S1. Supplemental Materials Part 1

Reproducibility in small-N treatment research: a tutorial through the lens of aphasiology

Introduction

This document details the code needed to reproduce the analysis in section 1 of the manuscript. For batch calculations of effect sizes across participants see part 2.

Setup

Load packages and functions

```
# Uncomment and run this line to install packages if needed
# Some packages are not used to generate the .pdf, but are used for table generation
# install.packages(c("here", "SingleCaseES", "lme4",
# "emmeans", "brms", "tidybayes", "ggdist", "tidyverse", "flextable", "officer"))

# Instructions for installing RStan are here:
# https://github.com/stan-dev/rstan/wiki/RStan-Getting-Started

library(here)           # for locating files
library(SingleCaseES)   # calculating SMD, Tau-U
library(lme4)           # frequentist mixed-effects models
library(emmeans)        # estimating effect sizes from lme4
library(brms)           # bayesian mixed-effects models
library(tidybayes)      # estimating effect sizes from brms
library(ggdist)         # Visualizing posterior distributions
library(tidyverse)      # data wrangling and plotting
library(flextable)      # creating tables
library(officer)        # saving to word


# set a seed for reproducibility
set.seed(42)
```

Read in data

Note that the current setup uses RStudio R projects (https://support.rstudio.com/hc/en-us/articles/200526207-Using-RStudio-Projects). One of the features of R projects is that the working directory is automatically set to the project root (the folder with the .Rproj). A discussion of R projects can be found at https://www.tidyverse.org/blog/2017/12/workflow-vs-script/. In this case `here("study-data")` refers to the /study-data folder inside the project.

```
# create a list of files
files <- list.files(
                here("data"), # look in the study-data folder
                full.names = TRUE,  # use the full paths of the files
                pattern = ".csv",   # only read in .csv files
                recursive = TRUE)   # include files within subfolders

# read in the files and combine them together
# map_df takes a function, in this case read_csv().
# show_col_types suppresses output since we're reading in many files
df <- files %>%
  map_dfr(read_csv, show_col_types = FALSE)
```

Preview the data

```
head(df)
```

```
## # A tibble: 6 x 11
##   participant condi~1 phoneme itemT~2 phase session item  trials spt2017 respo~3
##   <chr>       <chr>   <chr>   <chr>   <chr>   <dbl> <chr>  <dbl> <chr>     <dbl>
## 1 P1          blocked pr      tx      base~       1 pr-1      10 pre           0
## 2 P1          blocked pr      tx      base~       1 pr-12     10 pre           0
## 3 P1          blocked pr      tx      base~       1 pr-4      10 pre           0
## 4 P1          blocked pr      tx      base~       1 pr-15     10 pre           0
## 5 P1          blocked pr      tx      base~       1 pr-5      10 pre           0
## 6 P1          blocked pr      tx      base~       1 pr-7      10 pre           0
## # ... with 1 more variable: n_baselines <dbl>, and abbreviated variable names
## #   1: condition, 2: itemType, 3: response
## # i Use `colnames()` to see all variable names
```

Table 1: Data variables and their description

Variable	Description
participant	de-identified participant ID
condition	probe schedule (blocked or random)
phoneme	target_phoneme
itemType	item condition (treatment or generalization)
phase	treatment phase
session	session number from Wambaugh 2017
item	item identifier
trials	number of items in the list (per phoneme)
spt2017	phase used to calcualte effect sizes in Wambaugh et al., 2017
response	accuracy of participant response
n_baselines	Number of baseline sessions

Case example: Participant 10

Filter data for Participant 10

Starting from the entire dataset, filter for participant 10, treated items, and the blocked condition. Then to calculate session-level data (the number of correct responses per session), group by session, and use the summarize function to calculate the number of correct responses per session. The `group_by` function also includes phase and spt2017 because we want to keep these variables in the summary data frame, but their addition doesn't affect grouping. The .groups argument removes the grouping after summarize.

```
P10 <- df %>%
  # filter for participant 10, treated condition, blocked condition
  filter(participant == "P10",
         itemType == "tx",
         condition == "blocked") %>%
  # calculate the sum for each level of session, phase and spt2017
  group_by(session, phase, spt2017) %>%
  summarize(sum_correct = sum(response), .groups = "drop")
```

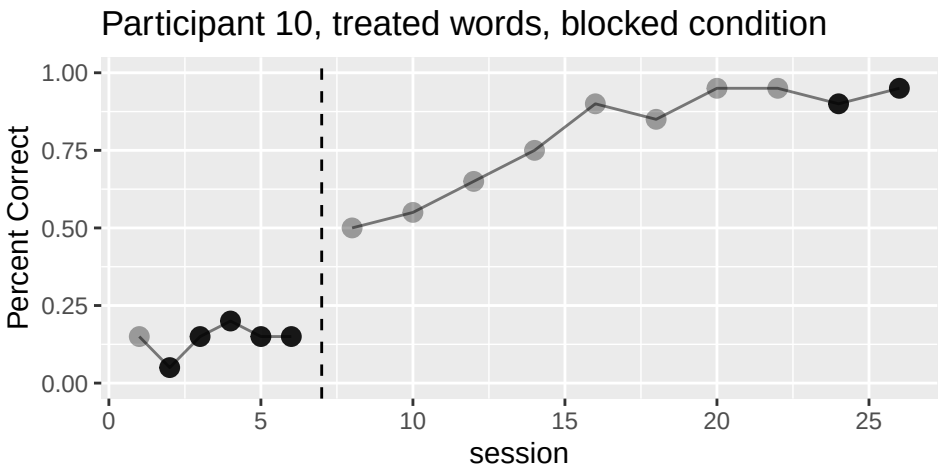
Plot performance over time

Plotting data from participant 10. First, we select only the baseline and treatment phases (ignoring the washout and maintenance phases for the purpose of this paper). The we create a dummy variable reflecting whether or not the session was included in the SMD/PMG calculations. Finally, we use the {ggplot2}

package to plot the data. A recent primer on {ggplot2} for researchers unfamiliar with R can be found here: https://doi.org/10.1177/25152459221074654

```
p1 = P10 %>%
  # filter for baseline and treatment phases
  filter(phase == "baseline" | phase == "treatment") %>%
  # create a new variable called Measure that has a value of
  # include if spt2017 is not an NA value and exclude if it is.
  # the levels argument indicates that exclude should be
  # the first level of the factor.
  mutate(Measure = factor(
    ifelse(!is.na(spt2017), "include", "exclude"),
           levels = c("exclude", "include"))) %>%
  # create a plot with session on the x axis, percent
  # correct on the y axis, and group by phase
  ggplot(aes(x = session, y = sum_correct/20, group = phase)) +
  # add points to the graph
  geom_point(aes(alpha = Measure), size = 3) +
  # add a line to the graph
  geom_line(alpha = 0.5) +
  # add a vertical line where x = 7
  geom_vline(aes(xintercept = 7), linetype = "dashed") +
  scale_x_continuous(breaks = seq(0,30,5)) +
  ylim(0, 1) +
  scale_alpha_discrete(range = c(0.35, 0.9)) +
  labs(title = "Participant 10, treated words, blocked condition",
       caption = "Dark circles represent data points used to calculate
       the within-case standardized mean difference in Wambaugh et al.,
       (2017)",
       y="Percent Correct") +
  guides(alpha = "none")

p1
```



Within-case standardized mean difference

There are any number of ways to calculate the within case standardized mean difference using R code. In this example, we have used the `SMD()` function from the established package {SingleCaseES} by James Pustejovsky because it includes additional functions that may be of interest to researchers in aphasiology.

Additionally, we do not show all information returned by the function, which also includes a 95% confidence interval, as it is not clear that this confidence interval applies to the the d_{BR} modification of the original within-case standardized mean difference.

$$d_{BR} = \frac{x_B - x_A}{S_A}$$

```
A = P10 %>% filter(spt2017 == "pre") %>% pull(sum_correct)
B = P10 %>% filter(spt2017 == "post") %>% pull(sum_correct)

SMD(A_data = A, B_data = B)
```

```
##    ES      Est       SE CI_lower CI_upper baseline_SD
## 1 SMD 11.46566 3.283023 5.031052 17.90027    1.095445
```

Proportion of potential maximal gain

There is no R package that includes a function to calculate PMG to our knowledge. However, creating such a function is relatively straightforward. A function that calculates PMG similar to the SMD() function from the {SingleCaseES} package might take the following form, with an additional argument for the number of items treated (nitems). The function calculates the mean of the A phase and B phase, and then calculates and returns the PMG value from the same data as d_{BR} above.

$$PMG = \frac{x_B - x_A}{n_{items} - x_A}$$

```
# the function is named PMG and takes 3 arguments:
# vectors of the a_data and b_data, and
# a single number indicating how many items were treated
PMG <- function(a_data, b_data, nitems){
  mean_a <- mean(a_data) # calculate mean of a_data
  mean_b <- mean(b_data) # calculate mean of b_data
  pmg <- (mean_b-mean_a)/(nitems-mean_a) # calculate PMG
  return(pmg) # return the PMG value.
```

```
}

PMG(a_data = A, b_data = B, nitems = 20)
```

```
## [1] 0.9127907
```

Tau-U

The Tau-U family of effect sizes (and several other non-overlap measures) can be calculated using the {SingleCaseES} package. In this case, we use all data summarized in the P10 data frame (and not just the data used to calculate d_{BR}).

First, we estimate the trend line during the baseline phase, which can be generated by creating a simple linear model using the `lm()` function. The model includes the number of correct responses as the dependent variable and the session number as the independent variable. The session coefficient reflects the slope during the baseline phase. The `coef()` function simple extracts the model coefficients.

```
P10 %>%
    filter(phase == "baseline") %>%
    lm(data = ., sum_correct~session) %>%
    coef()
```

```
## (Intercept)     session
##    2.133333    0.200000
```

The `Tau()` and `Tau_U()` functions take the same data structure as the `SMD()` and `PMG()` functions above.

Using the conservative benchmark of 0.33 recommended by Lee and Cherney (2018), we would calculate $Tau_{Avs.B}$ as the slope of the baseline phase is only 0.2 (from the result above). To calculate $Tau_{Avs.B}$, we can use the `Tau()` function.

```
A = P10 %>% filter(phase == "baseline") %>% pull(sum_correct)
B = P10 %>% filter(phase == "treatment") %>% pull(sum_correct)

Tau(A_data = A, B_data = B)
```

```
##    ES Est         SE CI_lower CI_upper
## 1 Tau   1 0.02710291        1        1
```

However, if we had elected to correct for baseline trends and use $Tau_{Avs.B-trendA}$, we can use the similar `Tau_U()` function.

```
Tau_U(A_data = A, B_data = B)
```

```
##      ES  Est
## 1 Tau-U 0.95
```

Mixed-effects model-based effect sizes

The mixed-effects model example for participant 10 uses item-level data, so we need to create a new dataframe for this model. As discussed in the manuscript, the model formula is based on a structure from Huitema & McKean (2000).

$$Y_t = \beta_0 + \beta_1 T_t + \beta_2 D_t + \beta_3 [T_t - (n_1 + 1)] D_t + \epsilon_t$$

The DV is predicted by the model intercept, baseline slope, level change, and slope change parameters.

After selecting data from participant 10, the coefficients are created by:

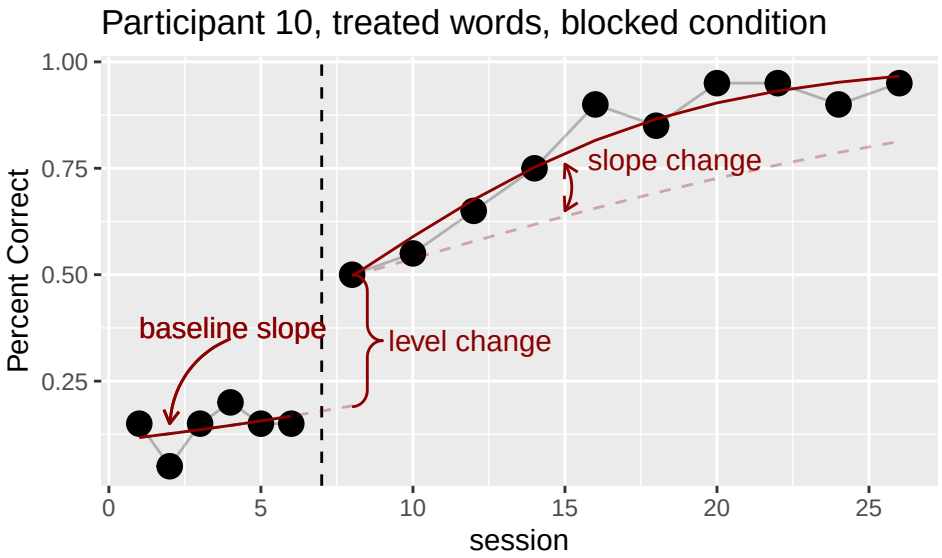
- setting `baseline_slope` equal to the session variable
- `level_change` is a dummy variable, 0 during baseline and 1 during treatment
- `slope_change` is created by subtracting the number of baselines plus 2 from the baseline slope value, and then multiplying the result with the `level_change` variable. Typically, if probing every session, the formula calls for subtracting the number of baselines plus 1. However, because Wambaugh et al., (2017) used intermittent probing schedules, and probed every other treatment session starting at the second, we need to add 2 to the number of baselines to ensure that the slope change variable starts at 0 on the first recorded treatment probe.

```
P10 <- df %>%
  filter(participant == "P10",
         condition == "blocked",
         itemType == "tx",
         phase == "baseline" | phase == "treatment") %>%
  mutate(baseline_slope = session,
         level_change = ifelse(phase == "baseline", 0, 1),
         slope_change = (baseline_slope - (6+2))*level_change,
         level_change = as.factor(level_change))
```

The resulting matrix looks as follows:

phase	baseline_slope	level_change	slope_change
baseline	1	0	0
baseline	2	0	0
baseline	3	0	0
baseline	4	0	0
baseline	5	0	0
baseline	6	0	0
treatment	8	1	0
treatment	10	1	2
treatment	12	1	4
treatment	14	1	6
treatment	16	1	8
treatment	18	1	10
treatment	20	1	12
treatment	22	1	14
treatment	24	1	16
treatment	26	1	18

Manuscript Figure 2. visualizes each parameter in this model structure. The code can be found in the .Rmd file, and is omitted from the pdf due to its length.



The model formula can be expressed as follows:

$$\text{response}_i \sim \text{Binomial}(n = 1, \text{prob}_{\text{response}=1} = \widehat{P})$$

$$\log\left[\frac{\hat{P}}{1-\hat{P}}\right] = \alpha_{j[i]} + \beta_{1j[i]}T_t + \beta_{2j[i]}D_t + \beta_{3j[i]}[T_t - (n_1 + 1)]D_t$$

$$\begin{pmatrix} \alpha_j \\ \beta_{1j} \\ \beta_{2j} \\ \beta_{3j} \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_{\alpha_j} \\ \mu_{\beta_{1j}} \\ \mu_{\beta_{2j}} \\ \mu_{\beta_{3j}} \end{pmatrix}, \begin{pmatrix} \sigma^2_{\alpha_j} & \rho_{\alpha_j\beta_{1j}} & \rho_{\alpha_j\beta_{2j}} & \rho_{\alpha_j\beta_{3j}} \\ \rho_{\beta_{1j}\alpha_j} & \sigma^2_{\beta_{1j}} & \rho_{\beta_{1j}\beta_{2j}} & \rho_{\beta_{1j}\beta_{3j}} \\ \rho_{\beta_{2j}\alpha_j} & \rho_{\beta_{2j}\beta_{1j}} & \sigma^2_{\beta_{2j}} & \rho_{\beta_{2j}\beta_{3j}} \\ \rho_{\beta_{3j}\alpha_j} & \rho_{\beta_{3j}\beta_{1j}} & \rho_{\beta_{3j}\beta_{2j}} & \sigma^2_{\beta_{3j}} \end{pmatrix}\right), \text{ for item j} = 1, \ldots, \text{J}$$

The first line of the equation indicates that the dependent variable takes a binomial distribution. The second line represents the extension of the Huitema & McKean (2000) model to the hierarchical case, where each fixed effect is estimated for each item, j[i]. The third line of the equation represents the random effect structure, indicating that the effect of each fixed effect (baseline slope, level change, and slope change) are allowed to vary for each item.

The following shows how we arrived at the final model for P10

1. First, we fit the maximal random effects structure. However, the model did not converge.

```
mod1 <-
    glmer(
      # fixed effects
      response ~ baseline_slope + level_change + slope_change +
      # random effects
       (1 + baseline_slope + level_change + slope_change | item),
          data = P10,
          family = binomial)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## unable to evaluate scaled gradient
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge: degenerate Hessian with 1 negative eigenvalues
```

2. Second, we tried specifying a different optimizer, following recommendations that can be found at https://bbolker.github.io/mixedmodels-misc/glmmFAQ.html#convergence-warnings.

```
mod1 <-
    glmer(response ~ baseline_slope + level_change + slope_change +
            (1 + baseline_slope + level_change + slope_change | item),
          data = P10,
```

```
          family = binomial,
          # for model convergence
          control = glmerControl(optimizer="bobyqa"))
```

We can now examine the model summary:

```
summary(mod1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: response ~ baseline_slope + level_change + slope_change + (1 +
##     baseline_slope + level_change + slope_change | item)
##    Data: P10
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##    249.5    302.3   -110.8    221.5      306
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.2995 -0.2462  0.0202  0.1563  3.6534
##
## Random effects:
##  Groups Name           Variance Std.Dev. Corr
##  item   (Intercept)    4.5354   2.1296
##         baseline_slope 0.3359   0.5795   -0.75
##         level_change1  5.6515   2.3773    0.34 -0.01
##         slope_change   0.8646   0.9298    0.66 -0.97 -0.14
## Number of obs: 320, groups:  item, 20
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -2.70900    1.33242  -2.033    0.042 *
## baseline_slope  0.01742    0.33249   0.052    0.958
## level_change1   2.50788    1.69433   1.480    0.139
## slope_change    0.39217    0.39651   0.989    0.323
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) bsln_s lvl_c1
## baselin_slp -0.861
## level_chng1  0.535 -0.723
## slope_chang  0.761 -0.916  0.499
```

We note that in this case, further reducing the random effects structure often returns a significant result for the level_change parameter, demonstrating how our choice of random effect structure can influence the statistical significance of model parameters. For this reason, we stress that researchers are forthcoming

of the model modifications made from the initial maximal random effect structure and also report what modifications result in parameters changing in significance.

Calculating an overall effect size for this participant requires contrasting performance either at the end of treatment with and without the level change and slope change parameters, or contrasting performance at the end of treatment with performance at the end of baseline. The former option assumes that any baseline trend would have continued throughout the treatment phase in the absence of treatment, is typically more conservative.

While there is a small, empirical baseline slope in this data, it may be reasonable to consider that this slope is largely driven by lower performance on the second probe session, and that performance in baseline sessions 3-6 are stable, and therefore estimate the difference in performance from the end of baseline to the end of treatment. Criteria for such decisions should ideally be made a-priori if possible, and reported in publications regardless.

1. First, we generate the marginal means for each combination of baseline slope, level change, and slope change.

```
# setup marginal means
#
marginal_means = emmeans(
                        # object refers to the glmer model object
                       object = mod1,
                       # specs refers to the model coefficients we're interested in
                       specs = c("baseline_slope", "level_change", "slope_change"),
                       # at indicates the values of the coefficients we're interested in
                       at = list(
                         baseline_slope = c(7, 26),
                         level_change = c("0", "1"),
                         slope_change = c(0, 19)
                       )
                     )

marginal_means
```

```
##  baseline_slope level_change slope_change  emmean   SE  df asymp.LCL asymp.UCL
##               7 0                        0 -2.5870 1.36 Inf    -5.252    0.0782
##              26 0                        0 -2.2560 7.53 Inf   -17.010   12.4976
##               7 1                        0 -0.0792 1.20 Inf    -2.427    2.2684
##              26 1                        0  0.2519 6.39 Inf   -12.263   12.7672
##               7 0                       19  4.8641 6.46 Inf    -7.801   17.5290
##              26 0                       19  5.1952 3.06 Inf    -0.803   11.1928
##               7 1                       19  7.3720 7.35 Inf    -7.041   21.7847
##              26 1                       19  7.7030 2.49 Inf     2.819   12.5873
##
```

```
## Results are given on the logit (not the response) scale.
## Confidence level used: 0.95
```

2. This returns a table of all possible comparisons, and we are only interested in contrasting the first row (beginning of treatment) with the last row (end of treatment). After selecting these two rows, we can then contrast their estimates.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
# There are 8 numbers corresponding to the 8 possible comparisons above
A = c(1, 0, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)

# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate  SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     10.3 3.1 Inf      4.22      16.4   3.322  0.0009
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

We could also make the more conservative assumption that any baseline trend continues by choosing the second row where baseline slope is set to the last treatment session.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
A = c(0, 1, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)

# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate   SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     9.96 8.51 Inf     -6.71      26.6   1.171  0.2417
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

Notice that there is much greater uncertainty in this contrast, evident by the increase in standard error, and as a result the p-value is no longer significant.

Group-level model

We can extend this individual model to all participants, still focusing on treated items in the blocked condition. First, we create a new data frame that includes all participants.

```
df_glmm <- df %>%
  # select the correct phase, condition, and itemType
  filter(phase == "baseline" | phase == "treatment",
         condition == "blocked",
         itemType == "tx") %>%
  # create the Huitema model parameters
  mutate(baseline_slope = session,
         level_change = ifelse(phase == "baseline", 0, 1),
         slope_change = (baseline_slope - (n_baselines+2))*level_change,
         level_change = as.factor(level_change))
```

Then we can start again with a relatively maximal random effect structures, noting that we could also include random slopes for items. However, it is unlikely that such a model structure could be supported by the data. In this case we have chosen to include the most theoretically important random effects (Matsucheck, 2018) that we expect to be supported by the data. A formula for the model is:

$$\text{response}_i \sim \text{Binomial}(n = 1, \text{prob}_{\text{response}=1} = \widehat{P})$$

$$\log\left[\frac{\hat{P}}{1-\hat{P}}\right] = \alpha_{j[i],k[i]} + \beta_{1k[i]}(\text{baseline\_slope}) + \beta_{2k[i]}(\text{level\_change}_1) + \beta_{3k[i]}(\text{slope\_change})$$

$$\alpha_j \sim N\left(\mu_{\alpha_j}, \sigma^2_{\alpha_j}\right), \text{ for item j} = 1, \ldots, \text{J}$$

$$\begin{pmatrix} \alpha_k \\ \beta_{1k} \\ \beta_{2k} \\ \beta_{3k} \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_{\alpha_k} \\ \mu_{\beta_{1k}} \\ \mu_{\beta_{2k}} \\ \mu_{\beta_{3k}} \end{pmatrix}, \begin{pmatrix} \sigma^2_{\alpha_k} & \rho_{\alpha_k\beta_{1k}} & \rho_{\alpha_k\beta_{2k}} & \rho_{\alpha_k\beta_{3k}} \\ \rho_{\beta_{1k}\alpha_k} & \sigma^2_{\beta_{1k}} & \rho_{\beta_{1k}\beta_{2k}} & \rho_{\beta_{1k}\beta_{3k}} \\ \rho_{\beta_{2k}\alpha_k} & \rho_{\beta_{2k}\beta_{1k}} & \sigma^2_{\beta_{2k}} & \rho_{\beta_{2k}\beta_{3k}} \\ \rho_{\beta_{3k}\alpha_k} & \rho_{\beta_{3k}\beta_{1k}} & \rho_{\beta_{3k}\beta_{2k}} & \sigma^2_{\beta_{3k}} \end{pmatrix}\right), \text{ for participant k} = 1, \ldots, \text{K}$$

The model takes a little longer to run, and returns a convergence warning

```
mod2 <-
 glmer(response ~ baseline_slope + level_change + slope_change +
       (1 + baseline_slope + level_change + slope_change | participant) +
```

```
       (1|item),
           data = df_glmm,
           family = binomial)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.088281 (tol = 0.002, component 1)
```

Again, we change the optimizer.

```
mod2 <-
 glmer(response ~ baseline_slope + level_change + slope_change +
       (1 + baseline_slope + level_change + slope_change | participant) +
       (1|item),
           data = df_glmm,
           family = binomial,
       control = glmerControl(optimizer = "bobyqa"))
```

Since the model appears to have converged, we can examine the model results

```
summary(mod2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: response ~ baseline_slope + level_change + slope_change + (1 +
##     baseline_slope + level_change + slope_change | participant) +
##     (1 | item)
##    Data: df_glmm
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   5652.2   5755.8  -2811.1   5622.2     7385
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.1144 -0.3551 -0.1630  0.3241 13.5509
##
## Random effects:
##  Groups      Name           Variance Std.Dev. Corr
##  item        (Intercept)    1.611095 1.26929
##  participant (Intercept)    0.404805 0.63624
##              baseline_slope 0.002648 0.05146   0.41
##              level_change1  2.390619 1.54616   0.18  0.86
##              slope_change   0.007304 0.08546  -0.78 -0.54 -0.52
## Number of obs: 7400, groups:  item, 322; participant, 20
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -3.44259    0.20649 -16.672  < 2e-16 ***
## baseline_slope  0.07624    0.01883   4.050 5.13e-05 ***
```

```
## level_change1   0.85948    0.39756   2.162 0.030628 *
## slope_change    0.09124    0.02582   3.534 0.000409 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) bsln_s lvl_c1
## baselin_slp -0.161
## level_chng1  0.096  0.310
## slope_chang -0.192 -0.666 -0.363
```

The summary table shows us there there are statistically significant effects for all three parameters: a small but significant trend at baseline, a fairly substantial level change on average, and an increase in slope from baseline that is slightly more than double the initial average trend. We also note that there there is much more variation in the level change parameter between participants relative to the baseline slope and slope change parameters. Finally, the correlation of fixed effects shows a positive association between individuals baseline trend and their level change, but a negative association between individuals baseline trend and slope change and level change.

We can calculate an overall effect size using the same approach as the individual model. In this case, we assume the median number of baseline sessions (11) and treatment sessions (20)

```
# setup marginal means
#
marginal_means = emmeans(
                        object = mod2,
                        specs = c("baseline_slope", "level_change", "slope_change"),
                        at = list(
                          baseline_slope = c(11, 31),
                          level_change = c("0", "1"),
                          slope_change = c(0, 20)
                        )
                      )

marginal_means
```

```
##  baseline_slope level_change slope_change  emmean    SE  df asymp.LCL asymp.UCL
##              11 0                        0 -2.6040 0.268 Inf   -3.1289  -2.07906
##              31 0                        0 -1.0792 0.587 Inf   -2.2294   0.07090
##              11 1                        0 -1.7445 0.545 Inf   -2.8120  -0.67696
##              31 1                        0 -0.2197 0.814 Inf   -1.8147   1.37518
##              11 0                       20 -0.7791 0.393 Inf   -1.5503  -0.00788
##              31 0                       20  0.7457 0.410 Inf   -0.0586   1.54996
##              11 1                       20  0.0804 0.480 Inf   -0.8613   1.02206
##              31 1                       20  1.6051 0.581 Inf    0.4668   2.74346
##
## Results are given on the logit (not the response) scale.
## Confidence level used: 0.95
```

Because the baseline trend, on average, was statistically reliable, we calculated an overall effect size assuming that it would have continued in the absence of treatment.

```
# code to select first and last rows
# The 1 indicates that the row should be selected
A = c(0, 1, 0, 0, 0, 0, 0, 0)
B = c(0, 0, 0, 0, 0, 0, 0, 1)

# contrast the marginal means
# infer argument returns a confidence interval and p value if
# both are set to TRUE.
contrast(marginal_means,
     method = list("Unadjusted effect size" = B-A),
     infer = c(TRUE, TRUE))
```

```
##  contrast               estimate    SE  df asymp.LCL asymp.UCL z.ratio p.value
##  Unadjusted effect size     2.68 0.525 Inf      1.66      3.71   5.112  <.0001
##
## Results are given on the log odds ratio (not the response) scale.
## Confidence level used: 0.95
```

This results in a statistically reliable group effect size of 2.7 logits. Given that the group model suggests a starting place of only around 3% (found by converting logits to percentage `plogis(-3.44)`), this indicates a gain of about 35 percentage points on average can be attributed to the level and slope changes. We can estimate change in percentage points by adding the intercept, the baseline slope times an average number of baseline sessions, and the predicted change, converting to the predicted percent correct, and then subtracting the percent correct at baseline plus the baseline change: `plogis(-3.44 + 0.07*5 + 2.68)-plogis(-3.44 + 0.07*5)`. However, we're not aware of a straightforward method of estimating individual effect sizes and confidence intervals using the frequentist approach.

Bayesian Mixed effects models

Bayesian mixed-effects models can be used in the same fashion as model 2 above to obtain both group and individual effect size estimates. First, a group-level model is estimated.

```
mod3 <-
 brm(
   # population level effects
   response ~ 0 + Intercept + baseline_slope + level_change + slope_change +
   # group level effects
     (1 + baseline_slope + level_change + slope_change | participant) +
     (1|item),
             data = df_glmm,
```

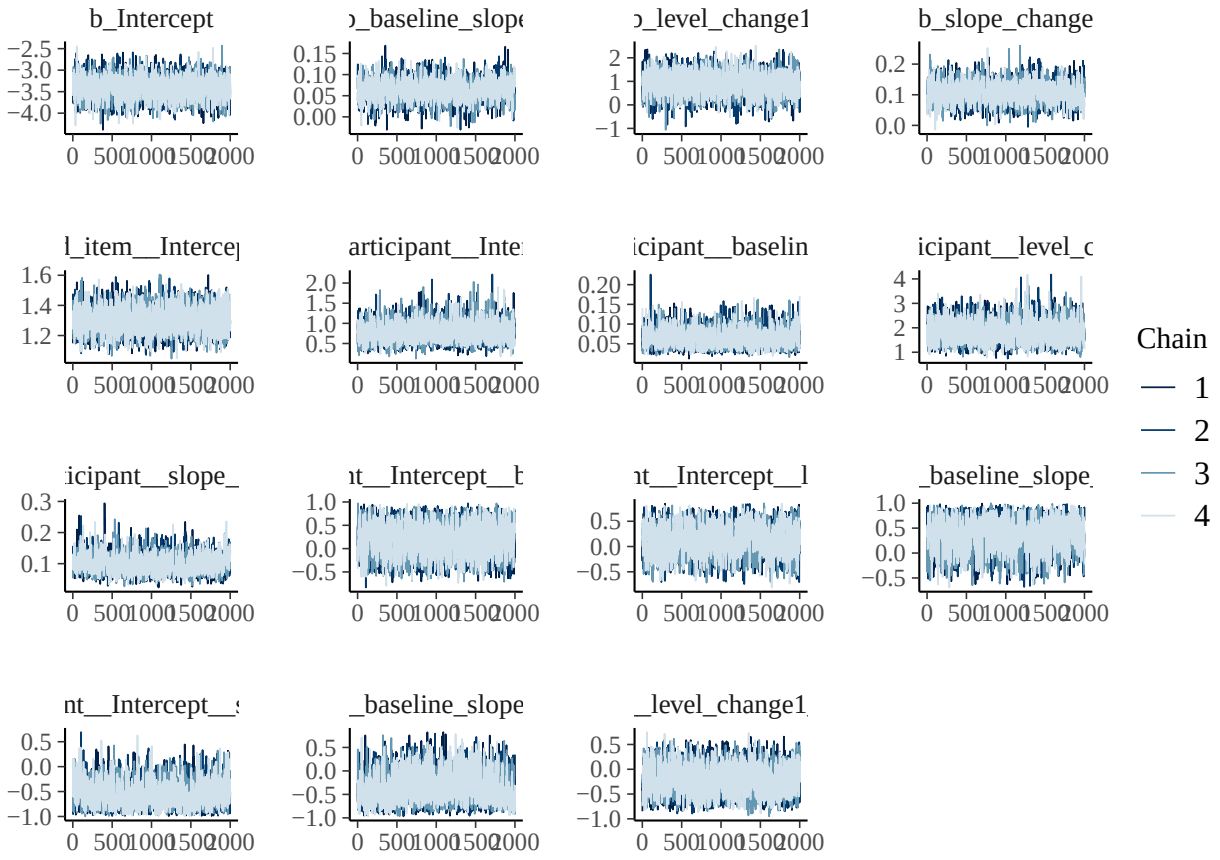
```
            family = bernoulli(), # special case of binomial with n=1 trials
            iter = 3000, # number of draws per chain
            warmup = 1000, # number of draws to toss on "burn in"
            chains = 4, # total number of chains
            seed = 42, # set a seed
            prior = c( # prior distributions
              prior(normal(-1, 2), class = b, coef = Intercept),
              prior(normal(0, 2.5), class = b)
            ),
            # extra arguments, see rmd file
            cores = 4,
            file = here("output", "group_brm"),
            file_refit = "on_change")
```

We can check several aspects of model fit and convergence.

1. Traceplots: Note that there are four lines within each plot, representing all the samples from each Markov Chain Monte Carlo (MCMC) simulation from the above model. There should be no noticeable variations in the patterns of the estimates across each model. Visual inspection of convergence is achieved by affirming that the MCMC estimates were consistently sampled from a range of values. Graphically, this is presented by the horizontal structure in each trace plot. For example, all estimates of model 3s intercept, `b_Intercept` fall within a range of about -4 and -2.5 logits. "Spikes" observed in traceplots are normal, as can be seen in the `participant_slope` plot. These spikes represent estimates at the tail end of a probability distribution. Traceplots with positive or negative slopes or sinusoidal wave shapes do not reflect convergence.

```
brms::mcmc_plot(mod3, type = "trace")
```

```
## No divergences to plot.
```



2. Check that the model can successfully re-estimate the data. In this case, we conducted posterior predictive checks on three statistics that can be estimated from the model. The posterior_predict function creates n number of posterior distributions, where n is the total number of post warmup iterations for one MCMC chain. We label these distributions `yrep` in the code below.

Then, we estimate posterior predictive p-values for the mean, standard deviation, and kurtosis of the n number of posteriors included in yrep. We note, that this is only one type of method of posterior checking, and was chosen because graphical posterior predictive checks of the predicted number outcome estimates from logistic regression models involving a large number of 0s can be misleading.

```
y = mod3$data$response
yrep = posterior_predict(mod3)
mean_ppc = mean(apply(yrep, 1, mean) > mean(y))
sd_ppc = mean(apply(yrep, 1, sd) > sd(y))
kurtosis_ppc = mean(apply(yrep, 1, e1071::kurtosis) > e1071::kurtosis(y))

print(c(mean_ppc, sd_ppc, kurtosis_ppc))
```

```
## [1] 0.506625 0.510625 0.489125
```

`mean_ppc`, `sd_ppc`, and `kurtosis_ppc` represent the proportion of mean, standard deviation, and kurtosis estimates less than the observed estimates of these parameters. The p-value is the proportion of posterior samples with greater prediction error than the actual data. Low p-values mean worse model fit than expected if the model were correct. Values near 0.5 are ideal.

3. Rhat statistic should be < 1.05, ideally < 1.01

The rhat stastistic represents the potential scale reduction factor across the split chains, the case of this model across the four chains. A value of 1.0 represents perfect convergence and estimates between 1 and 1.05, and ideally between 1 and 1.01 represent adequate convergence. Model fitting should be reevaluated with rhat statistics greater than 1.05.

```
max(rhat(mod3))
```

```
## [1] 1.003422
```

We can preview the model results using `summary()` again. Notably, the model estimates are largely similar to the frequentist model.

```
summary(mod3)
```

```
##  Family: bernoulli
##   Links: mu = logit
## Formula: response ~ 0 + Intercept + baseline_slope + level_change + slope_change + (1 + baseline_slo
##    Data: df_glmm (Number of observations: 7400)
##   Draws: 4 chains, each with iter = 3000; warmup = 1000; thin = 1;
##          total post-warmup draws = 8000
##
## Group-Level Effects:
## ~item (Number of levels: 322)
##               Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## sd(Intercept)     1.30      0.08     1.15     1.46 1.00     2366     4243
##
## ~participant (Number of levels: 20)
##                                   Estimate Est.Error l-95% CI u-95% CI Rhat
## sd(Intercept)                         0.77      0.23     0.40     1.28 1.00
## sd(baseline_slope)                    0.06      0.02     0.03     0.11 1.00
## sd(level_change1)                     1.80      0.40     1.12     2.70 1.00
## sd(slope_change)                      0.10      0.03     0.05     0.16 1.00
## cor(Intercept,baseline_slope)         0.21      0.33    -0.45     0.79 1.00
## cor(Intercept,level_change1)          0.15      0.28    -0.43     0.65 1.00
## cor(baseline_slope,level_change1)     0.36      0.32    -0.32     0.87 1.00
## cor(Intercept,slope_change)          -0.56      0.25    -0.93     0.02 1.00
## cor(baseline_slope,slope_change)     -0.37      0.31    -0.85     0.33 1.00
## cor(level_change1,slope_change)      -0.22      0.28    -0.72     0.35 1.00
```

```
##                                   Bulk_ESS Tail_ESS
## sd(Intercept)                         2463     3834
## sd(baseline_slope)                    2559     3731
## sd(level_change1)                     3090     4799
## sd(slope_change)                      1180     2898
## cor(Intercept,baseline_slope)         1883     3701
## cor(Intercept,level_change1)          1310     2675
## cor(baseline_slope,level_change1)      915     1814
## cor(Intercept,slope_change)           1053     2559
## cor(baseline_slope,slope_change)       954     1958
## cor(level_change1,slope_change)       1657     2921
##
## Population-Level Effects:
##                Estimate Est.Error l-95% CI u-95% CI Rhat Bulk_ESS Tail_ESS
## Intercept         -3.41      0.24    -3.89    -2.96 1.00     2118     3857
## baseline_slope     0.06      0.02     0.02     0.11 1.00     1309     2171
## level_change1      0.90      0.45    -0.01     1.78 1.00     2587     3970
## slope_change       0.11      0.03     0.05     0.17 1.00     1582     2628
##
## 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).
```

Finally, we can calculate effect sizes from the model for each individual

To do this efficiently, we wrote a function which takes arguments for the model object, and an argument we called "adjust" that can be TRUE if we would like to extrapolate the baseline slope through the end of treatment and FALSE otherwise. The default is FALSE.

The function works be selecting rows for each participant and item in the data and estimating the posterior distribution for the values in each row.

Then the data is transformed and the posterior distribution at the beginning of treatment (or at the end of treatment without the level change and slope change parameters) is subtracted from the posterior distribution at the end of treatment.

The resulting posterior distribution characterized the magnitude of change, the mean or median can be used as a point estimate and the middle 95% of the distribution is the 95% credible interval.

```r
glmmES = function(fit, adjust = FALSE){

  # start with the data that went into the model,
  # for each participant and phase (here we just used level_change
  # because they are equivalent) make a new variable called last_session
  # which is the highest value of the baseline slope coefficient
  # then filter for only rows where the baseline slope is
  # equal to the highest value in the phase (in other words
  # this selects the last baseline and last treatment session).
```

```r
# Remove the response column, reduce the data frame to only the
# unique rows
data = fit$data %>%
    group_by(level_change, participant) %>%
    mutate(last_session = max(baseline_slope)) %>%
    filter(baseline_slope == last_session) %>%
    select(-response) %>%
    distinct()
# If adjust is TRUE, then for each participant,
# set the baseline slope to always equal the highest value
# of baseline slope.
# in other words, we end up with a row where baseline slope
# represents the last treatment session, but level and slope
# change are still zero.
if(adjust){
 data = data %>%
    group_by(participant) %>%
    mutate(baseline_slope = max(baseline_slope))
}

# calculate an effect size in logits or log-odds
# start with the data frame we just created and add
# model draws from the linear/link-level predictor
# change timepoint to entry if level change is
# 0 and exit if 1. Select only the needed columns
# and take the data from long to wide. using pivot_wider
# Then subtract the value for the draw for entry from
# the draw for exit.
# then for each participant, calculate the point
# estimate and interval using point_interval
# the last line just adds a column indicating this effect size is in logits
linepred = data %>%
  add_linpred_draws(fit) %>%
  ungroup() %>%
  mutate(timepoint = ifelse(level_change == 0, "entry", "exit")) %>%
  select(timepoint, item, .draw, .linpred, participant) %>%
  pivot_wider(names_from = "timepoint", values_from = .linpred) %>%
  mutate(ES = exit-entry) %>%
  group_by(participant) %>%
  point_interval(ES) %>%
  mutate(unit = "logit")

# This block does the same thing, except using the expectation of the
# posterior (i.e., in percent correct terms)
epred = data %>%
  add_epred_draws(fit) %>%
  ungroup() %>%
  mutate(timepoint = ifelse(level_change == 0, "entry", "exit")) %>%
  select(timepoint, .draw, item, .epred, participant) %>%
  pivot_wider(names_from = "timepoint", values_from = .epred) %>%
  mutate(ES = exit-entry) %>%
  group_by(participant) %>%
  point_interval(ES) %>%
```

participant	ES	.lower	.upper	.width	.point	.interval	unit
P1	2.1895495	-0.3676083	4.988229	0.95	median	qi	logit
P10	4.0086213	1.5187068	6.748027	0.95	median	qi	logit
P11	2.9714045	1.5204160	4.609804	0.95	median	qi	logit
P12	3.0598628	1.9864749	4.130898	0.95	median	qi	logit
P13	5.8589987	4.3140540	7.506057	0.95	median	qi	logit
P14	3.9626795	2.6624822	5.335825	0.95	median	qi	logit
P15	2.2477880	0.6808346	3.828825	0.95	median	qi	logit
P16	7.2926630	4.4957225	11.309594	0.95	median	qi	logit
P17	6.2582404	3.5445388	10.266857	0.95	median	qi	logit
P18	1.5851814	-0.0863041	3.492992	0.95	median	qi	logit
P19	5.4483097	2.8257044	8.691806	0.95	median	qi	logit
P2	5.7103733	2.8251632	9.211314	0.95	median	qi	logit
P20	3.4349465	2.2776165	4.745630	0.95	median	qi	logit
P3	5.0591621	2.4579499	8.537965	0.95	median	qi	logit
P4	4.6388253	2.4688143	7.444193	0.95	median	qi	logit
P5	2.8040056	1.1536990	4.869518	0.95	median	qi	logit
P6	3.7047279	1.2918042	6.876803	0.95	median	qi	logit
P7	0.2205605	-2.6562366	2.434720	0.95	median	qi	logit
P8	2.2802549	-0.1926789	4.786389	0.95	median	qi	logit
P9	1.1980474	-1.1147377	3.580816	0.95	median	qi	logit

```
    mutate(unit = "percent")

  return(bind_rows(linepred, epred))
}
```

Here's how we might use the function, adjusting for baseline slope

```
bayesian_es = glmmES(mod3, adjust = TRUE)
```

Examine the results for logits

```
head(bayesian_es %>% filter(unit == "logit"), 20) %>% kable(format = "latex", booktabs = TRUE) %>%
  kable_styling(position = "center")
```

Examine the results for percent

```
head(bayesian_es %>% filter(unit == "percent"), 20) %>% kable(format = "latex", booktabs = TRUE) %>%
  kable_styling(position = "center")
```

To convert to odds ratios, exponentiate the logit effect sizes. To convert to number of items correct, multiply the percent gain by the number of items treated

participant	ES	.lower	.upper	.width	.point	.interval	unit
P1	0.3181775	-0.0483361	0.7443294	0.95	median	qi	percent
P10	0.4250740	0.0199313	0.8940472	0.95	median	qi	percent
P11	0.0869928	0.0082954	0.6206214	0.95	median	qi	percent
P12	0.5262244	0.1291868	0.7416138	0.95	median	qi	percent
P13	0.8543259	0.5419068	0.9424602	0.95	median	qi	percent
P14	0.6685233	0.2632803	0.8466362	0.95	median	qi	percent
P15	0.3431008	0.0433472	0.6799142	0.95	median	qi	percent
P16	0.8872033	0.5491569	0.9820621	0.95	median	qi	percent
P17	0.5681732	0.0907430	0.9129990	0.95	median	qi	percent
P18	0.0834094	-0.0056740	0.3588588	0.95	median	qi	percent
P19	0.7742871	0.1480239	0.9677798	0.95	median	qi	percent
P2	0.5197516	0.0352241	0.9583102	0.95	median	qi	percent
P20	0.4809946	0.0834361	0.7640616	0.95	median	qi	percent
P3	0.7847071	0.2275039	0.9540422	0.95	median	qi	percent
P4	0.7240730	0.1448200	0.9362476	0.95	median	qi	percent
P5	0.4817848	0.0932917	0.7780038	0.95	median	qi	percent
P6	0.6157490	0.2034815	0.8813546	0.95	median	qi	percent
P7	0.0327783	-0.2965442	0.5060892	0.95	median	qi	percent
P8	0.3184706	-0.0086866	0.7888893	0.95	median	qi	percent
P9	0.1379460	-0.1858118	0.6055480	0.95	median	qi	percent

```
sessionInfo()
```

```
## R version 4.2.1 (2022-06-23)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Monterey 12.5
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] officer_0.4.3      flextable_0.7.2    forcats_0.5.1      stringr_1.4.0
##  [5] dplyr_1.0.9        purrr_0.3.4        readr_2.1.2        tidyr_1.2.0
##  [9] tibble_3.1.8       ggplot2_3.3.6      tidyverse_1.3.2    ggdist_3.2.0
## [13] tidybayes_3.0.2    brms_2.17.0        Rcpp_1.0.9         emmeans_1.7.5
## [17] lme4_1.1-30        Matrix_1.4-1       SingleCaseES_0.6.1 here_1.0.1
## [21] kableExtra_1.3.4
##
## loaded via a namespace (and not attached):
##   [1] uuid_1.1-0           readxl_1.4.0         backports_1.4.1
##   [4] systemfonts_1.0.4    plyr_1.8.7           igraph_1.3.4
##   [7] splines_4.2.1        svUnit_1.0.6         crosstalk_1.2.0
##  [10] rstantools_2.2.0     inline_0.3.19        digest_0.6.29
```

```
##  [13] htmltools_0.5.3      fansi_1.0.3          magrittr_2.0.3
##  [16] checkmate_2.1.0      ggbrace_0.1.0        googlesheets4_1.0.0
##  [19] tzdb_0.3.0           modelr_0.1.8         RcppParallel_5.1.5
##  [22] matrixStats_0.62.0   vroom_1.5.7          xts_0.12.1
##  [25] svglite_2.1.0        prettyunits_1.1.1    colorspace_2.0-3
##  [28] rvest_1.0.2          haven_2.5.0          xfun_0.31
##  [31] jsonlite_1.8.0       callr_3.7.1          crayon_1.5.1
##  [34] zoo_1.8-10           glue_1.6.2           gtable_0.3.0
##  [37] gargle_1.2.0         webshot_0.5.3        distributional_0.3.0
##  [40] pkgbuild_1.3.1       rstan_2.21.5         abind_1.4-5
##  [43] scales_1.2.0         mvtnorm_1.1-3        DBI_1.1.3
##  [46] miniUI_0.1.1.1       viridisLite_0.4.0    xtable_1.8-4
##  [49] proxy_0.4-27         bit_4.0.4            stats4_4.2.1
##  [52] StanHeaders_2.21.0-7 DT_0.23              htmlwidgets_1.5.4
##  [55] httr_1.4.3           threejs_0.3.3        arrayhelpers_1.1-0
##  [58] posterior_1.2.2      ellipsis_0.3.2       pkgconfig_2.0.3
##  [61] loo_2.5.1            farver_2.1.1         dbplyr_2.2.1
##  [64] utf8_1.2.2           labeling_0.4.2       tidyselect_1.1.2
##  [67] rlang_1.0.2          reshape2_1.4.4       later_1.3.0
##  [70] munsell_0.5.0        cellranger_1.1.0     tools_4.2.1
##  [73] cli_3.3.0            generics_0.1.3       broom_1.0.0
##  [76] ggridges_0.5.3       evaluate_0.15        fastmap_1.1.0
##  [79] yaml_2.3.5           bit64_4.0.5          processx_3.7.0
##  [82] knitr_1.39           fs_1.5.2             zip_2.2.0
##  [85] nlme_3.1-157         mime_0.12            projpred_2.1.2
##  [88] xml2_1.3.3           compiler_4.2.1       bayesplot_1.9.0
##  [91] shinythemes_1.2.0    rstudioapi_0.13      gamm4_0.2-6
##  [94] e1071_1.7-11         reprex_2.0.1         stringi_1.7.8
##  [97] ps_1.7.1             Brobdingnag_1.2-7    gdtools_0.2.4
## [100] lattice_0.20-45      nloptr_2.0.3         markdown_1.1
## [103] shinyjs_2.1.0        tensorA_0.36.2       vctrs_0.4.1
## [106] pillar_1.8.0         lifecycle_1.0.1      bridgesampling_1.1-2
## [109] estimability_1.4     data.table_1.14.2    httpuv_1.6.5
## [112] R6_2.5.1             promises_1.2.0.1     gridExtra_2.3
## [115] codetools_0.2-18     boot_1.3-28          colourpicker_1.1.1
## [118] MASS_7.3-57          gtools_3.9.3         assertthat_0.2.1
## [121] rprojroot_2.0.3      withr_2.5.0          shinystan_2.6.0
## [124] mgcv_1.8-40          parallel_4.2.1       hms_1.1.1
## [127] grid_4.2.1           class_7.3-20         coda_0.19-4
## [130] minqa_1.2.4          rmarkdown_2.14       googledrive_2.0.0
## [133] lubridate_1.8.0      shiny_1.7.2          base64enc_0.1-3
## [136] dygraphs_1.1.1.6
```