Online Supplement F: Bayesian Hierarchical Modeling in brms - Log Model

Myrthe Veenman

This online supplement provides a tutorial on log-normal Bayesian Hierarchical models (BHM) in `brms` (Bürkner, 2017, 2018). In addition, it contains information about the Number Classification Task that could not be included in the paper. As `brms` runs on `Stan` (Carpenter et al., 2017), it is very similar to `rstan` (Stan Development Team, 2019). However, there are some differences that will be discussed in this tutorial. Only the fit of the full model will be shown as the fitting the others should follow logically from this tutorial. Technical details, such as explanations of statistical terms (e.g., $\hat{R}$), will not be discussed here but can be found in the paper. In concrete, the tutorial will cover the following:

Required Packages

In this practical, the following packages will be used:

- For the document layout: *rmarkdown* [Version 1.16; Xie, Allaire, & Grolemund (2018)], *papaja* [Version 0.1.0.9842; Aust & Barth (2018)], *knitr* [Version 1.25; Xie (2015)], *kableExtra* [Version 1.1.0; Zhu (2019)]
- For data structuring: *LaplacesDemon* [Version 16.1.4; Statisticat & LLC. (2020)], *plyr* [Version 1.8.4; Wickham (2011)], *dplyr* [Version 0.8.3; Wickham, François, Henry, & Müller (2019)], *readr* [Version

1.3.1; Wickham, Hester, & Francois (2018)], *truncnorm* [Version 1.0.8; (**truncnorm?**)]

- To fit the model and for model comparison: *brms* [Version 2.13.0; Bürkner (2017); Bürkner (2018)]
- For the visualization of the results: *lattice* [Version 0.20.38; Sarkar (2008)], *ggplot2* [Version 3.3.2; Wickham (2016)], *Rmisc* [Version 1.5; Hope (2013)], *devtools* [Version 2.2.1; Wickham, Hester, & Chang (2019)], *gghalves* [Version 0.1.0; Tiedemann (2020)], *bayesplot* [Version 1.7.1; Gabry, Simpson, Vehtari, Betancourt, & Gelman (2019)], *gridExtra* [Version 2.3; Auguie (2017)], *ggridges* [Version 0.5.3; (**ggridges?**)], *ggbeeswarm* [Version 0.6.0; (**ggbeeswarm?**)], *tibble* [Version 3.0.2; Müller & Wickham (2019)]

```
# Load the packages
library("papaja")
library("LaplacesDemon")
library("rstan")
library("brms")
library("plyr")
library("lattice")
library("ggplot2")
library("dplyr")
library("readr")
library("rmarkdown")
library("Rmisc")
library("devtools")
library("gghalves")
library("bayesplot")
library("bridgesampling")
library("gridExtra")
library("tibble")
library("kableExtra")
library("bayestestR")
library("ggridges")
library("truncnorm")
library("ggbeeswarm")
```

Input

To fit a log-normal BHM in *brms*, we use the *brm* function, as shown below.

```
model_fit <- brm(formula = formula,                    # Model formula
                 data = myData_dataframe,              # Dataset (parameters, observations)
                 family = lognormal(),                  # Family, distribution dependent variable
                 prior = priorsmodel,                  # Specification of the priors
                 warmup = 1000,                        # Nr. iterations for warmup (per chain)
                 iter = 4000,                          # Nr. iterations (per chain)
                 chains = 4,                           # Nr. chains
                 core = 4)                             # Nr. of cores for parallel estimation
```

We can see from the code above that the function requires the following input:

- Formula
- Family
- Data
- Priors

The other options do not require coding and are discussed in the paper.

Formula

The formula is an object that specifies the dependent variable as a function of the independent variables. The `~` (tilde) separates the dependent variable (`rt`) on the left side from the independent variables on the right side. The formula distinguishes between general effects and individual deviations. The first part of the equation on the predictor side of the formula shown below (i.e., `1 + side + dif1 + dif2 + dif3 + dif4`) represents the general effects. The `1` represents the intercept, in the case of the symbolic distance effect μ_γ, `side`, and `dif1` to `dif4` represent the general effects of side and distance (μ_β and μ_δ). Within the round brackets individual deviations and grouping variables are specified. In this case the grouping variable (`ind`) indicates that each participant has an individual effect. In other cases effects might vary per item, block, or even the combination of participant and item. The double bar (`||`) indicates that the correlation between the parameters should not be modeled. When using a single vertical bar (`|`), correlations across all individual effects (intercept, side and distance) are modeled. This approach, however, is a bit more difficult to interpret and would also not correspond to the implemented `Stan` model.

```
formula <- rt2 ~ 1 + side + dif1 + dif2 + dif3 + dif4 +   # General effects
          (1 + side + dif1 + dif2 + dif3 + dif4 || ind)   # Individual effects
```

Family

The argument `family` in the `brm` function specifies the exponential family of distributions according to which the dependent variable is assumed to be distributed. For the log-normal model, this argument takes the value `lognormal`.

Data

The `brm` function requires a data frame (here called `myData_dataframe`) that contains all variables mentioned in the formula. This means that every parameter in the formula should have an element in the data frame that we put in the function. In our case, this should result in seven elements in the data frame (i.e., `rt2`, `side`, `dif1`, `dif2`, `dif3`, `dif4`, and `ind`).

First, we will load the data and clean it according to the requirements in the paper.

```
indat=read.table(url('https://raw.githubusercontent.com/PerceptionCognitionLab/data0/master/lexDec-dist5/ld5.all'))
colnames(indat)=c('sub','block','trial','stim','resp','rt','error')

## Cleaning the data according to criteria discussed in paper
# (code retrieved from Julia Haaf:
# https://github.com/PerceptionAndCognitionLab/bf-order/blob/public/papers/submission/R-scripts/ld5.R)
clean=function()
{
  indat=read.table(url('https://raw.githubusercontent.com/PerceptionCognitionLab/data0/master/lexDec-dist5/ld5.all'))
  colnames(indat)=c('sub','block','trial','stim','resp','rt','error')

  bad1=indat$sub%in%c(34,43)
  bad2=indat$rt<250 | indat$rt>2000
  bad3=indat$err==1
  bad4=indat$block==0 & indat$trial<20
  bad5=indat$trial==0

  bad=bad1 | bad2 | bad3 |bad4 |bad5
  dat=indat[!bad,]
  return(dat)
}

indat1 <- clean()
myData_dataframe <- indat1  # Final dataset
```

Now, we will expand the data.frame we provide the *brm* function. We specify our dependent variable. In our case, these are the RTs. The RTs are given in milliseconds, but we would like them in seconds. Therefore, we transform the original RT (i.e., divide them by 1000) and save this in our new dependent variable `rt2`.

```
### Response time from milliseconds to seconds
myData_dataframe$rt2 <- indat1$rt/1000
```

The next step is to specify the intercept (`inter`) and indicators (`side`, `dif1`, `dif2`, `dif3`, and `dif4`). These are the variables that specify per observation of a variable applies or not. For the side parameter this means that the side indicator is $\frac{1}{2}$ for observations where the digit was smaller than 5, and $-\frac{1}{2}$ for observations where the digit was greater than 5. For the digit parameters it means that the digit indicator is 1 when for the observation the digit was equal to 7, 6, 4 or 3, and 0 when it was not.

```
# create variable inter for the intercept gamma
myData_dataframe$inter <- rep(1, nrow(indat1))

### Add variable with information of the side of the digit to the dataframe, for parameter beta
# j = condition, if j > 5 than x = -1/2, if j < 5 than x = 1/2
# indat1$stim 0 = 2, 1=3, 2=4, 3=6, 4=7, 5=8
# so if bigger than 2 than 1/2, else than -1/2
for (j in 1:nrow(indat1)){
  if (indat1$stim[j] < 3) {
```

```
    myData_dataframe$side[j] <- 1/2}
  else myData_dataframe$side[j] <- -1/2
}

### Add variable with information about difference between digits to the dataframe, for parameters delta's
#### Difference 8 and 7
for (j in 1:nrow(indat1)){
  if (indat1$stim[j] == 5) {  # 5 = 8 , reference: https://github.com/PerceptionCognitionLab/data0/blob/master/lexDec-dist5/ld5.txt
    myData_dataframe$dif1[j] <- 0}
  else if (indat1$stim[j] == 4) {
    myData_dataframe$dif1[j] <- 1}
  else myData_dataframe$dif1[j] <- 0
}

#### Difference 7 and 6
for (j in 1:nrow(indat1)){
  if (indat1$stim[j] == 4) {  # 4 = 7 , reference: https://github.com/PerceptionCognitionLab/data0/blob/master/lexDec-dist5/ld5.txt
    myData_dataframe$dif2[j] <- 0}
  else if (indat1$stim[j] == 3) {  # 3 = 6
    myData_dataframe$dif2[j] <- 1}
  else myData_dataframe$dif2[j] <- 0
}

#### Difference 4 and 3
for (j in 1:nrow(indat1)){
  if (indat1$stim[j] == 1) {  # 1 = 3, 2 = 4 , reference: https://github.com/PerceptionCognitionLab/data0/blob/master/lexDec-dist5/ld5.txt
    myData_dataframe$dif3[j] <- 0}
  else if (indat1$stim[j] == 2) {
    myData_dataframe$dif3[j] <- 1}
  else myData_dataframe$dif3[j] <- 0
}

#### Difference 3 and 2
for (j in 1:nrow(indat1)){
  if (indat1$stim[j] == 0) {  # 0 = 2, 1 = 3 , reference: https://github.com/PerceptionCognitionLab/data0/blob/master/lexDec-dist5/ld5.txt
    myData_dataframe$dif4[j] <- 0}
  else if (indat1$stim[j] == 1) {
    myData_dataframe$dif4[j] <- 1}
  else myData_dataframe$dif4[j] <- 0
}
```

Lastly, we add an indicator to the data frame (`ind`) that specifies which observation belongs to which individual (or group).

```
## Data add necessary information
### Add participant number
myData_dataframe$ind <- indat1$sub + 1
```

Priors

Finally, we specify the priors. In `brms`, priors can be set on the general effects represented by `class = b` and on the individual variation represented by `class = sd`. It is important to note that priors can only be set on the specific parameters that are part of the brms model specification. For example, `brms` only allows setting priors on standard deviations and not on variances.

Therefore, for the normal model of the symbolic distance effect, we will use the priors for standard deviations specified in Equation 9. To set the priors, we use the `set_prior` function, as shown below. In this function, first, the distribution is specified, using the `Stan` programming language. Then, the type of parameter to which the prior applies is defined by `class`. The parameter to which this prior applies is specified by `coef`. For the priors on the individual effects, an additional element has to be specified, namely, the grouping variable (i.e., `group`). For example, the prior distributions on the general side effect, μ_β, and on its variability between individuals, σ_β, can be specified as follows:

```
set_prior("normal(-0.5, 1)", class = "b", coef="side")  # general effect
set_prior("inv_gamma(13.8, 4.1)", class = "sd",       # individual variation
          coef="side", group = "ind")
```

All priors are saved in an R `data.frame` that is used in the `brm` function.

```
priorslogmodel <- c(set_prior("normal(-0.5, 1)", class = "Intercept"),
                    set_prior("inv_gamma(13.8, 4.1)", class = "sd", coef="Intercept", group = "sub1"),
                    set_prior("normal(0, 0.07)", class = "b", coef="side"),
                    set_prior("inv_gamma(13.8, 0.7)", class = "sd", coef="side", group = "sub1"),
                    set_prior("normal(0, 0.07)", class = "b", coef="dif1"),
                    set_prior("inv_gamma(13.8, 0.7)", class = "sd", coef="dif1", group = "sub1"),
                    set_prior("normal(0, 0.07)", class = "b", coef="dif2"),
                    set_prior("inv_gamma(13.8, 0.7)", class = "sd", coef="dif2", group = "sub1"),
                    set_prior("normal(0, 0.07)", class = "b", coef="dif3"),
                    set_prior("inv_gamma(13.8, 0.7)", class = "sd", coef="dif3", group = "sub1"),
                    set_prior("normal(0, 0.07)", class = "b", coef="dif4"),
                    set_prior("inv_gamma(13.8, 0.7)", class = "sd", coef="dif4", group = "sub1"),
                    set_prior("inv_gamma(13.8, 4.1)", class = "sigma"))
```

As described in the paper, we could check whether these priors are reasonable considering our expectations by performing prior prediction.

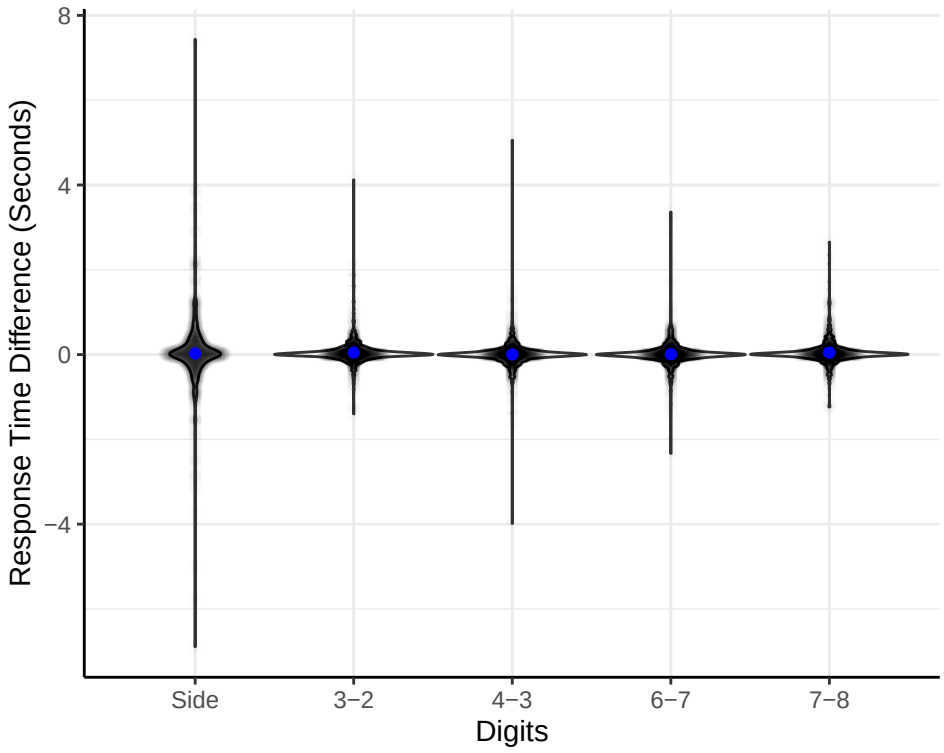


Figure 1: Predicted mean RT difference between conditions per trial. The blue line represents the aggregated trend.

Fit the Model

Now that we have the required input, we can estimate the model.

```
logmodel_fit <- brm(formula = logformula,                    # Model formula
                    data = myData_dataframe,                 # Dataset (parameters, observations)
                    family = lognormal(),                    # Family, distribution dependent variable
                    prior = priorslogmodel,                  # Specification of the priors
                    # sample_prior = "yes",                  # To calculate Bayes factors for point hypotheses
                    warmup = 1000,                           # Nr. iterations for warmup (per chain)
                    iter = 4000,                             # Nr. iterations (per chain)
                    chains = 4,                              # Nr. chains
                    core = 4)                                # Nr. of cores for parallel estimation
```

We have already estimated the model and saved it in the R object `brmmodel8log13012022.rds`. In case you do not want to wait until the model has been fitted but continue immediately with the rest of this tutorial, you can load this R object that contains the model estimations.

```
logmodel_fit <- readRDS(file = "myPath/brmmodel8log13012022.rds")

# For model comparison - sample_prior was set to "yes"
```

```
logmodel_fit_bf <- readRDS("myPath/BRMSmodelfit18_26052022_bf.rds")
```

Output

The `summary` function provides an overview of the model fit. From this function, we can get a lot of information.

```
summary(logmodel_fit)
```

```
##  Family: lognormal
##   Links: mu = identity; sigma = identity
## Formula: rt2 ~ 1 + side + dif1 + dif2 + dif3 + dif4 + (1 + side + dif1 + dif2 + dif3 + dif4 || sub1)
##    Data: indat1 (Number of observations: 17031)
##   Draws: 4 chains, each with iter = 4000; warmup = 1000; thin = 1;
##          total post-warmup draws = 12000
##
## Group-Level Effects:
## ~sub1 (Number of levels: 52)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     0.16      0.02     0.14     0.20 1.00     1812     3635
## sd(side)          0.03      0.00     0.02     0.04 1.00     8860     9582
## sd(dif1)          0.03      0.00     0.02     0.04 1.00    10764     9364
## sd(dif2)          0.03      0.00     0.02     0.04 1.00    11432     9997
## sd(dif3)          0.04      0.01     0.03     0.05 1.00     9919     9905
## sd(dif4)          0.03      0.00     0.02     0.04 1.00     8952     9472
##
## Population-Level Effects:
##           Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept    -0.56      0.02    -0.61    -0.52 1.01      584     1013
## side         -0.01      0.01    -0.02     0.01 1.00    12910     9851
## dif1          0.02      0.01     0.00     0.03 1.00    13772     9994
## dif2          0.06      0.01     0.05     0.08 1.00    12740     9767
## dif3          0.08      0.01     0.07     0.10 1.00    11760    10025
## dif4          0.03      0.01     0.02     0.05 1.00    14047    10478
##
## Family Specific Parameters:
##       Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sigma     0.22      0.00     0.21     0.22 1.00    26172     8091
##
## Draws were sampled using sampling(NUTS). For each parameter, Bulk_ESS
## and Tail_ESS are effective sample size measures, and Rhat is the potential
## scale reduction factor on split chains (at convergence, Rhat = 1).
```

`brms` offers the `launch_shinystan` function which will load a shiny app with the results of the model fit. It is very extensive. It, for instance, includes model diagnostics as trace plots, posterior plots, prediction plots and way more.

```
launch_shinystan(logmodel_fit)
```

It also has the possibility to save plots. However, if you want the full control over the layout, it might be

better to code them yourself. Therefore, we will show how to create the figures presented in the paper.

Trace plot

The trace plot shows whether the posterior distribution of the parameter has converged. With the package `mcmc_plot` function (used to be `stanplot`) it is possible to obtain the trace plots per parameter by setting the option `type` to `trace`. With the `variable` argument, you can specify for which parameters you want to inspect the trace plots. With `variables`, you can check the names of the parameters in the `brms` output.

```
variables(logmodel_fit)
```

The trace plot will automatically remove the warm-up (burn-in) period from the chain (which is desirable), however, the x axis will indicate that it is showing iteration 0 to 4000 (in our case) while it is actually showing the 1000th to 5000th iteration. As the `plot` option returns a `ggplot` object, it is possible to manually adjust the plot using `ggplot` functions. We created the function `traceplotfunc` to obtain the trace plot for the parameter you are interested in. In the figure below, we show the trace plots for all the general effects and variances.

```
# Default trace plot for parameter mu4
# I think warmup already deleted, but then have to adjust labels x axis
trdif2brms <- mcmc_plot(logmodel_fit,       # Model fit
                        variable = "b_dif2",  # Select parameter,
                        # it is also possible to select more than one at the same time,
                        # such as c("mu2", "mu3","mu4")
                        type = "trace")   # Specify that you want trace plots


# Function
traceplotfunc <- function(parameter, title){
trdif2brms <- mcmc_plot(logmodel_fit,       # Model fit
                        variable = parameter,  # Select parameter,
                        type = "trace")   # Specify that you want trace plots


trdif2brms2 <- trdif2brms +
               scale_x_continuous(breaks=c(1000, 3000, 5000),
                                  labels=c(2000, 4000, 6000)) +
                theme_classic() +
                theme(axis.title.y=element_text(angle = 0,
                                                vjust = 0.5,
                                                hjust = -0.5,
                                                margin = margin(0, 0.7, 0, 0, "cm"), size = 14),
                      axis.line.x = element_line(color="black"),
                     axis.line.y = element_line(color="black"),
                     #legend.position = "none",
                     plot.margin = unit(c(0.3,0.1,0.4,0),"cm")) +
               scale_color_manual(values=c("#E66101", "#998EC3", "#542788", "#F1A340")) +
               ylab(title) +
               xlab("Iteration") +
               labs(subtitle = "") #+
```

```
              #ylim(c(0.01, 0.07)) +
              #scale_y_continuous(limits = c(0.01, 0.07),
                                  #breaks = c(0.02, 0.04, 0.06),
                                  #labels = c(0.02, 0.04, 0.06))

return(trdif2brms2)
}

# Show trace plot for one digit parameter
tr2 <- traceplotfunc(parameter = "b_side", title = expression(~mu[~beta]))
tr2 <- tr2 + theme(legend.position = "none")
tr3 <- traceplotfunc(parameter = "b_dif1", title = expression(~mu[~delta[~7]]))
tr3 <- tr3 + theme(legend.position = "none")
tr4 <- traceplotfunc(parameter = "b_dif2", title = expression(~mu[~delta[~6]]))
tr4 <- tr4 + theme(legend.position = "none")
tr5 <- traceplotfunc(parameter = "b_dif3", title = expression(~mu[~delta[~4]]))
tr5 <- tr5 + theme(legend.position = "none")
tr6 <- traceplotfunc(parameter = "b_dif4", title = expression(~mu[~delta[~3]]))
legendtr <- cowplot::get_legend(tr6)
tr6 <- tr6 + theme(legend.position = "none")

# Plot general effects together
grid.arrange(tr2, tr3, tr4, tr5, tr6, legendtr, nrow = 2, ncol = 3)
```

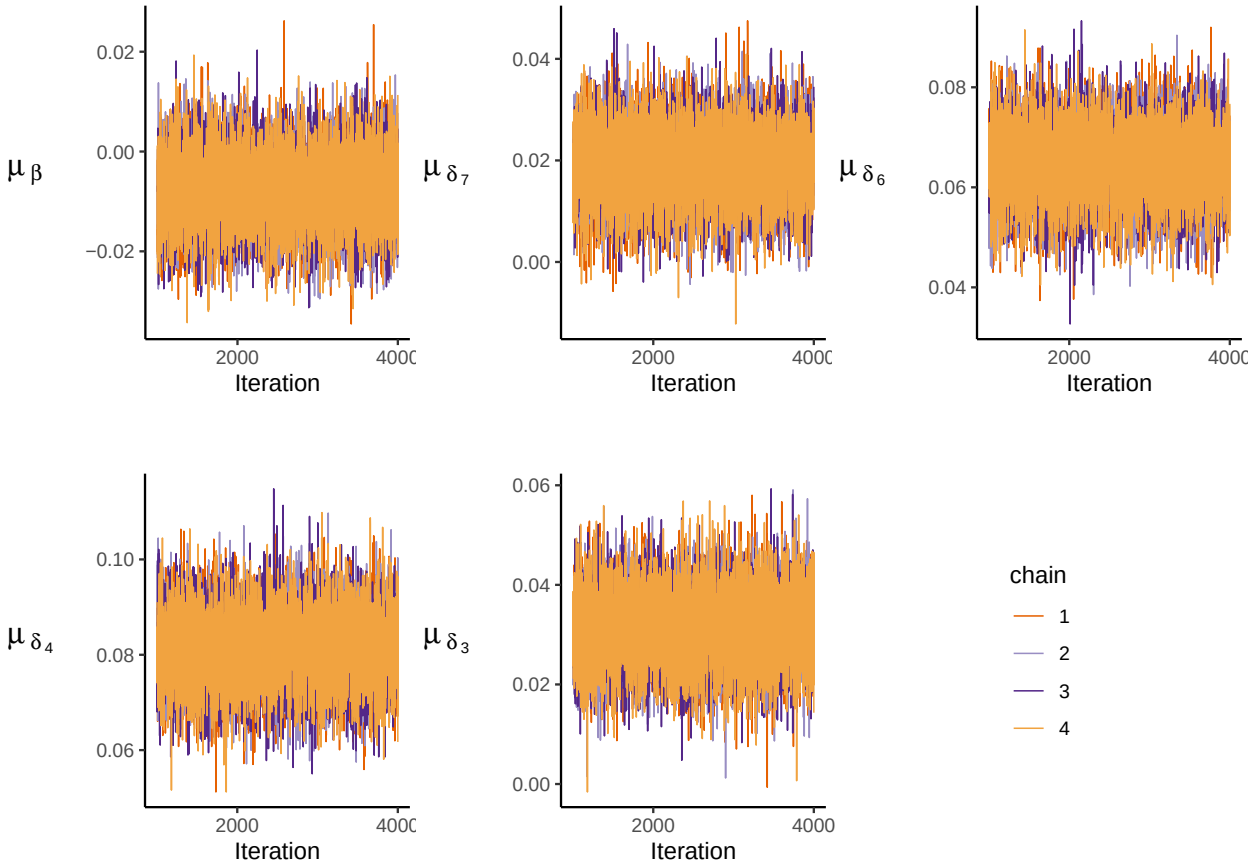
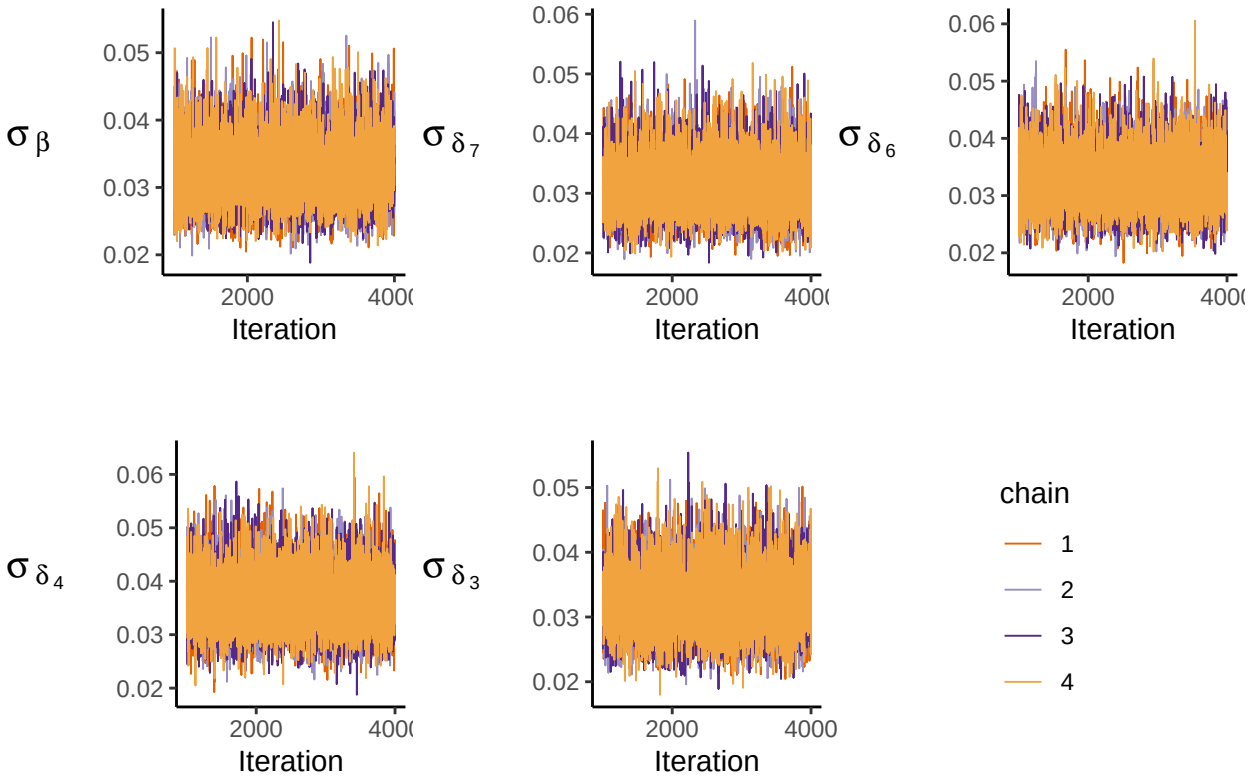


Figure 2: Trace plots of the general effect parameters.

```
# Show trace plot for one sd parameter
tr2 <- traceplotfunc(parameter = "sd_sub1__side", title = expression(~sigma[~beta]))
tr2 <- tr2 + theme(legend.position = "none")
tr3 <- traceplotfunc(parameter = "sd_sub1__dif1", title = expression(~sigma[~delta[~7]]))
tr3 <- tr3 + theme(legend.position = "none")
tr4 <- traceplotfunc(parameter = "sd_sub1__dif2", title = expression(~sigma[~delta[~6]]))
tr4 <- tr4 + theme(legend.position = "none")
tr5 <- traceplotfunc(parameter = "sd_sub1__dif3", title = expression(~sigma[~delta[~4]]))
tr5 <- tr5 + theme(legend.position = "none")
tr6 <- traceplotfunc(parameter = "sd_sub1__dif4", title = expression(~sigma[~delta[~3]]))
legendtr <- cowplot::get_legend(tr6)
tr6 <- tr6 + theme(legend.position = "none")

# Plot general effects together
grid.arrange(tr2, tr3, tr4, tr5, tr6, legendtr, nrow = 2, ncol = 3)
```



Because there are many parameters (each for every individual), it is hard to inspect them all individually. A more efficient way to check if the parameters have converged is by checking the $\hat{R}$ and the number of effective samples.

Rhat & Number of Effective Samples

As for the trace plots, the `mcmc_plot` function offers the possibility to plot the $\hat{R}$ (Rhat) and the ratio of effective sample size to the total posterior sample size (this is different from what is plotted in the paper).

```
# Plot frequency of rhat
mcmc_plot(logmodel_fit, type = "rhat_hist")   # Specify that you want trace plots

# Plot frequency of the ratio
mcmc_plot(logmodel_fit, type = "neff_hist")
```

However, these figures can be manually adjusted. To obtain the figures from the paper, the following steps can be followed. First, we save the values for $\hat{R}$ and parameter names.

```
# Obtain rhat and neff brms
## Rhat
rhatbrms <- mcmc_plot(logmodel_fit, type = "rhat_hist")$data$value  # save rhat values
rhatbrms_para <- mcmc_plot(logmodel_fit, type = "rhat_hist")$data$parameter  # save parameter names
rhatbrmsratio_comb <- data.frame(parameter = rhatbrms_para, value = rhatbrms)  # combine values and parameters names into one data.frame
```

Then, we obtain the $\hat{R}$ of the general effects.

```
# Obtain rhats of general effects
rhatbrms_v2 <- mcmc_plot(logmodel_fit, type = "rhat_hist")$data
fixedrhatbrms_v2 <- c(rhatbrms_v2$value[rhatbrms_v2$parameter == "b_Intercept"],
                      rhatbrms_v2$value[rhatbrms_v2$parameter == "b_side"],
                      rhatbrms_v2$value[rhatbrms_v2$parameter == "b_dif1"],
                      rhatbrms_v2$value[rhatbrms_v2$parameter == "b_dif2"],
                      rhatbrms_v2$value[rhatbrms_v2$parameter == "b_dif3"],
                      rhatbrms_v2$value[rhatbrms_v2$parameter == "b_dif4"])

deltameanrrbrms <- mean(fixedrhatbrms_v2[3:6])  # compute mean of delta parameters (digit effects)
```

We also save the $\hat{R}$ of the individual effects.

```
# Obtain rhats of all individual effects
rhatbrmsratio3 <- c(rhatbrmsratio_comb[grep(",Intercept]", rhatbrmsratio_comb$parameter),]$value,
                    rhatbrmsratio_comb[grep(",side]", rhatbrmsratio_comb$parameter),]$value,
                    rhatbrmsratio_comb[grep(",dif1]", rhatbrmsratio_comb$parameter),]$value,
                    rhatbrmsratio_comb[grep(",dif2]", rhatbrmsratio_comb$parameter),]$value,
                    rhatbrmsratio_comb[grep(",dif3]", rhatbrmsratio_comb$parameter),]$value,
                    rhatbrmsratio_comb[grep(",dif4]", rhatbrmsratio_comb$parameter),]$value)
```

Next, the data of the individual effects is restructured to the format used in *ggplot*.

```
# Restructure in a data frame we can use to create a figure with ggplot
rstanrhatneff_v2 <- data.frame(package = rep("Brms", 312),
                               parameter = rep(c("Gamma", "Beta", "Delta",                                    "Delta", "De
                                               each = 52),
                               rhat = rhatbrmsratio3)
```

We follow the same steps for the number of effective samples (Neff).

```
neffbrmsratio_value <- mcmc_plot(logmodel_fit, type = "neff_hist")$data$value   # save neff values, these are the values of the ration Neff/N, N re
neffbrmsratio_para <- mcmc_plot(logmodel_fit, type = "neff_hist")$data$parameter  # save parameter names
neffbrmsratio_comb <- data.frame(parameter = neffbrmsratio_para, value = neffbrmsratio_value)  # combine values and parameter names into data frame

# Neff of general effects
fixedneffbrms2 <- c(neffbrmsratio_comb[grep("b_Intercept", neffbrmsratio_comb$parameter),]$value[1],
                    neffbrmsratio_comb[grep("b_side", neffbrmsratio_comb$parameter),]$value[1],
                    neffbrmsratio_comb[grep("b_dif1", neffbrmsratio_comb$parameter),]$value[1],
                    neffbrmsratio_comb[grep("b_dif2", neffbrmsratio_comb$parameter),]$value[1],
                    neffbrmsratio_comb[grep("b_dif3", neffbrmsratio_comb$parameter),]$value[1],
                    neffbrmsratio_comb[grep("b_dif4", neffbrmsratio_comb$parameter),]$value[1])

fixedneffbrms3 <- fixedneffbrms2 * 12000  # we multiply values with total number of iterations (12000) to get Neff (used to be ration Neff/N, N rep
fixedneffbrmsmean <- mean(fixedneffbrms3[3:6])  # calculate mean Neff of delta parameters

# Neff of individual effects
neffbrmsratio3 <- c(neffbrmsratio_comb[grep(",Intercept]", neffbrmsratio_comb$parameter),]$value,
                    neffbrmsratio_comb[grep(",side]", neffbrmsratio_comb$parameter),]$value,
                    neffbrmsratio_comb[grep(",dif1]", neffbrmsratio_comb$parameter),]$value,
                    neffbrmsratio_comb[grep(",dif2]", neffbrmsratio_comb$parameter),]$value,
                    neffbrmsratio_comb[grep(",dif3]", neffbrmsratio_comb$parameter),]$value,
                    neffbrmsratio_comb[grep(",dif4]", neffbrmsratio_comb$parameter),]$value)
neffbrms2 <- neffbrmsratio3  * 12000  # we multiply values with total number of iterations (12000) to get Neff (used to be ration Neff/N, N represe
```

```
# Restructure in a data frame we can use to create a figure with ggplot
neffrstanbrms2 <- data.frame(package = rep("Brms", 312),
                             parameter = rep(c("Gamma", "Beta", "Delta", "Delta",                                    "Delta", "Del
                             neff = c(neffbrms2))
```

Finally, we visualize the $\hat{R}$ and Neff using violin plots.

```
# Create violin plots
## Rhat
  figrhatnew <- ggplot(rstanrhatneff_v2, aes(x = factor(parameter, level = c("Gamma", "Beta", "Delta")), y = rhat, colour = package)) +
    geom_violin(width = 1) +
    geom_quasirandom(alpha = 0.1, width = 0.2, dodge.width=1) +
    geom_point(aes(x = 1, y = fixedrhatbrms_v2[1]), shape = 8, colour="darkorange", size = 2) +
    geom_point(aes(x = 2, y = fixedrhatbrms_v2[2]), shape = 8, colour="darkorange", size = 2) +
    geom_point(aes(x = 3, y = deltameanrrbrms ), shape = 8, colour="darkorange", size = 2) +
    xlab("Parameter") + ylab(expression(hat(R))) +
    scale_colour_manual(values = "darkorange") +
    theme_classic() +
    theme(legend.position = "none") +
    scale_x_discrete(labels= c(
      expression(~gamma),
      expression(~beta),
      expression(~delta))) +
    labs(subtitle = "C") +
    theme(axis.title.y = element_text(angle = 0,
                                      vjust = 0.5,
                                      hjust = -0.5,
                                      margin = margin(0, 1.1, 0, 0, "cm")))


## Neff
figneffnew <- ggplot(neffrstanbrms2, aes(x = factor(parameter, level = c("Gamma", "Beta", "Delta")), y = neff, colour = package)) +
    geom_violin(width = 1) +
    geom_quasirandom(alpha = 0.1, width = 0.2, dodge.width=1) +
    geom_point(aes(x=1, y= fixedneffbrms3[1]), shape = 8, colour="darkorange", size = 2) +
    geom_point(aes(x=2, y= fixedneffbrms3[2]), shape = 8, colour="darkorange", size = 2) +
    geom_point(aes(x=3, y= fixedneffbrmsmean), shape = 8, colour="darkorange", size = 2) +
    xlab("Parameter") + ylab("Number of effective samples") +
    scale_colour_manual(values = "darkorange") +
    theme_classic() +
    theme(legend.position = "none") +
    scale_x_discrete(labels= c(
      expression(~gamma),
      expression(~beta),
      expression(~delta))) +
    labs(subtitle = "D") +
    geom_hline(yintercept = 20000, linetype = "dashed", color = "darkorange", size = .3)

# Plot together
grid.arrange(figrhatnew, figneffnew, nrow = 1, ncol = 2)
```

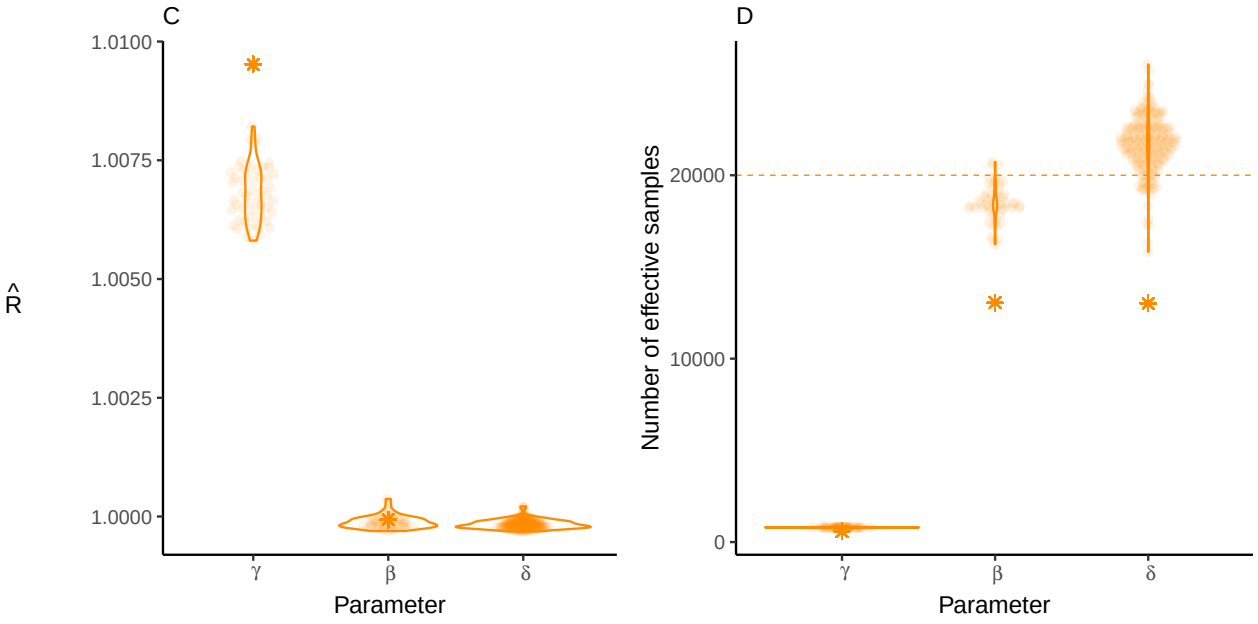


Figure 3: The frequency of the $\hat{R}$ statistic and the number of effective samples. A: The frequency of the $\hat{R}$. B: The frequency of the number of effective samples where the dashed lines represent the total number of iterations.

General effects

The general effects are represented by μ_{δ_7}, μ_{δ_6}, μ_{δ_4}, μ_{δ_3}, and μ_{β}. To investigate whether there is a digit and side effect, we will inspect the estimates for these parameters.

Posterior Distribution With the `mcmc_plot` function, the posterior distribution per parameter can be displayed.

```
# Seperate plot per parameter of posterior density
mcmc_plot(logmodel_fit,
     type = "dens")
```

Another possibility is to use the `bayesplot` package to display all the posterior densities together in one plot. This makes it easier to compare the posterior distributions of each parameter.

```
post8 <- as.array(logmodel_fit)  # change structure of model fit

color_scheme_set("orange")  # plot the distributions in orange

pairs8 <- mcmc_areas(
  post8,    # Structured model fit
```

```
  pars = c("b_side", "b_dif1", "b_dif2", "b_dif3", "b_dif4"),  # Select parameters
  prob = 0.95, # 80% intervals  # Plot 95% credible intervals
  prob_outer = 0.99, # 99%     # 99% outer interval
  point_est = "mean"           # Plot line in the middle
)
# Returns ggplot object

# Adjust ggplot object further
pairs82 <- pairs8 +
           theme_apa() +
           ggtitle("Brms") +
           xlim(-0.05, 0.095) +
           #scale_x_continuous(breaks = c(-0.025, 0, 0.025, 0.05, 0.075)) +
           scale_y_discrete(labels= c(
                                              expression(~mu[~beta]),
                                              expression(~mu[~delta[~7]]),
                                              expression(~mu[~delta[~6]]),
                                              expression(~mu[~delta[~4]]),
                                              expression(~mu[~delta[~3]])))

pairs82
```

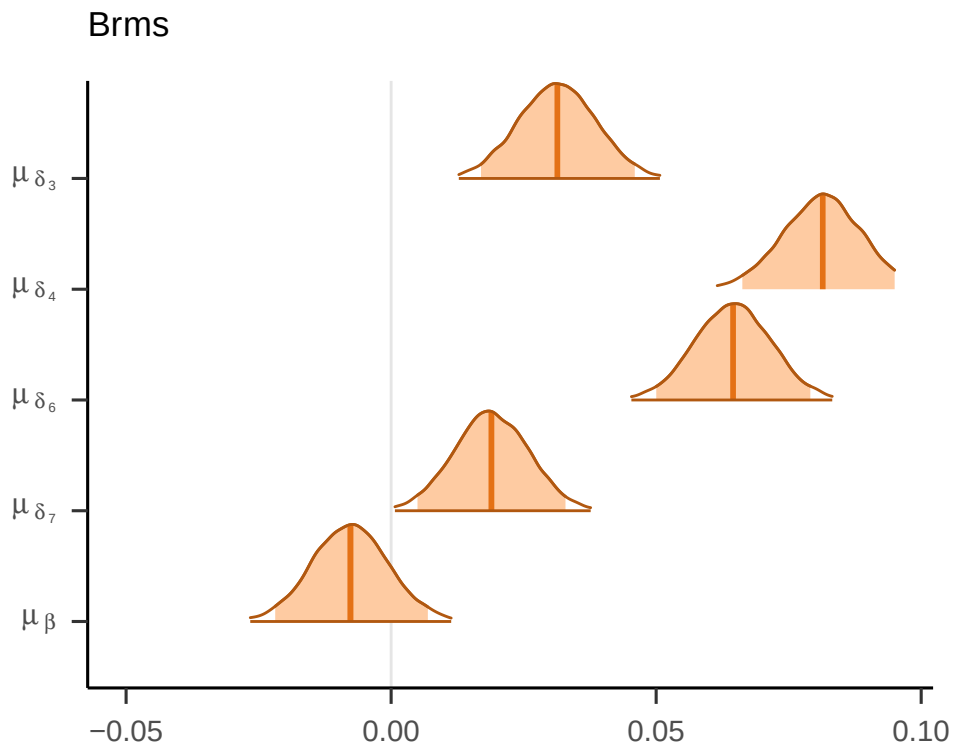


Figure 4: The posterior distributions for the general effects. The middle line within the distributions represents the mean posterior. The shaded area within the distributions represents 95% of the probability mass.

We can present more information about the distribution by providing a table with the estimated mean of the posterior distribution, the standard error of this mean, the lower and upper bound of the 95% credible

interval, the $\hat{R}$ and the number of effective samples.

```
# Obtain the mean, standard error, 95% credible interval, rhat and number of effective samples for the intercept, and the digit and side parameters
fitbrmshiertable <- summary(logmodel_fit)[[14]][,c("Estimate", "Est.Error", "l-95% CI", "u-95% CI", "Bulk_ESS", "Rhat")]  # variance

# The results are transformed to a dataframe that allows us to plot the results nicely in a table
fitbrmshiertable2 <- as.data.frame(fitbrmshiertable)

# We manually set the row names of the table using latex for mathematical symbols
rownamesfitbrmshiertable <-  c("$\\mu_{\\gamma}$","$\\mu_{\\beta}$", "$\\mu_{\\delta_{7}}$",
                               "$\\mu_{\\delta_{6}}$", "$\\mu_{\\delta_{4}}$", "$\\mu_{\\delta_{3}}$")

# We manually set the column names of the table
colnames(fitbrmshiertable2) <- c("Mean", "SE", "Lower Bound", "Upper Bound", "$n_{eff}$","$\\hat{R}$")

# The column with row names is added to the table
fitbrmshiertable3 <- add_column(fitbrmshiertable2,
                                Parameters = rownamesfitbrmshiertable,
                                .before = "Mean")

# Using the apa_table function from the papaja package the results are presented in an apa table
apa_table(fitbrmshiertable3,   # Data frame with results
          row.names = FALSE,    # Default row names are deleted (we set them ourself)
          caption = "Posterior Mean, Standard Error (SE) of the Mean, Lower and
          Upper Bound of the 95\\% Credible Interval, the Number of Effective
          Samples, and the $\\hat{R}$ of the General Effect Parameters. ",
          align = c("l", "c", "c", "c", "c", "c", "c"),   # How should results be presented, in the midle of the table or outlined left
          digits = 3,       # The number of digits behind the ".".
          placement = "h",  # Place after code
          escape = FALSE)   # This indicates that the results contain latex code that should be read as latex code. It is important to do this in c
```

Table 1: Posterior Mean, Standard Error (SE) of the Mean, Lower and Upper Bound of the 95% Credible Interval, the Number of Effective Samples, and the $\hat{R}$ of the General Effect Parameters.

Parameters	Mean	SE	Lower Bound	Upper Bound	n_{eff}	$\hat{R}$
μ_{γ}	-0.560	0.023	-0.608	-0.515	583.573	1.009
μ_{β}	-0.008	0.007	-0.022	0.007	12,909.505	1.000
μ_{δ_7}	0.019	0.007	0.005	0.033	13,771.944	1.000
μ_{δ_6}	0.065	0.007	0.050	0.079	12,739.547	1.000
μ_{δ_4}	0.081	0.008	0.066	0.097	11,759.558	1.000
μ_{δ_3}	0.031	0.007	0.017	0.046	14,046.688	1.001

Individual Effects

Next, we look at the individual deviations from the general effects. Therefore, we inspect the standard deviation and the individual estimates of the digit and side effects.

Standard Deviation Estimates First, we can show a table with the posterior distribution estimates of the standard deviation parameters. Note that this is different from *rstan* where we estimated the variances instead of the standard deviations.

```
# Obtain standard deviation estimates
fitbrmshiertable2 <- summary(logmodel_fit)[[17]]$sub1[,c("Estimate", "Est.Error", "l-95% CI", "u-95% CI", "Bulk_ESS", "Rhat")]

# The results are transformed to a dataframe that allows us to plot the results nicely in a table
fitbrmshiertable22 <- as.data.frame(fitbrmshiertable2)

# We manually set the row names of the table using latex for mathematical symbols
rownamesfitbrmshiertable2 <-  c("$\\sigma_{\\gamma}$","$\\sigma_{\\beta}$", "$\\sigma_{\\delta_{7}}$",
                                "$\\sigma_{\\delta_{6}}$", "$\\sigma_{\\delta_{4}}$", "$\\sigma_{\\delta_{3}}$")

# The column with row names is added to the table
colnames(fitbrmshiertable22) <- c("Mean", "SE", "Lower Bound", "Upper Bound", "$n_{eff}$","$\\hat{R}$")

# The column with row names is added to the table
fitbrmshiertable32 <- add_column(fitbrmshiertable22,
                                 Parameters = rownamesfitbrmshiertable2,
                                 .before = "Mean")

# Using the apa_table function from the papaja package the results are presented in an apa table
apa_table(fitbrmshiertable32,  # Data frame with results
          row.names = FALSE,    # Default row names are deleted (we set them ourself)
          caption = "Posterior Standard Deviation, Standard Error (SE) of the Standard Deviation,
          Lower and Upper Bound of the 95\\% Credible Interval, the Number of
          Effective Samples, and the $\\hat{R}$ of the Standard Deviation Parameters          as estimated by Brms",
          align = c("l", "c", "c", "c", "c", "c", "c"),  # How should results be presented, in the midle of the table or outlined left
          digits = 3,        # The number of digits behind the ".".
          place = "h",       # Place after code
          escape = FALSE)    # This indicates that the results contain latex code that should be read as latex code. It is important to do this in
```

Table 2: Posterior Standard Deviation, Standard Error (SE) of the Standard Deviation, Lower and Upper Bound of the 95% Credible Interval, the Number of Effective Samples, and the $\hat{R}$ of the Standard Deviation Parameters as estimated by Brms

Parameters	Mean	SE	Lower Bound	Upper Bound	n_{eff}	$\hat{R}$
σ_{γ}	0.165	0.017	0.136	0.201	1,811.737	1.001
σ_{β}	0.033	0.005	0.025	0.043	8,860.093	1.000
σ_{δ_7}	0.032	0.005	0.023	0.042	10,763.913	1.000
σ_{δ_6}	0.033	0.005	0.024	0.043	11,432.334	1.000
σ_{δ_4}	0.036	0.005	0.027	0.048	9,918.653	1.000
σ_{δ_3}	0.033	0.005	0.024	0.043	8,952.156	1.000

Individual Estimates Next, we can look at the individual estimates for the digit and side effect parameters in two different ways. First, we plot the individual estimates and display their 95% credible interval. The credible interval is displayed in pink when it contains zero, and displayed in blue when it does not contain zero. In addition, the dashed line represent the general effect estimate.

The code below shows you how to obtain this code for one parameter. The other parameters follow logically from this example and would result in the figure below.

```
# Gamma
# Extract the parameter estimates
brmsgammar <- coef(logmodel_fit)$sub1[,,"Intercept"][,c("Estimate", "Q2.5", "Q97.5")]
brmsgammar <- as.data.frame(brmsgammar)

# Extract the general effect estimate of the parameter
fixbremsgammar <- fixef(logmodel_fit)["Intercept","Estimate"]

# Order the estimates ascending
brmsgammar2 <- brmsgammar[order(brmsgammar$Estimate),]
brmsgammar2$order <- c(1:52)

# Add parameter to dataframe. When the credible interval contains zero, the parameter is set pink, otherwise to blue
for (i in 1:nrow(brmsgammar2)){  # 1 if CI contains zero, 0 otherwise
  if (brmsgammar2$Q2.5[i]*brmsgammar2$Q97.5[i] < 0) {
    brmsgammar2$zero2[i] <- "#F8766D"
  } else (brmsgammar2$zero2[i] <- "#00BFC4")
}

# Plot the individual estimates for gamma (intercept)
gammarandomplotbrms <- ggplot(brmsgammar2, aes(x = order, y = Estimate, ymin = Q2.5, ymax = Q97.5)) +
  geom_linerange(color = brmsgammar2$zero2) +
  geom_point() +
  geom_hline(yintercept = 0, linetype = "dashed", alpha = .3) +
  labs(title = expression(~gamma[~i]), y = "") +
  geom_hline(yintercept = fixbremsgammar, color = "red", alpha = .3) +
  theme(legend.position = "none",
        axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank(),)
```

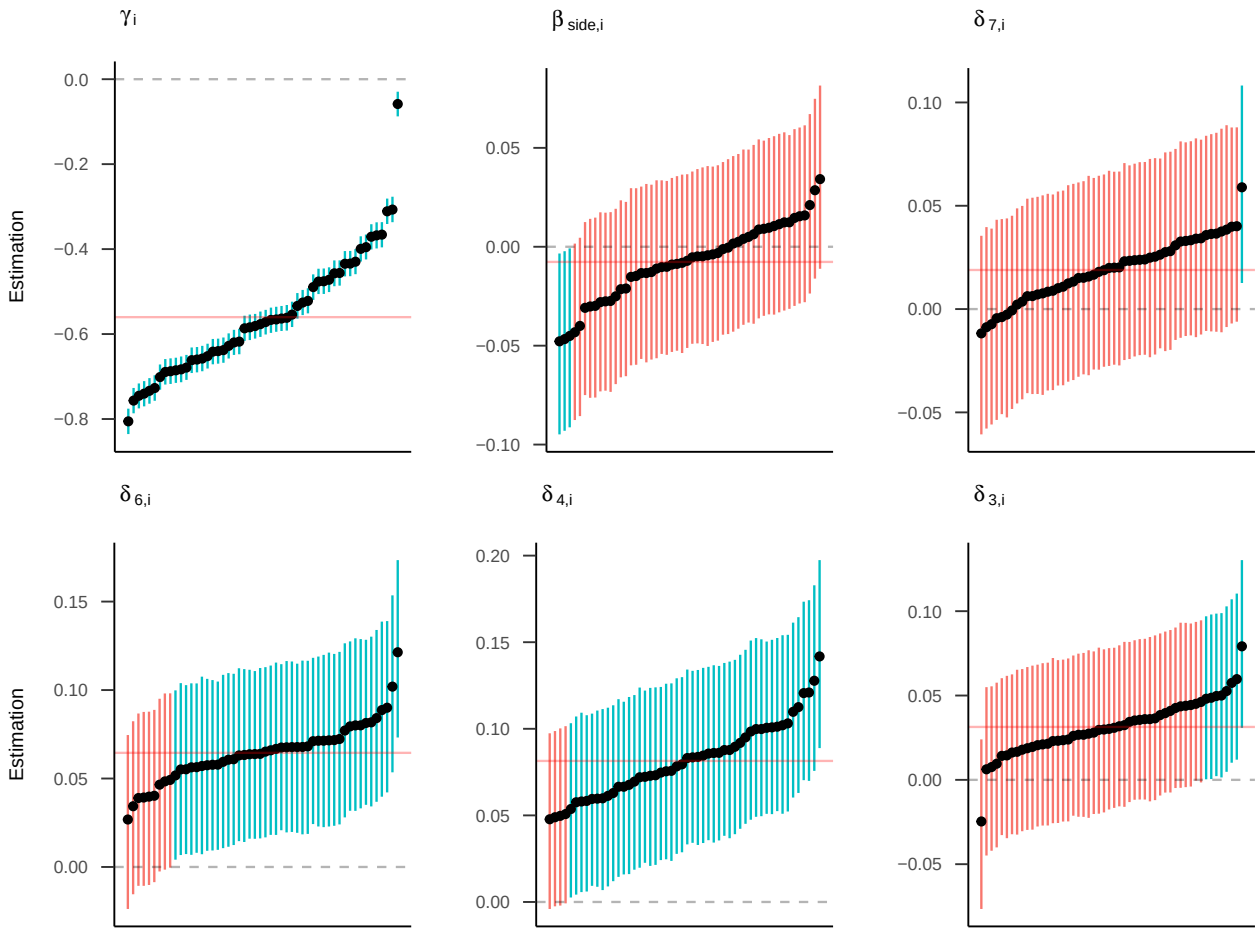


Figure 5: The posterior means with 95% credible interval in pink and red as estimated by *brms* for every individual shown in increasing order. The dashed line represents the general posterior mean. A pink interval means that the interval contains zero, a blue interval represents an interval that does not contain zero.

The second way is the plot the individual estimates and display the variation of these individual estimates. For this figure, we used the design and code from Langen (2020).

```
# Brms
## Create data.frame

### Delta's individual estimates
estimatesbrms <- coef(logmodel_fit)$sub1
#### Delta 1 - difference 8 and 7
del1estimatesbrms <- estimatesbrms[,,"dif1"]

#### Delta 2 - difference 7 and 6
del2estimatesbrms <- estimatesbrms[,,"dif2"]
```

```
#### Delta 3 - difference 4 and 3
del3estimatesbrms <- estimatesbrms[,,"dif3"]

#### Delta 4 - difference 3 and 2
del4estimatesbrms <- estimatesbrms[,,"dif4"]

#### create dataframe
parameter_m7 <- rep(c("Delta 1", "Delta 2", "Delta 3", "Delta 4"), each = 52 )

#### gamma_summary2 not using for plot
#### beta_summary2 not using for plot
parest_m7brms <- rbind(del1estimatesbrms[,1], del2estimatesbrms[,1], del3estimatesbrms[,1], del4estimatesbrms[,1])
parest_m7brms2 <- t(parest_m7brms)
parest_m7brms3 <- as.data.frame(parest_m7brms2)
parest_m7brms4 <- c(parest_m7brms3$V1, parest_m7brms3$V2, parest_m7brms3$V3, parest_m7brms3$V4)
datafr_m7brms <- data.frame(parameter_m7, parest_m7brms4)

# Plot
set.seed(321)
datafr_m7brms$x <- rep(c(1, 2, 3, 4), each = 52)
datafr_m7brms$xj <- jitter(datafr_m7brms$x, amount = 0.09)
datafr_m7brms$id <- rep(c(1:52), 4)

plotmodelest_m7brms <- ggplot(data=datafr_m7brms, aes(y=parest_m7brms4)) +

  #Add geom_() objects
  geom_point(data = datafr_m7brms %>% filter(x=="1"), aes(x=xj), color = 'dodgerblue', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7brms %>% filter(x=="2"), aes(x=xj), color = 'darkgreen', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7brms %>% filter(x=="3"), aes(x=xj), color = 'darkorange', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7brms %>% filter(x=="4"), aes(x=xj), color = 'gold2', size = 1.5,
             alpha = .6) +

  geom_line(aes(x=xj, group=id), color = 'lightgray', alpha = .3) +

  geom_line(data=data.frame(x=-1:4.5,y=0), aes(x = x, y = y), linetype = "dashed", size = .3) +

  geom_half_violin(
    data = datafr_m7brms %>% filter(x=="1"),aes(x = x, y = parest_m7brms4), position = position_nudge(x = 3.2),
    side = "r", fill = 'dodgerblue', alpha = .5, color = "dodgerblue", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7brms %>% filter(x=="2"),aes(x = x, y = parest_m7brms4), position = position_nudge(x = 2.2),
    side = "r", fill = "darkgreen", alpha = .5, color = "darkgreen", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7brms %>% filter(x=="3"),aes(x = x, y = parest_m7brms4), position = position_nudge(x = 1.2),
    side = "r", fill = "darkorange", alpha = .5, color = "darkorange", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7brms %>% filter(x=="4"),aes(x = x, y = parest_m7brms4), position = position_nudge(x = 0.2),
    side = "r", fill = "gold2", alpha = .5, color = "gold2", trim = TRUE) +
```

```
  #Define additional settings
  scale_x_continuous(breaks=c(1,2,3,4), labels=c(expression(~delta[~7]), expression(~delta[~6]), expression(~delta[~4]), expression(~delta[~3])))
  xlab("Parameter") + ylab("Value") +
  ggtitle('Brms') +
  theme_classic() +
  coord_cartesian(xlim = c(0.75, 5), ylim = c(-0.1, 0.17))

plotmodelest_m7brms
```

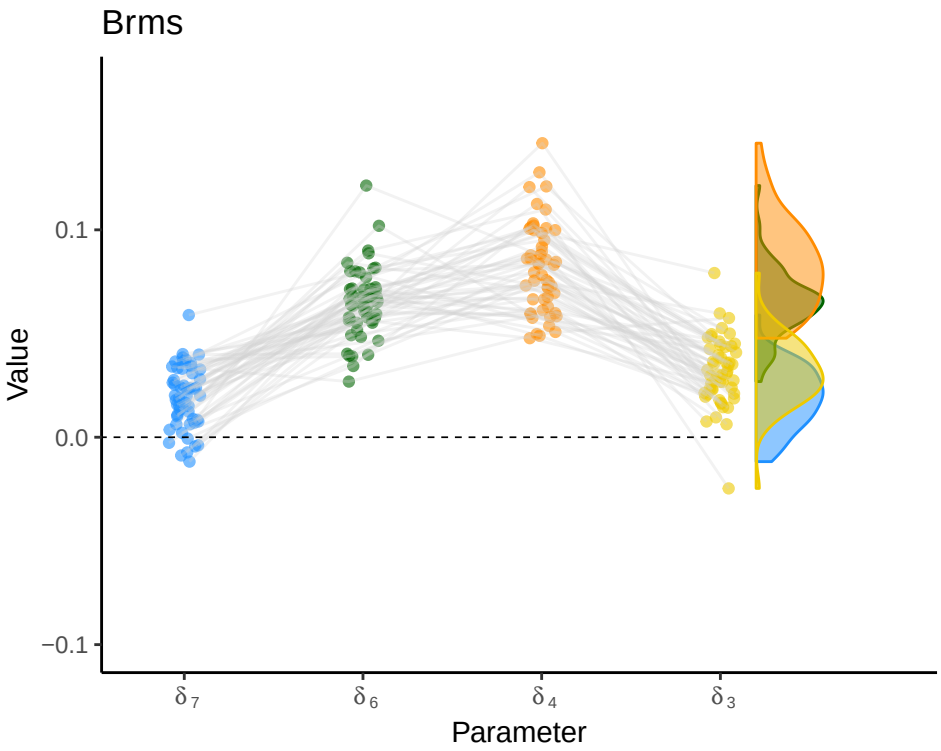


Figure 6: Model estimates for digit effect parameters. The points represent the mean parameter estimate for each individual. The violin plot on each right side shows the variance of the individual parameter estimates.

Model comparison

In case we have different models, we can compare them to check under which model the data is most likely. This can be done with the Bayes factor, as explained in the paper.

Savage-Dickey Density Ratio

In `brms` the Bayes factor, in the form of the Savage-Dickey density ratio, can be obtained by using the `hypothesis` function. To be able to use this function, it is important that the `sample_prior` option in the `brm` function when fitting the model is set to `TRUE`. | For every parameter we want to investigate, we compare the posterior against the null hypothesis. In the `hypothesis` function the null hypothesis is specified by the

`hypothesis` option. There, the parameter of interest is set to the null hypothesis, in this case, that the effect will be equal to zero.

```
# You have to set sample_prior = TRUE when fitting the model
# In case you haven't set sample_prior to TRUE, you could load the following model fit where sample_prior was set to TRUE
brmmodel8bf <- readRDS("myPath/brmslogmodelfit_BF_final.rds")

# Side effect
b1brms <- hypothesis(x = logmodel_fit_bf,           # Model fit
                     hypothesis = "a_side = 0")     # Null hypothesis
```

Then, we can retrieve the Savage-Dickey density ratio by extracting the `Evid.Ratio`.

```
b1brms$hypothesis$Evid.Ratio
```

Finally, we can plot the prior and posterior with the standard `plot` function. At the point of interest, in our case 0, we can check the height of the prior and posterior distribution. The ratio of these two represents the Savage-Dickey ratio. The function returns a `ggplot` object, that can then be manually adjusted.

```
plotsdratiob1 <- plot(b1brms,                       # Result hypothesis
                      plot = F,                     # Does not plot directly
                      theme = theme_get())[[1]]     # Theme used in plot

plotsdratiob1 + xlim(-0.07, 0.07)                   # Adjust x axis plot
```

We also created a function that provides this figure. The figure below shows the prior and posterior with the Savage-Dickey density ratio for the general side and digit effects.

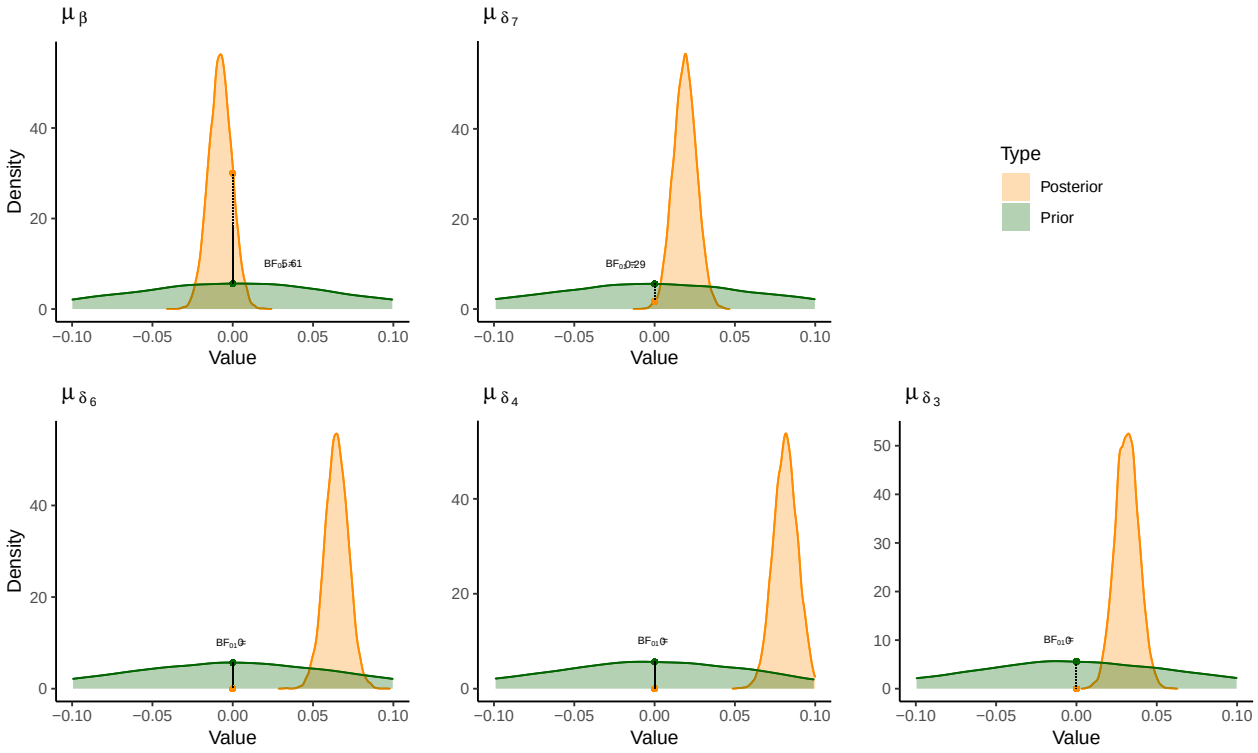


Figure 7: The prior and posterior distribution of the side and digit parameters estimated by the brms package. The dots represent the height of the distributions at zero. The ratio of these dots indicates the Savage-Dickey density ratio. The Bayes factor represents the evidence for the hypothesis that the effect equals zero. A Bayes factor of zero indicates that the evidence is in favor of the alternative hypothesis (the effect does not equal zero).

Bridge Sampling

To compare models that differ in more than one parameter, we can use bridge sampling. We could, for instance, compare the side model to the full model (for more details on these models see the paper). First, we need to estimate the models. We already estimated the full model. Next, we estimate the side model:

```
# Formula
formula_side <- rt2 ~ 1 + side +    # General effects
  (1 + side || sub1)

# Priors
priorsmodellog_side <- c(set_prior("normal(-0.5,1)", class = "Intercept"),
                  set_prior("inv_gamma(13.75136,4.102063)", class = "sd", coef="Intercept", group = "sub1"),
                  set_prior("normal(0,0.07071068)", class = "b", coef="side"),
                  set_prior("inv_gamma(13.75208,0.7489703)", class = "sd", coef="side", group = "sub1"),
                  set_prior("inv_gamma(13.75136, 4.102063)", class = "sigma"))

# Estimate side model
```

```
side_logmodel_fit <- brm(formula = formula_side,         # Model formula
                 data = myData_dataframe,              # Data frame with variables + dependent variable
                 family = lognormal(),        # Response distribution
                 prior = priorsmodellog_side,       # Priors
                 # sample_prior = TRUE, need to add this for model comparison
                 warmup = 1000,               # Iterations used for warmup
                 iter = 4000,                 # Total iterations per chain
                 chains = 4,                  # Number of chains
                 core = 4,                    # Cores for parallel estimation
                 control = list(adapt_delta = 0.97, max_treedepth = 15),  # Control sampler's behavior, this avoided problems of convergement
             save_pars = save_pars(all = TRUE))
```

Using `bayes_factor` function from the the `bridgesampling` package, we will estimate the Bayes factor of the full model against the side model.

```
bridgesampling::bayes_factor(logmodel_fit, side_logmodel_fit)[1]$bf  # BF of the full model against the side model
```

To check the stability of the estimated Bayes factor, we could repeat this process from model estimation to model comparison using bridge sampling a few times like we did in the paper.

Additional Resources

To learn more about fitting a Bayesian Hierarchical Model, we recommend the following resources:

- Stan forum
- Bayes factors with brms

References

Auguie, B. (2017). *gridExtra: Miscellaneous functions for "grid" graphics*. Retrieved from https://CRAN.R-project.org/package=gridExtra

Aust, F., & Barth, M. (2018). *papaja: Create APA manuscripts with R Markdown*. Retrieved from https://github.com/crsh/papaja

Bürkner, P.-C. (2017). brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, *80*(1), 1–28. doi:10.18637/jss.v080.i01

Bürkner, P.-C. (2018). Advanced Bayesian multilevel modeling with the R package brms. *The R Journal*, *10*(1), 395–411. doi:10.32614/RJ-2018-017

Carpenter, B., Gelman, A., Hoffman, M. D., Lee, D., Goodrich, B., Betancourt, M., ... Riddell, A. (2017). Stan: A probabilistic programming language. *Journal of Statistical Software*, *76*(1).

Gabry, J., Simpson, D., Vehtari, A., Betancourt, M., & Gelman, A. (2019). Visualization in Bayesian workflow. *Journal of the Royal Statistical Society*, *182*, 389–402. doi:10.1111/rssa.12378

Hope, R. M. (2013). *Rmisc: Rmisc: Ryan miscellaneous*. Retrieved from https://CRAN.R-project.org/package=Rmisc

Langen, J. van. (2020). Open-visualizations in r and python (Version v.1.0.4). Zenodo. doi:10.5281/zenodo.3715576

Müller, K., & Wickham, H. (2019). *Tibble: Simple data frames*. Retrieved from https://CRAN.R-project.org/package=tibble

Sarkar, D. (2008). *Lattice: Multivariate data visualization with r*. New York: Springer. Retrieved from http://lmdvr.r-forge.r-project.org

Stan Development Team. (2019). RStan: The R interface to Stan. Retrieved from http://mc-stan.org/

Statisticat, & LLC. (2020). *LaplacesDemon: Complete environment for Bayesian inference*. Bayesian-Inference.com. Retrieved from https://web.archive.org/web/20150206004624/http://www.bayesian-inference.com/software

Tiedemann, F. (2020). *Gghalves: Compose half-half plots using your favourite geoms*. Retrieved from https://github.com/erocoar/gghalves

Wickham, H. (2011). The split-apply-combine strategy for data analysis. *Journal of Statistical Software*, *40*(1), 1–29. Retrieved from http://www.jstatsoft.org/v40/i01/

Wickham, H. (2016). *ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from https://ggplot2.tidyverse.org

Wickham, H., François, R., Henry, L., & Müller, K. (2019). *Dplyr: A grammar of data manipulation*. Retrieved from https://CRAN.R-project.org/package=dplyr

Wickham, H., Hester, J., & Chang, W. (2019). *Devtools: Tools to make developing r packages easier*. Retrieved from https://CRAN.R-project.org/package=devtools

Wickham, H., Hester, J., & Francois, R. (2018). *Readr: Read rectangular text data*. Retrieved from https://CRAN.R-project.org/package=readr

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from https://yihui.name/knitr/

Xie, Y., Allaire, J. J., & Grolemund, G. (2018). *R markdown: The definitive guide*. Boca Raton, Florida: Chapman; Hall/CRC. Retrieved from https://bookdown.org/yihui/rmarkdown

Zhu, H. (2019). *kableExtra: Construct complex table with 'kable' and pipe syntax*. Retrieved from https://CRAN.R-project.org/package=kableExtra