

# Weiner Model

Binayaka Mishra

Project Manager, Competency & Data Analytics, Tech Mahindra, Electronics City, Phase II, Bangalore -560100, Karnataka, India

**Abstract:** This is the three-part series on the Wiener Model analysis, in which I will demonstrate various stages of model implementation using R code and R Package BRMS. The first instalment of the series demonstrates the basics of modelling and estimation. The second instalment of the series will demonstrate how to perform model diagnostics and access the model fit. Finally, the third section demonstrates how to test for differences in parameters between conditions.

**Keywords:** Wiener Model Analysis, Data Science, Bayesian Hypothesis Tests, Wiener Model Fit & Estimation, Wiener Model Diagnostics, Wiener Model Parameter Estimation

## 1. Introduction & Estimation

### 1.1 Introduction

Stan is, in my opinion, the most intriguing development in computational statistics in recent years. The Hamiltonian Monte-Carlo (HMC) version implemented in Stan is extremely efficient, and the range of probability distributions implemented in the Stan language allows for the fitting of an extremely wide range of models. Stan has significantly altered which models I believe can be realistically estimated in terms of both model complexity and data size. It is not an exaggeration to say that Stan (particularly rstan) has significantly altered the way I analyse data.

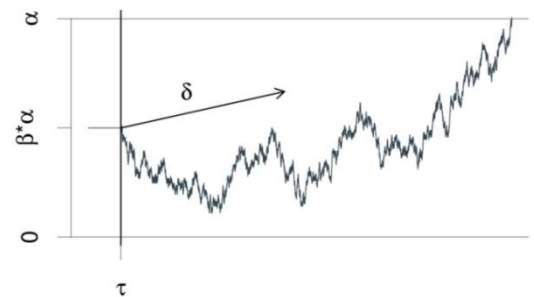
Brms is one of the R packages that allows for the simple implementation of Stan models and has recently gained popularity. Using the R formula interface, it is possible to specify a wide range of models. It generates the model code, compiles it, and then passes it along with the data to rstan for sampling based on the formula and a specification of the model family. I've avoided brms so far because I usually programme my models by hand (thanks to the excellent Stan documentation: [Stan - Documentation \(mc-stan.org\)](https://mc-stan.org)).

However, I recently discovered that brms can estimate the Wiener model for simultaneously accounting for responses and corresponding response times for data from two-choice tasks. Such information is common in psychology, and the model is one of the most popular cognitive models available. In this series, I'll show how to use brms to apply the Wiener model to some published data. The first section explains how to set up and estimate the model. The second section provides an overview of model diagnostics as well as an evaluation of model fit using posterior predictive distributions. The third section demonstrates how to inspect and compare the parameter posterior distributions.

This first part requires brms and a working C++ compiler, as well as the packages RWiener for generating the posterior predictive distribution within brms and rtdists for the data.

Library("brms")

### 1.2 Data & Model



**Figure 1.2.1:** The Wiener model for two-choice reaction times is depicted graphically. An evidence counter begins with the value ' $\alpha \cdot \beta$ ' 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 expect the reader to be familiar with the Wiener model and will only provide a brief overview here; for more information. For binary choice tasks, the Wiener model is a continuous-time evidence accumulation model. It is assumed that evidence is accumulated in each trial by a single accumulator in a noisy process. The accumulation of evidence begins at the start point and continues until the accumulator reaches one of the two decision bounds, at which point the corresponding response is given. The total response time is the sum of the accumulation process's decision time and non-decisional components. To summarise, the Wiener model allows for the decomposition of responses to binary choice tasks and corresponding 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, "binmishr/Weiner-Model-Analysis". The data file which has been used below for the figure 1.2.2 is predictions data and can be downloaded from link [https://github.com/binmishr/Weiner-Model-Analysis/blob/main/brms\\_wiener\\_example\\_predictions.rda](https://github.com/binmishr/Weiner-Model-Analysis/blob/main/brms_wiener_example_predictions.rda). The data comes from 17 participants who completed a lexical decision task in which they had to determine whether a given string was a word or not. In different experimental blocks, participants made decisions based on either speed or accuracy emphasis instructions. To reduce estimation time, we restrict the analysis to high-frequency words (frequency

= high) and the corresponding high-frequency non-words (frequency = nw\_high) after removing some extreme RTs(Response Time). To complete the model, we'll also need a numeric response variable with a value of 0 corresponding to responses at the lower response boundary and a value of 1 corresponding to responses at the upper response boundary. We do this by converting the categorical response variable response to a numeric value and subtracting one, so that a word response corresponds to the lower response boundary and a nonword response corresponds to the upper response boundary.

```
1. data(speed_acc, package = "rtdists")
2. speed_acc <- droplevels(speed_acc[!speed_acc$censor,]) # remove
   extreme RTs
3. speed_acc <- droplevels(speed_acc[ speed_acc$frequency %in%
4.                           c("high", "nw_high"),])
5. speed_acc$response2 <- as.numeric(speed_acc$response)-1
6. str(speed_acc)

1. 'data.frame':   10462 obs. of  10 variables:
2.  $ id          : Factor w/ 17 levels "1","2","3","4",...: 1 1 1 1 1 1 1
   1 1 1 ...
3.  $ block       : Factor w/ 20 levels "1","2","3","4",...: 1 1 1 1 1 1 1
   1 1 1 ...
4.  $ condition: Factor w/ 2 levels "accuracy","speed": 2 2 2 2 2 2 2
   2 2 ...
5.  $ stim        : Factor w/ 1611 levels "1001","1002",...: 1271 46 110
   666 422 ...
6.  $ stim_cat    : Factor w/ 2 levels "word","nonword": 2 1 1 1 1 1 2
   1 1 ...
7.  $ frequency: Factor w/ 2 levels "high","nw_high": 2 1 1 1 1 1 2
   1 1 ...
8.  $ response   : Factor w/ 2 levels "word","nonword": 2 1 1 1 1 1 1
   1 1 ...
9.  $ rt         : num  0.773 0.39 0.435 0.427 0.622 0.441 0.308 0.436
   0.412 ...
10. $ censor     : logi FALSE FALSE FALSE FALSE FALSE FALSE ...
11. $ response2  : num  1 0 0 0 0 0 0 0 0 0 ...
```

Figure 1.2.2: Sample Data from McKoon's. (2008)

### 1.3 Model Formula

The most important decision to make before creating a model is which parameters are allowed to vary between which conditions (i.e., factor levels). One constraint shared by the Wiener model (and other evidence-accumulation models) is that the parameters set before the evidence accumulation process begins (i.e., boundary separation, starting point, and non-decision time) cannot change based on stimulus characteristics unknown to the participant before the trial begins. As a result, the item-type, in this case word

versus non-word, is usually only allowed to influence the drift rate. We adhere to this constraint. Furthermore, as the speed and accuracy conditions are manipulated between blocks of trials, all four parameters are allowed to vary. It's also worth noting that all relevant variables are controlled within-subjects. As a result, the maximal random-effects structure includes random-effects parameters for each fixed-effect. To set up the model, we need to use the bf() function and create one formula for each of the Wiener model's four parameters.

```
1. formula <- bf(rt | dec(response2) ~ 0 + condition:frequency +
2.               (0 + condition:frequency|pid),
3.               bs ~ 0 + condition + (0 + condition|pid),
4.               ndt ~ 0 + condition + (0 + condition|pid),
5.               bias ~ 0 + condition + (0 + condition|pid))
```

Figure 1.3.1: Wiener Model Formula

The first formula is for the drift rate and is also used to specify the column on the left-hand side that contains the RTs (rt) and the response or decision (response2). On the right side, fixed and random effects can be specified in a manner similar to lme4. Because the drift rate is allowed to vary between both variables, condition and frequency (stim cat would be equivalent), we estimate fixed and random effects for both variables as well as their interaction. We must then create one formula for each of the remaining three parameters (which are only allowed to vary by condition). The parameter names are indicated on the left side of these formulas:

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

The fixed- and random-effects are again specified on the right side. It is worth noting that 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 contrast to the current approach in which the response

boundaries represent the actually two response options). In such a case, the starting point cannot be estimated and must be set to 0.5 (i.e., replace the formula with bias = 0.5).

Two more points are important in the formulas. To begin, I used an unusual parameterization and suppressed the intercept (e.g., 0 + condition instead of condition). When an intercept is present, categorical variables (i.e., factors) with k levels are coded with k-1 deviation variables, which represent deviations from the intercept. As a result, in a Bayesian setting, the prior for these deviation variables must be considered. In contrast, when the intercept is suppressed, the model can be configured so that each factor level (or design cell, if more than one factor is involved) receives its own parameter, as shown here. This allows for the same prior for each parameter (as long as one does not expect the parameters to vary dramatically). Furthermore, this is a common parameterization when programming a model, oneself. Compare the following two calls to see the differences between parameterizations (model.matrix is the function that creates the parameterization internally). Only the first generates a unique parameter for each condition.

```
1. unique(model.matrix(~0+condition, speed_acc))
2. ##      conditionaccuracy conditionspeed
3. ## 36              0              1
4. ## 128             1              0
5. unique(model.matrix(~condition, speed_acc))
6. ##      (Intercept) conditionspeed
7. ## 36              1              1
8. ## 128             1              0
```

Figure 1.3.2: Model Matrix Function

It should be noted that if more than one factor is involved and this parameterization is to be used, the factors must be combined using the: and not the \*. This is visible when the code below is executed. Also, when the factors with: are combined without suppressing the intercept, the resulting model has one parameter more than can be estimated (i.e., the model-matrix is rank deficient). As a result, caution is required at this stage. Second, brms formulas allow for the estimation of correlations between random-effects parameters of different formulas. To accomplish this, insert an identifier in the middle of the random-effects formula, separated by | on both sides. Correlations between random-effects formulas will then be estimated for all random-

effects formulas with the same identifier. In our case, we want to use the "latent-trait approach" to estimate the full random-effects matrix with correlations among all model parameters. As a result, we use the same identifier (p) in all formulas. As a result, correlations between all individual-level deviations across all four Wiener parameters will be estimated. Simply omit the identifier (e.g., (0 + condition|id)) to estimate correlations only among the random-effects parameters of each formula. Furthermore, brms, like afex, allows you to suppress correlations between categorical random-effects parameters using || (e.g., (0 + condition||id)).

```
1. unique(model.matrix(~ 0 + condition:frequency, speed_acc))
2. unique(model.matrix(~ 0 + condition*frequency, speed_acc))
3. unique(model.matrix(~ condition:frequency, speed_acc))
```

Figure 1.3.3: Model Matrix Sample Code with Parameters

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

The following step is to configure the priors. To accomplish this, we can use the get\_prior function. This function requires the formula, data, and model family to be specified. The family argument is the one in which we tell brms that we want to use the wiener model. It is also used to define the link function for the four Wiener parameters. Because the

drift rate can be any value (ranging from -Inf to Inf), the default link function is "identity" (i.e., no transformation), which we keep. The other three parameters are all limited in their range. The boundary must be greater than zero, the non-decision time must be greater than zero but less than the smallest RT, and the starting point must be between 0 and 1. These constraints are respected by the default link-functions, which use "log" for the first two parameters and "logit" for

the bias. This is certainly an option, but it has a number of drawbacks that lead me to use the "identity" link function for all parameters. To begin, priors must be specified on the untransformed scale when parameters are transformed. Second, the individual-level deviations (i.e., the random-effects estimates) are assumed to be multivariate normal distributions. Individual deviations are only normally distributed on the untransformed scale if the parameters are transformed. Similarly, correlations of parameter deviations across parameters would be on the untransformed scale as well. Both complicate the interpretation of the random effects. When specifying the parameters without transformation (i.e., link = "identity"), care must be taken to

ensure that the priors place the greatest weight on values within the allowed range. Similarly, starting values must be within the permitted range. Using the identity link function has some drawbacks, which are discussed further below. However, as long as parameters outside the allowed range occur only infrequently, such a model can successfully converge, making interpretation easier. The get prior function returns a data.frame containing all model parameters. If parameters have default priors, these are also listed. Priors must be defined for individual parameters, parameter classes, parameter classes for specific groups, or dpar. It should be noted that all parameters that do not have a default prior should be assigned a specific prior.

```
1. get_prior(formula,
2.           data = speed_acc,
3.           family = wiener(link_bs = "identity",
4.                           link_ndt = "identity",
5.                           link_bias = "identity"))
```

Figure 1.4.1: Configuration of Priors

	group	resp	dpar	prior	class	coef
1.						
2.					b	
3.					b	conditionaccuracy:frequencyhigh
4.					b	conditionaccuracy:frequencynw_high
5.					b	conditionspeed:frequencyhigh
6.					b	conditionspeed:frequencynw_high
7.				lkj(1)	cor	
8.						
9.						
10.						
11.					sd	conditionaccuracy:frequencyhigh
12.					sd	conditionaccuracy:frequencynw_high
13.					sd	conditionspeed:frequencyhigh
14.					sd	conditionspeed:frequencynw_high
15.					b	
16.					b	conditionaccuracy
17.					b	conditionspeed
18.						
19.					sd	
20.					sd	conditionaccuracy
21.					sd	conditionspeed
22.					b	
23.					b	conditionaccuracy
24.					b	conditionspeed

Figure 1.4.2: Output Of get\_prior Function

```
1. prior <- c(
2.   prior("cauchy(0, 5)", class = "b"),
3.   set_prior("normal(1.5, 1)", class = "b", dpar = "bs"),
4.   set_prior("normal(0.2, 0.1)", class = "b", dpar = "ndt"),
5.   set_prior("normal(0.5, 0.2)", class = "b", dpar = "bias")
6. )
```

Figure 1.4.3: Defining the priors with data

With this knowledge, we can use the make\_stancode function to inspect the entire model code. It is critical to ensure that all parameters listed in the parameters block have a prior listed in the model block. We can also see at the start

of the model block that none of our parameters have been transformed exactly as we would like.

```

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 Wiener Model with make\_stancode Function

```

// generated with brms 1.10.2
functions {

  /* Wiener 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 wiener_model_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 += wiener_model_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 Wiener Model log-PDF for a single response

Before we can finally estimate the model, we need a function that generates initial values. Estimation will not begin unless initial values that lead to an identifiable model for all data points are provided. The function must provide initial values for all parameters listed in the model's parameters block. It is worth noting that many of those parameters have at least one dimension with a parameterized extent (e.g., K). To obtain the required information, we can use `make_standata` to create the data set used by brms for estimation. Then, in function `initfun`, we use this data object (i.e., a list) to generate the appropriately sized initial values (note that `initfun` relies on the fact that `tmp_dat` is in the global environment, which is a 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**Estimation (Sampling)**

Finally, we have all of the pieces in place and can use the `brm` function to estimate the Wiener model. Please keep in mind that this will take approximately a full day, and may take longer depending on the speed of your PC. We've also increased the maximum treedepth to 15. We should have probably increased `adapt_delta` above the default value of 0.8 because there are a few divergent transitions, but that is up to the reader. After we finish estimating, we see that there are a few (10) divergent transitions. If this were a real analysis rather than an example, we would need to increase `adapt_delta` to a higher value (e.g., 0.95 or 0.99) and rerun the estimation. In this case, however, we immediately proceed to the second step and use `predict` to obtain samples from the posterior predictive distribution. It is critical to specify the number of posterior samples in this case (here we use 500). Furthermore, for obtaining the actual posterior predictive distribution rather than a summary of the posterior predictive distribution, set `summary = FALSE` and `negative_rt = TRUE`. The latter ensures that predicted responses to the lower boundary are negative, whereas predicted responses to the upper boundary are positive.

```
fit_wiener <- 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_wiener <- predict(fit_wiener,
                      summary = FALSE,
                      negative_rt = TRUE,
                      nsamples = NPRED)
```

**Figure 1.5.1:** Wiener Model Estimation

We save the results of both steps because they are both time consuming (estimation takes a day, obtaining posterior predictives takes a few hours). Given the relative size of both objects, using the 'xz' compression (the strongest in R) appears to be a good idea.

```
save(fit_wiener, file = "brms_wiener_example_fit.rda",
     compress = "xz")
save(pred_wiener, file = "brms_wiener_example_predictions.rda",
     compress = "xz")
```

**Figure 1.5.2:** Saving Results of Wiener Model Estimation

## 2. Model Fit & Diagnostics

### 2.1 Introduction

This second section is concerned with perhaps the most important steps in any model-based data analysis, model diagnostics and model fit assessment. It should be noted that the code in this part is completely self-contained and can be run without requiring the code from Part I.

### 2.2 Setup

We begin by loading a large number of packages that we will require later on. Obviously, brms, but also some of the tidyverse packages (i.e., dplyr, tidyr, tibble, and ggplot2). It took me a while to get on board with tidyverse, but now that I'm using it more and more, I can't deny its usefulness. If your data can be made 'tidy,' the tidyverse's cohesive set of tools makes many seemingly difficult tasks relatively simple. A few examples will be provided below. GridExtra is also required for combining plots, as is DescTools for the concordance correlation coefficient CCC, which is used below.

```
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've used a binary R data file that contains the fitted model object as well as the generated posterior predictive distributions below, which we can download directly into R. It's worth noting that I had to go through a temporary folder to get there. For your convenience, I have attached both Model fit and predictions .RDA files in my Github

account/repository "binmishr/Weiner-Model-Analysis". You can download the attached files to your PC location, change the file path in the download. File function and then load the files into temporary folder per the syntax given below in figure 2.2.3

```
tmp <- tempdir()
download.file("https://github.com/binmishr/Weiner-Model-Analysis/blob/main/brms_wiener_example_fit.rda",
             file.path(tmp, "brms_wiener_example_fit.rda"))
download.file("https://github.com/binmishr/Weiner-Model-Analysis/blob/main/brms_wiener_example_predictions.rda",
             file.path(tmp, "brms_wiener_example_predictions.rda"))
load(file.path(tmp, "brms_wiener_example_fit.rda"))
load(file.path(tmp, "brms_wiener_example_predictions.rda"))
```

**Figure 2.2.3:** Downloading the Model & Prediction data into Temporary folder

### 2.3 Model Diagnostics

Part I already informed us that there are a few divergent transitions. If this were a real analysis, we would be dissatisfied with the current fit and would attempt to rerun brm with a higher adapt\_delta in the hope of removing the divergent transitions. According to the Stan warning

guidelines, "the validity of the estimates is not guaranteed if there are post-warmup divergences." However, it is unclear how the small number of divergent transitions (<10) observed here affects the posterior. It's also unclear what to do if adapt\_delta can't be increased any longer and the model can't be reparametrized. Should all fits with any divergent transitions be ignored entirely. Returning to our fit, we



check the R-hat statistic as well as the number of effective samples as a first step in our model diagnostics. We focus on the parameters with the highest  $R^2$  and the fewest effective

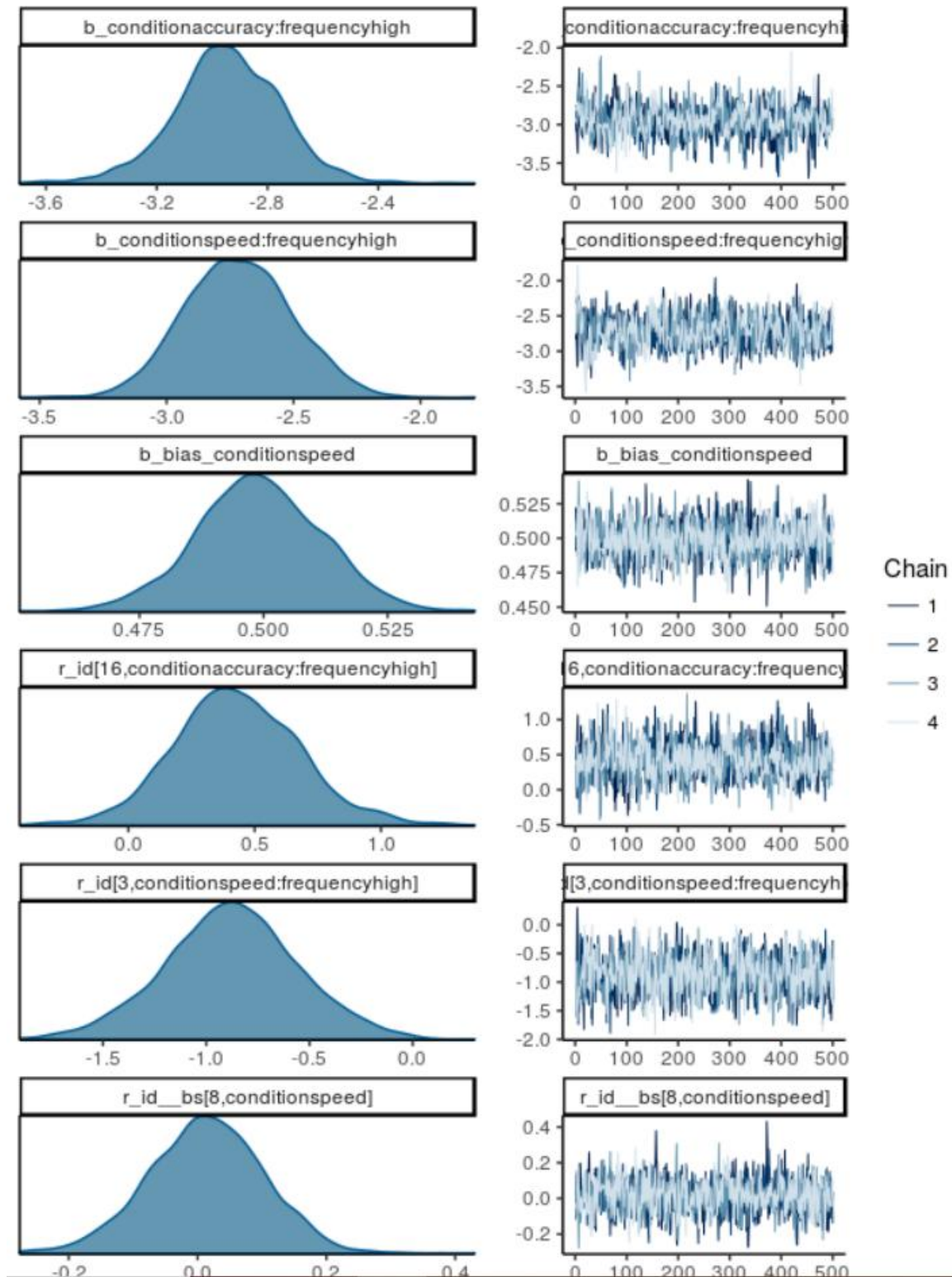
samples. Both are unproblematic (R-hat 1.05 and n eff > 100), indicating that the sampler has converged on the stationary distribution.

```
tail(sort(rstan::summary(fit_wiener$fit)$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_wiener$fit)$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_wiener)
pars_sel <- c(sample(pars[1:10], 3), sample(pars[-(1:10)], 3))
plot(fit_wiener, pars = pars_sel, N = 6,
     ask = FALSE, exact_match = TRUE, newpage = TRUE, plot = TRUE)
```

**Figure 2.3.2:** Plotting of the data



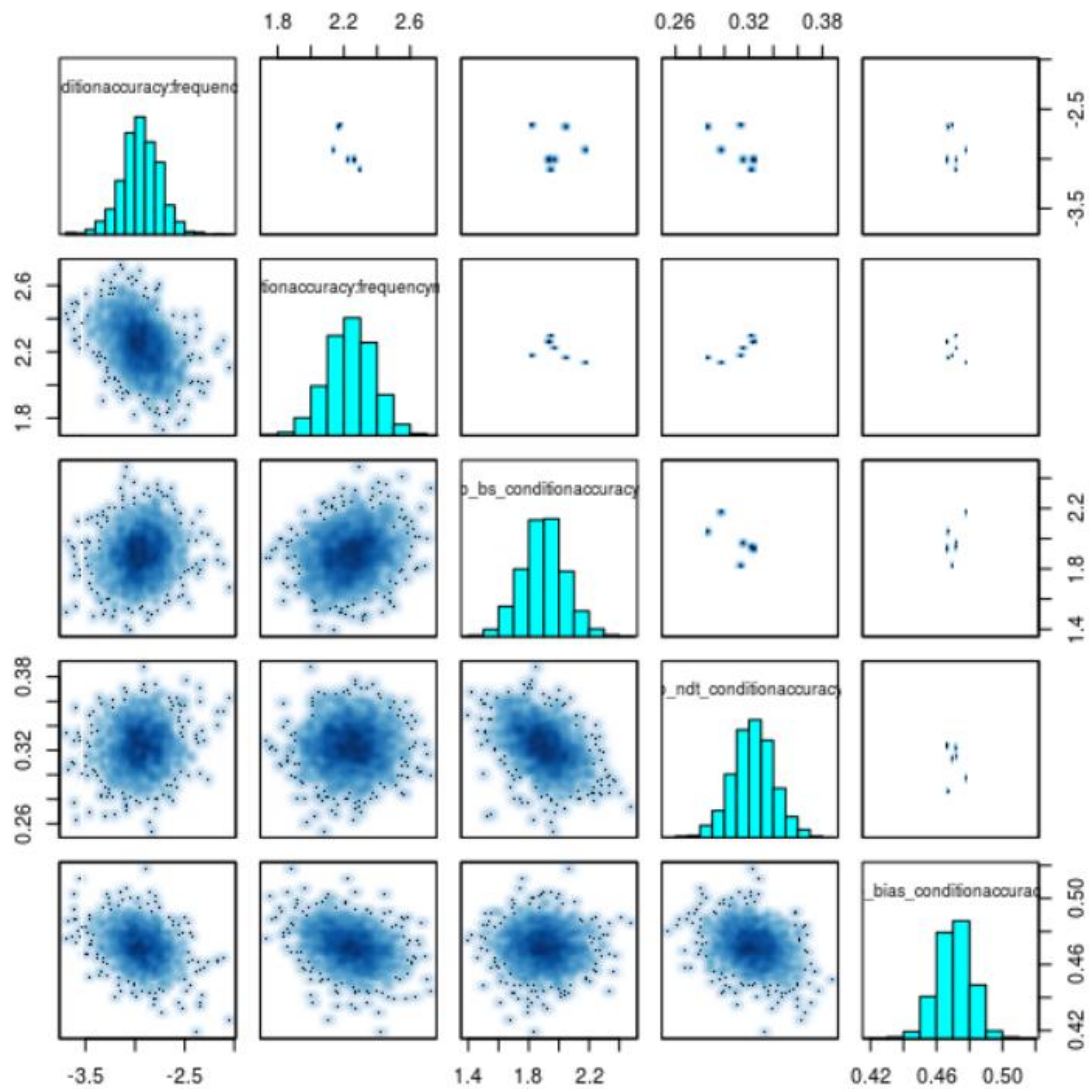
**Figure 2.3.3:** Graph of Chain behaviour of the Sampled data

Finally, there are some discussions in the literature about parameter trade-offs for the related models. These trade-offs are said to make fitting the model in a Bayesian setting particularly difficult. We look at the joint posterior of the fixed-effects of the main Wiener parameters for the accuracy condition to see if fitting the Wiener model with HMC as

implemented in Stan (i.e., NUTS) also shows this pattern. For this, we use the pairs function's stanfit method with the condition set to "divergent\_\_." This plot shows the few divergent transitions above the diagonal and the remaining samples below it.

```
pairs(fit_wiener$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 below table displays the ten largest absolute values of correlations among posteriors for all pairwise parameter combinations. The value in column Freq is the observed correlation between the posteriors of the two parameters

listed in the two previous columns, which is somewhat counterintuitive. To make this table, I used a trick called `as.table`, which is in charge of labelling the column containing the correlation value Freq.

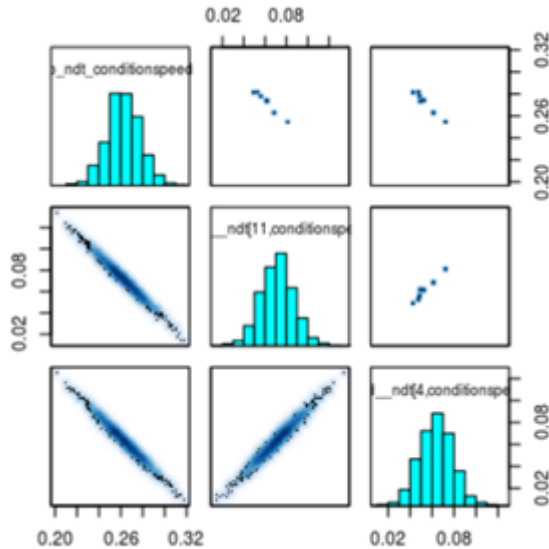
```
posterior <- as.mcmc(fit_wiener, combine_chains = TRUE)
cor_posterior <- cor(posterior)
cor_posterior[lower.tri(cor_posterior, diag = TRUE)] <- NA
cor_long <- as.data.frame(as.table(cor_posterior))
cor_long <- na.omit(cor_long)
tail(cor_long[order(abs(cor_long$Freq)),], 10)
#           Var1           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_wiener$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

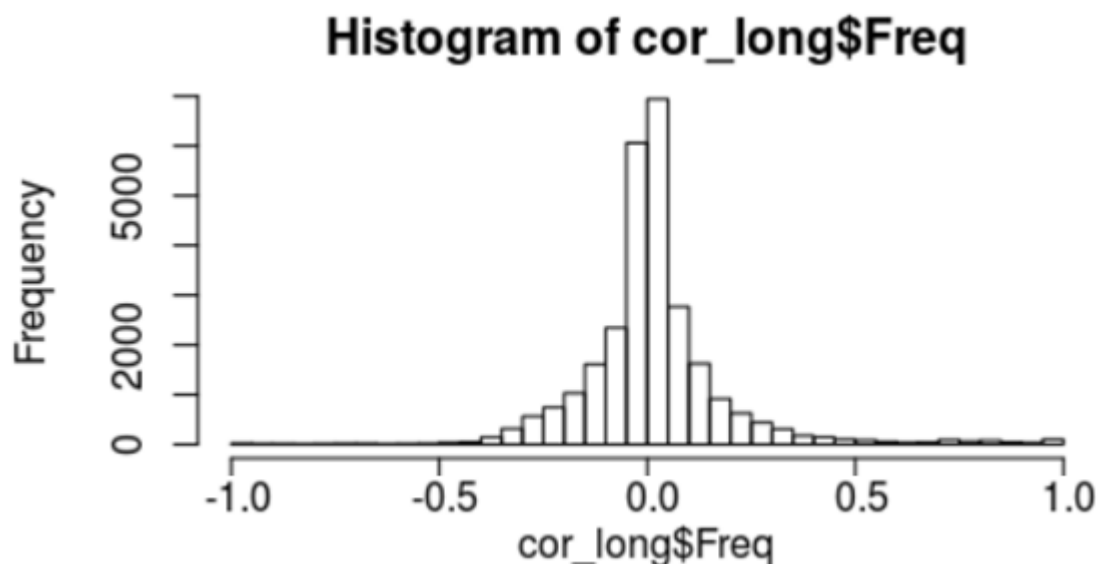


**Figure 2.3.8:** Graph of the Correlations values among posteriors for all pairwise parameter combinations

Overall, the model diagnostics show no particularly troubling behaviour (with the exception of the divergent transitions). We've discovered that some of the individual-level estimates for some of the parameters aren't very reliable. This, however, does not rule out the overall fit. The main take away from this fact is that we must exercise caution when interpreting individual-level estimates. As a result, we assume the fit is satisfactory and proceed to the next step of the analysis.

```
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

We will now look into the model fit. That is, we will investigate whether the model adequately describes the observed data. We will primarily do so through graphical checks. To accomplish this, we must first prepare the posterior predictive distribution and the data. We begin by combining the posterior predictive distributions with the data. Then, for each cell of the design (i.e., a combination of

condition and frequency factors), we compute three important measures (or test statistics  $T()$ ):

- 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.

This is first computed for each sample of the posterior predictive distribution. The median and some additional

quantiles across the posterior predictive distribution are then calculated to summarise these three measures. We compute

all of this in a single step using a lengthy combination of dplyr and tidyr magic.

```
d_speed_acc <- as_tibble(cbind(speed_acc, as_tibble(t(pred_wiener))))
```

**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

Following that, we compute the three measures for the data and combine them with the results of the posterior predictive distribution in a single data set. frame created with 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

The first critical question is whether our model can adequately describe the aggregated patterns in the data across participants. We simply use mean to aggregate the results obtained in the previous step (i.e., the summary results from the posterior predictive distribution as well as

the data test statistics). The summaries are then used to plot predictions (in grey and black) as well as data (in red) for the three measures. The inner (fat) error bars represent the 80% credibility intervals (CIs), while the outer (thin) error bars represent the 95% CIs. The median of the posterior predictive distributions is depicted by the black circle.

```
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 response") + xlab("") +
  scale_x_discrete(labels = new_x)

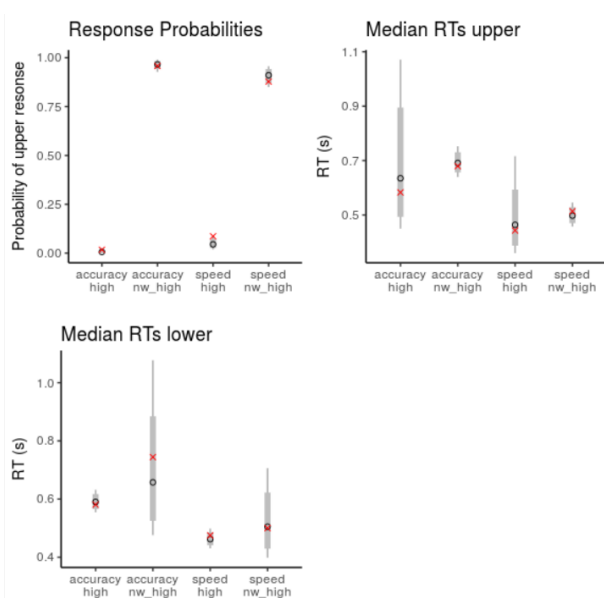
p2 <- ggplot(d_speed_acc_agg2, aes(x = condition:frequency)) +
  geom_linerange(aes(ymin = medrt.upper_lll, ymax = medrt.upper_hhh),
    col = "darkgrey") +
  geom_linerange(aes(ymin = medrt.upper_ll, ymax = medrt.upper_hh),
    size = 2, col = "grey") +
  geom_point(aes(y = medrt.upper_median), shape = 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

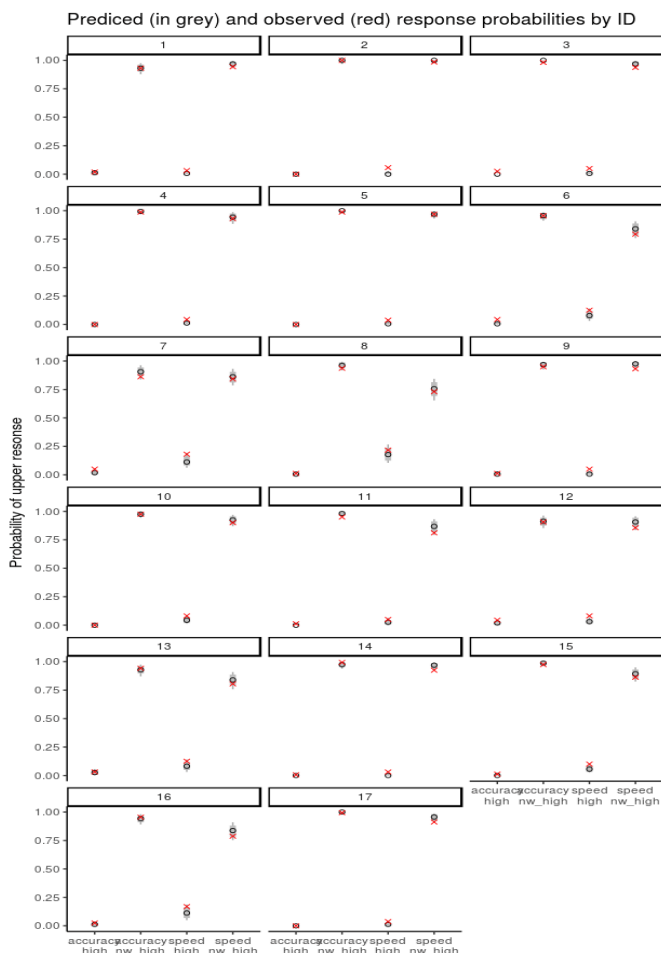
A close examination of the plots reveals no dramatic misalignment. Overall, the model appears to be capable of describing the data's general patterns. Only the response probabilities for words (frequency = high) appear to be overestimated. The red x appears to be outside the 80 percent confidence intervals, but possibly also outside the 95 percent confidence intervals. The RT plots reveal an intriguing (but not surprising) pattern. The posterior predictive distributions for rare responses (i.e., "word" responses to upper/non-word stimuli and "nonword" responses to lower/word stimuli) are relatively broad. The posterior predictive distributions for the common responses, on the other hand, are relatively narrow. In each case, the observed median is within the 80 percent confidence interval and also very close to the predicted median.

## 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_III, ymax =
    prob.upper_hhh),
    col = "darkgrey") +
  geom_linerange(aes(ymin = prob.upper_II, 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("Predicted (in grey) and observed (red) response
    probabilities by ID") +
  ylab("Probability of upper response") + 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 above graph follows the same pattern as the aggregated data. We see no dramatic misfits among the participants. Furthermore, response probabilities to non-word stimuli appear to be fairly well predicted. Response probabilities for word stimuli, on the other hand, are predicted to be lower than observed. This misfit, on the other hand, does not appear to be overly powerful. Following that, we examine the coverage probabilities of our three measures across individuals. That is, for each of the measures, each of the design cells, and each of the CIs (50 percent, 80 percent, 95 percent, and 99 percent), we calculate the proportion of participants whose observed test statistics fall within the corresponding CI.

```
speed_acc_agg %>%
  mutate(prob.upper_99 = (prob.upper >= prob.upper_III) &
    (prob.upper <= prob.upper_hhhh),
    prob.upper_95 = (prob.upper >= prob.upper_II) &
    (prob.upper <= prob.upper_hhh),
    prob.upper_80 = (prob.upper >= prob.upper_I) &
    (prob.upper <= prob.upper_hh),
    prob.upper_50 = (prob.upper >= prob.upper_1) &
    (prob.upper <= prob.upper_h),
    medrt.upper_99 = (medrt.upper > medrt.upper_III) &
    (medrt.upper < medrt.upper_hhhh),
    medrt.upper_95 = (medrt.upper > medrt.upper_II) &
    (medrt.upper < medrt.upper_hhh),
    medrt.upper_80 = (medrt.upper > medrt.upper_I) &
    (medrt.upper < medrt.upper_hh),
    medrt.upper_50 = (medrt.upper > medrt.upper_1) &
    (medrt.upper < medrt.upper_h),
    medrt.lower_99 = (medrt.lower > medrt.lower_III) &
    (medrt.lower < medrt.lower_hhhh),
    medrt.lower_95 = (medrt.lower > medrt.lower_II) &
    (medrt.lower < medrt.lower_hhh),
    medrt.lower_80 = (medrt.lower > medrt.lower_I) &
    (medrt.lower < medrt.lower_hh),
    medrt.lower_50 = (medrt.lower > medrt.lower_1) &
    (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

As can be seen, the coverage probability for the RTs is generally in line with or even above the width of the CIs. Furthermore, the coverage probability for the common response (i.e., upper for frequency = nw high and lower for frequency = high) is 1 for the 99 percent CIs in all cases. Unfortunately, the coverage for response probabilities is not

very good. Particularly in high-speed conditions and for tighter CIs. However, the coverage probabilities are at least acceptable for the wide CIs. So far, the results indicate that the model provides an adequate account. There are some misfits to be aware of if one wishes to extend the model or fit it to new data, but overall it provides a satisfactory account.

## 2.7 QQ-Plots: RTs

The final method for assessing model fit will be based on more quantiles of the RT distribution (i.e., so far we only

looked at the .5 quantile, the median). Individual observed versus predicted (i.e., mean from posterior predictive distribution) quantiles will then be plotted across conditions. To accomplish this, we first compute the quantiles per sample from the posterior predictive distribution and then aggregate across samples. This is accomplished through the use of `dplyr::summarise` at with a list column and `tidyr::unnest` to unstack the columns. The aggregated predicted RT quantiles are then combined with the observed 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

To assess the agreement between observed and predicted quantiles, we compute the concordance correlation coefficient for each cell and quantile. The CCC is a measure of absolute agreement between two values, making it more suitable than simple correlation. It is scaled from -1 to 1, with 1 representing perfect agreement, 0 representing no

relationship, and -1 representing a -1 correlation with the same mean and variance of the two variables. The code below generates QQ-plots for each condition and quantile separately for responses to the upper and lower boundaries. The CCC measures of absolute agreement are indicated by the value in the upper left corner of each plot.

```
plot_text <- rt_pp %>%
  group_by(condition, frequency, rowname, boundary) %>%
  summarise(ccc = format(
    CCC(observed, predicted, na.rm = TRUE)$rho.c$est,
    digits = 2))

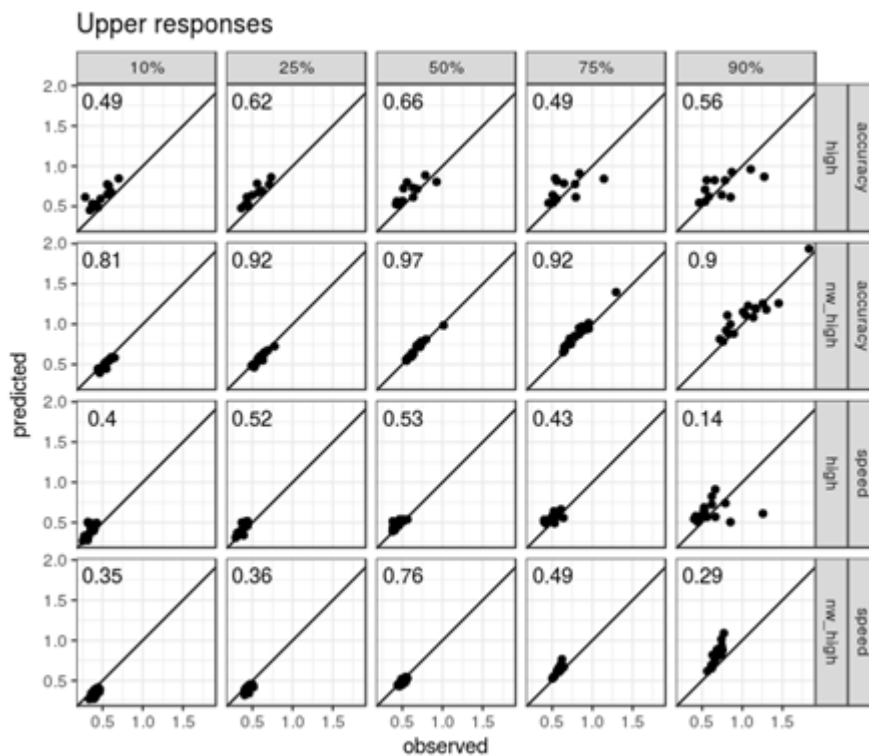
p_upper <- rt_pp %>%
  filter(boundary == "upper") %>%
  ggplot(aes(x = observed, predicted)) +
  geom_abline(slope = 1, intercept = 0) +
  geom_point() +
  facet_grid(condition+frequency~ rowname) +
```

```
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



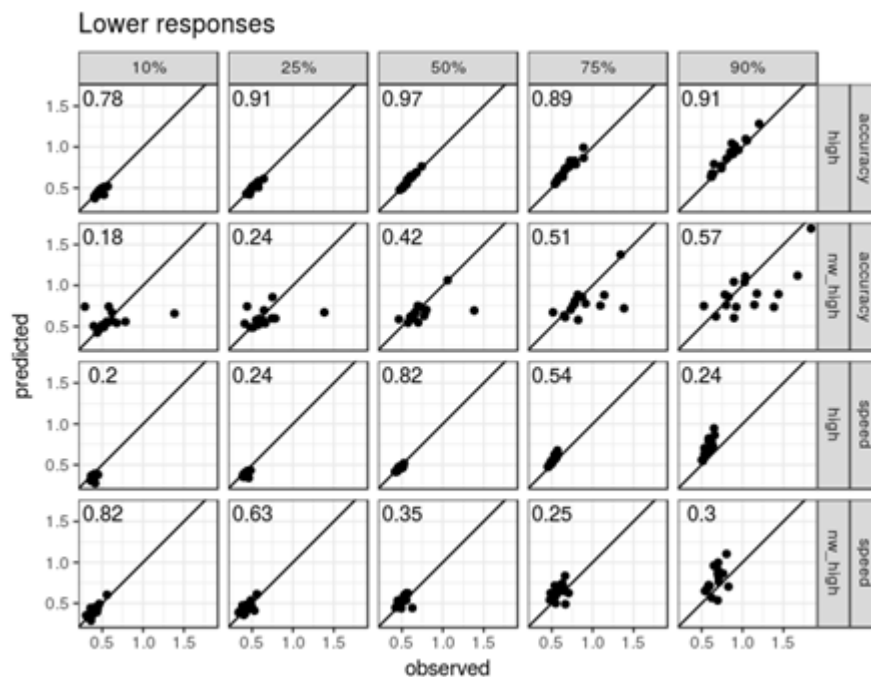


Figure 2.7.3: Graph of QQ-plot of the Quantiles with Upper and Lower response

### 3. Parameter Estimation & Hypothesis Tests

#### 3.1 Introduction

This is the third instalment of the series on fitting the Wiener 4-parameter model with brms. The first section went over how to set up the data and model. The second section dealt with (mostly graphical) model diagnostics and the evaluation of the model's adequacy (i.e., fit). This third section will examine the model's parameter estimates to see if there is any evidence for differences between the conditions. As before, this part is completely self-contained and can be run without needing to run the code from Parts I or II. Due to the length of this section, I will provide a brief overview. The following section provides a brief explanation of how we will conduct hypothesis testing. This is followed by a brief section that loads some packages and the fitted model object before providing a brief recap of the model. Following this is a relatively long section that examines the drift rate parameters in a variety of ways. Then we'll look at each of the other three parameters one by one. The section on non-decision time will be especially important. As I'll explain further below, I believe this parameter cannot be interpreted. Finally, I provide a brief overview of some of the current model's limitations and how it could be improved.

#### 3.2 Bayesian Hypothesis Testing

The purpose of this section is to show that there are differences in parameter estimates between conditions. Importantly, different methods of producing such evidence are only meant in a technical sense. In statistical terms, we will always inspect difference distributions resulting from linear combinations of cell-wise posterior distributions of group-level model parameter estimates. The slightly technical phrase "linear combinations of cell-wise posterior distributions" is frequently used to simply mean the

difference between two distributions. The difference distribution, for example, is the result of subtracting the posterior of the speed condition from the posterior of the accuracy condition. To recap, a posterior distribution is the probability distribution of a parameter based on data and model (where the latter includes the parameter priors). It provides an answer to the question of which parameters are most likely given our prior knowledge and data. As a result, the posterior distribution of the difference answers questions such as whether the difference values between two conditions are likely or not. With such a disparity in distribution, we can do two things as stated below:

- First, we can see if the difference distribution's  $x$  percent-highest posterior density (HPD) or credibility interval includes 0. If 0 falls within the 95 percent HPD interval, it may be considered a plausible value. If 0 is outside the 95 percent interval, we may consider it insufficiently plausible and conclude that there is evidence for a difference.
- Second, we can determine how much of the difference distribution is on one side of 0 and how much is on the other. If this value deviates significantly from 50%, there is evidence of a difference. For example, if all of the posterior samples for a particular difference are greater than zero, this provides strong evidence that the difference is greater than zero.

The investigation of posterior distributions to assess differences between conditions is only one method for hypothesis testing in a Bayesian setting. And, at least in the psychological literature, it is not the most widely accepted. Many of the more vocal supporters of Bayesian statistics in the psychological literature advocate hypothesis testing using Bayes factors. In general, I agree with many of the arguments in favour of the Bayes factor, particularly in cases like this one, where all relevant hypotheses or competing models are nested within a single large (super) model. The



main challenge with Bayes factors is their extreme sensitivity to parameter priors. In the case of nested models, this is not a major issue. This approach has been extended to general ANOVA designs. It has been applied to accumulator models. The general idea is to reparametrize the model using effect parameters that have been normalised, such as the residual variance. For a two-sample design, for example, parameterize the model with a standardised difference. Then, putting a prior on the standardised effect size measure is relatively simple and uncontroversial. In the current case, where the model lacks a residual variance parameter, such a normalisation could be accomplished by using the variance estimate of the group-level distribution for each parameter.

Unfortunately, to the best of my knowledge, brms does not support specifying a parameterization and prior distribution in accordance with default Bayes factor. And, it's also unlikely that brms will ever get this ability. As a result, I believe that brms is not the best tool for model selection using Bayes factors. While it now technically supports this capability (via our bridge sampling package), it only supports models with unnormalized parameterization. I believe that such a parameterization is inappropriate for Bayes factors-based model selection in most cases because the priors cannot be specified in a 'default' manner. As a result, I am unable to recommend brms for Bayes factor-based model selection at this time. To summarise, the reason we are basing our inferences solely on posterior distributions in this case is due to practical constraints rather than philosophical considerations.

One final word of caution for the psychological audience. While Bayes factors are clearly popular in psychology, they are not in many other scientific disciplines. As far as I can tell, the difference stems from the different types of data that different people work with. When working with observational (or correlational) data, tests for the presence of effects (or nullity) are either a no-no (e.g., when trying to do causal inference) or simply uninteresting. We all know that the real world is full of arbitrary relationships, especially small ones. So simply increasing N to get effects is not interesting, and estimation is the more interesting approach. For experimental data, on the other hand, we frequently have true null hypotheses, and testing those makes a lot of sense. However, as far as I can tell, the effect is completely insignificant. In this case, hypothesis testing is critical.

### 3.3 Getting Started

We begin by loading some packages for posterior analysis. Since the beginning of this series, I've grown increasingly fond of the entire tidyverse, so we've imported it entirely. Of course, we require brms as well. As shown below, we will require a few more packages (particularly emmeans and tidybayes), but these are only loaded when necessary. Then we'll need the posterior samples, which we can load in the same way we did in 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/Weiner-Model-
Analysis/blob/main/brms_wiener_example_fit.rda",
file.path(tmp, "brms_wiener_example_fit.rda"))
load(file.path(tmp, "brms_wiener_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

As a reminder, we have data from a lexical decision task (i.e., participants must decide whether presented strings are words or not), and frequency is the factor determining a

string's true status, with high referring to words and nw\_high referring to non-words. 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), with the drift rate for words (rows 1 and 2) being clearly negative (i.e., those trials mostly hit the lower boundary for the word decision) and the drift rate for non-words (rows 3 and 4) being clearly positive (i.e., those trials mostly hit the upper boundary for non-word decisions). Furthermore, there may be differences in drift rates depending on the accuracy or speed conditions. In particular, drift rates appear to be less extreme (i.e., closer to 0) in the speed conditions when compared to the accuracy conditions.

The only difference between the other three parameters is the condition factor. Given the experimental manipulation of the accuracy versus speed condition, we anticipate differences in the boundary separation parameters beginning with `bs_`. There appears to be a small effect for the non-decision time, parameters beginning with `ndt_`, as the 95 percent overlaps only slightly. However, as discussed in greater detail below, we must exercise caution in interpreting this distinction. Finally, there may or may not be a difference for bias parameters beginning with `bias_`. Furthermore, there appears to be a bias for "word" responses, at least in the accuracy condition.

The hypothesis function in `brms` can be used to test differences between conditions. But I couldn't get it to work with the current model. I believe this is due to the somewhat unusual parameterizations in which each cell receives one parameter (in some sense each cell has its own intercept, but there is no overall intercept). In contrast, a more "standard" parameterization has one intercept (for either the unweighted means or one of the cells) and the remaining parameters capture the differences between the intercept and the cell means. As a reminder, I chose this unconventional parameterization in the first place to make it easier to specify the parameter priors. Furthermore, when programming cognitive models by hand, this is a common parameterization.

### 3.3 Emmeans & tidybayes : Differences In Drift Rate

Another option is to use excellent `emmeans` package. I am a huge fan of `emmeans` and use it frequently when working with "normal" statistical models (e.g., ANOVAs, mixed models), regardless of whether I use frequentist (e.g., via `afex`) or Bayesian methods (e.g., `rstanarm` or `brms`). Unfortunately, it appears that `emmeans` can only analyse the main parameter of the response distribution for models estimated with `brms` at the moment, which in our case is the drift rate. In any case, I strongly advise you to review the `emmeans` vignettes to get a sense of what types of follow-up tests are all possible with this fantastic package.

As I recently discovered, `emmeans` plays well with `tidybayes`, a package that allows you to work with posterior draws within the `tidyverse`. `tidybayes` has an unusually large package footprint (i.e., it imports a large number of other packages) for a package with such limited functionality. I suppose this is a result of being a part of the `tidyverse`. In any case, many of the imported packages are already in the search path as a result of loading the `tidyverse` above, so attaching should be quick.

We start with `emmeans` only to ensure that everything works as expected. We get the estimated marginal means plus 95 percent -highest posterior density (HPD) intervals that match the output of the fixed effects for the central tendency estimate (which is the median of the posterior samples in both cases). As a reminder, the fact that the cell estimates match the parameter estimates is due to the unusual parameterization, which `emmeans` correctly detects. The lower and upper bounds of the intervals differ slightly between the summary outputs of `brms` and `emmeans`, as a result of the different methods for calculating the intervals (i.e., quantiles versus HPD intervals).

```
library("emmeans")
library("tidybayes")
```

Figure 3.3.1: Loading of the R Libraries

```
fit_wiener %>%
  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 3.3.2: Extracting Data with HPD Interval Probability

### 3.4 HPD Intervals & Histograms

As a first test, we want to see if there is evidence for a difference in speed and accuracy conditions for both words (frequency = high) and non-words (frequency = nw\_high).

There are several ways to accomplish this with `emmeans`, one of which is through the `by` argument and the `pairs` function. We don't have much evidence that there is a difference for either stimulus type here, because both HPD intervals include 0.

```
fit_wiener %>%
```

```

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

Instead of using emmeans to get the summary of the distribution, we can use tidybays to extract the samples in a tidy manner. The samples are then aggregated based on the same conditioning variable using one of the handy aggregation functions included with tidybays. After

experimenting with a few different options, I believe emmeans' hpd.summary() function uses the same method for calculating HPD intervals as tidybays, as both results match.

```

samp1 <- fit_wiener %>%
  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 3.4.2:** 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 3.4.3:** Calculating HPD Intervals with Mode

```

get_hdi <- function(x, level = 95) {
  tmp <- hdi::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 3.4.4:** 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 3.4.4:** Calculating HPD Intervals For Sample Point Estimation with Mode and User Defined Function with Level 80%

We can now assess whether there is evidence for a drift rate difference between conditions for both word and non-word stimuli because we have the samples in a convenient format. One issue with this is that the direction of the effect varies between words and non-words. This is due to the fact that word stimuli necessitate a response at the lower decision boundary and non-word stimuli necessitate a response at the

upper decision boundary. As a result, for one of the conditions, we must multiply the effect by -1. Then we can take the average of both conditions. We accomplish this using tidyverse magic, and we also add the number of values aggregated in this manner to the table. This is just a check to ensure that our logic is correct and that we always aggregate exactly two values. This is confirmed by the final check.

```

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 3.4.5:** Calculation of Drift Rate Difference between conditions for both word and non-word stimuli

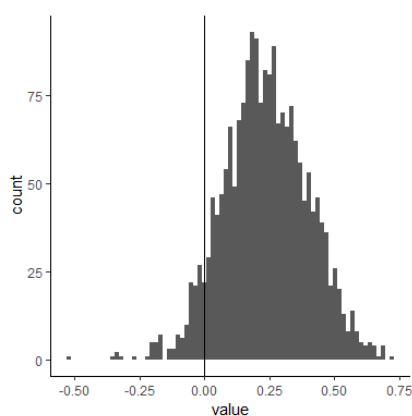
The resulting difference distribution can then be investigated. A histogram is one method for doing so graphically. It's a good idea to experiment with the number of bins until the figure looks right. Given the large number of samples, 75 bins seemed reasonable. There wasn't enough granularity with fewer bins, and there were too many small peaks with more bins.

```

ggplot(samp2, aes(value)) +
geom_histogram(bins = 75) +
geom_vline(xintercept = 0)

```

**Figure 3.4.6:** Code Plot of the Histogram of Drift Rate Difference between conditions for both word and non-word stimuli



**Figure 3.4.6:** Histogram of Drift Rate Difference between conditions for both word and non-word stimuli

The above histogram in figure 3.4.6 demonstrates that, while a significant portion of the posterior mass is to the right of 0,

a significant portion is still to the left. So there is some evidence for a difference, but it is not very strong, even when words and non-words are considered together. The HPD intervals can also be used to investigate this difference distribution. To get a better picture, consider the following interval sizes. This demonstrates that 0 is excluded only for the 85 percent interval and smaller intervals. To get a graphical overview of the output, use `hdcrcde::hdr.den` instead of `hdcrcde::hdr.den`.

```

hdcrcde::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
#
# $alpha
# 1% 5% 10% 15% 20% 50%
# 0.116 0.476 0.757 0.984 1.161 1.857

```

**Figure 3.4.7:** Difference Distribution via Different HPD Intervals

### 3.5 Bayesian P-values

Calculating the actual proportion of samples below 0 is a method that requires fewer arbitrary cutoffs than HPDs (for which the width must be defined). As previously stated, if

this proportion is small, this would be evidence of a difference. In this case, the proportion of samples that are less than 0 is .067. Unfortunately, .067 is slightly higher than the magical cutoff of .05, which is universally accepted as distinguishing small from large numbers, or, perhaps more accurately, likely from unlikely probabilities.

```
mean(samp2$value < 0)
# [1] 0.0665
```

**Figure 3.5.1:** Calculating Actual proportion of samples below 0

Let's take a closer look at this proportion. If two posterior distributions are stacked exactly on top of each other, the resulting difference distribution is centred on 0, with exactly half of the difference distribution on either side of 0. As a result, a proportion of 50% corresponds to the least evidence for a difference or, alternatively, the strongest evidence for the absence of a difference. Another implication is that both values near 0 and values near 1 indicate a difference, albeit in opposite directions. To facilitate interpretation of these proportions, I recommend that they be calculated in such a way that small values represent evidence for a difference (e.g., by subtracting the proportion from 1 if it is above .5). But what exactly does this proportion tell us? It denotes the likelihood of a difference in a specific direction. As a result, it is one-sided evidence for a difference. In contrast, for a 95% HPD, we subtract 2.5 percent from both sides of the difference distribution. We must multiply this proportion by 2 to ensure that it has the same two-sided property as our HPD intervals. Another advantage of this multiplication is

that it extends the range to the entire probability scale (i.e., from 0 to 1).

As a result, the resulting value is a probability (ranging from 0 to 1), with values close to zero indicating evidence for a difference and values close to one indicating evidence against a difference. Thus, unlike a traditional p-value, it is a continuous measure of evidence for (when near 0) or against (when near 1) a difference in parameter estimates. Given its superficial resemblance to classical p-values (low values are regarded as evidence for a difference), we could refer to it as a Bayesian p-value, or pB for short. In this case, we could say: The pB value for a difference in drift rate between speed and accuracy conditions across word and non-word stimuli is .13, indicating that the evidence for a difference is at best weak. Of course, Bayesian p-values can be abused in the same way that classical p-values can. For example, you could introduce arbitrary cutoff values, such as .05. Consider for a moment that we want to see if there are any differences in the absolute amount of evidence as measured by drift rate for any of the four cells in the design (I am not suggesting that is particularly sensible). This would necessitate transforming the posterior for all drift rates onto the same side (note, we do not want to take the absolute values as we still want to retain the information of switching from positive to negative drift rates or the other way around). For instance, multiply the drift rate for words by -1. We do so, and then we examine what the cell means. According to an examination of the four cell means, the drift rate values for words are greater than the values for non-words.

```
samp3 <- fit_wiener %>%
  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 3.5.2:** Inspection of the four cell means of drift rate values for words and non-words

I wrote two functions that return a compact letter display of all pairwise comparisons in order to get an overview of all pairwise differences using an arbitrary cut-off value. The functions require data in a wide format, with each column

representing one parameter's draws. It's worth noting that the compact letter display is calculated by another package, multcompView, which must be installed before you can use these functions.

```
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 3.5.3:** Using Functions to Find Compact Letter Display of all Pairwise Comparisons

Conditions that share a common letter in a compact letter display do not differ based on the criterion. Conditions that do not share a common letter differ based on the criterion. The compact letter display in this case is not very informative and simply repeats what we saw above. The drift rates for words are divided into two groups, and the drift rates for non-words are divided into two groups.

Compact letter displays can be quite informative in cases with more conditions or more complicated difference patterns. We could have also used tidybayes' functionality to inspect all pairwise comparisons. It is critical to use ungroup before calling the compare levels function. Otherwise, we get a difficult-to-understand error (the grouping appears to be a consequence 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 3.5.4:** Using Tidybayes to Find Compact Letter Display of all Pairwise Comparisons

### 3.6 Differences In Other Parameters

As previously discussed, we appear to be unable to use emmeans to examine the differences in the other parameter. Fortunately, tidybayes still allows you to extract posterior

samples in a tidy manner by using either gather\_draws or spread\_draws. It appears that you must pass the specific variable names you want to extract for either of those. We obtain them using get\_variables:

```

get_variables(fit_wiener)[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 3.6.1:** Extract Posterior Samples in Tidy Manner

### 3.7 Boundary Separation

Spread\_draws will be used to analyse the boundary separation. We begin by extracting the draws and then immediately compute the difference distribution between the two.

```
samp_bs <- fit_wiener %>%
  spread_draws(b_bs_conditionaccuracy,
b_bs_conditionspeed) %>%
  mutate(bs_diff = b_bs_conditionaccuracy -
b_bs_conditionspeed)
```

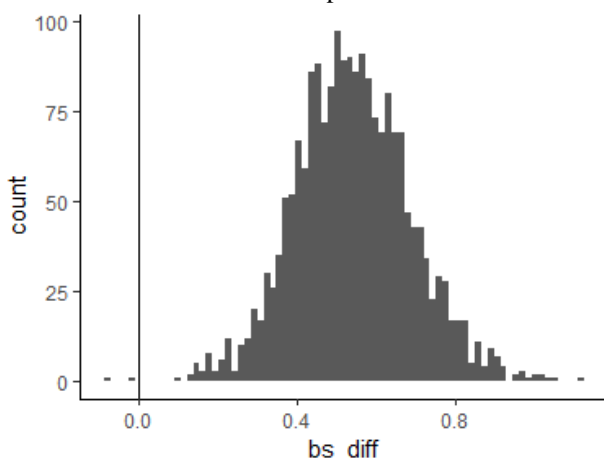
```
samp_bs
## A tibble: 2,000 x 6
#   .chain .iteration .draw b_bs_conditionaccuracy
b_bs_conditionspeed bs_diff
#   <int><int><int><dbl><dbl><dbl>
# 1     1     1     1         1.73         1.48 0.250
# 2     1     2     2         1.82         1.41 0.411
# 3     1     3     3         1.80         1.28 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 3.7.1:** Analysis of Boundary Separation of the Samples

Of course, we can now use the same tools as before. Take a look at the histogram, for example. I chose 75 bins once more. Of course, we can now use the same tools as before. Take a look at the histogram, for example. I chose 75 bins once more. Overall, we can be fairly certain that varying the speed versus accuracy conditions affects the boundary separation in the current data set. Everything went exactly as planned.

```
samp_bs %>%
  ggplot(aes(bs_diff)) +
  geom_histogram(bins = 75) +
  geom_vline(xintercept = 0)
```

**Figure 3.7.2:** Histogram Code of Boundary Separation of the Samples



**Figure 3.7.3:** 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 3.7.4:** Sample Data of Boundary Separation of the Samples below 0

### 3.8 Non-Decision Time

We use gather\_draws to compare differences in non-decision time. One advantage of this function over spread\_draws is that it makes obtaining marginal estimates simple. As previously stated, the HPD intervals overlap only very slightly, indicating that there is a difference between the conditions. The resulting marginal estimates are saved for later use in new data.frame. ndt\_mean is a data frame.

```
samp_ndt <- fit_wiener %>%
  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
```

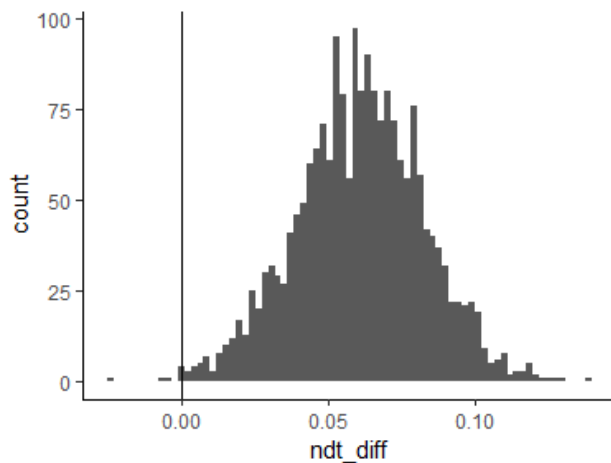
**Figure 3.8.1:** Accessing Differences in Non-decision time to obtain Marginal Estimates

To calculate the difference, it appears to me that the simplest approach is to spread the two variables across rows and then calculate the difference (similar to starting with spread draws in the first place). The resulting difference distribution can then be plotted again.

```
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 3.8.2:** Calculating the spread of two variables across rows and then calculate the difference from results of Figure 3.8.1



**Figure 3.8.3:** Plotting Histogram of the results from Figure 3.8.2

As previously speculated, there appears to be compelling evidence for a distinction. We can confirm this further using the Bayesian p-value:

```
mean(samp_ndt2$ndt_diff < 0) * 2
# [1] 0.005
```

**Figure 3.8.4:** Confirming Evidence of Difference of results of Figure 3.8.3 via Bayesian p-values

So far, it appears that we discovered another significant difference in parameter estimates as a result of the manipulation. This, however, would be a hasty conclusion. In fact, examining the non-decision time estimated by the 4-parameter Wiener model in this manner is completely misleading. The non-decision time parameter is only sensitive to a few data points, rather than capturing a meaningful feature of the response time distribution. In particular, the non-decision time reflects a specific feature of the distribution of minimum response times per participant and condition or cell for which it is estimated. For our example data, I will demonstrate this in the following. We must first load the data in the same manner as described in previous posts. The minimum RTs are then calculated for each participant and condition.

```
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 3.8.5:** Calculating Minimum RTs per participant and Conditions

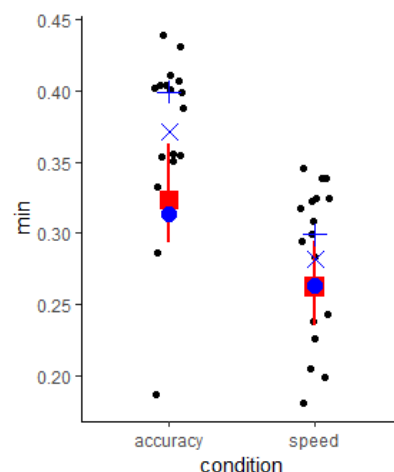
To investigate the issue, we want to compare the distribution of minimum RTs with non-decision time estimates graphically. To accomplish this, we must add a condition column with matching condition names to the ndt\_mean data.frame that we created earlier. Then we can combine them into a single plot. We also include a few summary

statistics on the distribution of individual minimum RTs. The black points represent the individual minimum RTs for each of the two conditions; 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; and the red square represents the point estimate of the non-decision time parameter with corresponding 95 percent HPD intervals.

```
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 3.8.6:** Comparison of the distribution of Minimum RTs with Estimates for Non-Decision Times



**Figure 3.8.7:** Graph Plot of the distribution of Minimum RTs with Estimates for Non-Decision Times

This graph in Figure 3.8.7 shows that the estimated non-decision time almost perfectly matches the midpoint between the largest and smallest minimum RT (i.e., the blue dot). To put this into context, consider comparing the number of minimum data points (i.e., the number of participants) to the total number of data points.

```
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

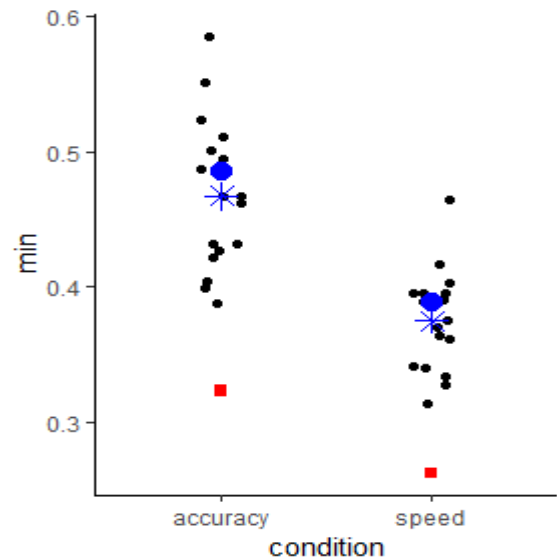
**Figure 3.8.8:** Comparing the Number of data points with Total number of data points

This demonstrates that the non-decision time parameter, one of only four model parameters, is largely determined by less than .5% of the data. If any of these minimum RTs is an outlier (which seems likely in the accuracy condition), a single response time can have a huge impact on the parameter estimate. In other words, it is unlikely that the non-decision time parameter reflects an actual latent process with the current implementation. Instead, it simply reflects the midpoint between the smallest and largest minimum RTs per participant and condition, slightly weighted toward the mass of the minimum RT distribution. This parameter estimate should not be used to draw any meaningful conclusions.

In the present case, this blunder does not appear to be too significant. If only one of the data points in the accuracy condition is an outlier and the other data points are faithful representations of the response time distribution's leading edge (which is essentially what the non-decision time is supposed to capture), the current parameter estimates understate the true difference. This conclusion is supported further by using a more robust ad hoc measure of the leading edge, specifically the 10% trimmed mean of the 40 fastest RTs per participant and condition plotted below. This graph also no longer contains any obvious outliers. The non-decision time estimates are still included for reference. However, 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, and the interpretation of non-decision times obtained with such a model is not recommended.

```
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 3.8.9:** Plotting of Data Points with 10% trimmed mean of the 40 fastest RTs per participant and condition of Samples**Figure 3.8.10:** Graph of Data Points with 10% trimmed mean of the 40 fastest RTs per participant and condition Of Samples

It is important to note that this confound does not apply to all model implementations, but only to the 4-parameter Wiener model as implemented here. There are solutions to this problem, two of which I'd like to highlight here. To begin, one could add trial variability in non-decision time across trials. This variability is frequently assumed to follow a uniform distribution, which can capture outliers at the response time distribution's leading edge. Second, rather than fitting only a model, one could assume that some of the responses are contaminants from a different process, such as random responses from a uniform distribution ranging from the absolute minimum to the maximum RT. Technically, this is a mixture model of the process and a uniform distribution with either a free or fixed mixture/contamination rate. It should be relatively simple to implement such a mixture model in brms using a custom\_family, and I hope to find the time to do so at some point in the future.

Of course, I am not the first to notice this behaviour of the 4-parameter Wiener model. However, this problem appears to be particularly prevalent in a Bayesian setting because the 4-parameter model variant is readily available while model variants dealing with this problem are not. I found some time ago what would be the best way to address this issue, and one thing I remember that using the 4-parameter Wiener model, we can simply ignore the non-decision time parameter. That still appears to be the best option to me.

I hope there aren't too many papers that use the 4-parameter model in this manner to interpret differences in the non-decision time parameter.

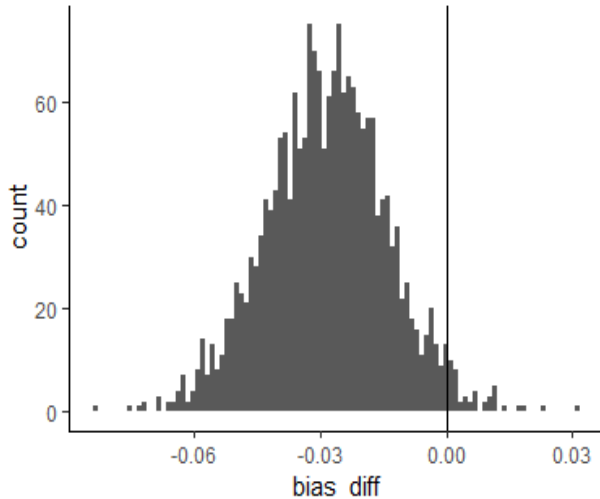
### 3.9 Starting Point / Bias

Finally, we can examine the starting point or bias. We repeat this process with spread\_draws and plot the resulting difference distribution.

```
samp_bias <- fit_wiener %>%
  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 3.9.1:** Examining the Starting Point or Bias of Sampling Data Distribution



**Figure 3.9.2:** Histogram of Starting Point or Bias of Sampling Data Distribution

The difference distributions imply that there could be a difference. As a result, we compute the Bayesian p-value next. This time, we calculate the difference in the opposite direction, so evidence for a difference is represented by small values.

```
mean(samp_bias$bias_diff > 0) *2
# [1] 0.046
```

**Figure 3.9.3:** Calculating the Bayesian p-values of Sampling Data Distribution

Together with the evidence for a difference, we can now more confidently postulate that there is a bias toward the lower boundary and "word" responses in the accuracy condition, whereas evidence accumulation begins unbiased in the speed condition. We are fortunate in that our Bayesian p-value is just below .05, leading us to believe that the difference is real. To wrap things up, here are some more estimates:

```
fit_wiener %>%
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 3.9.4:** Estimating the Starting Point or Bias of Sampling Data Distribution

For the current data, we find a specific pattern that is commonly perceived as typical. In the accuracy condition, error RTs are significantly slower than correct RTs, as shown below. This effect does not exist in the speed condition, where error RTs are faster than correct RTs.

```
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><lg1><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 3.9.5:** Examining of Error RTs & Correct RTs

Given the difference in the relative speeds of correct and error responses in the accuracy condition, it may come as no surprise that the accuracy condition also has a measurable bias. Specifically, a preference for word responses. However, as can be seen by inserting stim\_cat into the above group by call, the difference in relative error rate is especially pronounced for non-words, where a bias toward words should result in faster errors. As a result, it appears that the current model variant does not fully account for some of the more subtle effects in the data.

The standard method for dealing with differences in the relative speed of errors in weiner modelling is to use across-trial variability in model parameters. Variability in the starting point enables errors to be faster than correct RTs. Error RTs can be slower than correct RTs due to drift rate variability. However, as will be discussed further below, introducing these variables into a Bayesian framework has its own set of issues.

## 4. Conclusion

Overall, the fit is better for accuracy than for speed conditions, according to the results. Fit is also better for the common response (i.e., nw\_high for upper and high for lower responses). This latter observation is, once again, unsurprising. When the fit for the different quantiles is compared, it appears that at least the median (i.e., 50%) shows acceptable values for the common response. However, especially in the speed condition, the other quantiles are not well taken into account. Nonetheless, dramatic misfit is only seen in rare responses. The comparatively low variances in some of the cells could explain some of the low CCCs in the speed conditions. For example, for both common speed conditions (i.e., speed and nw\_high for upper responses and speed and high for lower responses), a visual inspection of the plot suggests an acceptable account while some CCC values are low (i.e., .5). Only in the speed conditions do we see systematic



deviations for the 90 percent quantile (and slightly less for the 75 percent quantile). The model predicts slower RTs than what has been observed.

As I've mentioned several times throughout this series, the model used here is the 4-parameter Wiener model. While this allows for estimation in the first place, it does have a few drawbacks. One of them has been extensively discussed in this section. The non-decision time parameter estimate essentially captures a feature of the distribution of minimum RTs. If these are contaminated by responses that cannot be assumed to be the result of the same process as the other responses (which I believe a priori to be quite likely), the estimate loses its meaning. I believe that the risks far outweigh the benefits. Another feature of the 4-parameter Wiener model is that it predicts equal mean response times for correct and error responses in the absence of a bias for any of the response options. This is possibly the most important theoretical constraint that has led to the development of many of the more highly parameterized model variants, such as the full (i.e., 7-parameter) model. While this series concludes here, there are a few more things that appear to be important, interesting, or viable. Here they are:

- We haven't yet looked at the estimates of the group-level parameters, which is a crucial step (i.e., standard deviations and correlations). They may contain important information about the specific data set and research question, but they may also contain information about the model parameters' trade-offs.
- To interpret the non-decision time, replace the pure Wiener process with a mixture of a Wiener and a uniform distribution. As previously stated, this should be possible with a custom family in brms.
- As previously stated, differences in the relative speed of error and correct RTs were one of the driving forces behind modern response time models. Variabilities in model parameters are typically used to explain these. The hierarchical structure is a relatively simple way to implement these variables in a Bayesian setting. For example, for the drift rate, each participant receives a by-trial random intercept,  $+ (0 + id || trial)$  (the double bar notation should ensure that these are uncorrelated across participants). While this appears to be a simple concept, I doubt such a model will converge in a reasonable timeframe. Given the theoretical significance of this approach, it appears to be an extremely important angle to investigate.
- It takes a long time to fit the Wiener model. It would be interesting to compare the fit using full Bayesian inference (i.e., sampling as done here) with variational Bayes (i.e., posterior parametric approximation), which is also implemented in Stan. I expect it to be ineffective, but the comparison would be fascinating. Diagnostics for variational Bayes were recently introduced.

## 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.