Online Supplement E: Bayesian Hierarchical Modeling in rstan - Log Model

This online supplement provides a tutorial on log-normal Bayesian Hierarchical models (BHM) in `rstan` (Stan Development Team, 2019). In addition, it contains information about the digit classification task that could not be included in the paper. 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. This tutorial will cover the following:

1. Required packages
2. Input

   2.1 Stan file

   2.2 Data
3. Fit model
4. Output

   4.1 General Effects

   4.2 Individual Effects
5. Model comparison
6. Additional resources

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; Mersmann, Trautmann, Steuer, & Bornkamp (2018)]
- To fit the model: *rstan* [Version 2.19.2; Stan Development Team (2019)]
- 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)], *brms* [Version 2.13.0; Bürkner (2017); Bürkner (2018)], *gridExtra* [Version 2.3; Auguie (2017)], *ggbeeswarm* [Version 0.6.0; Clarke & Sherrill-Mix (2017)], *tibble* [Version 3.0.2; Müller & Wickham (2019)]

- For model comparison: *bridgesamping* [Version 1.0.0; Gronau, Singmann, & Wagenmakers (2020)]

```r
# 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("truncnorm")
library("ggbeeswarm")
```

Input

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

```r
model_fit <- stan(file = "./myLogModel.stan",  # Stan file with model
                  data = myData_list,       # List with observed data and constants
                  iter = 4000,              # Number (Nr.) of iterations per chain
                  chains = 4,               # Nr. of chains
                  warmup = 1000)            # Nr. iterations for warmup per chain
```

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

- Stan file
- A list with the data

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

Stan file

The stan file `myLogModel.stan` is used to fit the full model and can be found under R objects > rstan > linear model. It contains descriptions for every line of code. The stan file is divided in three sections (in our case):

1. Data
2. Parameters
3. Model

As the name says, the Data section specifies the variables we will provide with data. It contains the number of total observations, the number of groups (in our case the number of individuals), the response times, the condition for every response time (side of the digit and the digit indicator) and the prior settings. There are different types of variables. A vector indicates that the variable consists of several numbers and between `[]` the length of the vector is given. An integer indicates that the variable is one full number (i.e., no half numbers such as 0.3, 1.5, 6.33). Real means that the number can be a half number, so it allows numbers as 0.5, 1.3, 5.66, etc. With `<lower>` and `<upper>` the lower and upper bound of the parameter can be specified. For instance, for a variance it could be indicated that this variable cannot be below 0 with `<lower = 0>`.

In the Parameters section, all the parameters that need to be estimated are given. These are the general and individual effects. The specification of the parameters works in the same way as the variable specification in the Data section. Therefore, the general effects are reals, while the individual parameters are vectors with a length of the number of groups (in our case individuals, this means that there will be an estimate for every individual).

The last section, Model, contains the formula to obtain the estimates. We use the target specification as it saves the constants, necessary for model comparison later on. For example, `target += normal_lpdf(delta2 | mu4, sqrt(g4))` basically means that "delta2" is normally distributed with a mean of "mu4" and a standard deviation of "g4." Note that we want to estimate the variances, not the standard deviations. However, in the normal distribution function, the standard deviation is asked for. Therefore, we computed the square root of the variance parameters. Furthermore, it is important that you end every line with `;`, and end the stan file with an empty line. With `//` you can add a comment to the code.

Data

In the function, we put a list that contains all the information corresponding to the Data section of the stan file. This means that every parameter set in the Data section of the stan file, should have an element in the list we put in the function. In our case, this should result in 23 elements in the list.

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()  # Final dataset
```

Now, we will construct the list we provide the `stan` function. The first element of the list is N, corresponding to the number of groups in the data. In our case these are the number of individuals. The second element is the number of total observations in the data. Then, we specify the dependent variable/observations. In our case these are the response times (RTs). Next, we add an indicator to the list that specifies which observation belongs to which individual (or group).

```
# Data into list and fill the list with necessary information
# Number of groups (individuals)
myData_list <- list(N = length(unique(indat1$sub)))

# Number of total observations
myData_list$All <- nrow(indat1)

# Observations
myData_list$y <- indat1$rt/1000  # rt in seconds instead of milliseconds

# Need a variable with group number that are chronological (that doesn't skip numbers)
for (j in 1:nrow(indat1)){
  if (j == 1) {
    indat1$sub1[j] <- 1}
  else if (indat1$sub[j] == indat1$sub[j-1]) {
    myData_list$group_inds[j] <- indat1$sub1[j-1]}
  else myData_list$group_inds[j] <- indat1$sub1[j-1] + 1
}

# myData_list$group_inds <- indat1$sub1  # Add variable with indicator person to the list used for the analysis
```

The next step is to specify the indicators. 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.

```
### Add variable with information of the side of the digit to the list, 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_list$side[j] <- 1/2}
  else myData_list$side[j] <- -1/2
}

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

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

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

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

# datareal$dif1 <- indat1$dif1
# datareal$dif2 <- indat1$dif2
# datareal$dif3 <- indat1$dif3
# datareal$dif4 <- indat1$dif4
```

Finally, we specify the priors. The prior specification is discussed in the paper.

```
### Set the priors and add them to the list
a <- -0.5        # mean for mu, mu is the mean of theta
b <- 1          # variance for mu
c <- 3          # mean for sigma
d <- 0.3        # variance for sigma, sigma is the variance of observations
e <- 3          # mean for g, g is the variance of theta
f <- 0.3        # variance for g

#### Priors for beta
a2 <- 0        # mean for mean of beta
b2 <- 0.005     # variance for mean of beta, used to be 1
e2 <- 3        # mean for variance of beta, used to be .7, adjusted
f2 <- 0.01      # variance for variance of beta, used to be .2., adjusted

#### Prior for deltas, same for every delta
a3 <- 0        # mean for mean of delta
b3 <- 0.005     # sd for mean of delta, 0.5, 0.3
dd3 <- 3       # mean for variance of delta, adjusted
f3 <- 0.01      # variance for variance of delta, adjusted

myData_list$a <- a
myData_list$b <- b
myData_list$c <- c
myData_list$d <- d
myData_list$dd <- e
myData_list$f <- f
myData_list$a2 <- a2
myData_list$b2 <- b2
myData_list$dd2 <- e2
myData_list$f2 <- f2
myData_list$a3 <- a3
myData_list$b3 <- b3
myData_list$dd3 <- dd3
myData_list$f3 <- f3
```

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

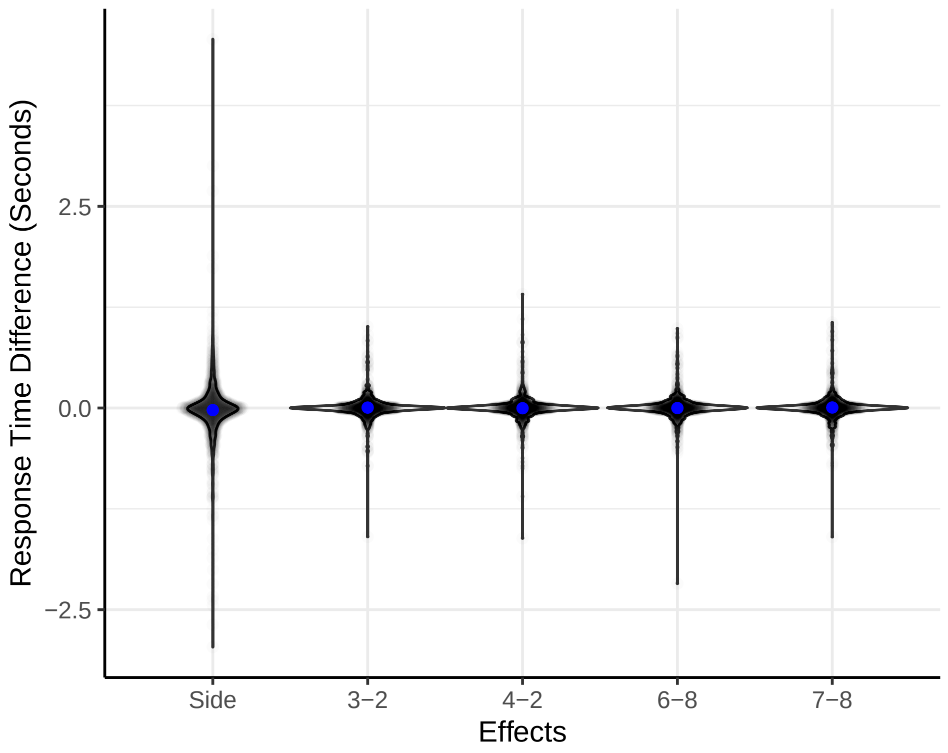


Figure 1: Predicted mean RT difference between conditions per trial. The blue dot represents the aggregated effect.

Fit the Model

Now we have the required input, we can fit the model.

```
library(rstan)
model_fit <- stan(file = "myPath/myLogModel.stan",
                  data = myData_list,
                  iter = 4000,
                  chains = 4,
                  warmup = 1000,
                  cores = 4)
```

We have already fitted the model and saved it in the R object `logmodel_fit`. 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 estimates.

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

Output

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 so you do not have to code them yourself. However, if you want the full control over the layout, it might be better to code them yourself. Therefore, we will show you how the program the plots presented in the paper.

The `summary` function provides an overview of the model fit. From this function, we can get a lot of information. As the output is structured in a specific way, it sometimes is hard to figure out how to isolate specific outcomes. This document nicely explains how to obtain specific output from the model fit.

Trace plot

The trace plot shows whether the posterior distribution of the parameter has converged. With the package's `plot` function it is possible to obtain the trace plots per parameter by setting the option `plotfun` to `trace`. By default, it will show the trace plots of the first ten parameters. With the `pars` function you can specify for which parameters you want to inspect the trace plots. The trace plot will automatically remove the warm-up (burn-in) period from the chain (which is desirable), however, if you would like to see that as well you have to set the option `inc_warmup` to `TRUE`. 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.

```
# Trace plot for parameter mu4
trdif2rstan <- plot(logmodel_fit,     # Model fit
                    plotfun = "trace",   # Specify that you want trace plots
                    pars = c("mu4"))     # Select parameter,
#it is also possible to select more than one at the same time, such as c("mu2", "mu3","mu4")

# In case you want to see the warmup period as well
trdif2rstanwarmup <- plot(logmodel_fit,    # Model fit
                          plotfun = "trace",  # Specify that you want trace plots
                          pars = c("mu4"),    # Select parameter
                          inc_warmup = TRUE)  # Show warmup period as well

# We created a function, so you can easily check all our parameters
traceplotfunc <- function(parameter, title){

# Trace plot for parameter mu4
trdif2rstan <- plot(logmodel_fit,     # Model fit
                    plotfun = "trace",   # Specify that you want trace plots
                    pars = c(parameter))     # Select parameter,
# it is also possible to select more than one at the same time, such as c("mu2", "mu3","mu4")

# Adjust the plot (without warmup) with ggplot
trdif2rstan2 <- trdif2rstan +   # Object returned from plot function
                ylab(title) +
                theme(axis.title.y=element_text(angle = 0,
                                          vjust = 0.5,
                                          hjust = -0.5,
                                          margin = margin(0, 0.7, 0, 0, "cm"), size = 14),
                      #legend.position = "none",
                      axis.line.x  = element_line(size = 0.6),
                      axis.line.y  = element_line(size = 0.6),
                      plot.margin = unit(c(0.3,0.1,0.3,0.05),"cm"),
                      plot.subtitle = element_text(size = 10.5),
                      axis.ticks = element_line(size = 0.6),
                      axis.text=element_text(size=8.5),
                       axis.ticks.length=unit(.1, "cm")) +
                labs(subtitle = "") +
                #ylim(c(0.01, 0.07)) +
                xlab("Iteration")

return(trdif2rstan2)
}
# Show trace plot for one digit parameter
tr2 <- traceplotfunc(parameter = "mu2", title = expression(~mu[~beta]))
tr2 <- tr2 + theme(legend.position = "none")
tr3 <- traceplotfunc(parameter = "mu3", title = expression(~mu[~delta[~7]]))
tr3 <- tr3 + theme(legend.position = "none")
tr4 <- traceplotfunc(parameter = "mu4", title = expression(~mu[~delta[~6]]))
tr4 <- tr4 + theme(legend.position = "none")
tr5 <- traceplotfunc(parameter = "mu5", title = expression(~mu[~delta[~4]]))
tr5 <- tr5 + theme(legend.position = "none")
tr6 <- traceplotfunc(parameter = "mu6", 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)
```

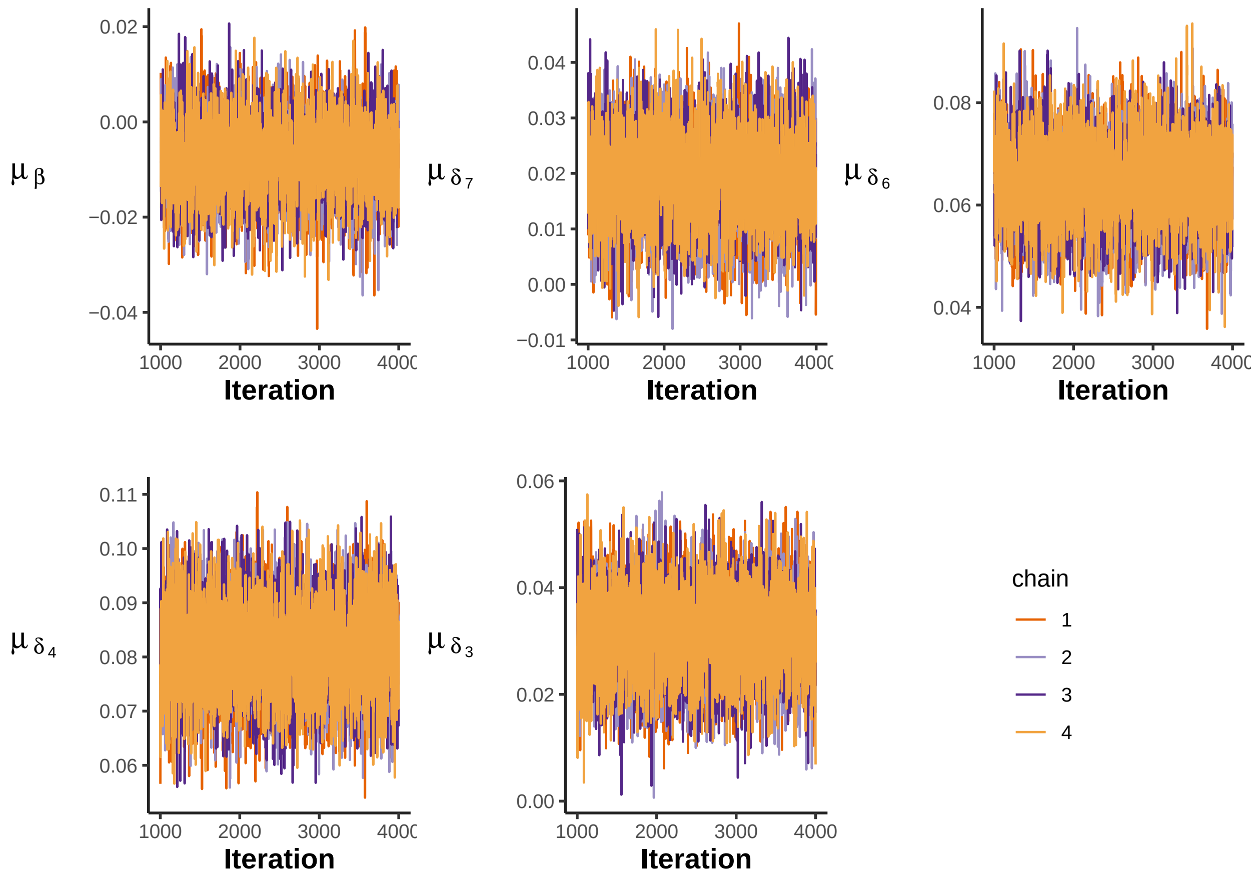
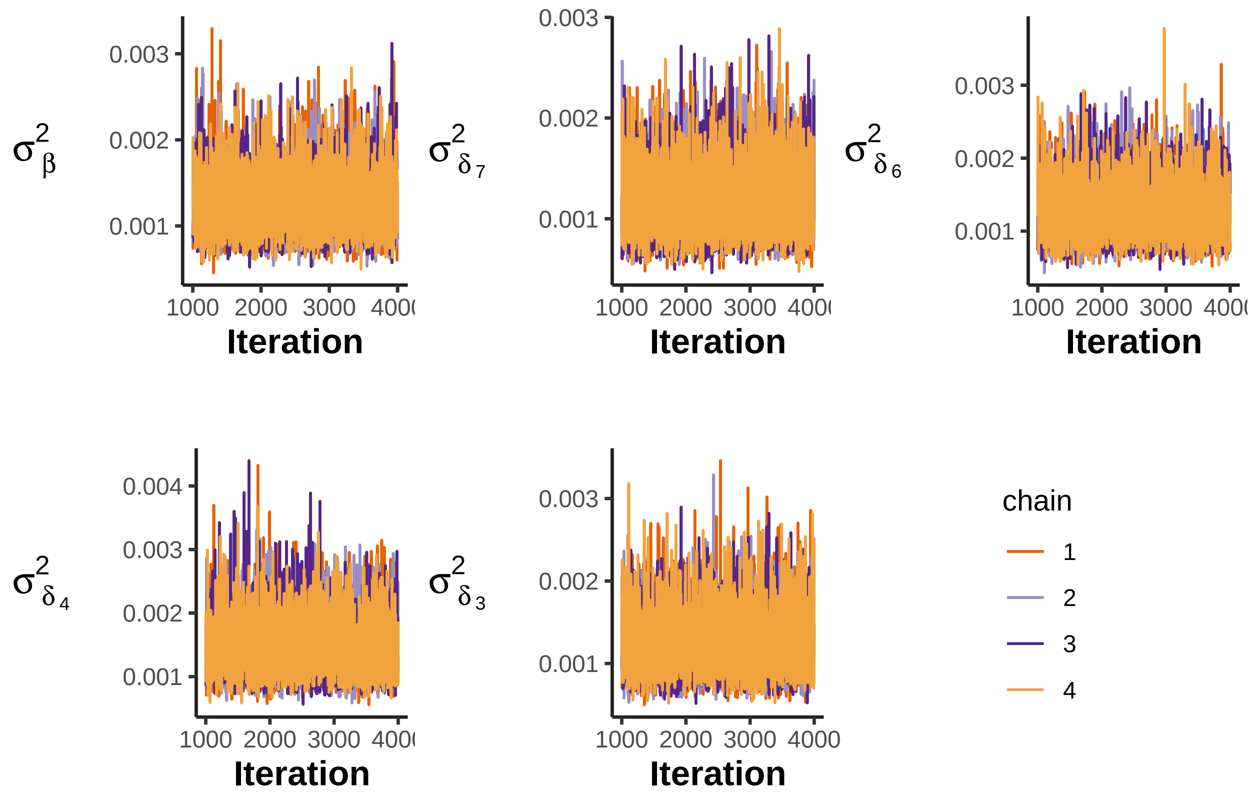


Figure 2: Trace plots of the general effect parameters.



Because there are so 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 standard *plot* function offers the possibility to plot the $\hat{R}$ (Rhat) and the ratio of the effective sample size to the total posterior sample size (this is different from what is plotted in the paper).

```r
# Plot frequency of rhat
plot(logmodel_fit, plotfun= "rhat")

# Plot frequency of the ratio
plot(logmodel_fit, plotfun = "ess")
```

However, you can also manually do this. To obtain the plots from the paper, the following code can be run. Instead of the ratio of the effective sample size to the total posterior sample size, we obtain the frequency of the numbers of effective samples (as shown in the paper).

```r
# Obtain rhat for every parameter from model fit
rhatrstan <- summary(logmodel_fit)$summary[,"Rhat"]
fixedrhatrstan <- rhatrstan[1:13]  # for general effects
```

```r
deltameanrrstan <- mean(fixedrhatrstan[3:6])  # calculate the mean for delta parameters

# Obtain number of effective samples for every parameter from model fit
neffrstan <- summary(logmodel_fit)$summary[,"n_eff"]
fixedneffrstan <- neffrstan[1:13]  # for general effects
fixedneffrstanmean <- mean(fixedneffrstan[3:6])  # calculate the mean for delta parameters

# Save as dataframe so we can plot it with ggplot
rstanrhatneff <- data.frame(rhat = rhatrstan, neff = neffrstan)

# Combine rhat into one data.frame
rhatrstan_trans <- t(rhatrstan)
rhatrstan_trans2 <- as.data.frame(rhatrstan_trans)

# remove lp and fixed effects
# rhat
rhatrstan2 <- rhatrstan[-c(1:13, 326)]

rhatrstan4 <- t(rhatrstan2)
rhatrstan5 <- as.data.frame(rhatrstan4)
rhatrstan6 <- unlist(rhatrstan5)
rstanrhatneff_v2 <- data.frame(package = rep("Rstan", 312),
                                 parameter = rep(c("Gamma", "Beta", "Delta", "Delta",
                                                   "Delta", "Delta", "Gamma", "Beta",
                                                   "Delta", "Delta", "Delta", "Delta"),
                                                 each = 52),
                                 rhat = rhatrstan6)

# Create violin plot
  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= fixedrhatrstan[1]), shape = 8, colour="darkgreen", size = 2) +
    geom_point(aes(x=2, y= fixedrhatrstan[2]), shape = 8, colour="darkgreen", size = 2) +
    geom_point(aes(x=3, y= deltameanrrstan), shape = 8, colour="darkgreen", size = 2) +
    xlab("Parameter") + ylab(expression(hat(R))) +
    scale_colour_manual(values = "darkgreen") +
    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
  # Combine neff into one data.frame
  neffrstan2 <- data.frame(neff = neffrstan)
  neffrstan3 <- neffrstan2$neff[-c(1:13, 326)]

  neffrstanbrms2 <- data.frame(package = rep("Rstan", 312),
                               parameter = rep(c("Gamma", "Beta", "Delta", "Delta",
                                                 "Delta", "Delta", "Gamma", "Beta",
                                                 "Delta", "Delta", "Delta", "Delta"), each = 52),
                               neff = neffrstan3)

  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 = fixedneffrstan[1]), shape = 8, colour="darkgreen", size = 2) +
    geom_point(aes(x = 2, y = fixedneffrstan[2]), shape = 8, colour="darkgreen", size = 2) +
    geom_point(aes(x = 3, y = fixedneffrstanmean), shape = 8, colour="darkgreen", size = 2) +
    xlab("Parameter") + ylab("Number of effective samples") +
    scale_colour_manual(values = "darkgreen") +
    theme_classic() +
    theme(legend.position = "none") +
    scale_x_discrete(labels= c(
      expression(~gamma),
      expression(~beta),
      expression(~delta))) +
    labs(subtitle = "D") +
    geom_hline(yintercept = 12000, linetype = "dashed", color = "darkgreen", size = .3)

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

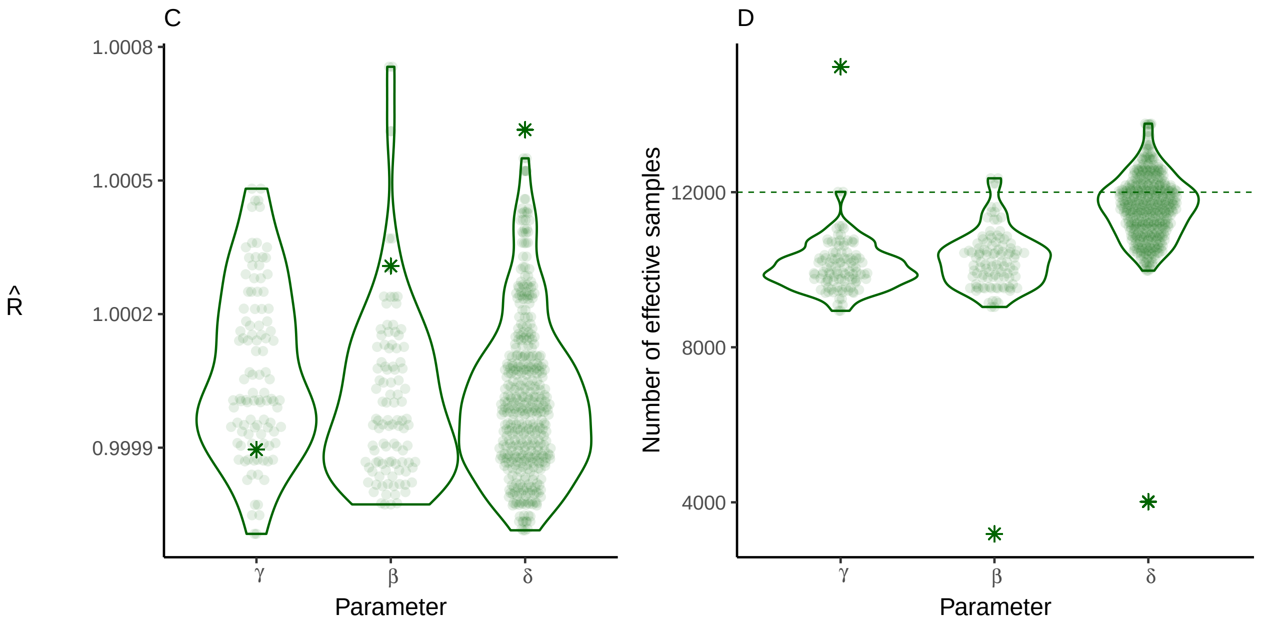


Figure 3: The frequency of the $\hat{R}$ statistic and the number of effective samples. A: The frequency of the $\hat{R}$. B: The frequency of the number of effective samples where the dashed lines represent the total number of iterations.

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 `plot` function, the posterior distribution per parameter can be displayed.

```
# Seperate plot per parameter of posterior density
plot(logmodel_fit,
     plotfun = "dens")
# without specifying pars, the posterior distributions of the first ten parameters are displayed
```

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

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

# With dimnames(postc) we could check the names of parameters
# The ones useful for us are "mu2", "mu3", "mu4", "mu5", and "mu6"

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

pairsc <- mcmc_areas(
  postc,                                      # Structured model fit
  pars = c("mu2", "mu3", "mu4", "mu5", "mu6"),  # Select parameters
  prob = 0.95,                                 # Plot 80% credible intervals
  prob_outer = 0.99,                          # 99% outer interval
  point_est = "mean"                          # Plot line in middle    distribution representing the mean
)
# Returns ggplot object

# Adjust ggplot object further
pairsc2 <- pairsc +                           # ggplot object
            ggtitle("Posterior Distribution") +                 # Add title
            scale_x_continuous(breaks = c(-0.025, 0, 0.025, 0.05, 0.075)) +   # specify the breaks and labels on x-axis, this is analysis dependen
            theme_apa() +                     # Select theme
            scale_y_discrete(labels= c(        # Customize labels to y axis
                                       expression(~mu[~beta]),
                                       expression(~mu[~delta[~7]]),
                                       expression(~mu[~delta[~6]]),
                                       expression(~mu[~delta[~4]]),
                                       expression(~mu[~delta[~3]])))

pairsc2
```

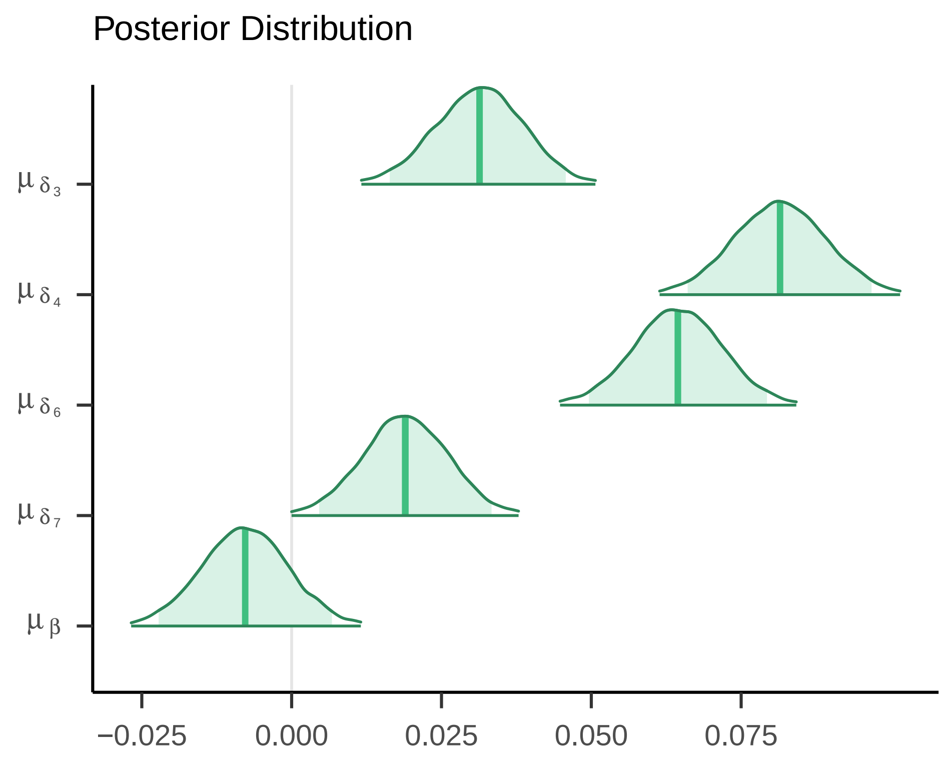


Figure 4: The posterior distributions for the general effects. The middle line within the distributions represents the mean posterior. The shaded area within the distributions represents 95% of the probability mass.

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
fitrstanhiertable <- summary(logmodel_fit,  # model fit
                             # Select parameters
                             pars = c("mu", "mu2", "mu3", "mu4", "mu5",
                                      "mu6"))$summary[,c("mean", "se_mean",
                                                         "2.5%", "97.5%",
                                                         "n_eff", "Rhat")]

# The results are transformed to a dataframe that allows us to plot the results nicely in a table
fitrstanhiertable2 <- as.data.frame(fitrstanhiertable)

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

# We manually set the column names of the table
colnames(fitrstanhiertable2) <- c("Mean", "SE", "Lower Bound", "Upper Bound", "$n_{eff}$","$\\hat{R}$")

# The column with row names is added to the table
```

```
fitrstanhiertable3 <- add_column(fitrstanhiertable2,
                                 Parameters = rownamesfitrstanhiertable,
                                 .before = "Mean")

# Using the apa_table function from the papaja package the results are presented in an apa table
apa_table(fitrstanhiertable3,   # 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 as
          estimated by Rstan.",
          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 as estimated by Rstan.

Parameters	Mean	SE	Lower Bound	Upper Bound	n_{eff}	$\hat{R}$
μ_γ	-0.558	0.000	-0.605	-0.512	15,235.072	1.000
μ_β	-0.008	0.000	-0.022	0.007	3,185.941	1.000
μ_{δ_7}	0.019	0.000	0.005	0.033	3,873.976	1.001
μ_{δ_6}	0.064	0.000	0.050	0.079	3,786.405	1.001
μ_{δ_4}	0.081	0.000	0.066	0.097	4,255.289	1.000
μ_{δ_3}	0.031	0.000	0.016	0.046	4,143.743	1.000

Individual Effects

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

Variance Estimates First, we can show a table with the posterior distribution estimates of the variance parameters. This indicates the variation of the general effect. We called our variances `g` (`g` until `g6`).

```
fitrstanhiertable2 <- summary(logmodel_fit,   # model fit
                              # Select variance parameters (we called them g)
                              pars = c("g", "g2", "g3", "g4", "g5",
                                       "g6"))$summary[,c("mean", "se_mean",
                                                         "2.5%", "97.5%",
                                                         "n_eff", "Rhat")]

# The results are transformed to a dataframe that allows us to plot the results nicely in a table
fitrstanhiertable22 <- as.data.frame(fitrstanhiertable2)

# We manually set the row names of the table using latex for mathematical symbols
rownamesfitrstanhiertable2 <-  c("$\\sigma_{\\gamma}^2$","$\\sigma_{\\beta}^2$", "$\\sigma_{\\delta_{7}}^2$", "$\\sigma_{\\delta_{6}}^2$", "$\\sig
```

```r
# The column with row names is added to the table
colnames(fitrstanhiertable22) <- c("Mean", "SE", "Lower Bound", "Upper Bound", "$n_{eff}$","$\\hat{R}$")

# The column with row names is added to the table
fitrstanhiertable32 <- add_column(fitrstanhiertable22,
                                  Parameters = rownamesfitrstanhiertable2,
                                  .before = "Mean")

# Using the apa_table function from the papaja package the results are presented in an apa table
apa_table(fitrstanhiertable32,  # Data frame with results
          row.names = FALSE,    # Default row names are deleted (we set them ourself)
          caption = "Posterior Variance, Standard Error (SE) of the Variance,
          Lower and Upper Bound of the 95\\% Credible Interval, the Number of
          Effective Samples, and the $\\hat{R}$ of the Variance Parameters           as estimated by Rstan.",
          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 Variance, Standard Error (SE) of the Variance, Lower and Upper Bound of the 95% Credible Interval, the Number of Effective Samples, and the $\hat{R}$ of the Variance Parameters as estimated by Rstan.

Parameters	Mean	SE	Lower Bound	Upper Bound	$n_{eff}$	$\hat{R}$
σ^2_γ	0.030	0.000	0.021	0.043	15,617.383	1.000
σ^2_β	0.001	0.000	0.001	0.002	7,815.681	1.000
$\sigma^2_{\delta_7}$	0.001	0.000	0.001	0.002	6,805.297	1.000
$\sigma^2_{\delta_6}$	0.001	0.000	0.001	0.002	7,325.299	1.000
$\sigma^2_{\delta_4}$	0.001	0.000	0.001	0.002	6,237.497	1.000
$\sigma^2_{\delta_3}$	0.001	0.000	0.001	0.002	6,841.198	1.001

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.

```r
# Gamma
## Create object with parameter names
gamnamesstan <- paste("gammai", "[",1:52,"]", sep="")

# Extract the parameter estimates using the names from before
gamstan <- as.data.frame(summary(logmodel_fit,
                                 pars = c( gamnamesstan))$summary)

# Extract the general effect estimate of the parameter
gamstanmu <- as.data.frame(summary(logmodel_fit,
                                    pars = c("mu"))$summary)

# Add the parameter names to the data.frame
gamstan$parameters <- c(gamnamesstan)

# Add parameter to dataframe. When the credible interval contains zero, the parameter is set pink, otherwise to blue
for (i in 1:nrow(gamstan)){  # 1 if CI contains zero, 0 otherwise
  if (gamstan$`2.5%`[i]*gamstan$`97.5%`[i] < 0) {
    gamstan$zero2[i] <- "pink"
  } else (gamstan$zero2[i] <- "blue")
}

# Order the estimates ascending
gamstan2 <- gamstan[order(gamstan$mean),]
gamstan2$order <- c(1:52)  # Add parameter to data frame with the order

# Plot the individual estimates for gamma (intercept)
gammarandomplot <- ggplot(gamstan2, aes(x = order, y = mean, ymin = `2.5%`, ymax = `97.5%`)) +
  geom_linerange(aes(color = "blue"), color = "#00BFC4") +
  geom_point() +
  geom_hline(yintercept = gamstanmu$mean, linetype = "dashed", alpha = .3) +
  labs(title = expression(~gamma[~i]), y = "Estimation") +
  theme(legend.position = "none",
        axis.title.x=element_blank(),
        axis.text.x=element_blank(),
        axis.ticks.x=element_blank())
```

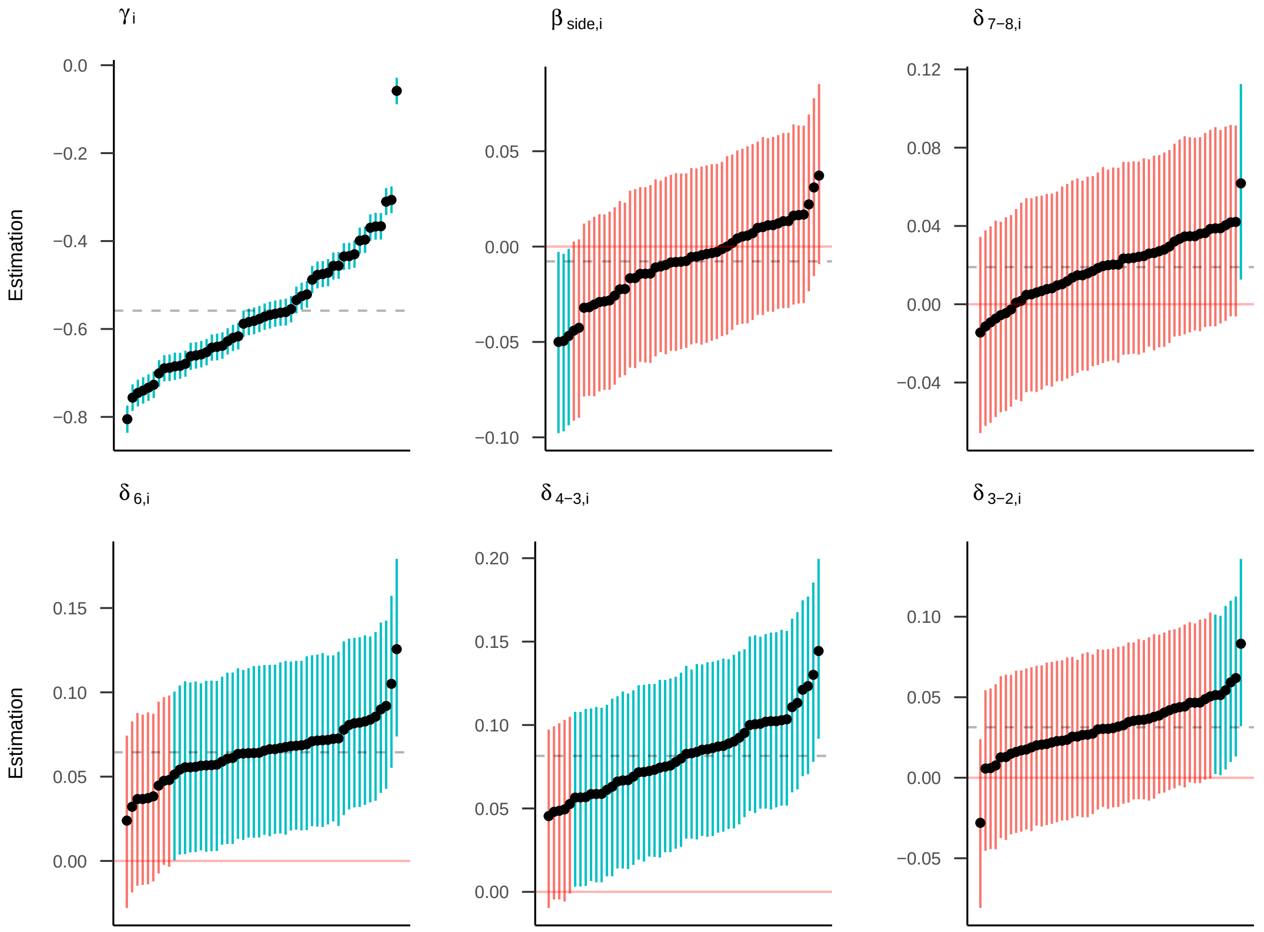


Figure 5: The posterior means with 95% credible interval in pink and red as estimated by *rstan* for every individual shown in increasing order. The dashed line represents the general posterior mean. A pink interval means that the interval contains zero, a blue interval represents an interval that does 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).

```
## Delta's
## Create dataframe
parameter_m7 <- rep(c("Delta 1", "Delta 2", "Delta 3", "Delta 4"), each = 52 )

### Obtain all individual estimates delta
hier_modeld <- logmodel_fit  # model fit

# Obtain individual estimates
delta1_summary <- summary(hier_modeld, pars = c("delta1"), probs = c(0.1, 0.9))$summary

delta2_summary <- summary(hier_modeld, pars = c("delta2"), probs = c(0.1, 0.9))$summary
```

```r
delta3_summary <- summary(hier_modeld, pars = c("delta3"), probs = c(0.1, 0.9))$summary

delta4_summary <- summary(hier_modeld, pars = c("delta4"), probs = c(0.1, 0.9))$summary

parest_m7 <- rbind(delta1_summary, delta2_summary, delta3_summary, delta4_summary)
datafr_m7 <- data.frame(parameter_m7, parest_m7)

## Plot
set.seed(321)
datafr_m7$x <- rep(c(1, 2, 3, 4), each = 52)
datafr_m7$xj <- jitter(datafr_m7$x, amount = 0.09)
datafr_m7$id <- rep(c(1:52), 4)

plotmodelest_m7 <- ggplot(data=datafr_m7, aes(y=mean)) +

  #Add geom_() objects
  geom_point(data = datafr_m7 %>% filter(x=="1"), aes(x=xj), color = 'dodgerblue', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7 %>% filter(x=="2"), aes(x=xj), color = 'darkgreen', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7 %>% filter(x=="3"), aes(x=xj), color = 'darkorange', size = 1.5,
             alpha = .6) +
  geom_point(data = datafr_m7 %>% 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_m7 %>% filter(x=="1"),aes(x = x, y = mean), position = position_nudge(x = 3.2),
    side = "r", fill = 'dodgerblue', alpha = .5, color = "dodgerblue", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7 %>% filter(x=="2"),aes(x = x, y = mean), position = position_nudge(x = 2.2),
    side = "r", fill = "darkgreen", alpha = .5, color = "darkgreen", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7 %>% filter(x=="3"),aes(x = x, y = mean), position = position_nudge(x = 1.2),
    side = "r", fill = "darkorange", alpha = .5, color = "darkorange", trim = TRUE) +

  geom_half_violin(
    data = datafr_m7 %>% filter(x=="4"),aes(x = x, y = mean), 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("") + ylab("Value") +
  ggtitle('') +
  theme_classic() +
  coord_cartesian(xlim = c(0.75, 5), ylim = c(-0.05, 0.12))

plotmodelest_m7
```

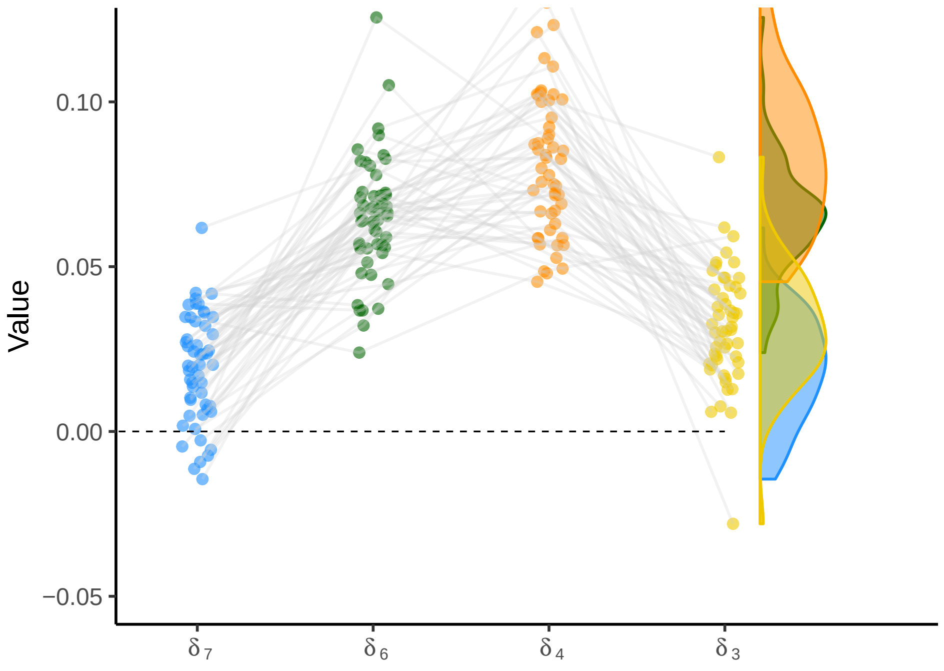


Figure 6: Model estimates for digit effect parameters. The points represent the mean parameter estimate for each individual. The violin plot on each right side shows the variance of the individual parameter estimates.

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. In `rstan` the Bayes factor can be obtained by using the `bridgesampling` package. To be able to use this package, it is important that the `.stan` file is written in a way that it saves the constants (i.e., `target += normal_lpdf(mu4 | a3, sqrt(b3));`).

First, we fit all the models (except for the full model that we already fitted.

```
# Fit the other models
## Null model
BFnulllogmodel <- stan(file = "myPath/logstanmodelh0.stan",
                    data = myData_list,
                    iter = 4000,
                    chains = 4,
                    warmup = 1000,
                    cores = 4)

## Side model
BFsidelogmodel <- stan(file = "myPath/logstanmodelhside.stan",
                       data = myData_list,
                       iter = 4000,
                       chains = 4,
                       warmup = 1000,
                       cores = 4)
```

```r
## Digit model
BFdigitlogmodel <- stan(file = "myPath/logstanmodelhdigit.stan",
                        data = myData_list,
                        iter = 4000,
                        chains = 4,
                        warmup = 1000,
                        cores = 4)
```

Then, we use the `bridge_sampler` function to obtain the marginal likelihoods. In case you loaded the full model from the provided R object, you will need an extra step as described in this issue.

```r
# Bridge sampling
# library(bridgesampling)
bridge_Hnulllog <- bridge_sampler(BFnulllogmodel)
bridge_Hsidelog <- bridge_sampler(BFsidelogmodel)
bridge_Hdigitlog <- bridge_sampler(BFdigitlogmodel)
bridge_Hfulllog <- bridge_sampler(logmodel_fit)  # in case you just fitted the model yourself

## In case you loaded the full model from the R object, you will need some preparation (https://github.com/quentingronau/bridgesampling/issues/7)

### Fit model again but do not run any chains
new_logmod <- stan(file = "myPath/myLogModel.stan",
                   data = myData_list,
                   chains = 0)

### Now do the bridgesampling
bridge_Hfulllog <- bridge_sampler(logmodel_fit, new_logmod)
```

Finally, we can obtain the Bayes factor. You obtain the evidence in favor of the first model that you specify in the `bf` function.

```r
# Obtain BF (in this way evidence in favor of the null hypothesis)
bridgesampling::bf(bridge_Hnulllog, bridge_Hfulllog)   # Null against full

# Obtain BF (in this way evidence in favor of the full hypothesis)
bridgesampling::bf(bridge_Hfulllog, bridge_Hnulllog)   # Full against null
bridgesampling::bf(bridge_Hfulllog, bridge_Hsidelog)   # Full against side
bridgesampling::bf(bridge_Hfulllog, bridge_Hdigitlog)  # Full against digit
```

Additional Resources

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

- Stan forum
- Accessing the contents of a stanfit object

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

Clarke, E., & Sherrill-Mix, S. (2017). *Ggbeeswarm: Categorical scatter (violin point) plots*. Retrieved from https://CRAN.R-project.org/package=ggbeeswarm

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

Gronau, Q. F., Singmann, H., & Wagenmakers, E.-J. (2020). bridgesampling: An R package for estimating normalizing constants. *Journal of Statistical Software*, *92*(10), 1–29. doi:10.18637/jss.v092.i10

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

Mersmann, O., Trautmann, H., Steuer, D., & Bornkamp, B. (2018). *Truncnorm: Truncated normal distribution*. Retrieved from https://CRAN.R-project.org/package=truncnorm

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.bayesi an-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