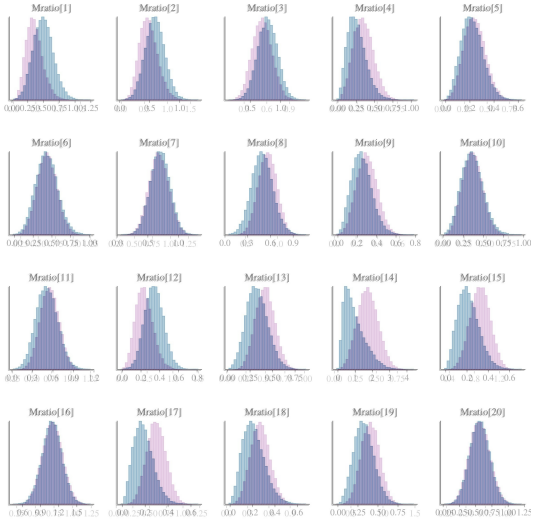


The original paper showed a strong positive correlation between orientation perception and orientation visual short term memory metacognition and no correlation between other tasks.
So I visually checked the results of all 3 tasks, though I didn't do statistical correlation analysis among them.

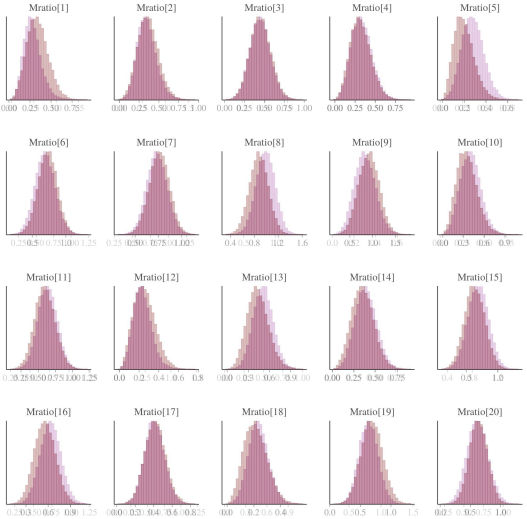


Estimation of hyperparameter looks similar, not that different

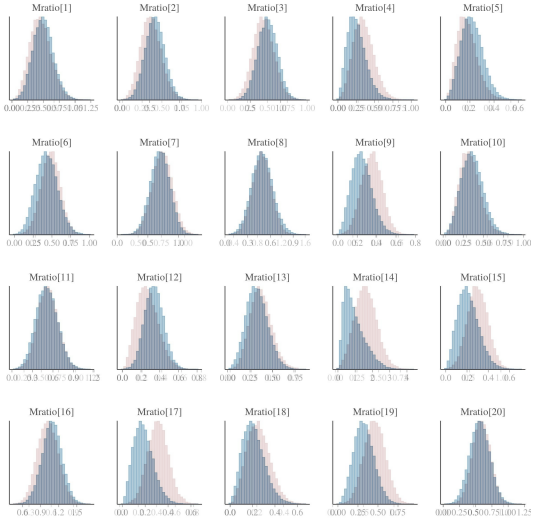
Additional Results (with 3 tasks)



Contrast perception
Orientation memory



Orientation perception
Orientation memory



Contrast perception
Orientation perception

Distributions of Mratio of individuals look similar!
(these distributions are from hierarchical Bayesian estimation without parameter expansion)

Conclusion

Conclusion

- Individual Bayesian estimation
  - Version 1 and Version 2 showed similar output, though the solution was different.
  - However, MLE results were quite different from each other.
  - It seems Version 2 works better with MLE function in Stan.
- Hierarchical Bayesian estimation
  - In Fleming 2017, parameter expansion helped sampling, but in my project, it worsened sampling and overall results (e.g. estimation of cS1 and cS2). → Couldn't figure out why..
  - In the model with parameter expansion, estimation of meta-d was good but only estimation of cS1 and cS2 was bad.
  - My Stan codes don't work well with MLE function, especially when estimating meta-d.

Reference

- Samaha, J., & Postle, B. R. (2017). Correlated individual differences suggest a common mechanism underlying metacognition in visual perception and visual short-term memory. *Proceedings of the Royal Society B: Biological Sciences*, *284*(1867), 20172035.
- Lee, M. D. & Wagenmakers, E.-J. (2014). *Bayesian Cognitive Modeling: A Practical Course*. Cambridge University Press.
- Maniscalco, B., & Lau, H. (2014). Signal detection theory analysis of type 1 and type 2 data: meta-d′, response-specific meta-d′, and the unequal variance SDT model. In *The cognitive neuroscience of metacognition* (pp. 25-66). Springer, Berlin, Heidelberg.
- Fleming, S. M. (2017). HMeta-d: hierarchical Bayesian estimation of metacognitive efficiency from confidence ratings. *Neuroscience of consciousness*, *2017*(1), nix007.