Assignment 2 - The Matching Pennies game

- **Describe the models you are working on (you can re-use text from assignment 1, if relevant, it is no plagiarism!)**

  Here we're implementing the Rescorla Wagner model with differing learning rates for winning and losing. This means that as agents play against each other, both start with a bias towards answering 1 on the first trial, after this initial trial they update their expected utility E(x) based on the following rule:

$$E(t|t>0) = \begin{cases} E_{t-1} + \alpha_W * (C_{t-1} - E_{t-1}) & fb_{t-1} = 1 \\ E_{t-1} + \alpha_L * (C_{t-1} - E_{t-1}) & fb_{t-1} = 0 \end{cases}$$

  where t is the trial, E is the expected value of the agent, W is the learning rate when winning, L is the learning rate when losing, C is the choice of the opponent, and fb is the feedback aka. winning or losing.
  Both of our agents follow this learning rule, however as it is opponent two's job to mismatch the pennies he does not respond based on his expected value E but based on the complement probability i.e. 1-E(t).

- **showcase a commented version of the stan model (what does each line do?)**

```
//defining the data block:
data {
  //number of trials
  int<lower=1> n;
  //choices for the first agent
  array[n] int rw1;
  //choices for the second agent
  array[n] int rw2;
  //feedback the first agent got
  array[n] int fb_rw1;
  //feedback the second agent got
  array[n] int fb_rw2;
  #whether to only sample from the priors or not
  //0 = no prior,
  //1 = prior only
  int <lower = 0, upper = 1> prior;

}

// The parameters of the model
parameters {

  //iases for the first trial the agents play
  real  bias_1;
  real  bias_2;
  //learning rates for winning and losing for the first agent
  real alpha_1w;
  real alpha_1l;
  //learning rates for winning and losing for the second agent
  real alpha_2w;
  real alpha_2l;
}


//here we calculate the expected values.
transformed parameters{
  // we start by defininf an array of beliefs i.e. expected values for each agent
  array[n] real <lower = 0, upper = 1> belief_1;
  array[n] real <lower = 0, upper = 1> belief_2;

  //the first trial's belief is just going to be equal to the bias in probability space
  //as we put a normal distribution with a mean and standard deviation for the prior for the bias' we need to constrain it to probability space
  //by using the inverse logit. The same logic goes for the other parameters (learning rates)
  belief_1[1] = inv_logit(bias_1);
  belief_2[1] = inv_logit(bias_2);

  //now we loop through all the trials where the learning rule is applied
  for (i in 2:n){
    //when agent1 wins use this
    if(fb_rw1[i-1])
      belief_1[i] = belief_1[i-1]+inv_logit(alpha_1w)*(rw2[i-1]-belief_1[i-1]);
      //if not then he lost so:
    else
      belief_1[i] = belief_1[i-1]+inv_logit(alpha_1l)*(rw2[i-1]-belief_1[i-1]);
      //same goes for agent 2.
    if(fb_rw2[i-1])
```

```
  //now we loop through all the trials where the learning rule is applied
  for (i in 2:n){
    //when agent1 wins use this
    if(fb_rw1[i-1])
      belief_1[i] = belief_1[i-1]+inv_logit(alpha_1w)*(rw2[i-1]-belief_1[i-1]);
      //if not then he lost so:
    else
      belief_1[i] = belief_1[i-1]+inv_logit(alpha_1l)*(rw2[i-1]-belief_1[i-1]);
      //same goes for agent 2.
    if(fb_rw2[i-1])
      belief_2[i] = belief_2[i-1]+inv_logit(alpha_2w)*(rw1[i-1]-belief_2[i-1]);
    else
      belief_2[i] = belief_2[i-1]+inv_logit(alpha_2l)*(rw1[i-1]-belief_2[i-1]);
  }
}

// The model to be estimated. here we defined our priors and likelihood
model {

  //priors: weakly informative priors N(0,1) for all parameters as a start
  target += normal_lpdf(bias_1 | 0,1);
  target += normal_lpdf(bias_2 | 0,1);

  target +=normal_lpdf(alpha_1w | 0,1);
  target +=normal_lpdf(alpha_1l | 0,1);

  target +=normal_lpdf(alpha_2w | 0,1);
  target +=normal_lpdf(alpha_2l | 0,1);

  //sample from the posterior (aka. prior == 0) or just sample from the prior?
  if(prior == 0){
  //likelihood agents beliefs are translated into choices
  //note that agent two uses the complement belief / expected value to guide his choice as he tries to mismatch:
    for (i in 1:n){
      target +=bernoulli_lpmf(rw1[i] | belief_1[i]);
      target +=bernoulli_lpmf(rw2[i] | (1-belief_2[i]));
    }
  }

}
```

```
88
89  // here we do some transformations and prior / posterior predictive checks:
90  generated quantities{
91
92
93    #convinient way to get the parameters in their native (0-1 space)
94    real <lower = 0, upper = 1> theta1_prior = inv_logit(bias_1);
95    real <lower = 0, upper = 1> theta2_prior = inv_logit(bias_2);
96
97    real <lower = 0, upper = 1> alpha1l_prior = inv_logit(alpha_1l);
98    real <lower = 0, upper = 1> alpha1w_prior = inv_logit(alpha_1w);
99
100   real <lower = 0, upper = 1> alpha2l_prior = inv_logit(alpha_2l);
101   real <lower = 0, upper = 1> alpha2w_prior = inv_logit(alpha_2w);
102
103
104   // we want to store the posterior / prior simulations for both agents
105   array[n] int sim_rw1t;
106   array[n] int sim_rw2t;
107
108   //now we simulate reponses of the agents as in the model block for each trial i.
109   for (i in 1:n){
110     sim_rw1t[i] = bernoulli_rng(belief_1[i]);
111     sim_rw2t[i] = bernoulli_rng(1-belief_2[i]);
112   }
113   //now we sum all the responses such that we have both all the trial level responses but also how many times each individual answered 1 and
114   //therefore also 0
115   int sim_rw1 = sum(sim_rw1t);
116   int sim_rw2 = sum(sim_rw2t);
117
118 }
```

- **describe a process of parameter recovery (why are you doing it?, how are you doing it?)**

A process of parameter recovery involves simulating agents with known parameter values, here the bias and the two learning rates for each agent. We then invert the problem by feeding the data that these parameters produced (choice and feedback) to the stan model that then gives us the probability distributions for the parameters. We can then investigate how well the parameters given by stan correspond to the known parameters that we simulated the responses from.

One would like to perform parameter recovery for two main reasons:
Firstly, parameter recovery ensures that the model at hand is both sensible and without coding errors. That the model is sensible here means that if we are actually capturing the underlying process of what we are trying to model, then we will be able to pick up on that, i.e. recover the parameters, if this is not the case then when we fit the model to actual data we cannot be sure that the parameters we receive are the “true” ones and therefore meaningful.
Secondly parameter recovery also serves as a way to estimate how many trials and or participants in a hierarchical model is needed to get good / precise estimates of the parameters. Running an experiment with 500 trials with two random bias agents, is unnecessarily resource heavy as the parameters (the bias’), can be estimated with good precision with way fewer trials, however with very few trials parameter recovery will give very uncertain estimates. Therefore even after confirming that the model is

sensible, i.e. no coding errors and parameters that can be recovered, parameter recovery serves as a form of power analysis to help determine the sample / trial size.

- **discuss the results: how many trials should be used at least to properly recover the parameters? What's the role of priors? Add relevant plot(s).**

Simulations:

Simulation for parameter recovery started with simulating on the single game for model quality checks, including "prior - posterior update check", in other words, what has the model learnt after fitting the data, and predictive checks on the agents' choices, i.e. how many trials the agents would choose 1 out of all trials (in this case 500 trails).

The actual matching pennies game in question has 30 trials per game. To test the minimum amount of trials that can be properly recovered, we started the simulation along 15 trials to 60 trials. Based on the result that was not promising, a broader range of trial amounts was tested, namely, from 2 trials to 500 trials, with different trial intervals, i.e. 2 - 10 trials, 100 - 110 trials, 200 - 210 trials, …, 500 - 510 trials.

Results:

Figure 1 - 3 below show the results of model quality checks on a single game with 500 trials, where the parameters in question are:

a) theta1, theta2 as the original biases of the matcher and non-matcher, and both sampled ;

b) alpha1l, alpha1w standing for the learning rates towards lose and win of the matcher respectively

c) similarly alpha2l, alpha2w as the two learning rates of the non-matcher.

And the prior for all the parameters are normal distribution with mean $= 0$ and standard deviation (sd) $= 1$ (i.e. $N(0,1)$).

Figure 4 - 10 presents the parameter recovery for alpha1l, alpha1w, alpha2l, and alpha2w with different trials, with 5 games simulated parallelly. The true values in the simulation are: alpha1l = 0.3, alpha1w = 0.8, alpha2l = 0.9, alpha2w = 0.1 as is evident by the horizontal lines in these plots.

Figure 10-14 shows our sensitivity analysis for ~400 and ~200 trials

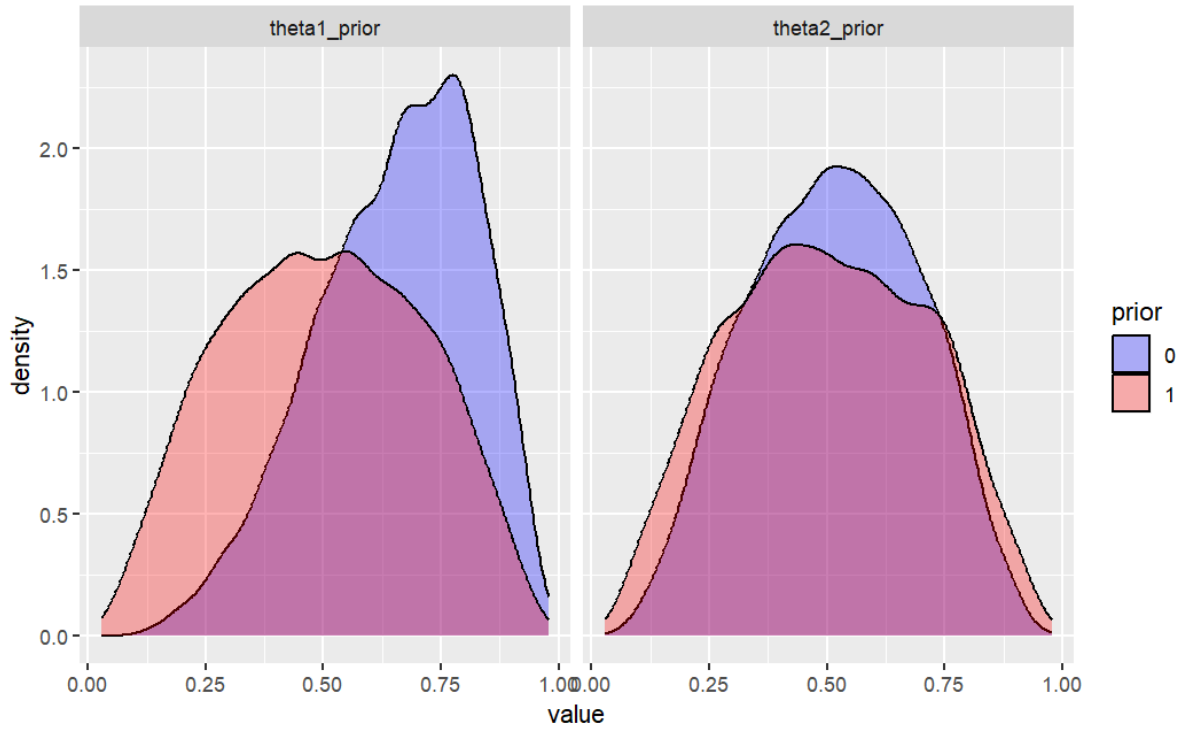


Figure 1 *Parameter recovery for theta1 of the matcher (left) and theta2 of the non-matcher (right), where the red presents the prior, while blue is the posterior. The true value of theta1 and theta2 is both 0.5.*

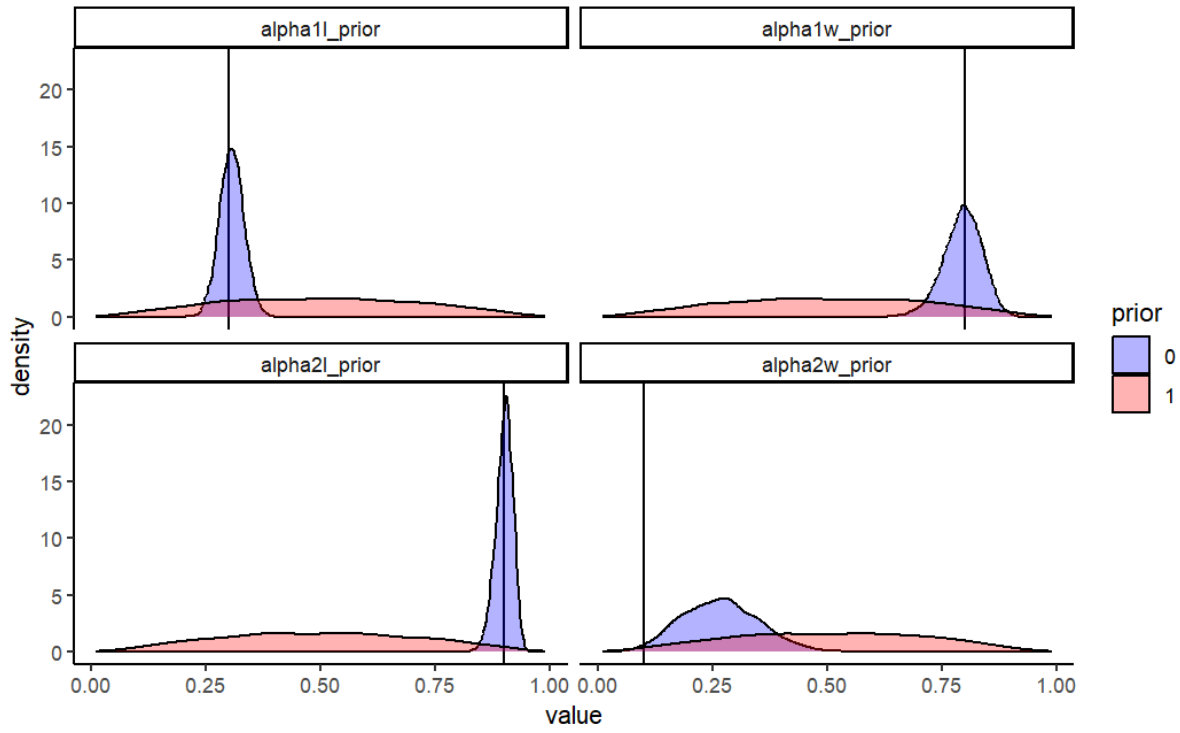


Figure 2 *Parameter recovery for alpha1l, alpha1w, alpha2l, and alpha2w, where the red presents the prior, the blue is for posterior, and the vertical black line stands for the true value for each parameter.*

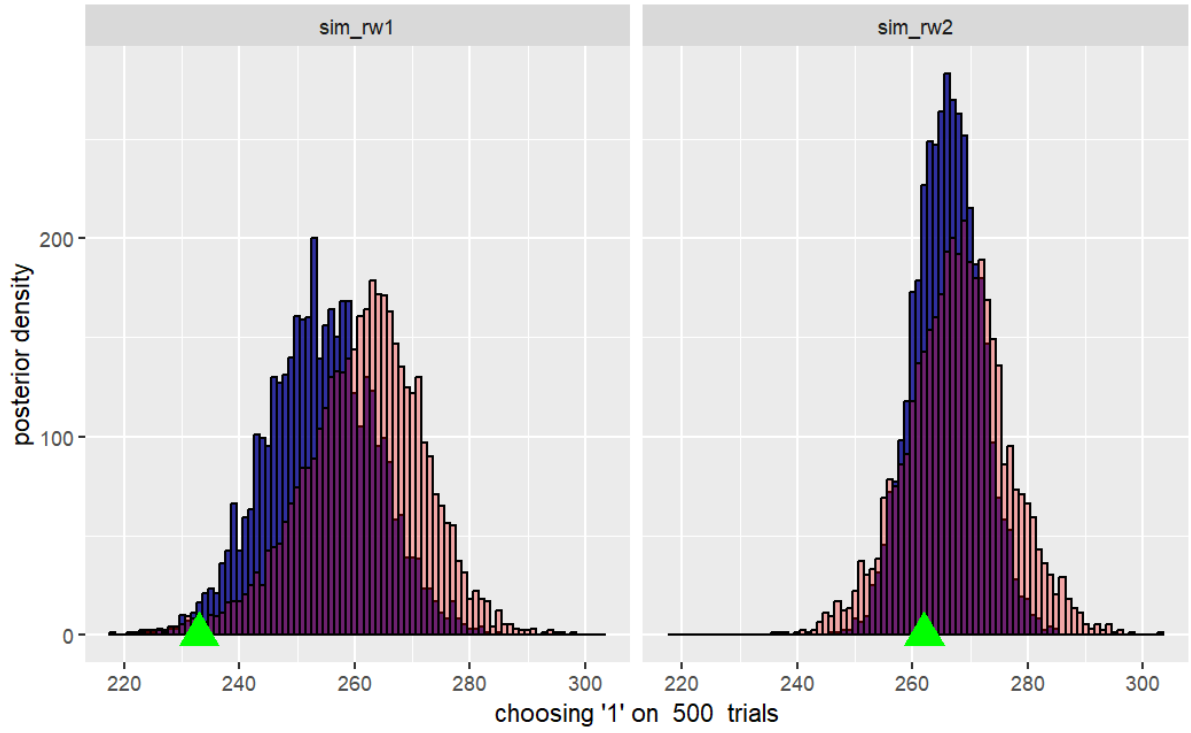


Figure 3 *Predictive checks on how many trails out of 500 the matcher rw1 agent (left)*

and the non-matcher rw2 agent (right) would choose 1. The green triangle points the actual choice of each agent.

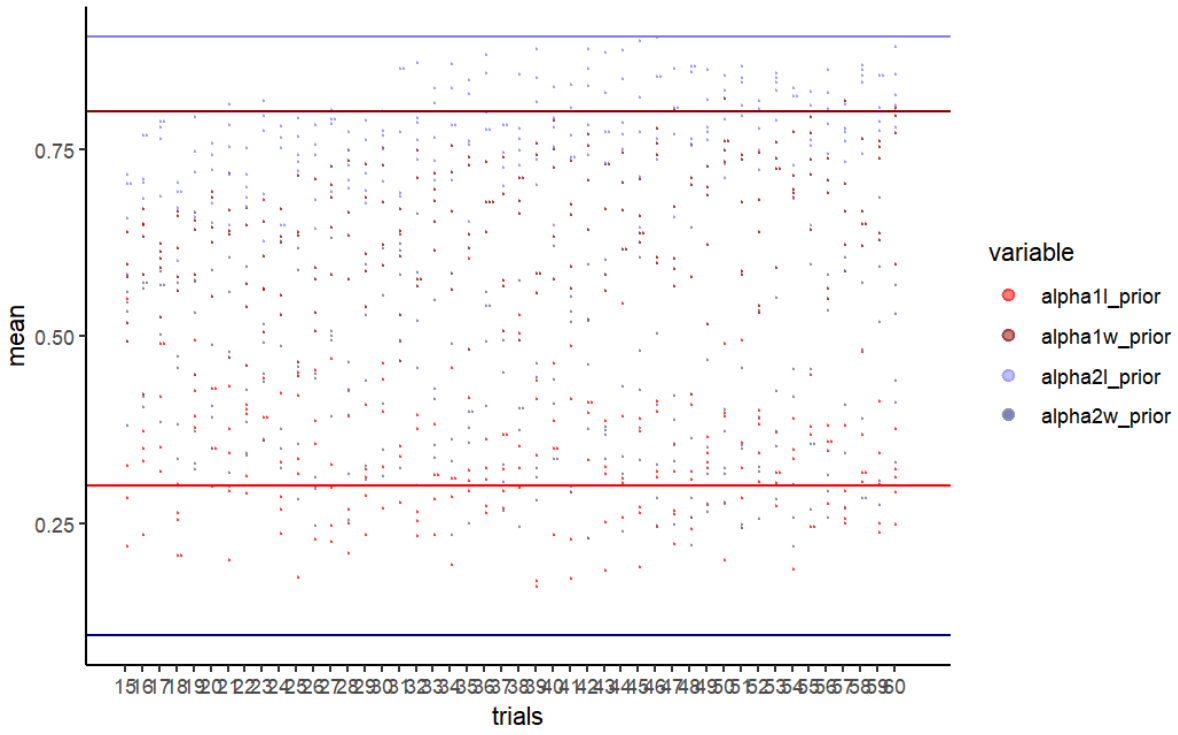


Figure 4 *Parameter recovery with 15 - 60 trials.*

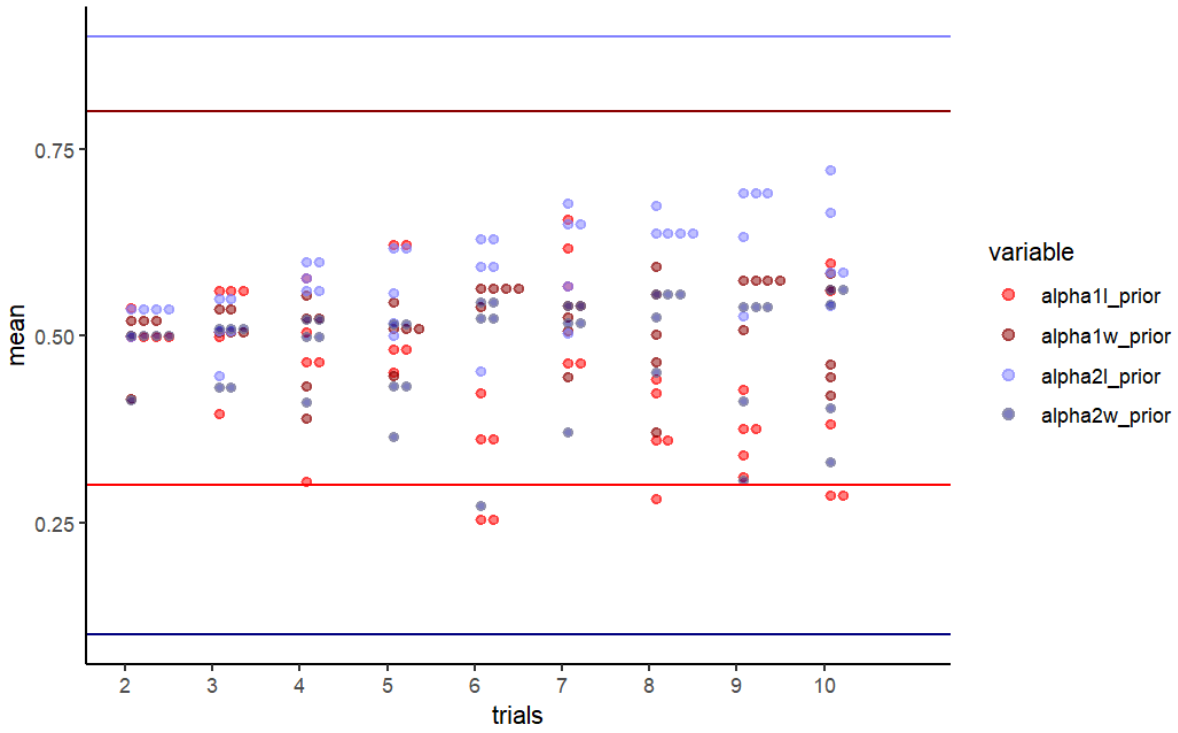


Figure 5 *Parameter recovery with 2 - 10 trials.*

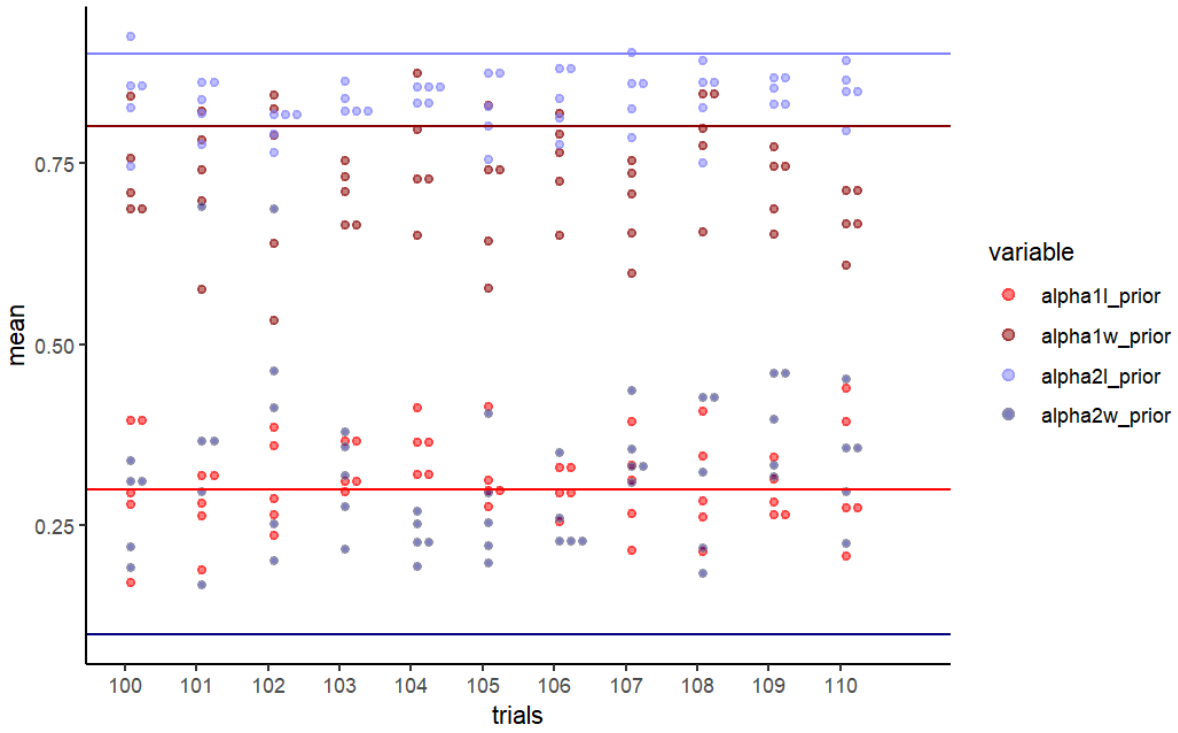


Figure 6 *Parameter recovery with 100 - 110 trials.*

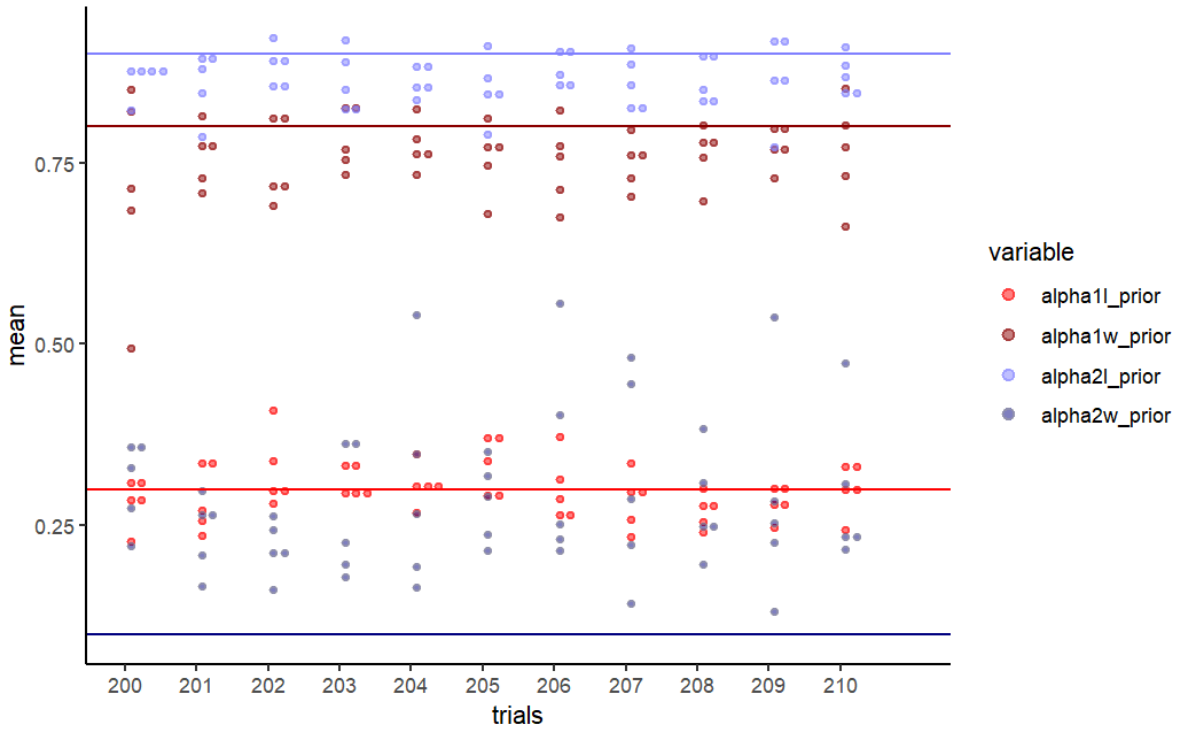


Figure 7 *Parameter recovery with 200 - 210 trials.*

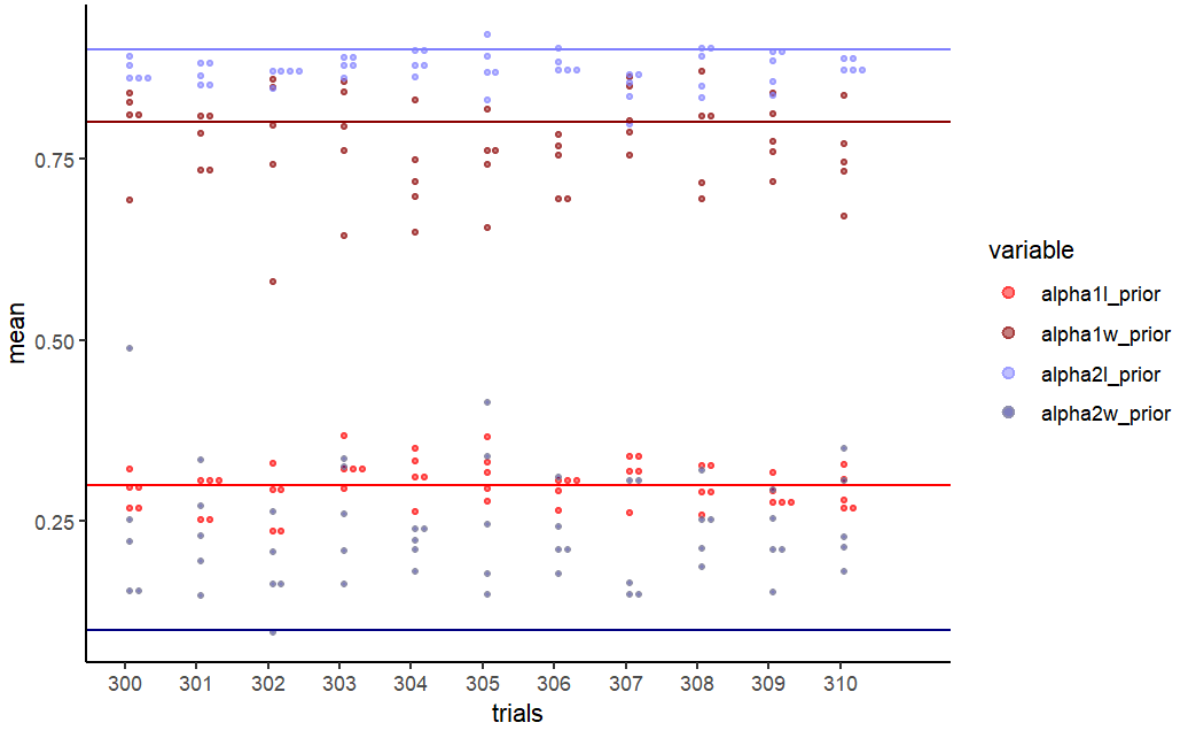


Figure 8 *Parameter recovery with 300 - 310 trials.*

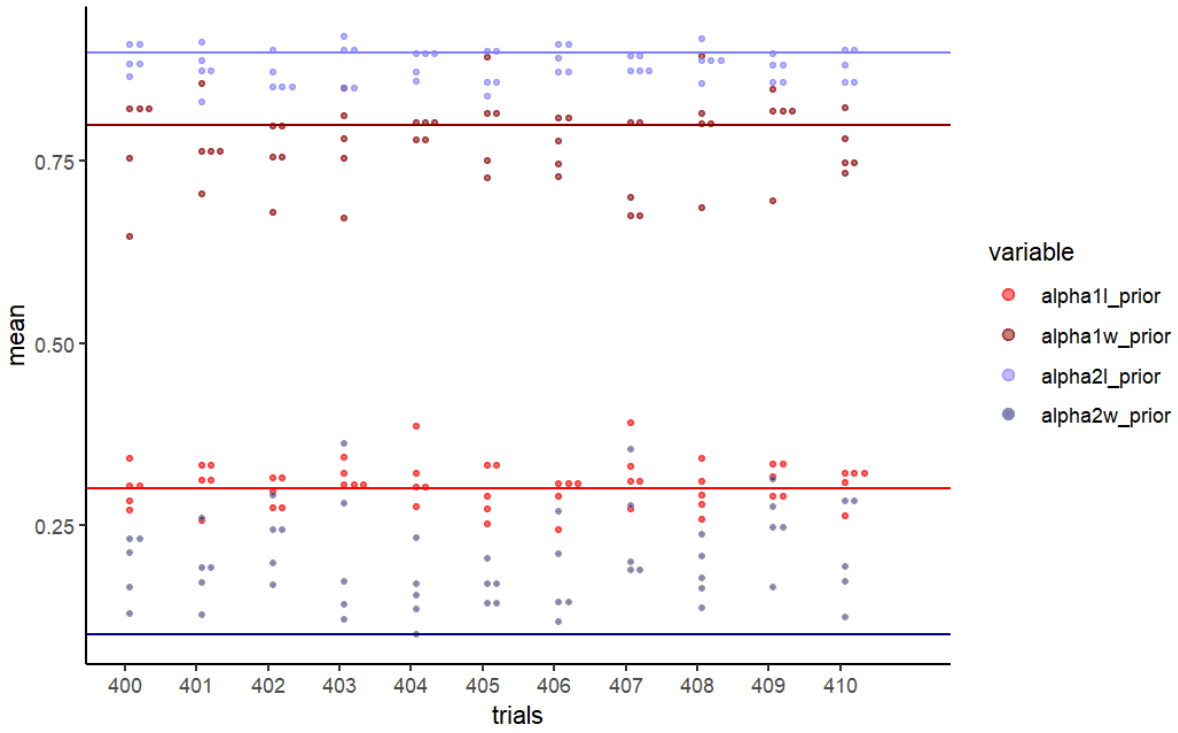


Figure 9 *Parameter recovery with 400 - 410 trials.*

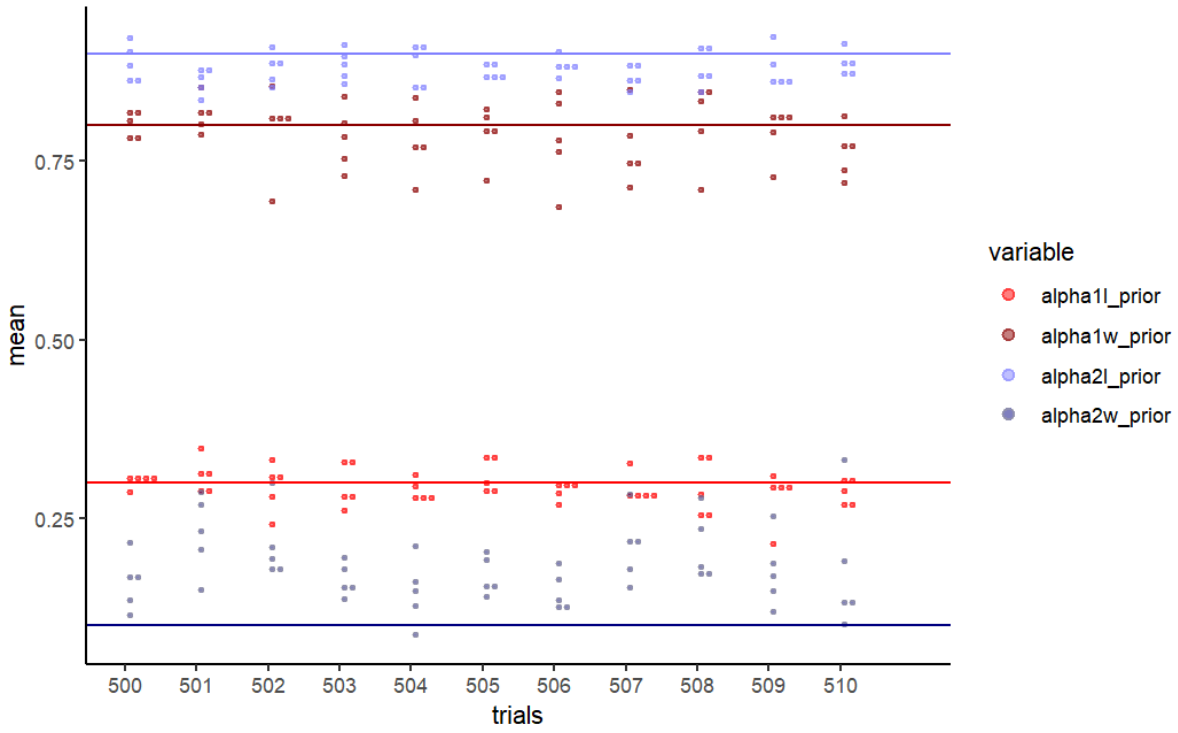


Figure 10 *Parameter recovery with 500 - 510 trials.*

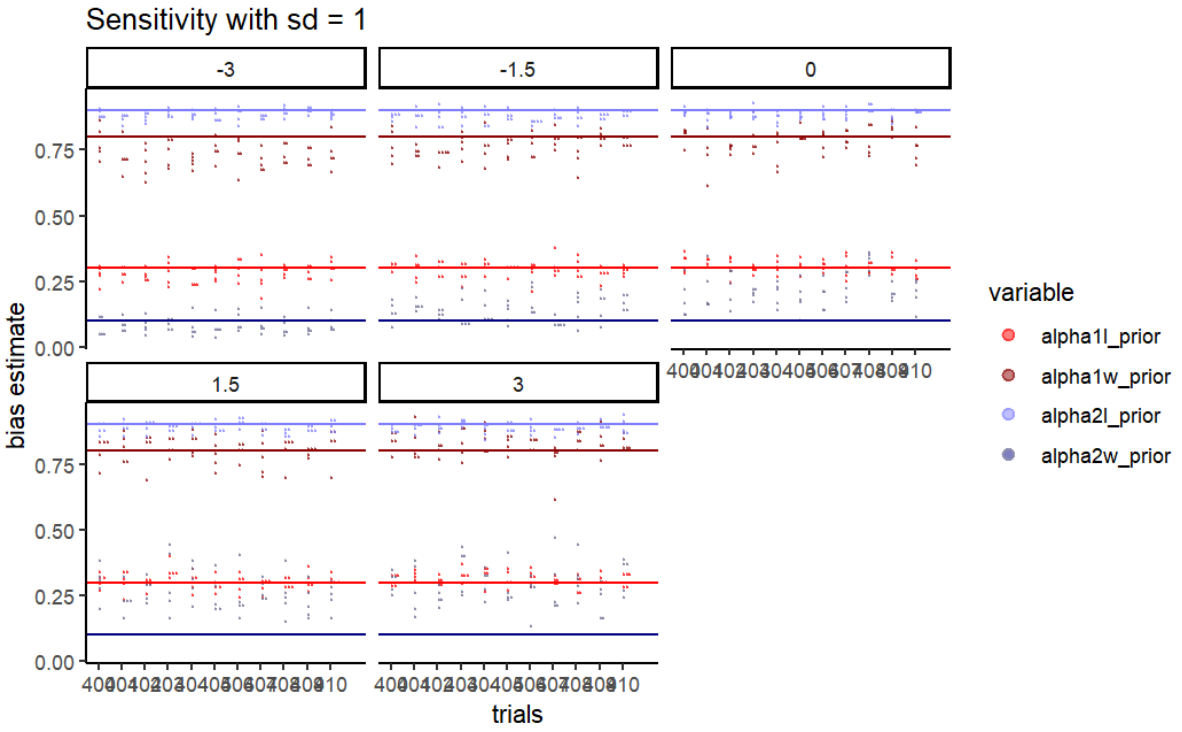


Figure 11 *Prior sensitivity check recovery with 400 - 410 trials, mean value between -3 to 3, sd = 1*

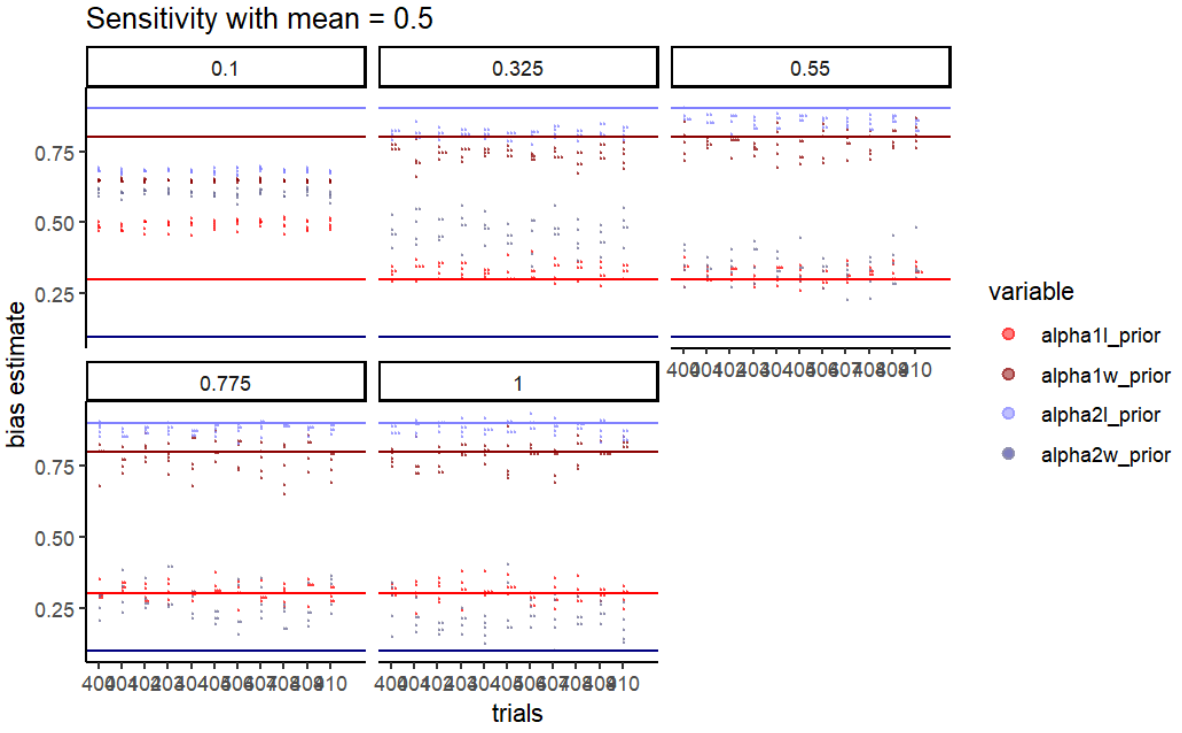


Figure 12 *Prior sensitivity check recovery with 400 - 410 trials, mean = 0.5, sd value between 0.1 to 1.*

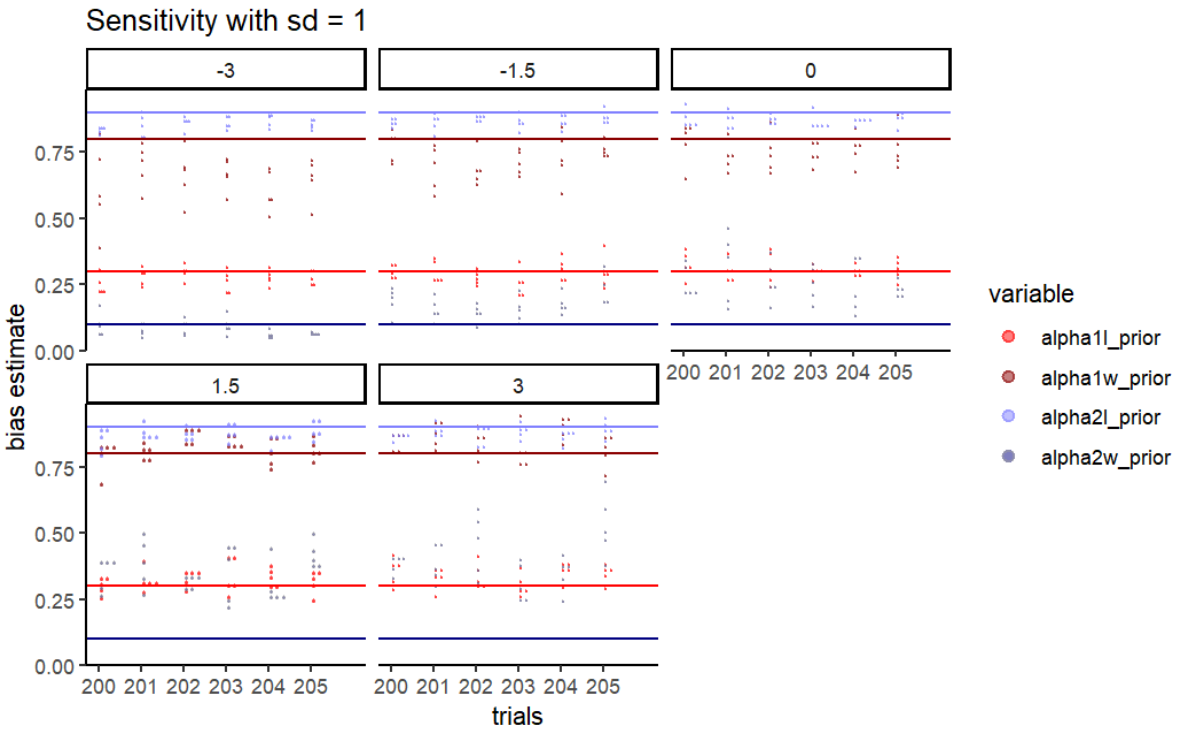


Figure 13 *Prior sensitivity check recovery with 200 - 210 trials, mean value between -3 to 3, sd = 1*

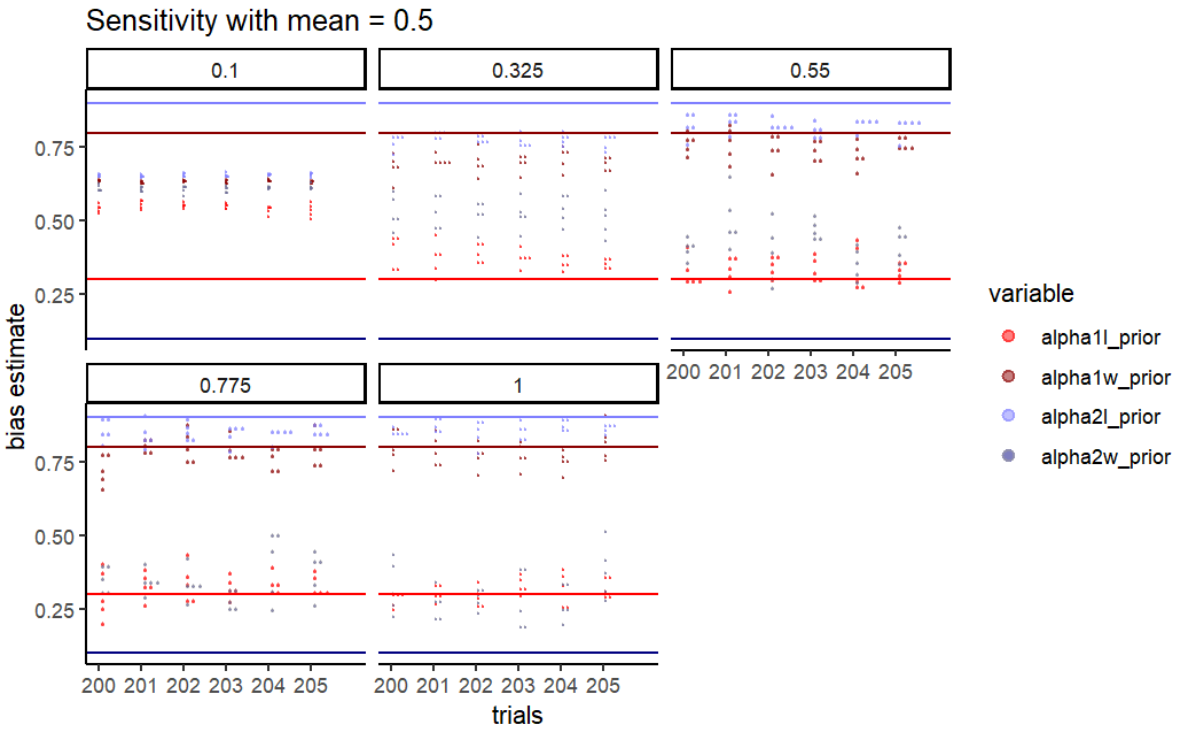


Figure 14 *Prior sensitivity check recovery with 200 - 205 trials, mean = 0.5, sd value between 0.1 to 1.*

Discussion:

The model quality checks with one single game imply that the model can learn after seeing the data and provide proper inferred values for all the parameters in question, within a reasonable estimation error. And the result on choice predictions shows that the model is also capable of properly predicting the agent's choice given a prior.

However, the following results on parameter recovery show that the performance is to a large extent relying on the amount of trials of the game, in another word, as the number of trials increases, the model gives more accurate inferred parameter values. This may due to the fact that the learning rate for one agent differs on losing and winning, it requires more trials for the model to capture the pattern therefore to differentiate them. It's also noticeable that for alpha1l = 0.3, alpha1w = 0.8, alpha2l = 0.9, the model needs at least 200 trials to give a proper recovery, while for alpha2w = 0.1, it takes at least 400 trials. It may be caused by the big gap between alpha2l and alpha2w. The high alpha2l makes the agent learn fast from the present loss then adjust the following

choice quickly, as a contrast, if it wins, a rather low alpha2w doesn't make as much effect on the next choice because it updates the belief much more slowly. So in general, when the agent has two learning rates, a high learning rate can make a more salient influence on the agent's choice which can be sooner reflected by the agent's choice while the much lower learning rate requires more trials to reveal the pattern. This may explain why only alpha2w requires more trials to estimate.

Besed on the results from parameter recovery, the prior sensitivity check was implemented on 400 trails level and 200 trials level, in order to examine not only how prior influences the performance but also if a better prior range can compensate for the fewer number of trials. As shown in Figure 11 and 12, when simulated on 400 trails and with a fixed sd = 1, the model achieves the optimal performance with mean = -3. As the mean value increases, it doesn't affect much on recovering alpha1l = 0.3, alpha1w = 0.8, alpha2l = 0.9, but the performance on recovering a rather low parameter i.e. alpha2w = 0.1 gets significantly decreased. When the mean value is fixed, in this case, mean = 0.5, the performance gets significantly improved as sd increases from 0.1 to 1. Results on 200 - 205 trails show the same pattern. In addition all parameters can be well recovered with 200 trials if the mean value is between -1.5 to -3, then the model fails to infer alpha2w if the mean value is higher than -1.5, which aligns with the results from the parameter recovery. It also demonstrates that when the prior is very conservative, meaning a rather low sd, i.e. sd = 1, the model fails to infer any parameter, where increasing trials doesn't make much difference. All in all, the results suggest that mean, sd and number of trials all matter to the model's performance. Therefore, it is recommended to set a quite low mean prior value say [-3 ; 0] and a rather wide standard deviation i.e. sd = [0.5 ; 1] to get all parameter well recovered, with relatively fewer number of trails, in this case, 200 trails would be enough.

Second model

For our second model we decided to use the hierarchical gaussian filter (HGF) which has been introduced in our other course. Here we decided to keep the other agent simple, i.e. a random bias agent with different biases in given blocks of trials. These experiments there closely resemble (they are mirrors) of the experiments from several papers investigating sensory associative learning. The HGF closely resembles the Rescorla Wagner model in that expectations on each trial are updates based on weighted prediction errors. However the HGF implements a dynamic learning rate that is dependent on the precision (inverse variance) that varies by trial. Here we specifically implement the 3-level HGF for binary outcomes. Here there are 3 parameters that have to be estimated, , and , which governs the belief trajectory and therefore the choice of the agent. The update equations are beyond the scope of this assignment, but can be found in the repository. The HGF is structured in hierarchies as the names suggest. In the commonly used 3-level HGF the first level is where the belief / expected values evolve in time. The second level governs the first level through a sigmoid (inverse logit) transformation and evolves in time as a gaussian random walk with a step size equal to exp(3t-1* +) where 3t-1 is the mean of the third level at time (t-1). The third level then also evolves in time as a gaussian random walk with a constant step size of . Therefore the parameters of the model are as follows as well as their restrictions:

is the one component of the step size of the second level. [-∞;∞]

is the coupling between the second and third level [0;∞]

is the step size of the third level [0;∞]

The restrictions of theta and kappa being strictly positive is that theta is the step size of the gaussian random walk of the third level. Secondly, kappa is strictly positive as the direction of influence of the third level is already incorporated in the mean of the third level and opposite influence does not make any sense. Previous literature as well as the original paper of Mathys et al 2011 has provided some suggestion to which range of values these parameters take. which is summarized below

[-12;0]

[0;3]

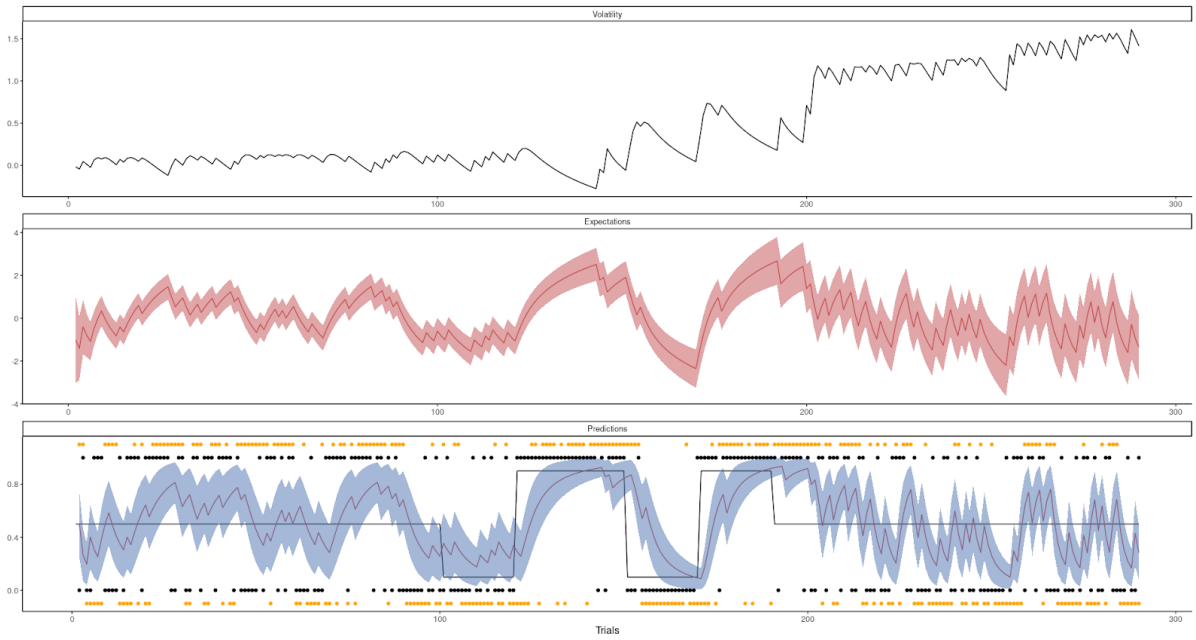
[0;1]

We therefore first simulate an agent with these parameters and see the output. For this demonstration the belief / expected value is run through a bernoulli distribution as the probability parameter of answering 1 to generate responses:

To make this agent resemble what was done in the original HGF paper we let this agent play against a random bias agent that switches bias as that paper (see solid black line below): Here we show an agent with parameters of:

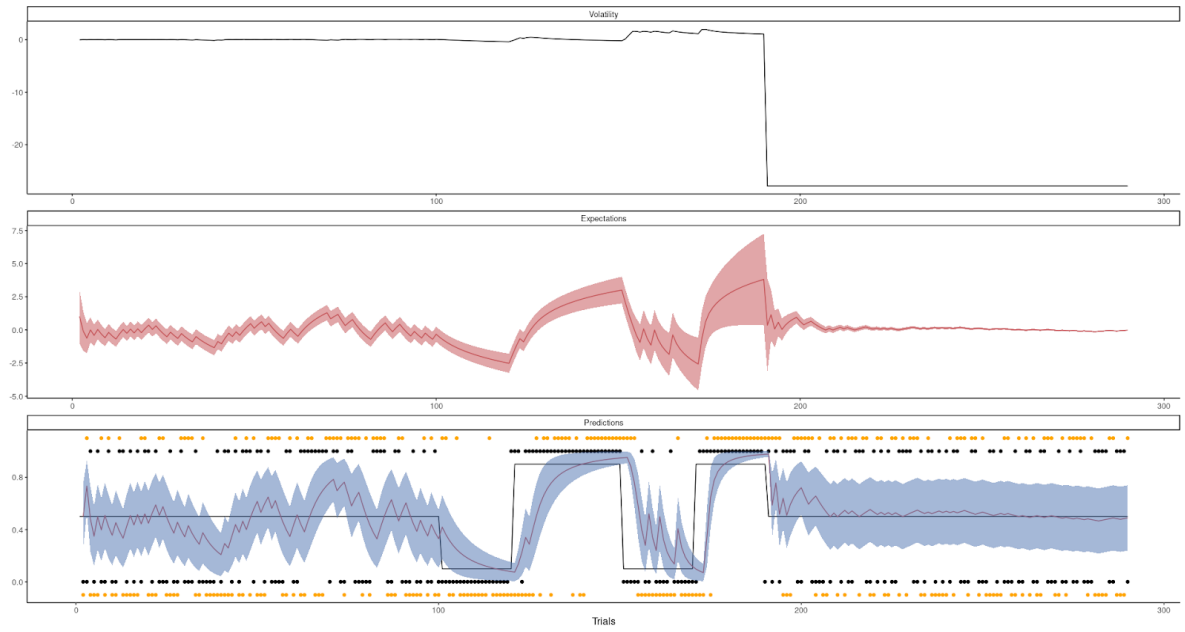
$= -2.5, \ = 1.4, \ = 0.5$

Closely resembling those presented in the 2011 HGF paper:

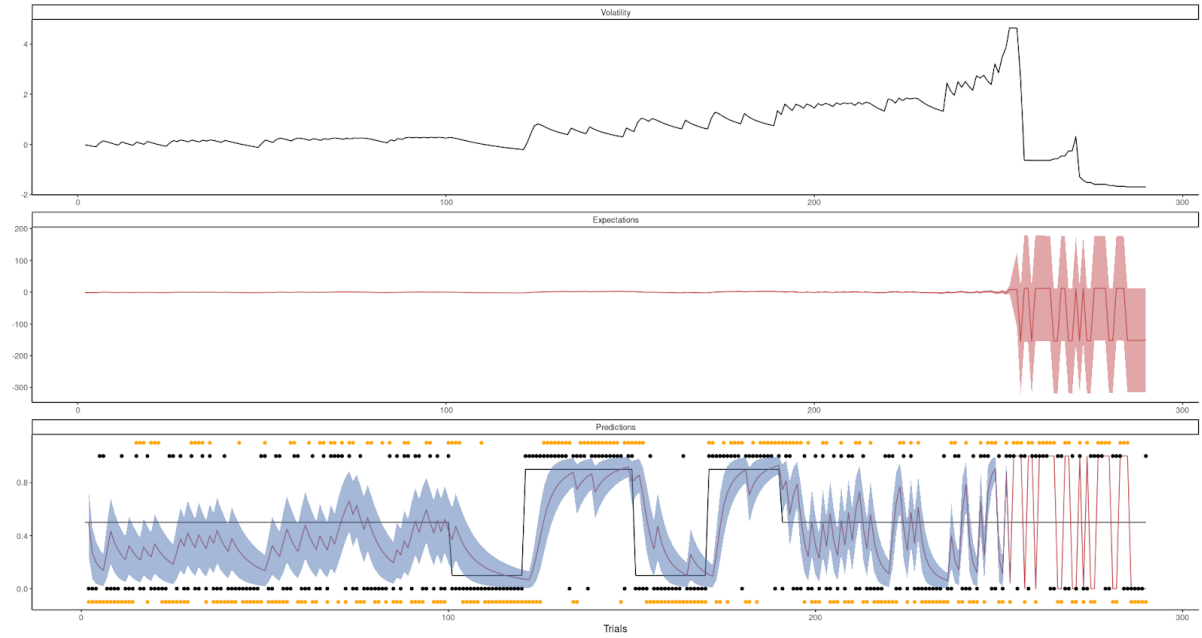


Here the lowest panel represents the belief of the agent i.e their probability of answering 1 with an uncertainty given by a Bernoulli distribution. The second level is the logit transformed first level and evolves in time as a gaussian random walk with a variance equal to exp(*3t-1+) where 3t-1 is the mean of the third level at time step t-1. This level has a constant variance of

However sometimes the model does not behave so nicely:



or:



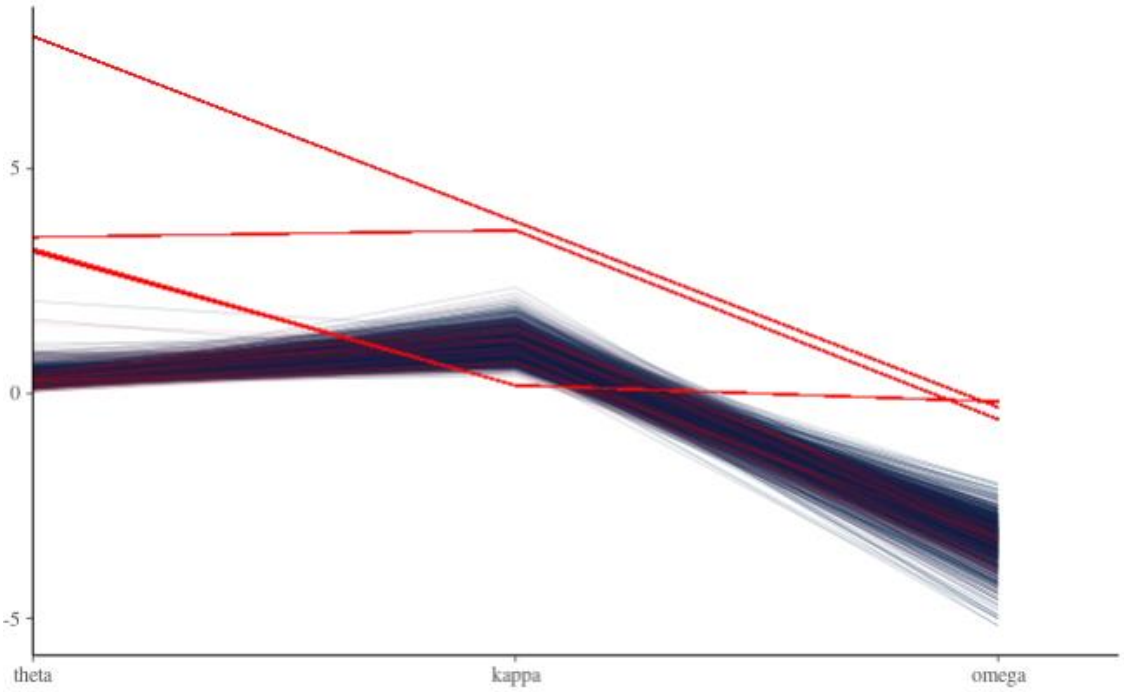
It therefore seems like the HGF implementation is highly sensitive to how the random bias behaves, meaning when the random bias agent picks the option that goes against his bias. We tried fitting this but even with quite informative priors we got a lot of divergent chains:

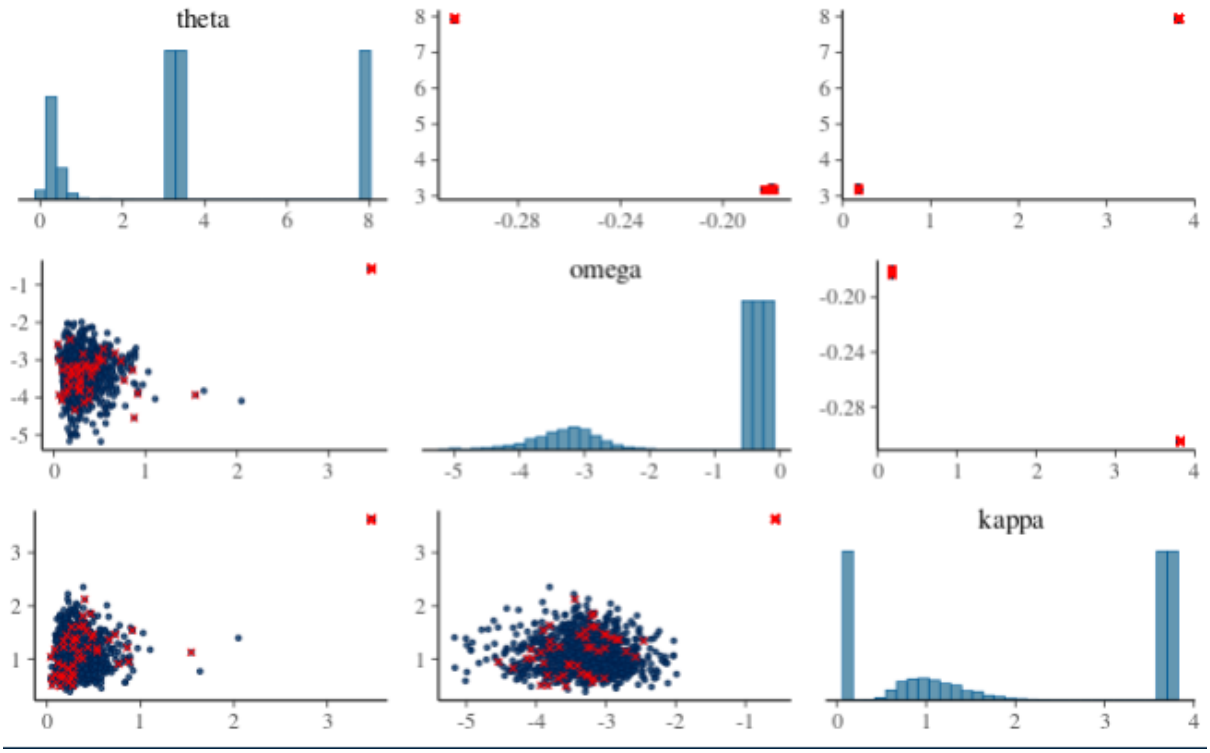
Here we show what happened when restricting the omega parameters to be below 0, however the pattern was the same if this restraint wasn't there.

~ Normal(-3,2)

~LogNormal(-1.3,0.4)

~ LogNormal(0,0.3)



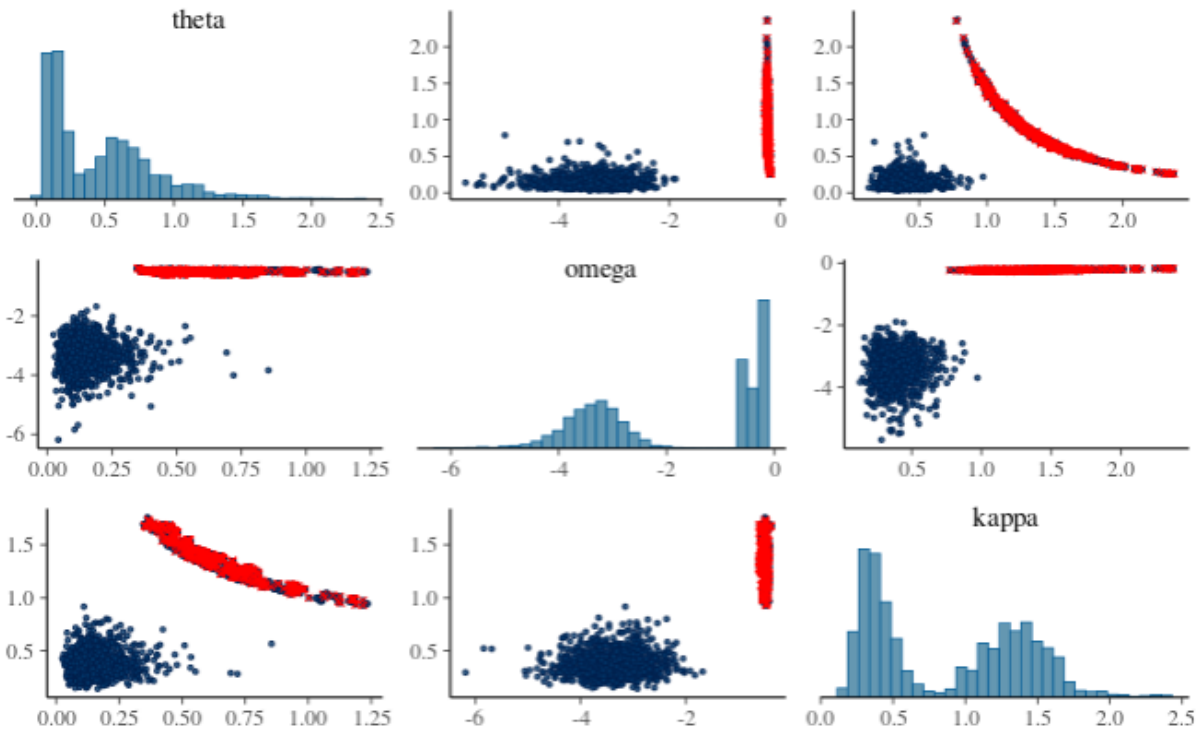


Seen from the plot above it looks like the divergences happen when the sampler gets to the tails (high values) of the or parameters, which made us try to limit their range i.e constraining them with priors as:

~LogNormal(-2,0.4)

~ LogNormal(-1,0.3)

which helped but it seemed as if we needed to restrain the parameters even more which at this point doesn't make any sense:

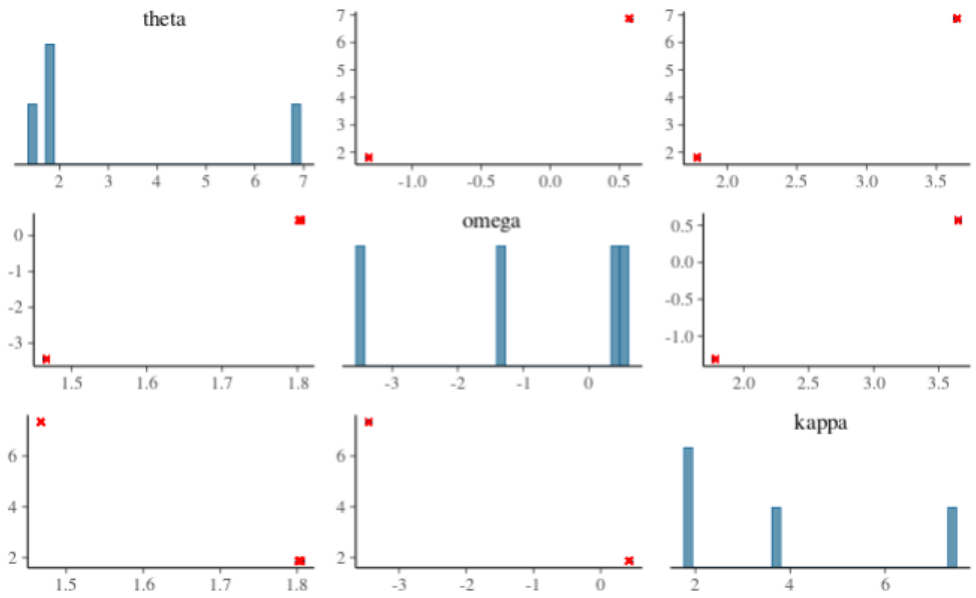


Widening the priors made the model even worse at converging.

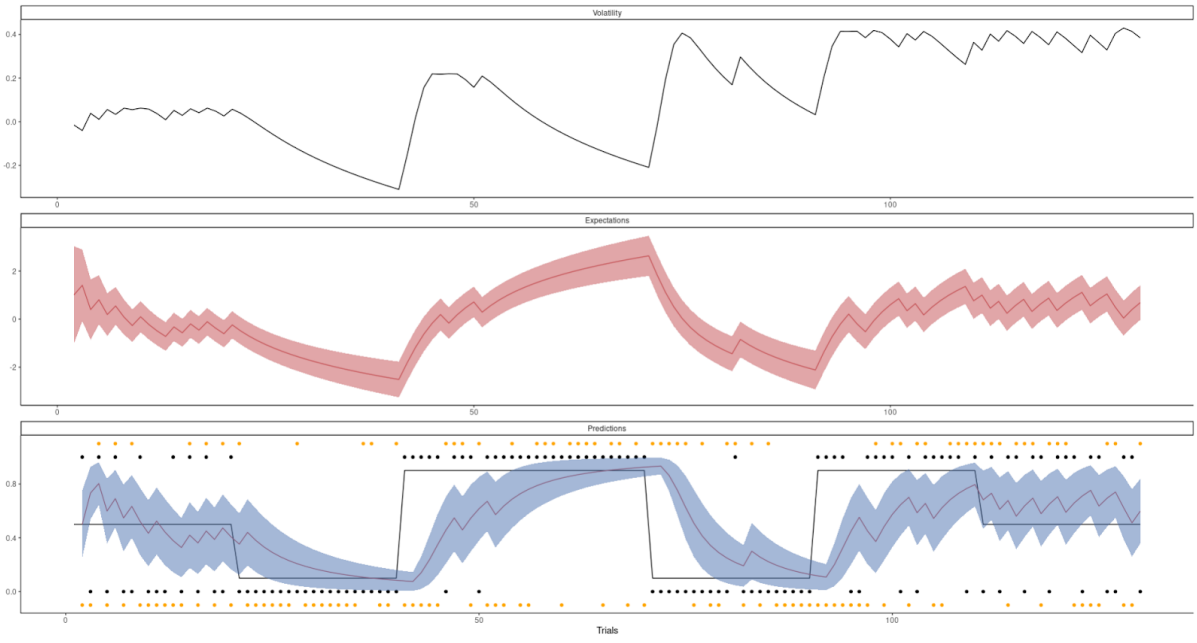
~LogNormal(0,1)

~ LogNormal(0,1)

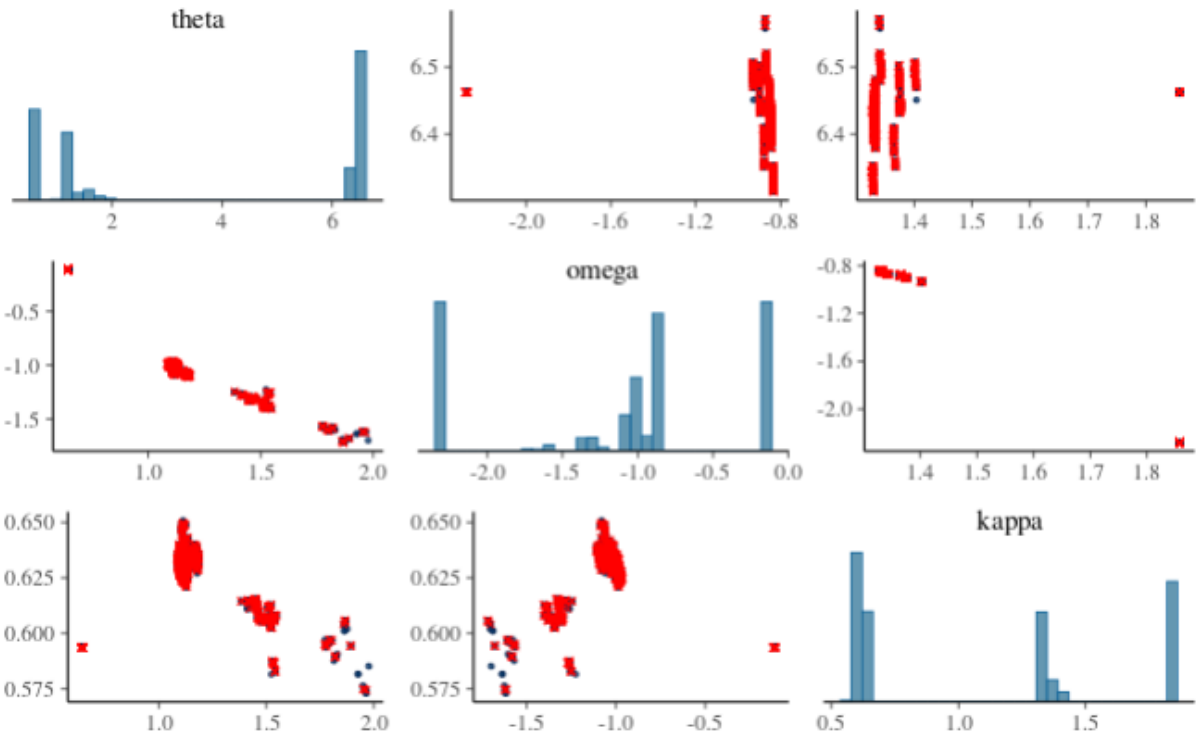
~ Normal(-3,4)



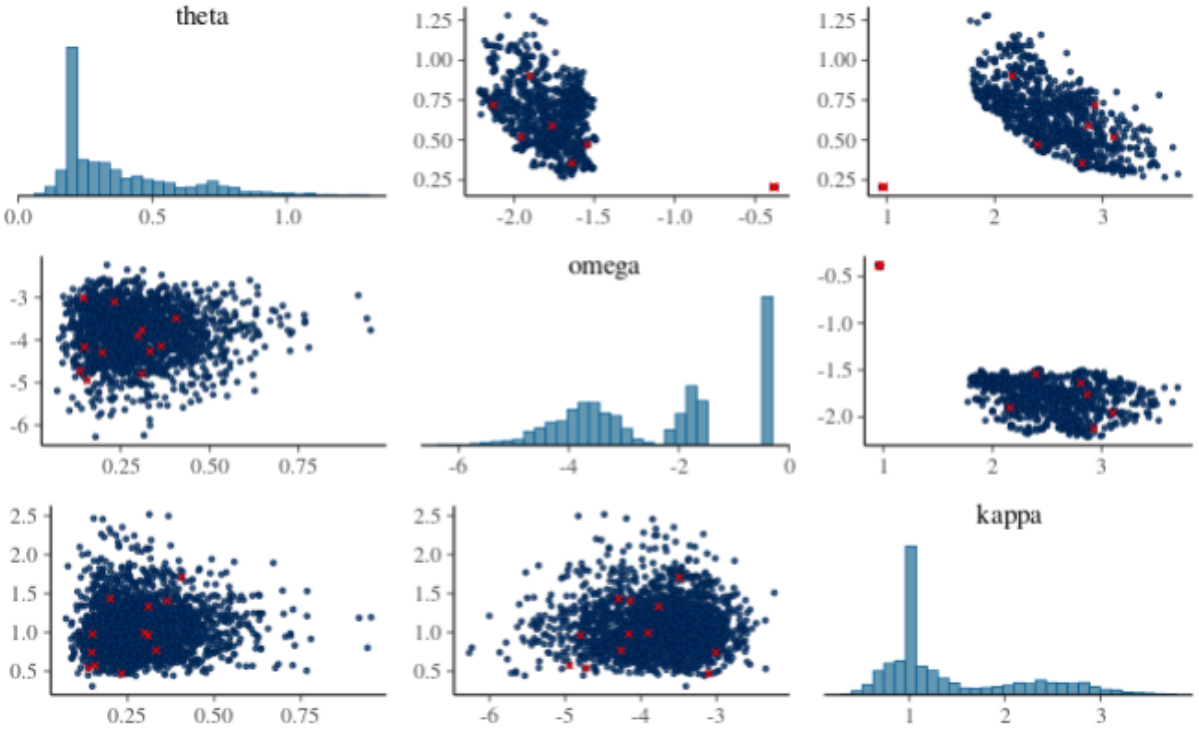
Lastly we tried to reduce the number of trials as we saw that even running the model feedforward produced some weird behavior sometime which mostly happened in the end of the trial sequence:

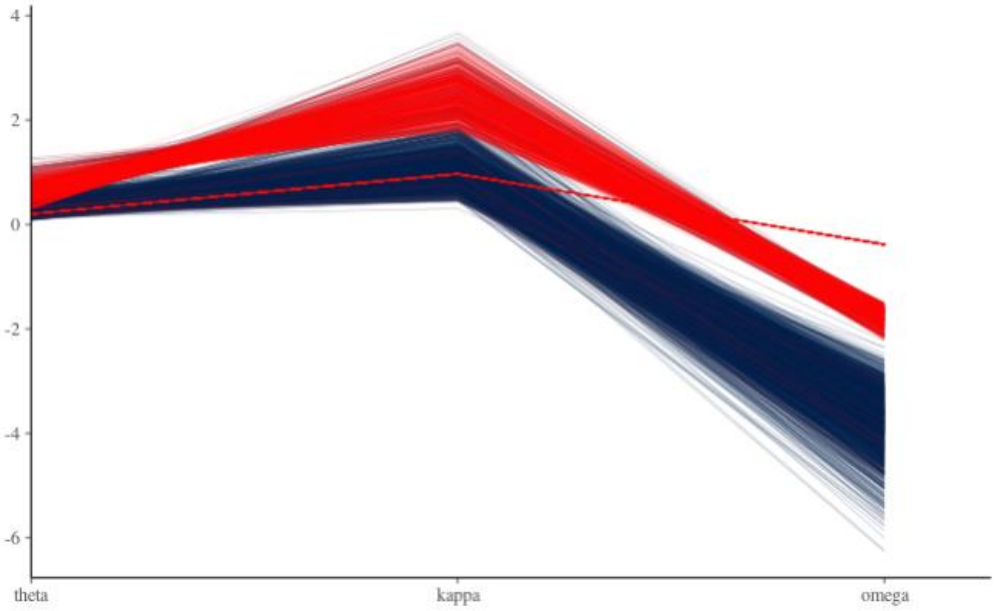


first with the wide priors



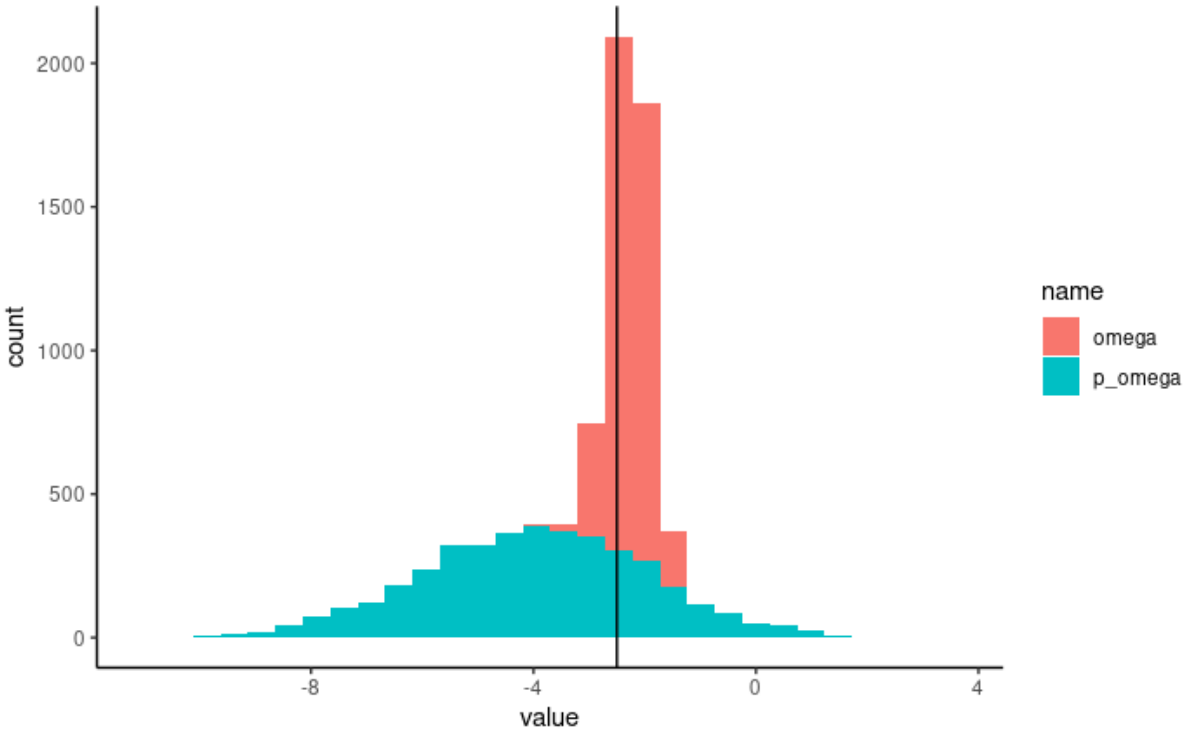
Then the informed priors:



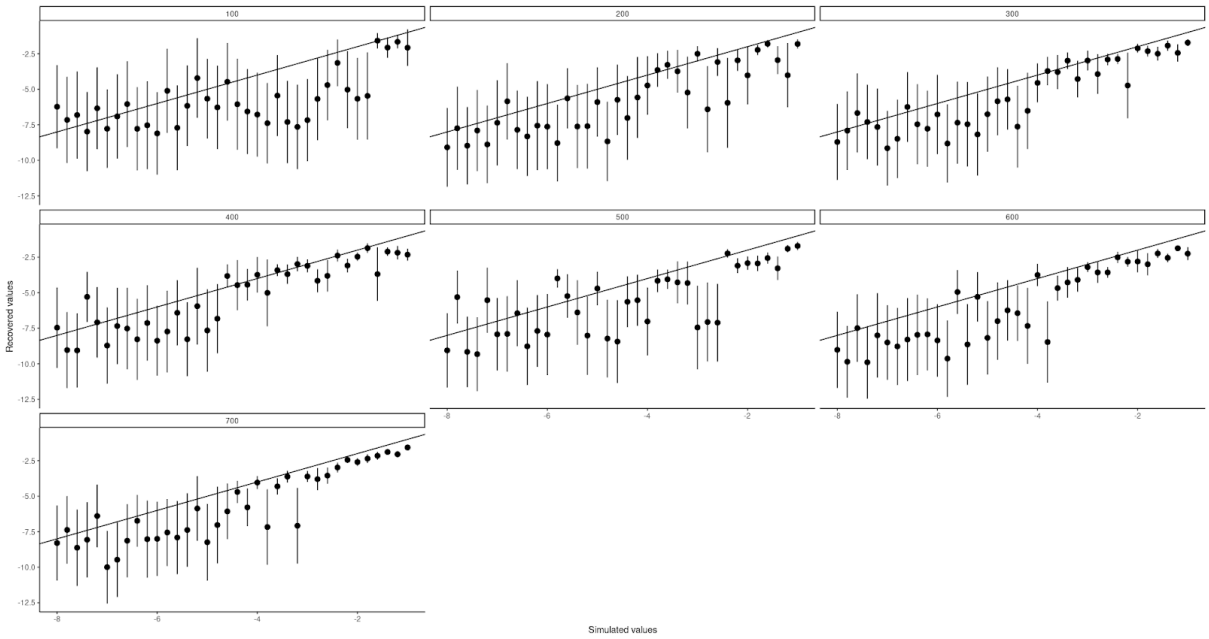


again a pattern emerges that when the sampler gets to the tails it fails, we therefore also tried with the restrictive priors which did not help.

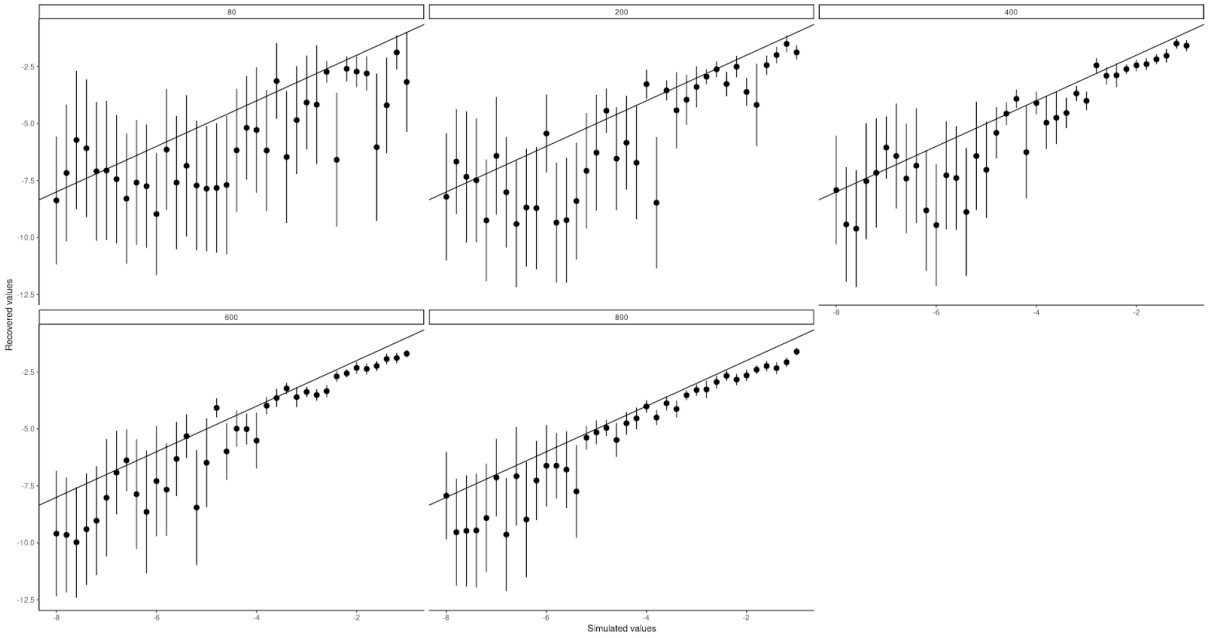
Next, we therefore tried to fix either the or parameter which resulted in similar patterns. However, fixing both resulted in no divergence and good fit. Making the model reduce to a 2-level HGF with 1 free parameter. Below is a prior posterior update plot of this parameter.



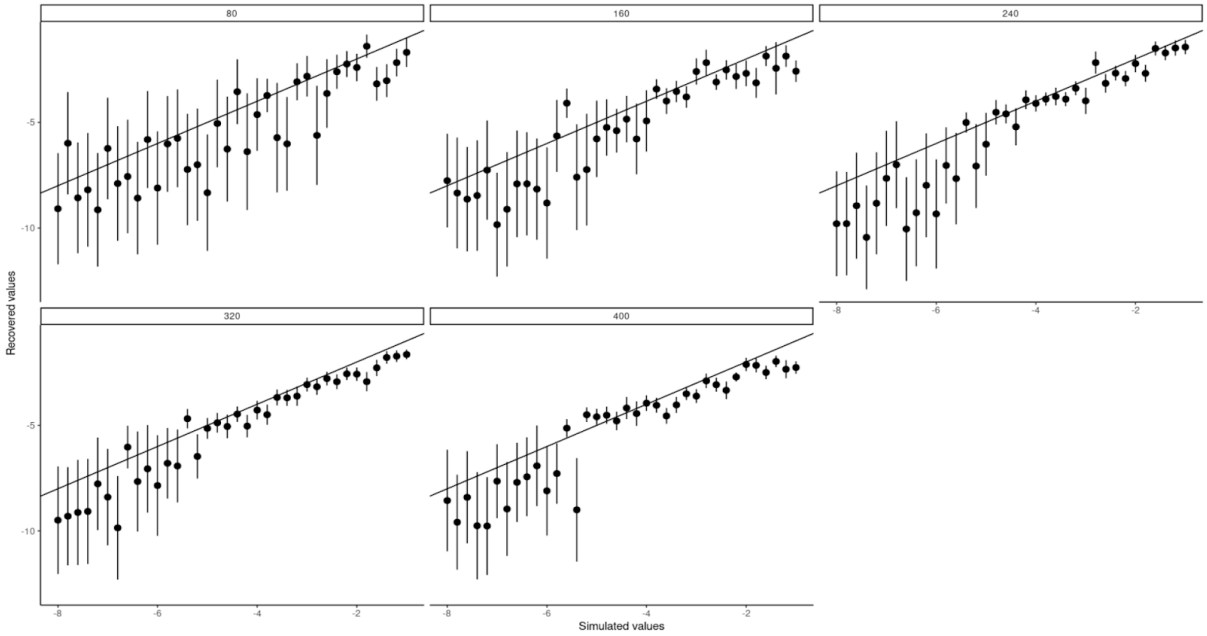
We therefore continued with parameter recovery and sensitivity analysis of this model. Given the reduced complexity we decided to have the random bias agent swap bias every 10 trials and therefore having to decide how many 20 trial sequences we need to recover the parameter. The biases of the random agent were set to 0.8 and 0.2 (i.e 10 trials of 0.8 and 10 trials of 0.2). We initialize our parameter recovery with a normal prior that is very wide i.e $\sim$ Normal(-3,5), and then simulate values from -8 to -1 with an increment of 0.2. We did this simulation for 100 until 700 trials with 100 trials increments shown below:



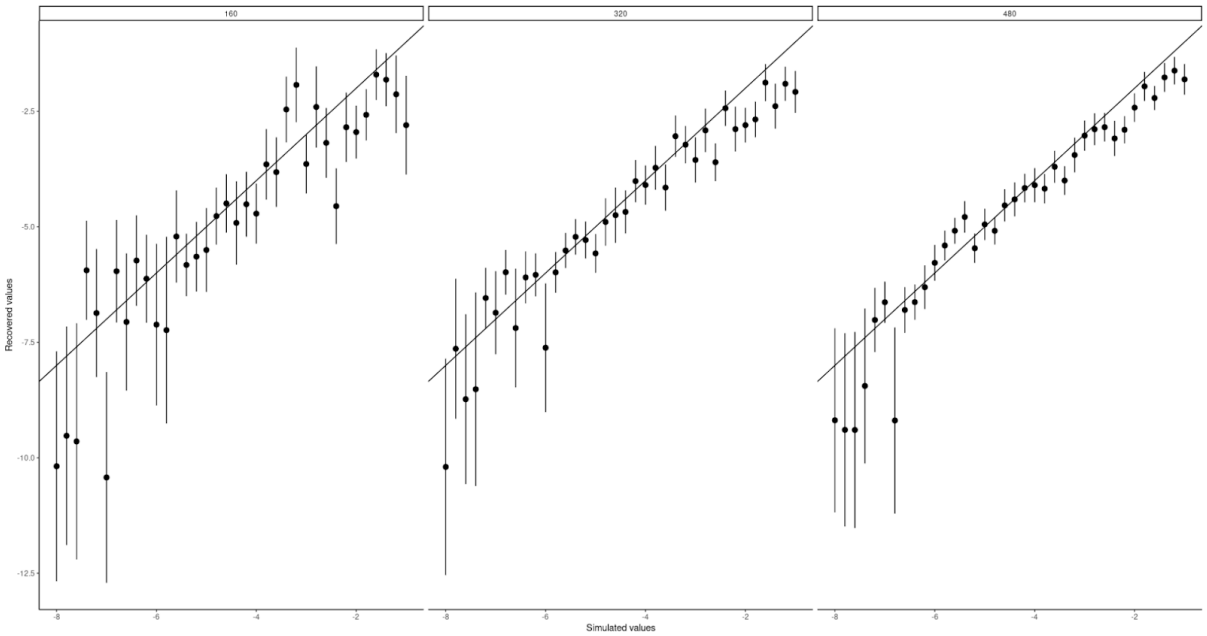
next we tried with 20 trials of each bias before switching for the random bias agent



with 40 trials between switching



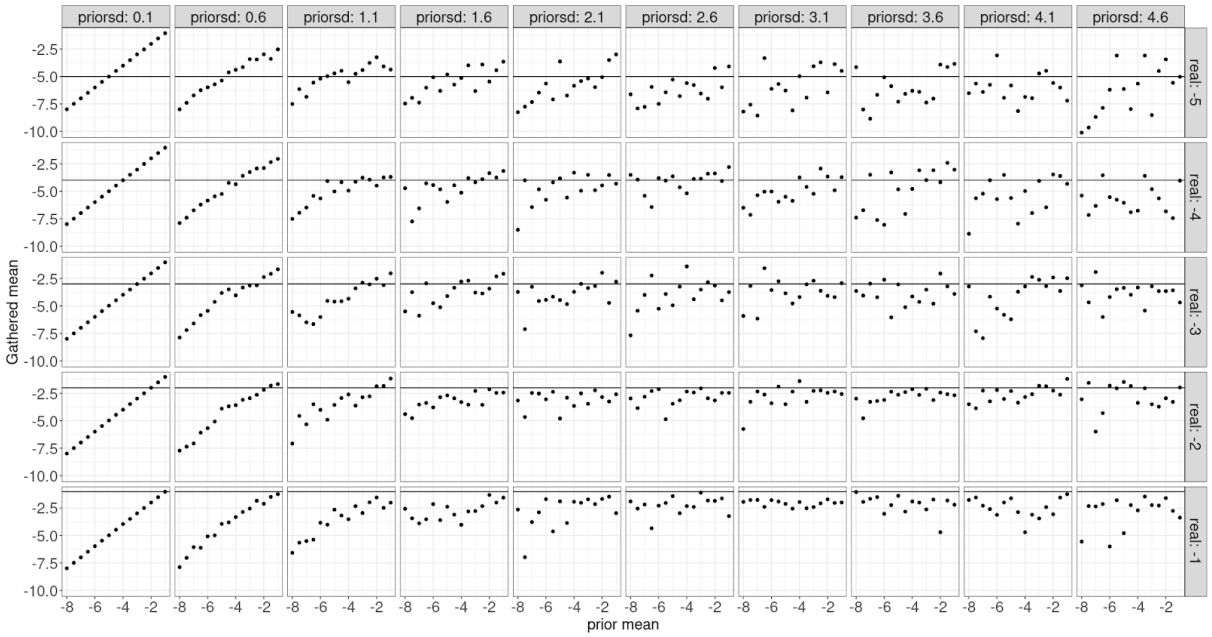
and lastly with 80 trials between switching



Looking at these plots we decided to go with 40 trials between switches and do each bias once, equating to 80 trials.

Now we did prior sensitivity analysis on this agents' omega parameter, simulating means from -8 to -1 as well as standard deviations from 5 to 0.1 by increments of 0.5

while varying the true underlying omega parameter from -5 to -1. Note in the plot below the vertical line is the same as the row-facets.



Discussion

From this prior sensitivity analysis we see that very strict priors (leftmost panels) constrain the model's ability to recover the underlying parameter as the gathered mean and the prior mean are mostly on a straight line. When the the prior standard deviation becomes very large (rightmost panels), we see that only the model with low real values of seem to be affected (i.e. real = -5 sd = 4.6) in this panel the prior mean seems to dictate the posterior mean when it's low. Overall it seems that prior standard deviations between 2 and 4 are fine for most prior means and that if one wanted wider priors then having more informative prior means would help.