**Digital Distribution Model**

**ORCID: 0000-0002-7919-4017**

**Abstract: This three-part series on the analysis of the digital distribution model uses R code and the BRMS package to show various stages of model implementation. The series' first installment shows the fundamentals of modelling and estimation. The second part of the series will show how to access the model fit and run model diagnostics. The third section shows how to conduct a test for parameter differences between conditions.**

**Note:** The Font Family and Size are Aerial & 8,10,12 respectively and Consolas & 10 respectively for better reading of Contents.

**Part I – Introduction & Estimation**

**1.1 Introduction**

The most intriguing advancement in computational statistics in recent years, in my opinion, is Stan for fitting Digital models. The range of probability distributions supported by the Stan language enables the fitting of a very broad range of models, and the Hamiltonian Monte-Carlo (HMC) version implemented in Stan is extremely efficient. Stan has significantly changed which models, in terms of both model complexity and data size, I think can be realistically estimated. Saying that Stan, and especially rstan, have significantly changed the way I analyse data is not an exaggeration.

One of the R packages, brms, has recently grown in popularity because it makes it easy to implement Stan models. There is a large variety of models that can be specified using the R formula interface. Based on the formula and a description of the model family, it generates the model code, compiles it, and then sends it, along with the data, to rstan for sampling. Since I typically programme my models by hand (thanks to the excellent Stan documentation: Stan - Documentation (mc-stan.org)), I have avoided brms thus far.

But I just learned that for data from two-choice tasks, brms can estimate the Digital Distribution model for simultaneously accounting for responses and corresponding response times. Such data are frequently found in psychology, and the model is one of the most widely used ones. I'll demonstrate how to use brms to apply the Digital Distribution model to some published data in this series. How to set up and estimate the model is described in the first section. A description of model diagnostics and an assessment of model fit using posterior predictive distributions are given in the second section. The parameter posterior distributions are examined and compared in the third section.

Brms, a functional C++ compiler, the packages RWiener for creating the posterior predictive distribution within brms, and rtdists for the data are required for this first section. I shall use DDM (Digital Distribution Model) as an abbreviation throughout this paper where ever is needed. To avail the packages and function of R package BRMS, you need to load it into your R console windows via the command Library(“brms”)

Library(“brms”)

**1.2 Data & Model**

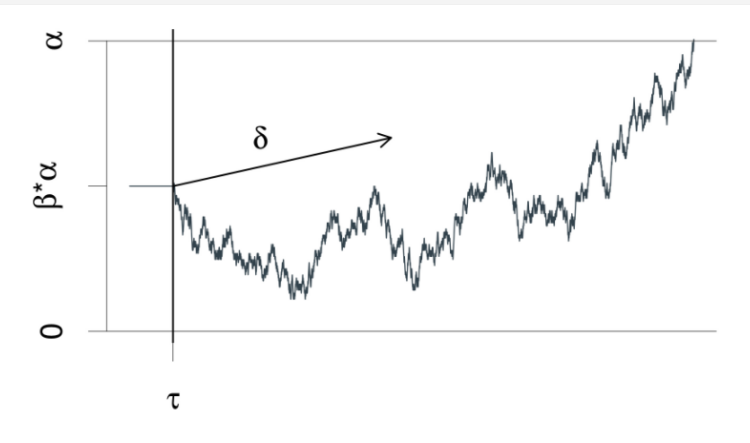


Figure 1.2.1 The Digital Distribution model for two-choice reaction times is depicted graphically. An evidence counter begins with the value 'alpha'\*'beta' and progresses with random increments. 'delta' is the mean increment. The process ends when the amount of evidence accumulated exceeds 'alpha' or exceeds 0. The decision process begins at time 'tau' after the stimulus is presented and ends at the reaction time.

I will only give a brief overview of the Digital Distribution model here. The model is a continuous-time evidence accumulation model for binary choice tasks. It is assumed that a single accumulator uses a noisy process to gather evidence for each trial. Once the accumulator reaches one of the two decision bounds, the corresponding response is given. The accumulation of evidence starts at the start point and continues until that point. The decision time and non-decisional components of the accumulation process are added to get the overall response time. In conclusion, the Digital Distribution model enables the division of responses to binary choice tasks and associated response times into four latent processes:

* The average slope of the accumulation process towards the boundaries is represented by the drift rate (delta). The greater the (absolute value of the) drift rate, the more evidence there is for the corresponding response option.
* The distance between the two decision bounds (alpha) is interpreted as a measure of response caution.
* The accumulation process's starting point (beta) is a measure of response bias towards one of the two response boundaries.
* Non-decision time (tau) encompasses all non-decisional processes such as stimulus encoding and response processes.

We will examine a portion of the data from my github account/repository ,“ https://github.com/binmishr/Digital-Distribution-Model/blob/main/speed\_acc.RData”. The data file which has been used below for the figure 1.2.2 is Responses and response times from an experiment in which instruction manipulated speed and accuracy between blocks. The information was gathered are whether or not a given string of letters was a word. The decisions based on either accuracy emphasis instructions or speed emphasis instructions in various experimental blocks. After eliminating some extremely long RTs (Response Times), we limit the analysis to high-frequency words (frequency = high) and the corresponding high-frequency non-words (frequency = nw\_high). A numeric response variable with a value of 0 for responses at the lower response boundary and a value of 1 for responses at the upper response boundary is also required to complete the model. A word response corresponds to the lower response boundary, and a nonword response corresponds to the upper response boundary. To achieve this, we convert the categorical response variable response to a numeric value and subtract one.

Figure 1.2.2 Sample Data

**1.3 Model Formula**

Which parameters are permitted to vary between which conditions (i.e., factor levels) must be decided upon before a model can be created. One restriction shared by the Digital Distribution model (and other evidence-accumulation models) is that the parameters chosen beforehand (i.e., boundary separation, starting point, and non-decision time) cannot change as a result of stimulus properties the participant is unaware of before the trial starts. The item-type, in this case word versus non-word, is therefore typically only permitted to affect drift rate. We abide by this restriction. All four parameters are allowed to vary, and the speed and accuracy conditions are changed between blocks of trials. Furthermore, it is important to note that all pertinent variables are controlled within-subjects. As a result, each fixed-effect has random-effects parameters included in the maximal random-effects structure. Using the bf() function, we must create a formula for each of the four parameters of the Digital Distribution model in order to set up the model.

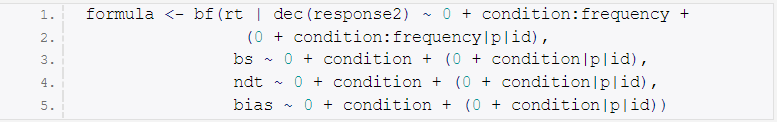


Figure 1.3.1 Digital Distribution Model Formula

The first formula specifies the column on the left that contains the RTs (rt) and the response or decision (response2) as well as the drift rate. Similar to lme4, fixed and random effects can be specified on the right side. We estimate fixed and random effects for both variables as well as their interaction because the drift rate is allowed to vary for both conditions and frequencies (stim cat would be equivalent). The last three parameters (which can only vary by condition) require that we develop a formula for each. These formulas' left sides display the parameter names as follows:

* bs stands for boundary separation (alpha)
* ndt stands for non-decision time (tau)
* bias: beginning point (beta)

On the right side, the fixed and random effects are once more described. It is important to note that, in contrast to the current approach, in which the response boundaries represent the two actual response options, one common approach for developing evidence accumulation models is to specify that one response boundary represents correct responses and the other response boundary denotes incorrect responses. In this scenario, the starting point cannot be estimated and must be set to 0.5 (replacing bias = 0.5 in the formula).

The formulas also require two additional considerations. In the beginning, I suppressed the intercept and used an unusual parameterization (for example, 0 + condition rather than condition). Categorical variables (i.e., factors) with k levels are coded with k-1 deviation variables, which represent deviations from the intercept, when an intercept is present. Therefore, the prior for these deviation variables must be taken into account in a Bayesian setting. As shown here, the model can be set up so that each factor level (or design cell, if more than one factor is involved) receives its own parameter when the intercept is suppressed. As long as one does not anticipate significantly different parameter values, this allows for the same prior for each parameter. Furthermore, when programming a model on one's own, this is a typical parameterization. To see the differences between parameterizations (models), compare the next two calls. The internal parameterization is created by the function matrix. For each condition, only the first generates a special parameter.

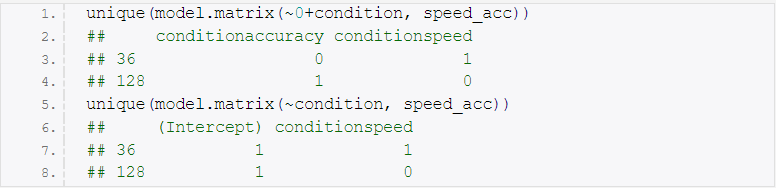


Figure 1.3.2 Model Matrix Function

It should be noted that the factors must be combined using the: and not the \* if more than one factor is involved and this parameterization is to be used. When the code below is run, this is evident. Additionally, the resulting model has one more parameter than can be estimated when the factors with: are combined without suppressing the intercept (i.e., the model-matrix is rank deficient). Therefore, caution is needed at this time. Second, the estimation of correlations between the random-effects parameters of various formulas is possible with brms formulas. To do this, add an identifier to the random-effects formula in the middle, separated by | on both sides. Then, for all random-effects formulas with the same identifier, correlations between random-effects formulas will be estimated. In this instance, we want to estimate the complete random-effects matrix with correlations between all model parameters using the "latent-trait approach". So, in all formulas, we employ the same identifier (p). Correlations between each individual-level deviation for each of the four Digital Distribution Model parameters will then be calculated. If you want to estimate correlations only among the random-effects parameters of each formula, simply omit the identifier (for example, (0 + condition|id)). Additionally, brms, like afex, enables you to use || (for example, (0 + condition||id)) to suppress correlations between categorical random-effects parameters.

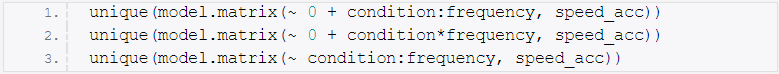


Figure 1.3.3 Model Matrix Sample Code with Parameters

**1.4 Family, Link-Functions and Priors**

Configuring the priors is the next step. The get\_prior function can be used to achieve this. The formula, data, and model family must all be specified for this function. We inform brms that we intend to use the digital distribution model parameter ( wiener) during the family argument. The link function for the four digital distribution model parameter ( wiener) parameters is also defined using it. Since the drift rate can take on any value (between -Inf and Inf), "identity" (i.e., no transformation) is the default link function, which we maintain. The ranges of the other three parameters are all constrained. The non-decision time must be greater than zero but less than the smallest RT, the boundary must be greater than zero, and the starting point must be between 0 and 1. The default link-functions, which use "log" for the first two parameters and "logit" for the bias, adhere to these constraints. Although this is undoubtedly a possibility, a number of disadvantages make me prefer to use the "identity" link function for all parameters. When parameters are transformed, priors must first be defined on the untransformed scale. Second, multivariate normal distributions are presumed to be the source of the individual-level deviations (i.e., the random-effects estimates). If the parameters are transformed, then individual deviations are only normally distributed on the untransformed scale. Parameter deviation correlations between parameters would also be on the untransformed scale. Both make it more difficult to interpret the random effects. Care must be taken to make sure that the priors give values within the permitted range the most weight when the parameters are specified without transformation (link = "identity"). The permitted range must also be met by the starting values. There are some disadvantages to using the identity link function, which are covered in more detail below. However, a model of this type can successfully converge, simplifying interpretation, provided that parameters outside the permitted range occur only infrequently. All model parameters are included in the data.frame that the get prior function returns. These are also listed if the parameters have default priors. Priors need to be specified for each parameter as well as for parameter classes, parameter classes for particular groups, and dpars. All parameters that lack a default prior should be given a specific prior, it should be noted.

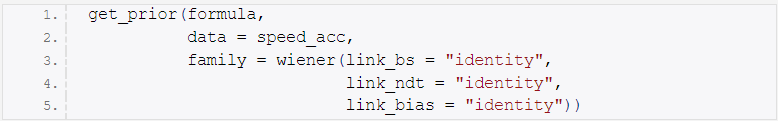


Figure 1.4.1 Configuration of Priors

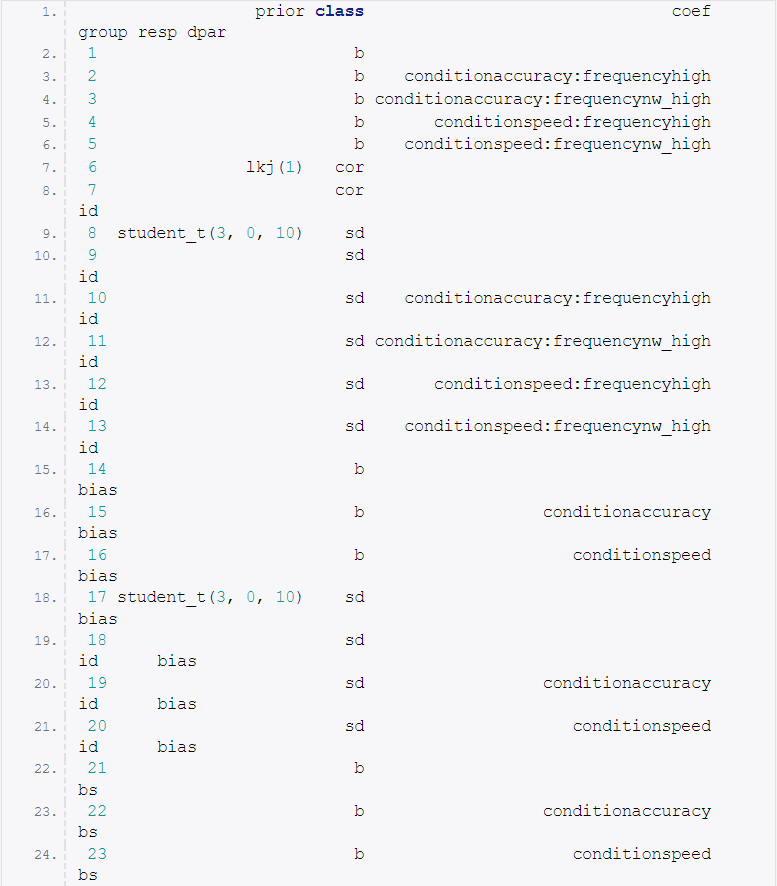


Figure 1.4.2 Output Of get\_prior Function

The prior or set\_prior functions, which offer varying degrees of control, can be used to define priors. One benefit of parameterizing the model is that it eliminates the need to distinguish between intercept and other parameters by allowing us to specify priors for just one set of parameters, b. We try to specify the priors in a weakly informative manner, but we do choose them based on prior knowledge of likely Digital Distribution model parameter values. In other words, they should keep the estimation unaffected while limiting the range to likely values. So that roughly 70% of the prior mass is between -10 and 10, we use a Cauchy distribution with location 0 and scale 5. For boundary separation, a normal prior with a mean of 1.5 and a standard deviation of one is used. For non-decision time and bias, a normal prior with a mean of 0.5 (i.e., no bias) and a standard deviation of 0.2 is used.

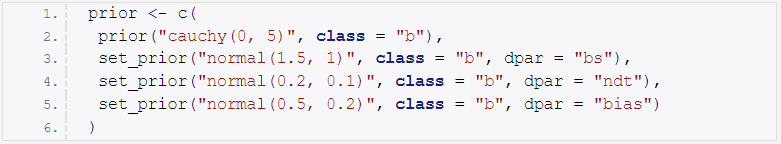


Figure 1.4.3 Defining the priors with data

We can use the make\_stancode function to inspect the entire model code once we have this information. The model block must contain a prior listing for every parameter listed in the parameters block. At the beginning of the model block, we can also see that none of our parameters have been transformed precisely as we had hoped. We have to use use function weiner\_lpdf functions of the BRMS package to inspect the data further which is shown below in the Figure 1.4.5

make\_stancode(formula,

family = wiener(link\_bs = "identity",

link\_ndt = "identity",

link\_bias = "identity"),

data = speed\_acc,

prior = prior)

Figure 1.4.4 Inspection of Digital Distribution Model with make\_stancode Function

// generated with brms version 1.10.2

functions {

/\* Digital Distribution Model log-PDF for a single response

\* Args:

\* y: reaction time data

\* dec: decision data (0 or 1)

\* alpha: boundary separation parameter > 0

\* tau: non-decision time parameter > 0

\* beta: initial bias parameter in [0, 1]

\* delta: drift rate parameter

\* Returns:

\* a scalar to be added to the log posterior

\*/

real Digital\_Distribution\_lpdf(real y, int dec, real alpha,

real tau, real beta, real delta) {

if (dec == 1) {

return wiener\_lpdf(y | alpha, tau, beta, delta);

} else {

return wiener\_lpdf(y | alpha, tau, 1 - beta, - delta);

}

}

}

data {

int<lower=1> N; // total number of observations

vector[N] Y; // response variable

int<lower=1> K; // number of population-level effects

matrix[N, K] X; // population-level design matrix

int<lower=1> K\_bs; // number of population-level effects

matrix[N, K\_bs] X\_bs; // population-level design matrix

int<lower=1> K\_ndt; // number of population-level effects

matrix[N, K\_ndt] X\_ndt; // population-level design matrix

int<lower=1> K\_bias; // number of population-level effects

matrix[N, K\_bias] X\_bias; // population-level design matrix

// data for group-level effects of ID 1

int<lower=1> J\_1[N];

int<lower=1> N\_1;

int<lower=1> M\_1;

vector[N] Z\_1\_1;

vector[N] Z\_1\_2;

vector[N] Z\_1\_3;

vector[N] Z\_1\_4;

vector[N] Z\_1\_bs\_5;

vector[N] Z\_1\_bs\_6;

vector[N] Z\_1\_ndt\_7;

vector[N] Z\_1\_ndt\_8;

vector[N] Z\_1\_bias\_9;

vector[N] Z\_1\_bias\_10;

int<lower=1> NC\_1;

int<lower=0,upper=1> dec[N]; // decisions

int prior\_only; // should the likelihood be ignored?

}

transformed data {

real min\_Y = min(Y);

}

parameters {

vector[K] b; // population-level effects

vector[K\_bs] b\_bs; // population-level effects

vector[K\_ndt] b\_ndt; // population-level effects

vector[K\_bias] b\_bias; // population-level effects

vector<lower=0>[M\_1] sd\_1; // group-level standard deviations

matrix[M\_1, N\_1] z\_1; // unscaled group-level effects

// cholesky factor of correlation matrix

cholesky\_factor\_corr[M\_1] L\_1;

}

transformed parameters {

// group-level effects

matrix[N\_1, M\_1] r\_1 = (diag\_pre\_multiply(sd\_1, L\_1) \* z\_1)';

vector[N\_1] r\_1\_1 = r\_1[, 1];

vector[N\_1] r\_1\_2 = r\_1[, 2];

vector[N\_1] r\_1\_3 = r\_1[, 3];

vector[N\_1] r\_1\_4 = r\_1[, 4];

vector[N\_1] r\_1\_bs\_5 = r\_1[, 5];

vector[N\_1] r\_1\_bs\_6 = r\_1[, 6];

vector[N\_1] r\_1\_ndt\_7 = r\_1[, 7];

vector[N\_1] r\_1\_ndt\_8 = r\_1[, 8];

vector[N\_1] r\_1\_bias\_9 = r\_1[, 9];

vector[N\_1] r\_1\_bias\_10 = r\_1[, 10];

}

model {

vector[N] mu = X \* b;

vector[N] bs = X\_bs \* b\_bs;

vector[N] ndt = X\_ndt \* b\_ndt;

vector[N] bias = X\_bias \* b\_bias;

for (n in 1:N) {

mu[n] = mu[n] + (r\_1\_1[J\_1[n]]) \* Z\_1\_1[n] + (r\_1\_2[J\_1[n]]) \* Z\_1\_2[n] + (r\_1\_3[J\_1[n]]) \* Z\_1\_3[n] + (r\_1\_4[J\_1[n]]) \* Z\_1\_4[n];

bs[n] = bs[n] + (r\_1\_bs\_5[J\_1[n]]) \* Z\_1\_bs\_5[n] + (r\_1\_bs\_6[J\_1[n]]) \* Z\_1\_bs\_6[n];

ndt[n] = ndt[n] + (r\_1\_ndt\_7[J\_1[n]]) \* Z\_1\_ndt\_7[n] + (r\_1\_ndt\_8[J\_1[n]]) \* Z\_1\_ndt\_8[n];

bias[n] = bias[n] + (r\_1\_bias\_9[J\_1[n]]) \* Z\_1\_bias\_9[n] + (r\_1\_bias\_10[J\_1[n]]) \* Z\_1\_bias\_10[n];

}

// priors including all constants

target += cauchy\_lpdf(b | 0, 5);

target += normal\_lpdf(b\_bs | 1.5, 1);

target += normal\_lpdf(b\_ndt | 0.2, 0.1);

target += normal\_lpdf(b\_bias | 0.5, 0.2);

target += student\_t\_lpdf(sd\_1 | 3, 0, 10)

- 10 \* student\_t\_lccdf(0 | 3, 0, 10);

target += lkj\_corr\_cholesky\_lpdf(L\_1 | 1);

target += normal\_lpdf(to\_vector(z\_1) | 0, 1);

// likelihood including all constants

if (!prior\_only) {

for (n in 1:N) {

target += Digital\_Distribution\_lpdf(Y[n] | dec[n], bs[n], ndt[n], bias[n], mu[n]);

}

}

}

generated quantities {

corr\_matrix[M\_1] Cor\_1 = multiply\_lower\_tri\_self\_transpose(L\_1);

vector<lower=-1,upper=1>[NC\_1] cor\_1;

// take only relevant parts of correlation matrix

cor\_1[1] = Cor\_1[1,2]; [...]

cor\_1[45] = Cor\_1[9,10]; }

Figure 1.4.5 Function of Digital Distribution Model log-PDF for a single response

We need a function to generate initial values before we can finally estimate the model. Estimation won't start until all of the initial values that result in a model that can be identified are given. All of the parameters listed in the model's parameters block must have initial values provided by the function. The fact that many of those parameters have at least one dimension with a parameterized extent (like K, for example) should be noted. We can use make\_standata to create the data set that brms uses for estimation in order to gather the necessary data. After that, we use this data object (a list) in the function initfun to create the initial values that are the right size (note that initfun depends on the fact that tmp\_dat is in the global environment, which is a little bit of a code smell).

tmp\_dat <- make\_standata(formula,

family = wiener(link\_bs = "identity",

link\_ndt = "identity",

link\_bias = "identity"),

data = speed\_acc, prior = prior)

str(tmp\_dat, 1, give.attr = FALSE)

## List of 26

## $ N : int 10462

## $ Y : num [1:10462(1d)] 0.773 0.39 0.435 ...

## $ K : int 4

## $ X : num [1:10462, 1:4] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_1 : num [1:10462(1d)] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_2 : num [1:10462(1d)] 0 1 1 1 1 1 0 1 1 0 ...

## $ Z\_1\_3 : num [1:10462(1d)] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_4 : num [1:10462(1d)] 1 0 0 0 0 0 1 0 0 1 ...

## $ K\_bs : int 2

## $ X\_bs : num [1:10462, 1:2] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_bs\_5 : num [1:10462(1d)] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_bs\_6 : num [1:10462(1d)] 1 1 1 1 1 1 1 1 1 1 ...

## $ K\_ndt : int 2

## $ X\_ndt : num [1:10462, 1:2] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_ndt\_7 : num [1:10462(1d)] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_ndt\_8 : num [1:10462(1d)] 1 1 1 1 1 1 1 1 1 1 ...

## $ K\_bias : int 2

## $ X\_bias : num [1:10462, 1:2] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_bias\_9 : num [1:10462(1d)] 0 0 0 0 0 0 0 0 0 0 ...

## $ Z\_1\_bias\_10: num [1:10462(1d)] 1 1 1 1 1 1 1 1 1 1 ...

## $ J\_1 : int [1:10462(1d)] 1 1 1 1 1 1 1 1 1 1 ...

## $ N\_1 : int 17

## $ M\_1 : int 10

## $ NC\_1 : num 45

## $ dec : num [1:10462(1d)] 1 0 0 0 0 0 0 0 0 0 ...

## $ prior\_only : int 0

initfun <- function() {

list(

b = rnorm(tmp\_dat$K),

b\_bs = runif(tmp\_dat$K\_bs, 1, 2),

b\_ndt = runif(tmp\_dat$K\_ndt, 0.1, 0.15),

b\_bias = rnorm(tmp\_dat$K\_bias, 0.5, 0.1),

sd\_1 = runif(tmp\_dat$M\_1, 0.5, 1),

z\_1 = matrix(rnorm(tmp\_dat$M\_1\*tmp\_dat$N\_1, 0, 0.01),

tmp\_dat$M\_1, tmp\_dat$N\_1),

L\_1 = diag(tmp\_dat$M\_1)

)

}

Figure 1.4.6 Functions to generate initial values for Model Estimation

**1.5 Estimation (Sampling)**

Finally, everything is in place, and we can estimate the Digital Distribution model using the brm function. Please be aware that depending on how quickly your PC runs, this could take longer than a full day. The maximum treedepth has also been raised to 15. There are a few divergent transitions, so we probably should have increased adapt\_delta above the default value of.8, but the reader decides. We discover that there are a few (10) divergent transitions after we have finished estimating. Instead of treating this as a hypothetical analysis, we would need to raise adapt\_delta to a higher number (like.95 or.99) and rerun the estimation. However, in this instance, we move straight to step two and use predict to gather samples from the posterior predictive distribution. In this situation, it is crucial to specify the number of posterior samples (500 are used here). Additionally, set summary = FALSE and negative rt = TRUE to obtain the actual posterior predictive distribution as opposed to a summary of the posterior predictive distribution. Due to the latter, predicted responses to the lower boundary are guaranteed to be negative while those to the upper boundary are guaranteed to be positive.

fit\_DDM <- brm(formula,

data = speed\_acc,

family = wiener(link\_bs = "identity",

link\_ndt = "identity",

link\_bias = "identity"),

prior = prior, inits = initfun,

iter = 1000, warmup = 500,

chains = 4, cores = 4,

control = list(max\_treedepth = 15))

NPRED <- 500

pred\_DDM <- predict(fit\_DDM,

summary = FALSE,

negative\_rt = TRUE,

nsamples = NPRED)

Figure 1.5.1 Digital Distribution Model Estimation

Because both processes take time (estimation takes a day, obtaining posterior predictive takes a few hours), we save the results of both. Using the 'xz' compression, the strongest in R, seems wise given the relative sizes of the two objects.

save(fit\_DDM, file = "brms\_DDM\_example\_fit.rda",

compress = "xz")

save(pred\_DDM, file = "brms\_DDM\_example\_predictions.rda",

compress = "xz")

Figure 1.5.2 Saving Results of Digital Distribution Model Estimation

**Part II – Model Fit & Diagnostics**

**2.1 Introduction**

Model diagnostics and model fit evaluation, arguably the most crucial steps in any model-based data analysis, are covered in this second section. It should be noted that the code in this section can be run entirely independently of the code in Part I.

**2.2 Setup**

We start by loading a sizable number of the packages we'll need later. Brms is a given, but there are also some tidyverse packages (namely, dplyr, tidyr, tibble, and ggplot2) that are useful. It took me some time to accept tidyverse, but now that I'm using it more frequently, I can't argue with its value. If your data can be made "tidy," the comprehensive toolkit of the tidyverse makes many tasks that appear challenging comparatively easy. The examples below are just a few. DescTools and GridExtra are both necessary for the concordance correlation coefficient CCC, which is used below, as well as for combining plots.

library("brms")

library("dplyr")

library("tidyr")

library("tibble") # for rownames\_to\_column

library("ggplot2")

library("gridExtra") # for grid.arrange

library("DescTools") # for CCC

Figure 2.2.1 Loading of the R Libraries

data(speed\_acc, package = "rtdists")

speed\_acc <- droplevels(speed\_acc[!speed\_acc$censor,]) # remove extreme RTs

speed\_acc <- droplevels(speed\_acc[ speed\_acc$frequency %in%

c("high", "nw\_high"),])

speed\_acc$response2 <- as.numeric(speed\_acc$response)-1

Figure 2.2.2 Loading of the data with rtdists package

I used a binary R data file that we can open in R and which contains the generated posterior predictive distributions below in addition to the fitted model object. It's worth noting that in order to get there, I had to navigate through a temporary folder. I've attached the Model fit and predictions for your convenience.RDA files can be found in my " binmishr/Digital-Distribution-Model " repository on Github. You can change the file path during the download and download the attached files to a different location on your PC. File function, followed by loading the files into the temporary folder in accordance with the syntax shown below in figure 2.2.3

tmp <- tempdir()

download.file("https://github.com/binmishr/Digital-Distribution-Model/blob/main/brms\_DDM\_example\_fit.rda",

file.path(tmp, "brms\_DDM\_example\_fit.rda"))

download.file("https://github.com/binmishr/ Digital-Distribution-Model /blob/main/brms\_DDM\_example\_predictions.rda",

file.path(tmp, "brms\_DDM\_example\_predictions.rda"))

load(file.path(tmp, "brms\_DDM\_example\_fit.rda"))

load(file.path(tmp, "brms\_DDM\_example\_predictions.rda"))

Figure 2.2.3 Downloading the Model & Prediction data into Temporary folder

**2.3 Model Diagnostics**

There are a few divergent transitions, as Part I already mentioned. We would be unsatisfied with the current fit if this were a real analysis, and we would try to rerun brm with a higher adapt\_delta in the hopes of removing the divergent transitions. The Stan warning guidelines state that "the validity of the estimates is not guaranteed if there are post-warmup divergences." However, it is unclear how the posterior is impacted by the few divergent transitions (10) found here. What to do if adapt delta can no longer be increased and the model cannot be reparametrized is also not clear. Should all fits and divergent transitions be completely disregarded. Returning to our fit, as a first step in our model diagnostics, we check the R-hat statistic and the number of effective samples. The parameters with the highest R2 and the fewest effective samples are the ones we concentrate on. The sampler has converged on the stationary distribution because both are unproblematic (R-hat 1.05 and n eff > 100).

tail(sort(rstan::summary(fit\_DDM$fit)$summary[,"Rhat"]))

# sd\_id\_\_conditionaccuracy:frequencyhigh

# 1.00

# r\_id\_\_bs[15,conditionaccuracy]

# 1.00

# b\_bias\_conditionaccuracy

# 1.00

# cor\_id\_\_conditionspeed:frequencyhigh\_\_ndt\_conditionaccuracy

# 1.00

# sd\_id\_\_ndt\_conditionspeed

# 1.00

# cor\_id\_\_conditionspeed:frequencynw\_high\_\_bs\_conditionspeed

# 1.01

head(sort(rstan::summary(fit\_DDM$fit)$summary[,"n\_eff"]))

# lp\_\_

# 462

# b\_conditionaccuracy:frequencyhigh

# 588

# sd\_id\_\_ndt\_conditionspeed

# 601

# sd\_id\_\_conditionspeed:frequencyhigh

# 646

# b\_conditionspeed:frequencyhigh

# 695

# r\_id[12,conditionaccuracy:frequencyhigh]

# 712

Figure 2.3.1 Measuring the R-hat statistics & Number of effective samples

pars <- parnames(fit\_DDM)

pars\_sel <- c(sample(pars[1:10], 3), sample(pars[-(1:10)], 3))

plot(fit\_DDM, pars = pars\_sel, N = 6,

ask = FALSE, exact\_match = TRUE, newpage = TRUE, plot = TRUE)

Figure 2.3.2 Plotting of the data

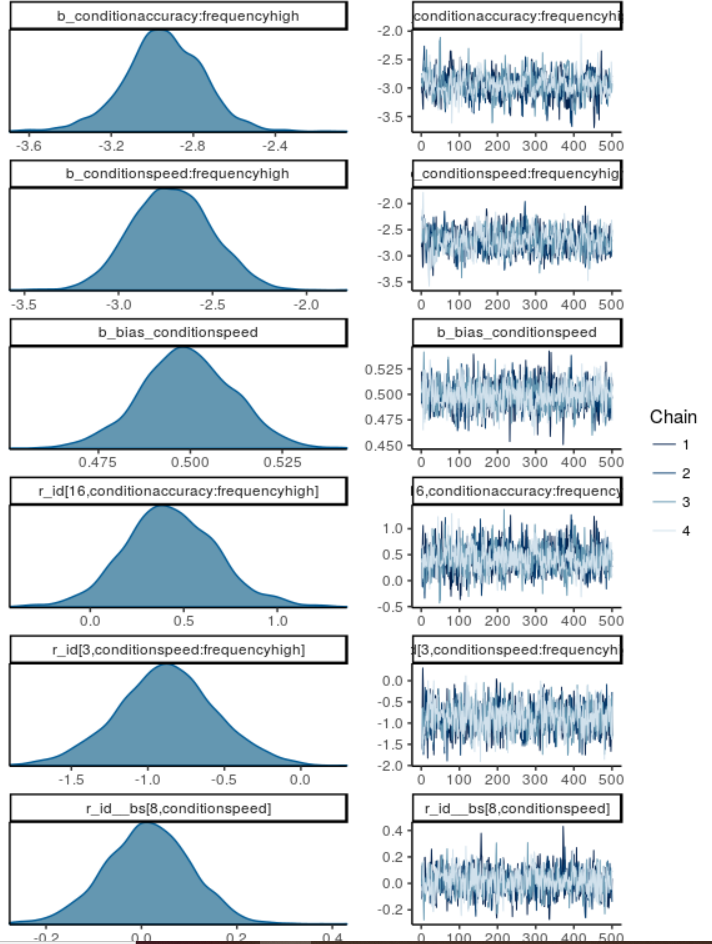


Figure 2.3.3 Graph of Chain behaviour of the Sampled data

The literature also includes discussions of parameter trade-offs for related models. It is claimed that these trade-offs make fitting the model in a Bayesian environment particularly challenging. In order to determine whether fitting the Digital Distribution model with HMC as implemented in Stan (i.e., NUTS) also exhibits this pattern, we examine the joint posterior of the fixed-effects of the primary parameters for the accuracy condition. We accomplish this using the stanfit method of the pairs function with the condition set to "divergent\_\_." This plot displays the remaining samples below the diagonal and the few divergent transitions above it.

pairs(fit\_DDM$fit, pars = pars[c(1, 3, 5, 7, 9)], condition = "divergent\_\_")

Figure 2.3.4 Plotting of the divergent Transitions of the Sampled data

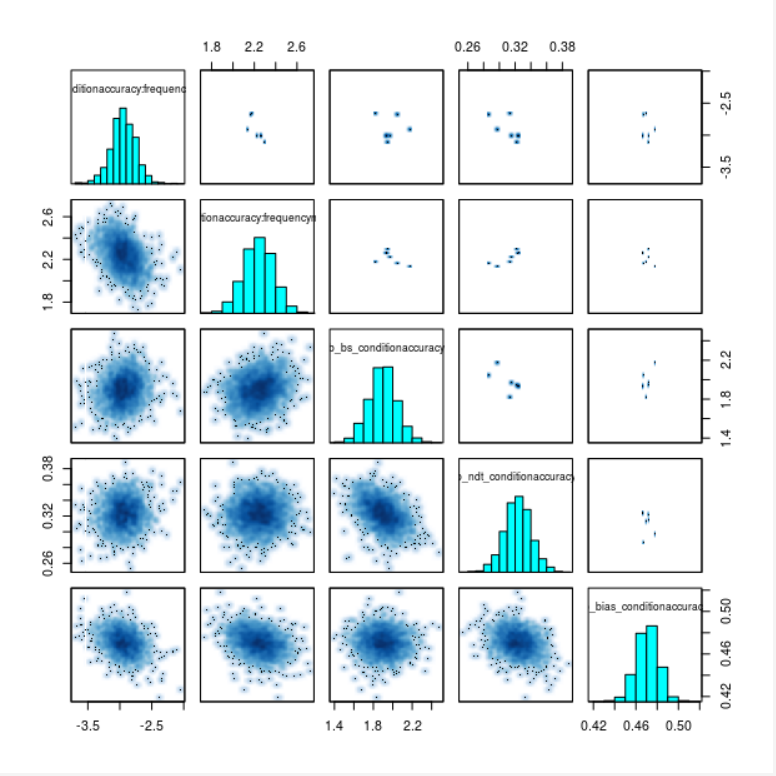


Figure 2.3.5 Graph of the Divergent Transitions

The ten highest absolute posterior correlation values for all pairwise parameter combinations are shown in the table below. It may seem counterintuitive, but the value in column Freq is the observed correlation between the posteriors of the two parameters listed in the two columns before it. As.table, a technique I used to create this table, is responsible for labelling the column containing the correlation value Freq.

posterior <- as.mcmc(fit\_DDM, combine\_chains = TRUE)

cor\_posterior <- cor(posterior)

cor\_posterior[lower.tri(cor\_posterior, diag = TRUE)] <- NA

cor\_long <- as.data.frame(as.table(cor\_posterior))

cor\_long <- na.omit(cor\_long)

tail(cor\_long[order(abs(cor\_long$Freq)),], 10)

# Var1 Var2 Freq

# 43432 b\_ndt\_conditionspeed r\_id\_\_ndt[1,conditionspeed] -0.980

# 45972 r\_id\_\_ndt[4,conditionspeed] r\_id\_\_ndt[11,conditionspeed] 0.982

# 46972 b\_ndt\_conditionspeed r\_id\_\_ndt[16,conditionspeed] -0.982

# 44612 b\_ndt\_conditionspeed r\_id\_\_ndt[6,conditionspeed] -0.983

# 46264 b\_ndt\_conditionspeed r\_id\_\_ndt[13,conditionspeed] -0.983

# 45320 b\_ndt\_conditionspeed r\_id\_\_ndt[9,conditionspeed] -0.984

# 45556 b\_ndt\_conditionspeed r\_id\_\_ndt[10,conditionspeed] -0.985

# 46736 b\_ndt\_conditionspeed r\_id\_\_ndt[15,conditionspeed] -0.985

# 44140 b\_ndt\_conditionspeed r\_id\_\_ndt[4,conditionspeed] -0.990

# 45792 b\_ndt\_conditionspeed r\_id\_\_ndt[11,conditionspeed] -0.991

Figure 2.3.6 Correlations values among posteriors for all pairwise parameter combinations

pairs(fit\_DDM$fit, pars =

c("b\_ndt\_conditionspeed",

"r\_id\_\_ndt[11,conditionspeed]",

"r\_id\_\_ndt[4,conditionspeed]"),

condition = "divergent\_\_")

Figure 2.3.7 Plotting of the Correlations values among posteriors for all pairwise parameter combinations

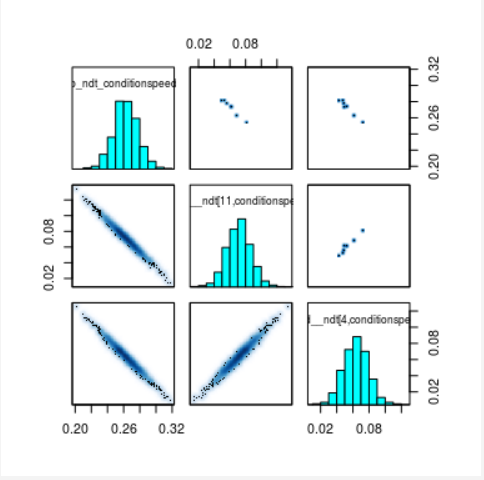


Figure 2.3.8 Graph of the Correlations values among posteriors for all pairwise parameter combinations

Overall, the model diagnostics (with the exception of the divergent transitions) do not indicate any particularly alarming behaviour. We've found that some of the parameter estimates at the individual-level aren't very accurate. However, this does not disprove the overall fit. This fact teaches us to be cautious when interpreting estimates at the individual level. Therefore, we proceed to the next stage of the analysis under the assumption that the fit is satisfactory.

hist(cor\_long$Freq, breaks = 40)

Figure 2.3.9 Plotting of the Histogram of Correlations values

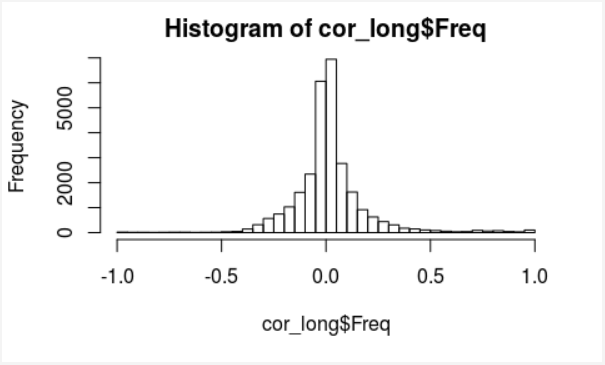


Figure 2.3.10 Histogram graph of the Correlations values among posteriors for all pairwise parameter combinations

**2.4 Accessing Model Fit**

Now let's examine the model fit. In other words, we will look into whether the model adequately accounts for the data that was observed. The main method we'll use to do this is graphical checks. We must first prepare the posterior predictive distribution and the data in order to achieve this. The posterior predictive distributions and the data are combined first. Then, we compute three significant measures (or test statistics T()) for each cell of the design (i.e., a combination of condition and frequency factors):

* Probability of responding with an upper boundary response (i.e., "nonword").
* Median response times (RTs) to the upper boundary.
* The lower boundary's median RTs.

For each sample of the posterior predictive distribution, this is first calculated. The posterior predictive distribution's median and a few additional quantiles are then calculated to combine these three metrics. With the help of a complex dplyr and tidyr magic combination, we compute all of this in a single step.

d\_speed\_acc <- as\_tibble(cbind(speed\_acc, as\_tibble(t(pred\_DDM))))

Figure 2.4.1 Combination of posterior predictive distribution with data

d\_speed\_acc\_agg <- d\_speed\_acc %>%

group\_by(id, condition, frequency) %>% # select grouping vars

summarise\_at(.vars = vars(starts\_with("V")),

funs(prob.upper = mean(. > 0),

medrt.lower = median(abs(.[. < 0]) ),

medrt.upper = median(.[. > 0] )

)) %>%

ungroup %>%

gather("key", "value", -id, -condition, -frequency) %>% # remove grouping vars

separate("key", c("rep", "measure"), sep = "\_") %>%

spread(measure, value) %>%

group\_by(id, condition, frequency) %>% # select grouping vars

summarise\_at(.vars = vars(prob.upper, medrt.lower, medrt.upper),

.funs = funs(median = median(., na.rm = TRUE),

llll = quantile(., probs = 0.01,na.rm = TRUE),

lll = quantile(., probs = 0.025,na.rm = TRUE),

ll = quantile(., probs = 0.1,na.rm = TRUE),

l = quantile(., probs = 0.25,na.rm = TRUE),

h = quantile(., probs = 0.75,na.rm = TRUE),

hh = quantile(., probs = 0.9,na.rm = TRUE),

hhh = quantile(., probs = 0.975,na.rm = TRUE),

hhhh = quantile(., probs = 0.99,na.rm = TRUE)

))

Figure 2.4.2 Calculate Posterior predictive distribution , Median & quantiles across measures

Then, we compute the three data metrics and combine them with the posterior predictive distribution's findings to create a single data set. frame built using a left join

speed\_acc\_agg <- speed\_acc %>%

group\_by(id, condition, frequency) %>% # select grouping vars

summarise(prob.upper = mean(response == "nonword"),

medrt.upper = median(rt[response == "nonword"]),

medrt.lower = median(rt[response == "word"])

) %>%

ungroup %>%

left\_join(d\_speed\_acc\_agg)

Figure 2.4.3 Calculating measures & Combining with Posterior predictive distribution

**2.5 Aggregated Model Fit**

Whether or not our model can accurately describe the data's aggregated patterns across participants is the first important question. The results from the previous step (i.e., the summary results from the posterior predictive distribution as well as the data test statistics) are simply aggregated using the mean. The predictions (in grey and black) and data (in red) for the three measures are then plotted using the summaries. The 80% credibility intervals (CIs) are represented by the inner (fat) error bars, while the 95% CIs are represented by the outer (thin) error bars. The black circle represents the median of the posterior predictive distributions.

d\_speed\_acc\_agg2 <- speed\_acc\_agg %>%

group\_by(condition, frequency) %>%

summarise\_if(is.numeric, mean, na.rm = TRUE) %>%

ungroup

Figure 2.5.1 Aggregation of the Posterior predictive distribution

new\_x <- with(d\_speed\_acc\_agg2,

paste(rep(levels(condition), each = 2),

levels(frequency), sep = "\n"))

p1 <- ggplot(d\_speed\_acc\_agg2, aes(x = condition:frequency)) +

geom\_linerange(aes(ymin = prob.upper\_lll, ymax = prob.upper\_hhh),

col = "darkgrey") +

geom\_linerange(aes(ymin = prob.upper\_ll, ymax = prob.upper\_hh),

size = 2, col = "grey") +

geom\_point(aes(y = prob.upper\_median), shape = 1) +

geom\_point(aes(y = prob.upper), shape = 4, col = "red") +

ggtitle("Response Probabilities") +

ylab("Probability of upper resonse") + xlab("") +

scale\_x\_discrete(labels = new\_x)

p2 <- ggplot(d\_speed\_acc\_agg2, aes(x = condition:frequency)) +

geom\_linerange(aes(ymin = medrt.upper\_lll, ymax = medrt.upper\_hhh),

col = "darkgrey") +

geom\_linerange(aes(ymin = medrt.upper\_ll, ymax = medrt.upper\_hh),

size = 2, col = "grey") +

geom\_point(aes(y = medrt.upper\_median), shape = 1) +

geom\_point(aes(y = medrt.upper), shape = 4, col = "red") +

ggtitle("Median RTs upper") +

ylab("RT (s)") + xlab("") +

scale\_x\_discrete(labels = new\_x)

p3 <- ggplot(d\_speed\_acc\_agg2, aes(x = condition:frequency)) +

geom\_linerange(aes(ymin = medrt.lower\_lll, ymax = medrt.lower\_hhh),

col = "darkgrey") +

geom\_linerange(aes(ymin = medrt.lower\_ll, ymax = medrt.lower\_hh),

size = 2, col = "grey") +

geom\_point(aes(y = medrt.lower\_median), shape = 1) +

geom\_point(aes(y = medrt.lower), shape = 4, col = "red") +

ggtitle("Median RTs lower") +

ylab("RT (s)") + xlab("") +

scale\_x\_discrete(labels = new\_x)

grid.arrange(p1, p2, p3, ncol = 2)

Figure 2.5.2 Plot of the Posterior Predictive Distribution

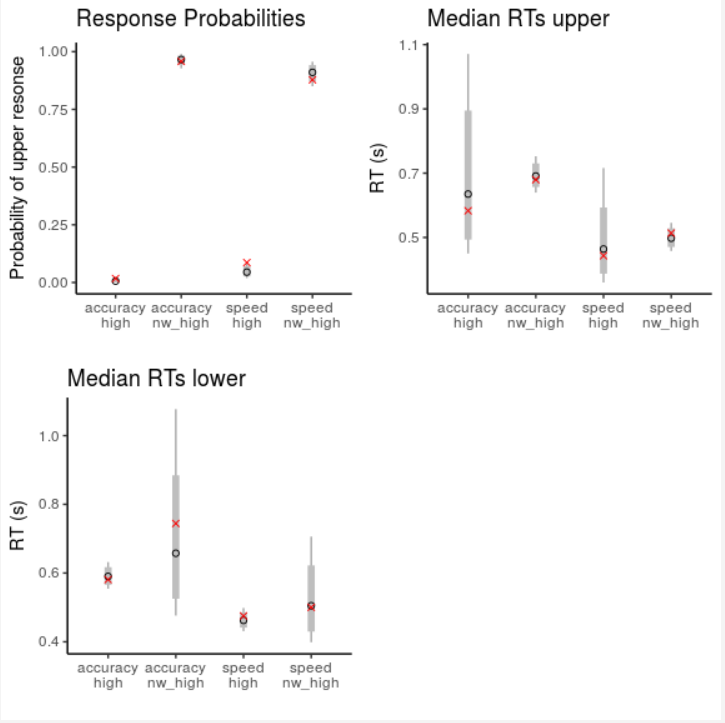


Figure 2.5.3 Graph of the Measures of the Posterior Predictive Distribution

There is no dramatic misalignment in the plots, according to a close examination. Overall, the model seems to be able to describe the broad trends in the data. Only the response probabilities for words with a high frequency seem to be inflated. The red x appears to be outside the 95% and 80% confidence intervals, but it's also possible that it's outside both of those. The RT plots display an interesting (yet predictable) pattern. For uncommon responses (i.e., "word" responses to upper/non-word stimuli and "nonword" responses to lower/word stimuli), the posterior predictive distributions are comparatively wide. On the other hand, the posterior predictive distributions for the typical responses are quite constrained. Each time, the observed median is very close to the predicted median and falls within the 80% confidence interval.

**2.6 Individual Level Fit**

We look at predicted response probabilities on an individual level to further investigate the pattern. We plot the response probabilities in the same manner as before, but this time by participant id.

ggplot(speed\_acc\_agg, aes(x = condition:frequency)) +

geom\_linerange(aes(ymin = prob.upper\_lll, ymax = prob.upper\_hhh),

col = "darkgrey") +

geom\_linerange(aes(ymin = prob.upper\_ll, ymax = prob.upper\_hh),

size = 2, col = "grey") +

geom\_point(aes(y = prob.upper\_median), shape = 1) +

geom\_point(aes(y = prob.upper), shape = 4, col = "red") +

facet\_wrap(~id, ncol = 3) +

ggtitle("Prediced (in grey) and observed (red) response probabilities by ID") +

ylab("Probability of upper resonse") + xlab("") +

scale\_x\_discrete(labels = new\_x)

Figure 2.6.1 Plot of the Predicted Response Probabilities

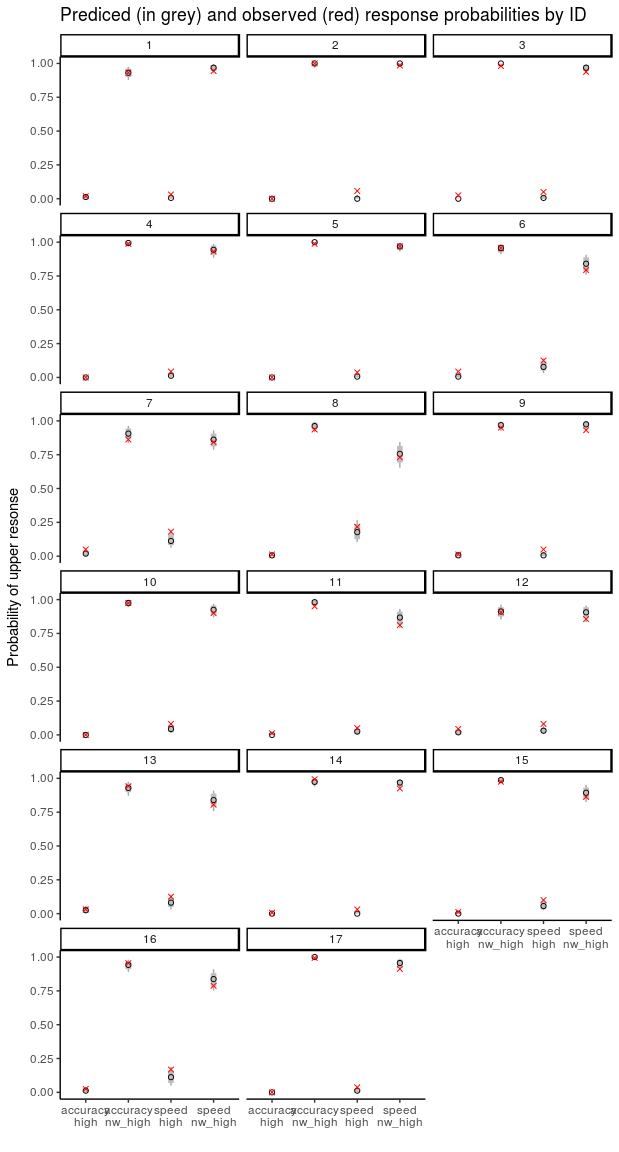


Figure 2.6.2 Graph of the Predicted Response Probabilities

The graph above exhibits the same pattern as the compiled data. We don't notice any obvious outcasts among the participants. Additionally, it appears that response probabilities to non-word stimuli can be fairly predicted. On the other hand, it is predicted that response probabilities to word stimuli will be lower than what has been found. On the other hand, this misfit doesn't seem to be particularly strong. The coverage probabilities of our three measures across individuals are then looked at. That is, we determine the percentage of participants whose observed test statistics fall within each CI (50 percent, 80 percent, 95 percent, and 99 percent) for each measure, design cell, and CI.

speed\_acc\_agg %>%

mutate(prob.upper\_99 = (prob.upper >= prob.upper\_llll) &

(prob.upper <= prob.upper\_hhhh),

prob.upper\_95 = (prob.upper >= prob.upper\_lll) &

(prob.upper <= prob.upper\_hhh),

prob.upper\_80 = (prob.upper >= prob.upper\_ll) &

(prob.upper <= prob.upper\_hh),

prob.upper\_50 = (prob.upper >= prob.upper\_l) &

(prob.upper <= prob.upper\_h),

medrt.upper\_99 = (medrt.upper > medrt.upper\_llll) &

(medrt.upper < medrt.upper\_hhhh),

medrt.upper\_95 = (medrt.upper > medrt.upper\_lll) &

(medrt.upper < medrt.upper\_hhh),

medrt.upper\_80 = (medrt.upper > medrt.upper\_ll) &

(medrt.upper < medrt.upper\_hh),

medrt.upper\_50 = (medrt.upper > medrt.upper\_l) &

(medrt.upper < medrt.upper\_h),

medrt.lower\_99 = (medrt.lower > medrt.lower\_llll) &

(medrt.lower < medrt.lower\_hhhh),

medrt.lower\_95 = (medrt.lower > medrt.lower\_lll) &

(medrt.lower < medrt.lower\_hhh),

medrt.lower\_80 = (medrt.lower > medrt.lower\_ll) &

(medrt.lower < medrt.lower\_hh),

medrt.lower\_50 = (medrt.lower > medrt.lower\_l) &

(medrt.lower < medrt.lower\_h)

) %>%

group\_by(condition, frequency) %>% ## grouping factors without id

summarise\_at(vars(matches("\\d")), mean, na.rm = TRUE) %>%

gather("key", "mean", -condition, -frequency) %>%

separate("key", c("measure", "ci"), "\_") %>%

spread(ci, mean) %>%

as.data.frame()

# condition frequency measure 50 80 95 99

# 1 accuracy high medrt.lower 0.706 0.8824 0.882 1.000

# 2 accuracy high medrt.upper 0.500 0.8333 1.000 1.000

# 3 accuracy high prob.upper 0.529 0.7059 0.765 0.882

# 4 accuracy nw\_high medrt.lower 0.500 0.8125 0.938 0.938

# 5 accuracy nw\_high medrt.upper 0.529 0.8235 1.000 1.000

# 6 accuracy nw\_high prob.upper 0.529 0.8235 0.941 0.941

# 7 speed high medrt.lower 0.471 0.8824 0.941 1.000

# 8 speed high medrt.upper 0.706 0.9412 1.000 1.000

# 9 speed high prob.upper 0.000 0.0588 0.588 0.647

# 10 speed nw\_high medrt.lower 0.706 0.8824 0.941 0.941

# 11 speed nw\_high medrt.upper 0.471 0.7647 1.000 1.000

# 12 speed nw\_high prob.upper 0.235 0.6471 0.941 1.000

Figure 2.6.3 Coverage Probabilities of the Measures

The coverage probability for the RTs is typically equal to or even higher than the width of the CIs, as can be seen. Additionally, for all 99 percent CIs, the coverage probability for the common response (i.e., upper for frequency = nw high and lower for frequency = high) is 1. Unfortunately, there is not much good coverage for response probabilities. specifically for tighter CIs and high-speed conditions. But for the wide CIs, the coverage probabilities are at least tolerable. According to the results so far, the model offers a sufficient account. Overall, it provides a satisfactory account, but there are some misfits to be aware of if one wishes to extend the model or fit it to new data.

**2.7 QQ-Plots: RTs**

Since we have only examined the median and the fifth quantile of the RT distribution thus far, the final method for evaluating model fit will be based on more quantiles of the RT distribution. Quantiles for each individual will then be plotted across conditions as observed versus predicted (i.e., mean from posterior predictive distribution). To do this, we first aggregate across samples and then compute the quantiles per sample from the posterior predictive distribution. This is achieved by using tidyr::unnest to unstack the columns and dplyr::summarise at with a list column. The observed RT quantiles are then added to the aggregated predicted RT quantiles.

quantiles <- c(0.1, 0.25, 0.5, 0.75, 0.9)

pp2 <- d\_speed\_acc %>%

group\_by(id, condition, frequency) %>% # select grouping vars

summarise\_at(.vars = vars(starts\_with("V")),

funs(lower = list(rownames\_to\_column(

data.frame(q = quantile(abs(.[. < 0]), probs = quantiles)))),

upper = list(rownames\_to\_column(

data.frame(q = quantile(.[. > 0], probs = quantiles ))))

)) %>%

ungroup %>%

gather("key", "value", -id, -condition, -frequency) %>% # remove grouping vars

separate("key", c("rep", "boundary"), sep = "\_") %>%

unnest(value) %>%

group\_by(id, condition, frequency, boundary, rowname) %>% # grouping vars + new vars

summarise(predicted = mean(q, na.rm = TRUE))

rt\_pp <- speed\_acc %>%

group\_by(id, condition, frequency) %>% # select grouping vars

summarise(lower = list(rownames\_to\_column(

data.frame(observed = quantile(rt[response == "word"], probs = quantiles)))),

upper = list(rownames\_to\_column(

data.frame(observed = quantile(rt[response == "nonword"], probs = quantiles ))))

) %>%

ungroup %>%

gather("boundary", "value", -id, -condition, -frequency) %>%

unnest(value) %>%

left\_join(pp2)

Figure 2.7.1 Calculation of quantiles per sample from posterior predictive distribution & Aggregation across samples

We calculate the concordance correlation coefficient for each cell and quantile to evaluate the consistency between observed and predicted quantiles. The CCC is more appropriate than simple correlation because it measures the degree of absolute agreement between two values. A -1 correlation with the same mean and variance of the two variables is represented by a scale from -1 to 1, with 1 denoting perfect agreement, 0 denoting no relationship, and -1 denoting a -1 correlation. With respect to responses to the upper and lower boundaries, the code below creates QQ-plots for each condition and quantile separately. The value in the upper left corner of each plot indicates the CCC measures of absolute agreement.

plot\_text <- rt\_pp %>%

group\_by(condition, frequency, rowname, boundary) %>%

summarise(ccc = format(

CCC(observed, predicted, na.rm = TRUE)$rho.c$est,

digits = 2))

p\_upper <- rt\_pp %>%

filter(boundary == "upper") %>%

ggplot(aes(x = observed, predicted)) +

geom\_abline(slope = 1, intercept = 0) +

geom\_point() +

facet\_grid(condition+frequency~ rowname) +

geom\_text(data=plot\_text[ plot\_text$boundary == "upper", ],

aes(x = 0.5, y = 1.8, label=ccc),

parse = TRUE, inherit.aes=FALSE) +

coord\_fixed() +

ggtitle("Upper responses") +

theme\_bw()

p\_lower <- rt\_pp %>%

filter(boundary == "lower") %>%

ggplot(aes(x = observed, predicted)) +

geom\_abline(slope = 1, intercept = 0) +

geom\_point() +

facet\_grid(condition+frequency~ rowname) +

geom\_text(data=plot\_text[ plot\_text$boundary == "lower", ],

aes(x = 0.5, y = 1.6, label=ccc),

parse = TRUE, inherit.aes=FALSE) +

coord\_fixed() +

ggtitle("Lower responses") +

theme\_bw()

grid.arrange(p\_upper, p\_lower, ncol = 1)

Figure 2.7.2 QQ-plot of the Quantiles

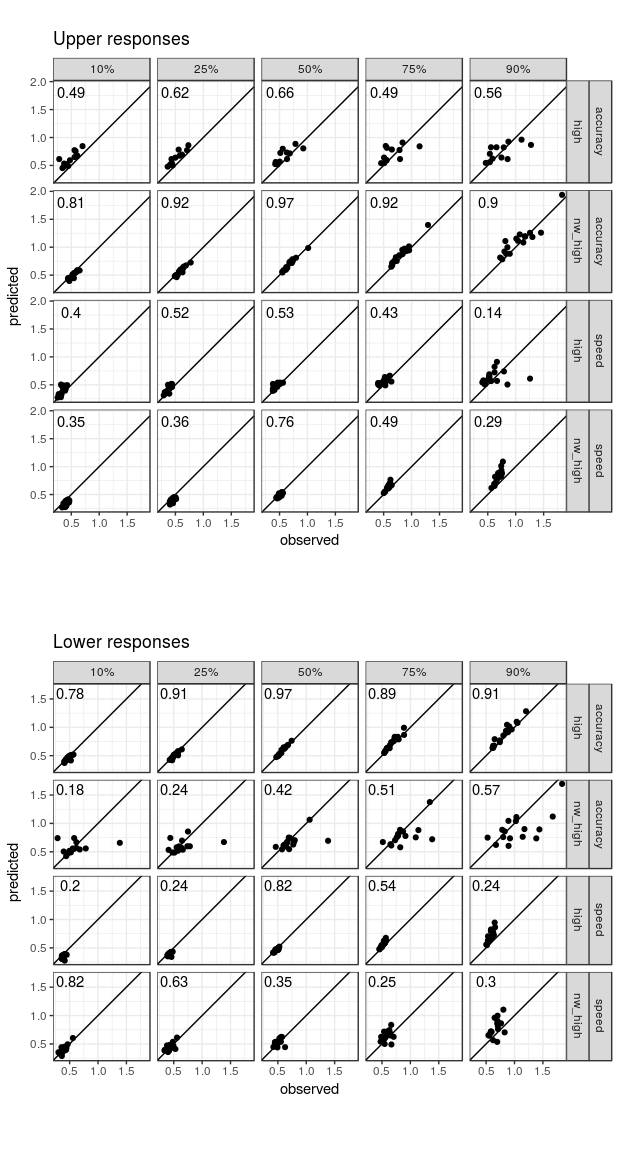


Figure 2.7.3 Graph of QQ-plot of the Quantiles with Upper and Lower response

**Part III – Parameter Estimation & Hypothesis Tests**

**3.1 Introduction**

The third section in a series on fitting the Digital Distribution 4-parameter model with brms is now available. How to set up the data and the model was covered in the first section. The evaluation of the model's suitability (i.e., fit) and (mostly graphical) model diagnostics were covered in the second section. The third section will look at the parameter estimates of the model to see if there is any proof that the conditions are different. As before, running this part doesn't require running the code from Parts I or II because it is entirely self-contained. Given the length of this section, I will give a succinct summary. We give a brief explanation of our methodology for hypothesis testing in the section that follows. The fitted model object and a short section that loads some packages are then loaded, and the model is briefly reviewed after that. This is followed by a lengthy section that explores various aspects of the drift rate parameters. The other three parameters will then be examined one at a time. There will be particular value in the section on non-decision time. I think it's impossible to interpret this parameter, as I'll explain further down. Finally, I give a brief overview of some of the shortcomings of the current model and some suggestions for how to make it better.

**3.2 Bayesian Hypothesis Testing**

This section demonstrates that the parameter estimates for the various conditions vary. It's important to note that the various ways of obtaining this proof are only being discussed in a technical sense. In terms of statistics, we will always look at the difference distributions produced by linear combinations of the posterior cell-wise distributions of the estimated group-level model parameters. It is common to refer to the difference between two distributions by the slightly technical term "linear combinations of cell-wise posterior distributions". For instance, the posterior of the accuracy condition is subtracted from the posterior of the speed condition to produce the difference distribution. A posterior distribution, to restate, is the probability distribution of a parameter based on data and model (where the latter includes the parameter priors). Which parameters are most likely given our prior knowledge and data is a question it answers. As a result, the posterior distribution of the difference provides answers to issues like the likelihood of the values of the difference between two conditions. Given the disparity in distribution, there are two things we can do:

* First, we can check to see if the credibility interval or highest posterior density (HPD) of the difference distribution includes 0. A plausible value might be 0 if it falls within the 95 percent HPD interval. If 0 is outside of the 95 percent range, we might think it's not plausible enough and come to the conclusion that there is evidence for a difference.
* Secondly, we can calculate the proportion of the difference distribution that lies on one side of zero and the other. If this number significantly differs from 50%, there is a difference. For instance, there is strong evidence that a difference is greater than zero if all of the posterior samples for that difference are higher than zero.

One technique for hypothesis testing in a Bayesian context is the examination of posterior distributions to gauge differences between conditions. It is also not the most widely accepted theory, at least not in the psychological literature. In the psychological literature, many of the more outspoken proponents of Bayesian statistics support Bayes factor hypothesis testing. In general, I concur with many of the justifications for the Bayes factor, especially in situations like this one where all pertinent hypotheses or rival models are nested inside a single large (super) model. The biggest problem with Bayes factors is how sensitive they are to parameter priors. This is not a serious problem for nested models. It has been modified to include general ANOVA designs. Models of accumulators have used it. The general concept is to reparametrize the model using normalised effect parameters, such as residual variance. Consider parameterizing the model with a standardised difference for a two-sample design. The standardised effect size measure can then be prioritised in a straightforward and consensus-based manner. The variance estimate of the group-level distribution for each parameter could be used to achieve this normalisation in the current scenario, where the model lacks a residual variance parameter.

Unfortunately, as far as I'm aware, brms does not allow for the specification of a parameterization and prior distribution in accordance with the default Bayes factor. Additionally, it's unlikely that brms will ever acquire this skill. As a result, I don't think brms is the best method for choosing a model that uses Bayes factors. Although it technically does so now (through our bridge sampling package), it only works with models that have unnormalized parameterization. Because the priors cannot be specified in a 'default' way, I think that such a parameterization is inappropriate for Bayes factors-based model selection in most instances. Therefore, at this time, I am unable to suggest brms for Bayes factor-based model selection. In conclusion, it is due to practical constraints rather than philosophical considerations that we are relying solely on posterior distributions for our inferences in this situation.

For the psychological audience, one last word of warning. Although Bayes factors are undoubtedly common in psychology, few other scientific fields use them. I believe that the variation is caused by the various types of data that are used by various individuals. Tests for the presence of effects (or nullity) when working with observational (or correlational) data are either uninteresting or a no-no when trying to infer causal relationships, for example. We all understand that arbitrary relationships, particularly small ones, are prevalent in the real world. As a result, merely increasing N to obtain effects is uninteresting, and estimation is a better strategy. On the other hand, testing true null hypotheses that frequently exist for experimental data makes a lot of sense. But from what I can tell, the impact is utterly negligible. Hypothesis testing is crucial in this situation.

**3.3 Getting Started**

In order to start, we load a few packages for posterior analysis. We've imported the entire tidyverse because I've grown to love it since the start of this series. Of course, we also need brms. We will also need a few more packages, as shown below (especially emmeans and tidybayes), but these are only loaded when they are required. The posterior samples are then required, which we can load similarly to Part II by loading into the Temporary folder, as shown in Figure 2.2.3.

library("brms")

library("tidyverse")

theme\_set(theme\_classic()) # theme for ggplot2

options(digits = 3)

tmp <- tempdir()

download.file("https://github.com/binmishr/Digital-Distribution-Model/blob/main/brms\_DDM\_example\_fit.rda",

file.path(tmp, "brms\_DDM\_example\_fit.rda"))

load(file.path(tmp, "brms\_DDM\_example\_fit.rda"))

Figure 3.3.1 Loading of R libraries & Data file into Temp Folder

# Estimate Est.Error l-95% CI u-95% CI

# conditionaccuracy:frequencyhigh -2.944 0.1971 -3.345 -2.562

# conditionspeed:frequencyhigh -2.716 0.2135 -3.125 -2.299

# conditionaccuracy:frequencynw\_high 2.238 0.1429 1.965 2.511

# conditionspeed:frequencynw\_high 1.989 0.1785 1.626 2.332

# bs\_conditionaccuracy 1.898 0.1448 1.610 2.186

# bs\_conditionspeed 1.357 0.0813 1.200 1.525

# ndt\_conditionaccuracy 0.323 0.0173 0.289 0.358

# ndt\_conditionspeed 0.262 0.0154 0.232 0.293

# bias\_conditionaccuracy 0.471 0.0107 0.449 0.491

# bias\_conditionspeed 0.499 0.0127 0.474 0.524

# Warning message:

# There were 7 divergent transitions after warmup. Increasing adapt\_delta above 0.8 may help.

Figure 3.3.2 Sample data of Group Level Posteriors

Recall that our data come from a lexical decision task in which participants are asked to determine whether or not presented strings are words. Frequency is the factor that establishes a string's true status, with high denoting words and nw\_high denoting non-words. The drift rate for words is clearly negative (rows 1 and 2), meaning that those trials mostly hit the lower boundary for the word decision, and the drift rate for non-words is clearly positive (rows 3 and 4), meaning that those trials mostly hit the upper boundary for non-word decisions. As a result, the frequency factor determines the sign of the parameter estimates for the drift rate (the first four rows in the results table). Furthermore, depending on the accuracy or speed conditions, there might be variations in drift rates. Particularly, compared to the accuracy conditions, the drift rates seem to be less extreme (i.e., closer to 0).

The condition factor is the only distinction between the other three parameters. We expect variations in the boundary separation parameters starting with bs\_ given the experimental manipulation of the accuracy versus speed condition. The small 95 percent overlap suggests that the non-decision time, parameters starting with ndt\_, only has a minor influence. We must be cautious when interpreting this distinction, as will be covered in more detail below. In addition, bias parameters with the prefix bias\_ may or may not differ. Furthermore, at least in the accuracy condition, there seems to be a bias for "word" responses.

To test for differences between conditions, use brms' hypothesis function. But using the current model, I was unable to make it work. This, in my opinion, is caused by the somewhat unusual parameterizations where each cell is given a single parameter (each cell, in a sense, has its own intercept, but there is no overall intercept). Contrarily, in a more "standard" parameterization, there is only one intercept (for either the unweighted means or one of the cells), and the remaining parameters are used to measure how the intercept differs from the means of the cells. Recall that the reason I initially selected this non-standard parameterization was to make it simpler to specify the parameter priors. Furthermore, this is a typical parameterization when manually programming cognitive models.

**3.4 emmeans & tidybayes : Differences In Drift Rate**

Utilising the top-notch emmeans package is an additional choice. Regardless of whether I use frequentist (e.g., via afex) or Bayesian methods (e.g., rstanarm or brms), I am a huge fan of emmeans and use it frequently when working with "normal" statistical models (e.g., ANOVAs, mixed models). Sadly, it seems that for models estimated with brms, emmeans can currently only analyse the key response distribution parameter, which in our case is the drift rate. In any case, I highly suggest that you read through the emmeans vignettes to get a sense of the different kinds of follow-up tests that are all possible with this fantastic package.

I've just learned that emmeans works well with tidybayes, a package that lets you work with posterior draws in the tidyverse. For a package with such limited functionality, tidybayes has an unusually large package footprint (i.e., it imports a sizable number of other packages). This is, I suppose, what it means to be a part of the tidyverse. In any case, after loading the tidyverse above, many of the imported packages are already in the search path, so attaching should be simple.

Only to make sure that everything functions as expected, we begin with emmeans. The results of the fixed effects for the central tendency estimate, which in both cases is the median of the posterior samples, are the estimated marginal means plus 95%-highest posterior density (HPD) intervals. Recall that the unusual parameterization, which emmeans correctly detects, is the reason why the cell estimates and parameter estimates match. Due to the different approaches used to calculate the intervals (i.e., quantiles versus HPD intervals), the lower and upper bounds of the intervals differ slightly between the summary outputs of brms and emmeans.

library("emmeans")

library("tidybayes")

Figure 3.4.1 Loading of the R Libraries

fit\_DDM %>%

emmeans( ~ condition\*frequency)

# condition frequency emmean lower.HPD upper.HPD

# accuracy high -2.94 -3.34 -2.56

# speed high -2.72 -3.10 -2.28

# accuracy nw\_high 2.24 1.96 2.50

# speed nw\_high 1.99 1.64 2.34

#

# HPD interval probability: 0.95

Figure 3.4.2 Extracting Data with HPD Interval Probability

**3.5 HPD Intervals & Histograms**

We are looking for evidence of a difference in the conditions for speed and accuracy for both words (frequency = high) and non-words (frequency = nw\_high) as a first test. This can be done in a number of ways with emmeans, one of which is by using the by argument and pairs function. Given that both HPD intervals contain 0, there isn't much evidence to suggest that there is a difference for either stimulus type in this situation.

fit\_DDM %>%

emmeans("condition", by = "frequency") %>%

pairs

# frequency = high:

# contrast estimate lower.HPD upper.HPD

# accuracy - speed -0.225 -0.6964 0.256

#

# frequency = nw\_high:

# contrast estimate lower.HPD upper.HPD

# accuracy - speed 0.249 -0.0647 0.550

#

# HPD interval probability: 0.95

Figure 3.5.1 Difference in speed and accuracy conditions for both (frequency = high) and non-words (frequency = nw\_high).

To extract the samples neatly, we can use tidybayes instead of emmeans to obtain the distribution's summary. A useful aggregation function provided by tidybayes is then used to group the samples based on the same conditioning variable. After trying a few different approaches, I've concluded that emmeans' hpd.summary() function calculates HPD intervals using the same methodology as tidybayes because the two results are consistent.

samp1 <- fit\_DDM %>%

emmeans("condition", by = "frequency") %>%

pairs %>%

gather\_emmeans\_draws()

samp1 %>%

median\_hdi()

# # A tibble: 2 x 8

# # Groups: contrast [1]

# contrast frequency .value .lower .upper .width .point .interval

# <fct> <fct> <dbl> <dbl> <dbl> <dbl> <chr> <chr>

# 1 accuracy - speed high -0.225 -0.696 0.256 0.95 median hdi

# 2 accuracy - speed nw\_high 0.249 -0.0647 0.550 0.95 median hdi

Figure 3.5.2 Calculating HPD Intervals with Median

samp1 %>%

mode\_hdi()

# # A tibble: 2 x 8

# # Groups: contrast [1]

# contrast frequency .value .lower .upper .width .point .interval

# <fct> <fct> <dbl> <dbl> <dbl> <dbl> <chr> <chr>

# 1 accuracy - speed high -0.190 -0.696 0.256 0.95 mode hdi

# 2 accuracy - speed nw\_high 0.252 -0.0647 0.550 0.95 mode hdi

Figure 3.5.3 Calculating HPD Intervals with Mode

get\_hdi <- function(x, level = 95) {

tmp <- hdrcde::hdr(x, prob = level)

list(data.frame(mode = tmp$mode[1], lower = tmp$hdr[1,1], upper = tmp$hdr[1,2]))

}

samp1 %>%

summarise(hdi = get\_hdi(.value)) %>%

unnest

# # A tibble: 2 x 5

# # Groups: contrast [1]

# contrast frequency mode lower upper

# <fct> <fct> <dbl> <dbl> <dbl>

# 1 accuracy - speed high -0.227 -0.712 0.247

# 2 accuracy - speed nw\_high 0.249 -0.0616 0.558

Figure 3.5.4 Calculating HPD Intervals For Sample Point Estimation with Mode and User Defined Function

samp1 %>%

summarise(hdi = get\_hdi(.value, level = 80)) %>%

unnest

# # A tibble: 2 x 5

# # Groups: contrast [1]

# contrast frequency mode lower upper

# <fct> <fct> <dbl> <dbl> <dbl>

# 1 accuracy - speed high -0.212 -0.540 0.0768

# 2 accuracy - speed nw\_high 0.246 0.0547 0.442

Figure 3.5.5 Calculating HPD Intervals For Sample Point Estimation with Mode and User Defined Function with Level 80%

Now that we have the samples in a convenient format, we can evaluate whether there is evidence for a drift rate difference between conditions for both word and non-word stimuli. The fact that the direction of the effect differs for words and non-words is one problem with this. This is because responses to word stimuli are required at the lower decision boundary, whereas responses to non-word stimuli are required at the upper decision boundary. As a result, we must multiply the effect by -1 for one of the conditions. The average of both conditions can then be calculated. Using tidyverse magic, we achieve this and update the table with the total number of values aggregated in this way. This is merely a verification that our reasoning is sound and that we consistently aggregate exactly two values. The last cheque verifies this.

samp2 <- samp1 %>%

mutate(val2 = if\_else(frequency == "high", -1\*.value, .value)) %>%

group\_by(contrast, .draw) %>%

summarise(value = mean(val2),

n = n())

all(samp2$n == 2)

# [1] TRUE

Figure 3.5.6 Calculation of Drift Rate Difference between conditions for both word and non-word stimuli

The difference distribution that results can then be studied. One tool for doing this graphically is a histogram. It's wise to try different bin counts until the figure looks just right. It seemed reasonable to use 75 bins given the large number of samples. With fewer bins, the granularity was insufficient, and with more bins, the number of small peaks was excessive.

ggplot(samp2, aes(value)) +

geom\_histogram(bins = 75) +

geom\_vline(xintercept = 0)

Figure 3.5.7 Code Plot of the Histogram of Drift Rate Difference between conditions for both word and non-word stimuli

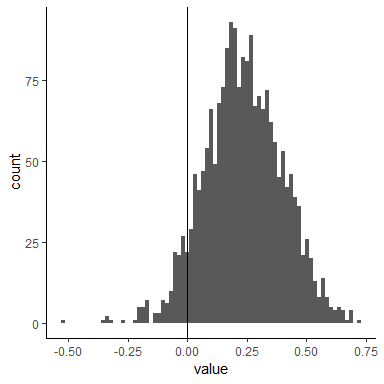


Figure 3.5.8 Histogram of Drift Rate Difference between conditions for both word and non-word stimuli

A significant portion of the posterior mass is to the right of 0, as shown by the histogram in figure 3.5.8, but a significant portion is still to the left. Therefore, even when words and non-words are taken into account together, there is some evidence for a difference, but it is not very strong. This difference distribution can also be studied using the HPD intervals. Take into account the following interval sizes to get a clearer picture. This demonstrates that 0 is only disregarded for intervals smaller than 85 percent. Use hdrcde::hdr.den instead of hdrcde::hdr.den to get a graphical summary of the output.

hdrcde::hdr(samp2$value, prob = c(99, 95, 90, 80, 85, 50))

# $`hdr`

# [,1] [,2]

# 99% -0.1825 0.669

# 95% -0.0669 0.554

# 90% -0.0209 0.505

# 85% 0.0104 0.471

# 80% 0.0333 0.445

# 50% 0.1214 0.340

#

# $mode

# [1] 0.225

#

# $falpha

# 1% 5% 10% 15% 20% 50%

# 0.116 0.476 0.757 0.984 1.161 1.857

Figure 3.5.9 Difference Distribution via Different HPD Intervals

**3.6 Bayesian P-values**

A method that requires fewer arbitrary cutoffs than HPDs (for which the width must be defined) is calculating the actual proportion of samples below 0. As was already mentioned, if this proportion is low, it would indicate a difference. The percentage of samples that are less than 0 in this instance is.067. Unfortunately, the magical cutoff of.05, which is widely acknowledged as separating small from large numbers, or perhaps more accurately, likely from unlikely probabilities, is slightly higher at.067 than it should be.

mean(samp2$value < 0)

# [1] 0.0665

Figure 3.6.1 Calculating Actual proportion of samples below 0

Let's examine this ratio more closely. The difference distribution that results from placing two posterior distributions exactly on top of one another is centred on 0, with exactly half of the difference distribution on either side of 0. Therefore, a percentage of 50% represents either the weakest evidence for the existence of a difference or, alternatively, the strongest evidence for its absence. Another implication is that there is a difference, even though it is in the opposite direction, between values close to 0 and values close to 1. I advise that these proportions be calculated so that small values serve as proof of a difference in order to make interpretation easier (for example, by deducting the proportion from 1 if it is higher than.5). But what does this ratio actually tell us? It indicates the chance of a difference in a particular direction. It is therefore biassed proof of a difference. In contrast, we deduct 2.5% from both sides of the difference distribution for a 95% HPD. This ratio must be multiplied by two in order to maintain the same two-sidedness as our HPD intervals. This multiplication also has the benefit of expanding the range to include the entire probability scale, from 0 to 1.

Thus, the resulting value is a probability (ranging from 0 to 1), where values close to zero denote evidence in favour of a difference and values close to 1 denote evidence in opposition to a difference. It is therefore a continuous measure of evidence for (when near 0) or against (when near 1) a difference in parameter estimates, unlike a traditional p-value. We could call it a Bayesian p-value, or pB, given its superficial similarity to classical p-values (low values are viewed as evidence for a difference). The pB value for a difference in drift rate across speed and accuracy conditions for word and non-word stimuli is.13 in this instance, indicating that the evidence for a difference is at best weak. Naturally, Bayesian p-values can be misused just like conventional p-values. You could, for instance, use arbitrary cutoff values like.05. Take into consideration for a moment that we want to determine whether there are any differences in the absolute amount of evidence for any of the four design cells as measured by drift rate (I am not suggesting that is particularly sane). In order to preserve the information about the transition from positive to negative drift rates, or the other way around, it would be necessary to transform the posterior for all drift rates onto the same side (note that we do not want to take the absolute values). For instance, add -1 to the word drift rate. After doing this, we look at what the cell means. The drift rate values for words are higher than the values for non-words, as shown by an analysis of the four cell means.

samp3 <- fit\_DDM %>%

emmeans( ~ condition\*frequency) %>%

gather\_emmeans\_draws() %>%

mutate(.value = if\_else(frequency == "high", -1 \* .value, .value),

intera = paste(condition, frequency, sep = "."))

samp3 %>%

mode\_hdi(.value)

# # A tibble: 4 x 8

# # Groups: condition [2]

# condition frequency .value .lower .upper .width .point .interval

# <fct> <fct> <dbl> <dbl> <dbl> <dbl> <chr> <chr>

# 1 accuracy high 2.97 2.56 3.34 0.95 mode hdi

# 2 accuracy nw\_high 2.25 1.96 2.50 0.95 mode hdi

# 3 speed high 2.76 2.28 3.10 0.95 mode hdi

# 4 speed nw\_high 2.00 1.64 2.34 0.95 mode hdi

Figure 3.6.2 Inspection of the four cell means of drift rate values for words and non-words

To get an overview of all pairwise differences using an arbitrary cut-off value, I created two functions that return a compact letter display of all pairwise comparisons. The functions need data in a wide format, where each column corresponds to the draws for one parameter. It's important to note that in order to use these functions, you must first install the package multcompView, which calculates the compact letter display.

get\_p\_matrix <- function(df, only\_low = TRUE) {

# pre-define matrix

out <- matrix(-1, nrow = ncol(df), ncol = ncol(df), dimnames = list(colnames(df), colnames(df)))

for (i in seq\_len(ncol(df))) {

for (j in seq\_len(ncol(df))) {

out[i, j] <- mean(df[,i] < df[,j])

}

}

if (only\_low) out[out > .5] <- 1- out[out > .5]

out

}

cld\_pmatrix <- function(model, pars, level = 0.05) {

p\_matrix <- get\_p\_matrix(model)

lp\_matrix <- (p\_matrix < (level/2) | p\_matrix > (1-(level/2)))

cld <- multcompView::multcompLetters(lp\_matrix)$Letters

cld

}

samp3 %>% ungroup() %>% ## to get rid of unneeded columns

select(.value, intera, .draw) %>%

spread(intera, .value) %>%

select(-.draw) %>% ## we need to get rid of all columns not containing draws

cld\_pmatrix()

# accuracy.high accuracy.nw\_high speed.high speed.nw\_high

# "a" "b" "a" "b"

Figure 3.6.3 Using Functions to Find Compact Letter Display of all Pairwise Comparisons

In a compact letter display, conditions that share a common letter do not differ based on the criterion. Depending on the criterion, conditions that do not share a common letter vary. The brief letter display in this instance is uninformative and just reiterates what we already saw. There are two groups for the drift rates for words and two groups for the drift rates for non-words. In situations with more conditions or more complex difference patterns, compact letter displays can be very informative. We could have also examined all pairwise comparisons using the tidybayes functionality. Prior to calling the compare levels function, ungroup must be used. If we don't, we get a confusing error (the grouping seems to be a result of using emmeans).

samp3 %>%

ungroup %>%

compare\_levels(.value, by = intera) %>%

mode\_hdi()

# # A tibble: 6 x 7

# intera .value .lower .upper .width .point .interval

# <fct> <dbl> <dbl> <dbl> <dbl> <chr> <chr>

# 1 accuracy.nw\_high - accuracy.high -0.715 -1.09 -0.351 0.95 mode hdi

# 2 speed.high - accuracy.high -0.190 -0.696 0.256 0.95 mode hdi

# 3 speed.nw\_high - accuracy.high -0.946 -1.46 -0.526 0.95 mode hdi

# 4 speed.high - accuracy.nw\_high 0.488 0.0879 0.876 0.95 mode hdi

# 5 speed.nw\_high - accuracy.nw\_high -0.252 -0.550 0.0647 0.95 mode hdi

# 6 speed.nw\_high - speed.high -0.741 -1.12 -0.309 0.95 mode hdi

Figure 3.6.4 Using Tidybayes to Find Compact Letter Display of all Pairwise Comparisons

**3.7 Differences In Other Parameters**

We appear to be unable to use emmeans to look at the variations in the other parameter, as was previously discussed. Fortunately, tidybayes still enables you to use gather\_draws or spread\_draws to extract posterior samples in a neat manner. It appears that for either of those, you must pass the names of the specific variables you wish to extract. We get them by utilising get variables:

get\_variables(fit\_DDM)[1:10]

# [1] "b\_conditionaccuracy:frequencyhigh" "b\_conditionspeed:frequencyhigh"

# [3] "b\_conditionaccuracy:frequencynw\_high" "b\_conditionspeed:frequencynw\_high"

# [5] "b\_bs\_conditionaccuracy" "b\_bs\_conditionspeed"

# [7] "b\_ndt\_conditionaccuracy" "b\_ndt\_conditionspeed"

# [9] "b\_bias\_conditionaccuracy" "b\_bias\_conditionspeed"

Figure 3.7.1 Extract Posterior Samples in Tidy Manner

**3.8 Boundary Separation**

We'll analyse the boundary separation using Spread\_draws. The draws are first extracted, after which we immediately compute the difference distribution between the two.

samp\_bs <- fit\_DDM %>%

spread\_draws(b\_bs\_conditionaccuracy, b\_bs\_conditionspeed) %>%

mutate(bs\_diff = b\_bs\_conditionaccuracy - b\_bs\_conditionspeed)

samp\_bs

# # A tibble: 2,000 x 6

# .chain .iteration .draw b\_bs\_conditionaccuracy b\_bs\_conditionspeed bs\_diff

# <int> <int> <int> <dbl> <dbl> <dbl>

# 1 1 1 1 1.73 1.48 0.250

# 2 1 2 2 1.82 1.41 0.411

# 3 1 3 3 1.80 1.28 0.514

# 4 1 4 4 1.85 1.42 0.424

# 5 1 5 5 1.86 1.37 0.493

# 6 1 6 6 1.81 1.36 0.450

# 7 1 7 7 1.67 1.34 0.322

# 8 1 8 8 1.90 1.47 0.424

# 9 1 9 9 1.99 1.20 0.790

# 10 1 10 10 1.76 1.19 0.569

# # ... with 1,990 more rows

Figure 3.8.1 Analysis of Boundary Separation of the Samples

We can still use the same tools, of course. Examine the histogram, for instance. I once again selected 75 bins. We can still use the same tools, of course. Examine the histogram, for instance. I once again selected 75 bins. In general, we can say with some confidence that changing the speed versus accuracy conditions has an impact on the boundary separation in the current data set. Everything went according to plan.

samp\_bs %>%

ggplot(aes(bs\_diff)) +

geom\_histogram(bins = 75) +

geom\_vline(xintercept = 0)

Figure 3.8.2 Histogram Code of Boundary Separation of the Samples

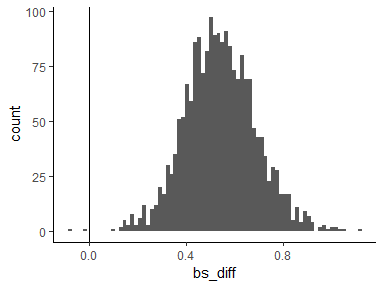


Figure 3.8.3 Histogram Plot Of Boundary Separation of the Samples

sum(samp\_bs$bs\_diff < 0)

# [1] 2

mean(samp\_bs$bs\_diff < 0) \*2

# [1] 0.002

Figure 3.8.4 Sample Data of Boundary Separation of the Samples below 0

**3.9 Non-Decision Time**

To compare differences in non-decision time, we use gather\_draws. This function has one benefit over spread\_draws: it makes getting marginal estimates simple. As mentioned earlier, there is a difference between the conditions because the HPD intervals only very slightly overlap. The resulting marginal estimates are kept for use with new data at a later time.frame. A data frame is ndt\_mean.

samp\_ndt <- fit\_DDM %>%

gather\_draws(b\_ndt\_conditionaccuracy, b\_ndt\_conditionspeed)

(ndt\_mean <- samp\_ndt %>%

median\_hdi())

# # A tibble: 2 x 7

# .variable .value .lower .upper .width .point .interval

# <chr> <dbl> <dbl> <dbl> <dbl> <chr> <chr>

# 1 b\_ndt\_conditionaccuracy 0.323 0.293 0.362 0.95 median hdi

# 2 b\_ndt\_conditionspeed 0.262 0.235 0.295 0.95 median hdi

Figure 3.9.1 Accessing Differences in Non-decision time to obtain Marginal Estimates

I believe that spreading the two variables across rows before calculating the difference is the simplest method to determine the difference (much like starting with spread draws in the first place). Plotting the resulting difference distribution once more.

samp\_ndt2 <- samp\_ndt %>%

spread(.variable, .value) %>%

mutate(ndt\_diff = b\_ndt\_conditionaccuracy - b\_ndt\_conditionspeed)

samp\_ndt2 %>%

ggplot(aes(ndt\_diff)) +

geom\_histogram(bins = 75) +

geom\_vline(xintercept = 0)

Figure 3.9.2 Calculating the spread of two variables across rows and then calculate the difference from results of Figure 3.9.1

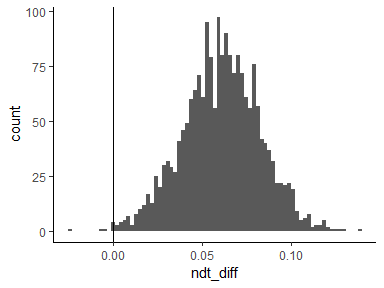


Figure 3.9.3 Plotting Histogram of the results from Figure 3.9.2

As was previously hypothesised, there seems to be strong support for a distinction. Using the Bayesian p-value, we can confirm this further:

mean(samp\_ndt2$ndt\_diff < 0) \* 2

# [1] 0.005

Figure 3.9.4 Confirming Evidence of Difference of results of Figure 3.9.3 via Bayesian p-values

So far, it seems that the manipulation led to the discovery of yet another substantial difference in parameter estimates. But this would be a hasty judgement. In reality, using this method to examine the non-decision time estimated by the 4-parameter Digital Distribution model is wholly deceptive. The non-decision time parameter fails to capture a significant aspect of the response time distribution because it is only sensitive to a small number of data points. The estimated non-decision time specifically reflects a characteristic of the distribution of minimum response times for each participant, condition, or cell. I'll illustrate this using the following example data. The data must first be loaded in the same way as previously discussed posts. Next, the minimum RTs for each participant and condition are determined.

data(speed\_acc, package = "rtdists")

speed\_acc <- droplevels(speed\_acc[!speed\_acc$censor,]) # remove extreme RTs

speed\_acc <- droplevels(speed\_acc[ speed\_acc$frequency %in%

c("high", "nw\_high"),])

min\_val <- speed\_acc %>%

group\_by(condition, id) %>%

summarise(min = min(rt))

Figure 3.9.5 Calculating Minimum RTs per participant and Conditions

We want to investigate the problem by visually contrasting the minimum RT distribution with non-decision time estimates. In order to do this, we must add a condition column to the ndt\_mean data.frame that contains condition names that match. A single plot can then be created by combining them. A few summary statistics on the distribution of specific minimum RTs are also provided. The blue + and blue x represent the median and mean of the individual minimum RTs; the blue circle represents the midpoint between the largest and smallest value of the minimum RT distributions; the red square represents the point estimate of the non-decision time parameter with corresponding 95 percent HPD intervals. The black points represent the individual minimum RTs for each of the two conditions. These have been depicted in Figure 3.9.7

ndt\_mean$condition <- c("accuracy", "speed")

ggplot(min\_val, aes(x = condition, y = min)) +

geom\_jitter(width = 0.1) +

geom\_pointrange(data = ndt\_mean,

aes(y = .value, ymin = .lower, ymax = .upper),

shape = 15, size = 1, color = "red") +

stat\_summary(col = "blue", size = 3.5, shape = 3,

fun.y = "median", geom = "point") +

stat\_summary(col = "blue", size = 3.5, shape = 4,

fun.y = "mean", geom = "point") +

stat\_summary(col = "blue", size = 3.5, shape = 16,

fun.y = function(x) (min(x) + max(x))/2,

geom = "point")

Figure 3.9.6 Comparison of the distribution of Minimum RTs with Estimates for Non-Decision Times

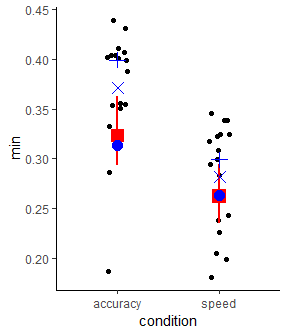


Figure 3.9.7 Graph Plot of the distribution of Minimum RTs with Estimates for Non-Decision Times

Figure 3.9.7's graph demonstrates that the estimated non-decision time (the blue dot) almost exactly corresponds to the middle point between the largest and smallest minimum RT. Consider comparing the number of minimum data points (i.e., the number of participants) to the overall number of data points to put this in perspective.

speed\_acc %>%

group\_by(condition) %>%

summarise(n())

# # A tibble: 2 x 2

# condition `n()`

# <fct> <int>

# 1 accuracy 5221

# 2 speed 5241

length(unique(speed\_acc$id))

# [1] 17

17 / 5000

# [1] 0.0034

Figure 3.9.8 Comparing the Number of data points with Total number of data points

This demonstrates that less than.5% of the data largely determine the non-decision time parameter, one of only four model parameters. A single response time can have a significant impact on the parameter estimate if one of these minimum RTs is an outlier (which seems likely under the accuracy condition). In other words, the current implementation makes it unlikely that the non-decision time parameter accurately captures a latent process. Instead, it merely represents the midpoint of the minimum RTs for each participant and condition, with a slight weighting towards the mass of the minimum RT distribution. It is not advised to use this parameter estimate to make any inferences.

This error does not seem to be particularly significant in the current situation. The current parameter estimates understate the true difference if only one of the data points in the accuracy condition is an outlier and the other data points are accurate depictions of the leading edge of the response time distribution (which is essentially what the non-decision time is supposed to capture). Using a more reliable ad hoc measure of the leading edge, specifically the 10% trimmed mean of the 40 fastest RTs per participant and condition plotted below, further supports this conclusion. Additionally, there are no longer any obvious outliers in this graph. For reference, the non-decision time estimates are still present. The interpretation of non-decision times obtained with such a model is not advised because having a parameter that is essentially driven by very few data points appears to be completely at odds with the general concept of cognitive modelling.

min\_val2 <- speed\_acc %>%

group\_by(condition, id) %>%

summarise(min = mean(sort(rt)[1:40], trim = 0.1))

ggplot(min\_val2, aes(x = condition, y = min)) +

geom\_jitter(width = 0.1) +

stat\_summary(col = "blue", size = 3.5, shape = 3,

fun.y = "median", geom = "point") +

stat\_summary(col = "blue", size = 3.5, shape = 4,

fun.y = "mean", geom = "point") +

stat\_summary(col = "blue", size = 3.5, shape = 16,

fun.y = function(x) (min(x) + max(x))/2,

geom = "point") +

geom\_point(data = ndt\_mean, aes(y = .value), shape = 15,

size = 2, color = "red")

Figure 3.9.9 Plotting of Data Points with 10% trimmed mean of the 40 fastest RTs per participant and condition of Samples

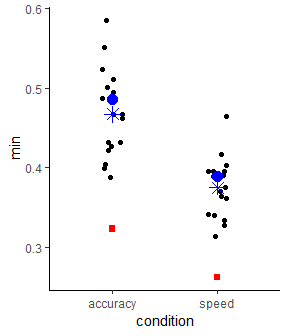


Figure 3.9.10 Graph of Data Points with 10% trimmed mean of the 40 fastest RTs per participant and condition Of Samples

It is significant to note that only the 4-parameter Digital Distribution model as applied here is affected by this confound; other model implementations are not affected. This issue has solutions, two of which I'd like to emphasise here. One could start by including trial variability in non-decision time between trials. Since outliers can be found at the leading edge of the response time distribution, it is frequently assumed that this variability follows a uniform distribution. Second, rather than assuming that all of the responses fit a single model, one might assume that some of the responses are byproducts of other processes, such as random responses from a uniform distribution spanning the RT's minimum and maximum. Technically speaking, this is a uniform distribution model of the process with a free or fixed mixture/contamination rate. A custom\_family should make it relatively easy to implement such a mixture model in brms, and I hope to find the time to do so in the future.

Naturally, I am not the first to notice this 4-parameter Digital Distribution model behaviour. However, the 4-parameter model variant is easily accessible while model variants addressing this problem are not, suggesting that this problem is particularly prevalent in a Bayesian setting. I figured out how to solve this problem most effectively a while back, and one thing I remembered was that we could ignore the non-decision time parameter in the 4-parameter Digital Distribution model. I still think that's the best choice.

I hope there aren't too many papers that interpret variations in the non-decision time parameter using the 4-parameter model in this way.

**3.10 Starting Point / Bias**

Finally, we can look at the bias or starting point. This procedure is repeated using spread\_draws, and the difference distribution produced is plotted.

samp\_bias <- fit\_DDM %>%

spread\_draws(b\_bias\_conditionaccuracy, b\_bias\_conditionspeed) %>%

mutate(bias\_diff = b\_bias\_conditionaccuracy - b\_bias\_conditionspeed)

samp\_bias %>%

ggplot(aes(bias\_diff)) +

geom\_histogram(bins = 100) +

geom\_vline(xintercept = 0)

Figure 3.10.1 Examining the Starting Point or Bias of Sampling Data Distribution

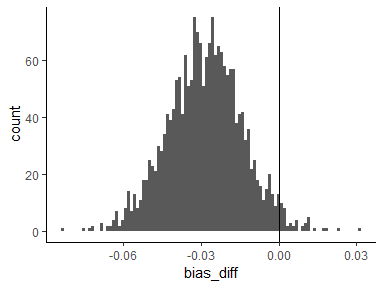


Figure 3.10.2 Histogram of Starting Point or Bias of Sampling Data Distribution

The distributions of the difference suggest that there might be a difference. We then calculate the Bayesian p-value as a result. This time, we calculate the difference in the reverse direction, so small values indicate the presence of a difference.

mean(samp\_bias$bias\_diff > 0) \*2

# [1] 0.046

Figure 3.10.3 Calculating the Bayesian p-values of Sampling Data Distribution

We can now more confidently postulate that there is a bias towards the lower boundary and "word" responses in the accuracy condition, whereas evidence accumulation starts out unbiased in the speed condition. This is due to the evidence for a difference. Our Bayesian p-value is fortunate to be just below.05, which gives us reason to think that the difference is real. Here are a few more estimates to finish:

fit\_DDM %>%

gather\_draws(b\_bias\_conditionaccuracy, b\_bias\_conditionspeed) %>%

summarise(hdi = get\_hdi(.value, level = 80)) %>%

unnest

# # A tibble: 2 x 4

# .variable mode lower upper

# <chr> <dbl> <dbl> <dbl>

# 1 b\_bias\_conditionaccuracy 0.470 0.457 0.484

# 2 b\_bias\_conditionspeed 0.498 0.484 0.516

Figure 3.10.4 Estimating the Starting Point or Bias of Sampling Data Distribution

We identify a particular pattern that is typically thought of as typical for the current data. As shown below, under the accuracy condition, error RTs move much more slowly than correct RTs. In the speed condition, where error RTs are faster than correct RTs, this effect does not exist.

speed\_acc %>%

mutate(correct = stim\_cat == response) %>%

group\_by(condition, correct, id) %>%

summarise(mean = mean(rt),

se = mean(rt)/sqrt(n())) %>%

summarise(mean = mean(mean),

se = mean(se))

# # A tibble: 4 x 4

# # Groups: condition [?]

# condition correct mean se

# <fct> <lgl> <dbl> <dbl>

# 1 accuracy FALSE 0.751 0.339

# 2 accuracy TRUE 0.693 0.0409

# 3 speed FALSE 0.491 0.103

# 4 speed TRUE 0.513 0.0314

Figure 3.10.5 Examining of Error RTs & Correct RTs

It should not come as a surprise that the accuracy condition also has a measurable bias given the difference in the relative speeds of correct and error responses in the accuracy condition. A preference for verbal responses in particular. Although a bias towards words should lead to faster errors, as can be seen by adding stim cat to the aforementioned group by call, the difference in relative error rate is particularly noticeable for non-words. It appears as a result that the current model variant does not fully account for some of the data's more subtle effects.

Using across-trial variability in model parameters is the norm for handling variations in the relative speed of errors in Digital Distribution modelling. Errors can be faster than correct RTs because the starting point can vary. Drift rate variability can cause error RTs to be slower than correct RTs. However, adding these variables to a Bayesian framework comes with its own set of problems, which will be covered in more detail below.

**4.Conclusion**

Overall, the results show that the fit is better for accuracy conditions than for speed conditions. The common response (i.e., nw\_high for upper and high for lower responses) also fits better. Again, this latter observation is not unexpected. It seems that at least the median (i.e., 50%) shows acceptable values for the common response when the fit for the various quantiles is compared. The other quantiles are not well considered, though, particularly in the speed condition. Dramatic misfit is, however, only occasionally observed in responses. Some of the low CCCs in the speed conditions may be explained by the comparatively small variances in some of the cells. For instance, despite some CCC values being low (i.e.,.5) for both common speed conditions (speed and nw\_high for upper responses and speed and high for lower responses), a visual inspection of the plot suggests an acceptable account. We only observe systematic deviations for the 90 percent quantile (and slightly less for the 75 percent quantile) under speed conditions. RTs predicted by the model are slower than those actually seen.

The model has some limitations even though it initially permits estimation. One of them has been covered in this section in great detail. A characteristic of the distribution of minimum RTs is essentially captured by the non-decision time parameter estimate. The estimate becomes meaningless if these are tainted by responses that cannot be assumed to be the outcome of the same process as the other responses, which I believe to be quite likely a priori. I think the risks outweigh the advantages by a wide margin. In the absence of a bias for any of the response options, the model also predicts equal mean response times for correct and incorrect responses. The full (i.e., 7-parameter) model is one of the more highly parameterized model variants that was developed as a result of this theoretical constraint, which is conceivably the most significant one. This series comes to an end, but there are a few additional ideas that seem significant, intriguing, or workable. Here they are:

* A crucial step that we haven't yet taken is to examine the estimates of the group-level parameters, such as correlations and standard deviations. They might include crucial details about the particular data set and research question, but they might also include details about the trade-offs between the model parameters

.

* Substitute a mix of a digital distribution and a uniform distribution for the pure digital distribution process in order to interpret the non-decision time. As was already mentioned, using a custom family in BRMs should make this possible.
* As was previously mentioned, one of the main motivations for modern response time models was differences in the relative speed of error and correct RTs. Usually, model parameter variations are used to explain these. The implementation of these variables in a Bayesian setting can be done quite easily using the hierarchical structure. For instance, each participant receives a by-trial random intercept for the drift rate, + (0+id||trial), and the double bar notation should guarantee that these intercepts are uncorrelated among participants. Although it seems like a straightforward idea, I don't think a model of this kind will converge in a timely manner. Given the theoretical importance of this strategy, it seems like a crucial area to research.
* Fitting the Digital Distribution model takes a long time. It would be interesting to compare the fit using variational Bayes (i.e., posterior parametric approximation), which is also implemented in Stan, with the fit using full Bayesian inference (i.e., sampling as done here). Although I anticipate it won't work, the contrast would be fascinating. Recently, variational Bayes diagnostics were released..

**5.References**

[1] Bürkner, P.-C. (2017). brms: An R Package for Bayesian Multilevel Models Using Stan. Journal of Statistical Software, 80(1), 1–28. https://doi.org/10.18637/jss.v080.i01

[2] Hoffman, M. D., & Gelman, A. (2014). The No-U-turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. J. Mach. Learn. Res., 15(1), 1593–1623. http://dl.acm.org/citation.cfm?id=2627435.2638586

[3] Klauer, K. C. (2010). Hierarchical Multinomial Processing Tree Models: A Latent-Trait Approach. Psychometrika, 75(1), 70–98. https://doi.org/10.1007/s11336-009-9141-0

[4] Monnahan, C. C., Thorson, J. T., & Branch, T. A. (2016). Faster estimation of Bayesian models in ecology using Hamiltonian Monte Carlo. Methods in Ecology and Evolution, n/a-n/a. https://doi.org/10.1111/2041-210X.12681

[5] Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. Journal of Memory and Language, 68(3), 255–278. https://doi.org/10.1016/j.jml.2012.11.001

[6] Rouder, J. N., Morey, R. D., Speckman, P. L., & Province, J. M. (2012). Default Bayes factors for ANOVA designs. Journal of Mathematical Psychology, 56(5), 356–374. https://doi.org/10.1016/j.jmp.2012.08.001

[7] Ratcliff, R. (1978). A theory of memory retrieval. Psychological Review, 85(2), 59–108.

[8] Ratcliff, R., & Smith, P. L. (2004). A Comparison of Sequential Sampling Models for Two-Choice Reaction Time. Psychological Review, 111(2), 333–367. https://doi.org/10.1037/0033-295X.111.2.333

[9] Jones, M., & Dzhafarov, E. N. (2014). Unfalsifiability and mutual translatability of major modelling schemes for choice reaction time. Psychological Review, 121(1), 1–32. https://doi.org/10.1037/a0034190

[10] Rouder, J. N., Haaf, J. M., & Vandekerckhove, J. (2018). Bayesian inference for psychology, part IV: parameter estimation and Bayes factors. Psychonomic Bulletin & Review, 25(1), 102–113. https://doi.org/10.3758/s13423-017-1420-7

[11] Kruschke, J. K. (2015). Doing Bayesian Data Analysis: A Tutorial Introduction with R, JAGS and Stan. Academic Press.