Compendium to "A Tutorial for Quantifying Within- and Between-Participant Variance in Multilevel Logistic Models"

Sean Devine, A. Ross Otto, James O. Uanhoro, Jessica Kay Flake

2022-10-12

What is this?

This is a supplemental companion piece for "A Tutorial for Quantifying Within- and Between-Participant Variance in Multilevel Logistic Models" (**PUBLICATION LINK**). This document is meant to contain all the code (code boxes and other miscellaneous analyses) used in the paper in one place for easy to follow use.

The goal of this document is to present all of the techniques presented in the paper in a reproducible and easy to follow manner. Throughout this compendium, we will refer to R functions. We assume readers are familiar with the basics of R syntax and some basic functionality, but will do our best to remain as clear as possible if they are not. This document is meant to be read *alongside* the original paper. We will not go into as much detail about the concepts being presented and, as such, urge readers to use this as a supplement to the original paper, rather than a wholly complete tutorial in its own right.

This document will proceed in the same order as the original paper and will include all of the same code and data. Note: in the paper, we included paranthetical referencing the associated compendium section to follow along with. These sections are listed here:

1. Loading and cleaning the data.
2. Fitting and summarizing `logistic_MLM0`.
3. Reproducing Figure 2 in the paper.
4. Computing the ICC using
    a. The Latent Threshold approach
    b. The Simulation approach
    c. The Linearization approach
    d. The Median Odds Ratio
5. Bootstrapping procedures
6. Computing the ICC with a model that contains predictors.
7. Supplemental Materials
    a. Linear regression of response times.
    b. Multilevel linear regression of response times.
    c. Reproduce Figure S1.

1. Loading and cleaning the data

First, the data is loaded from a .csv file provided on the OSF page for Bogdanov et al. (2022)[https://osf.io/26w4u/]. This is done with the `read.csv()` function that is built-in to R.

```
data <- read.csv('Bogdanovetal2021/DST_data_osf2.csv')
```

We then subset the data such that `data.Ctl` only contains data from participants in the control condition.

```r
data.Ctl <- data[data$condition=='control', ]
```

Finally, we reverse-code the `effort_choice` variable to make it easier to interpret, such that `0` refers to low effort choices and `1` refers to high effort choices.

```r
data <- data.Ctl$effort_choice <- abs(data.Ctl$effort_choice-1)
```

2. Fitting and summarizing `logistic_MLM0`.

Next, we fit `logisticMLM0` from the paper. As a reminder, this is the null model on which the variance parameters were estimated throughout the text.

To begin, we load the `lme4` package. If you do not have it installed already, run `install.packages('lme4')` to do so.

```r
# if the lme4 is not installed, run this command first:
# install.packages('lme4')
library(lme4)
```

Now we can fit the model described in the text. To do so, we use the `glmer()` function from `lme4`. The first variable before the `~` is the binary outcome variable, `effort_choice` in our case. After the `~`, we input the predictor variables, which in this case is just the intercept, designated by a `1`. Finally, we specify the random effects and grouping variable. Here, we only estimate random intercepts, hence we input a `1` before the `|` followed by the grouping factor, which here is `PID`. We then specify the data to be used, using the `data=data.Ctl` argument, and finally specify the distribution to be used, which in this case is the `binomial` distribution. For more information on `lme4` syntax in R, see https://www.learn-mlms.com/.

```r
logistic_MLM0 <- glmer(effort_choice ~ 1 + (1|PID), data=data.Ctl, family='binomial')
```

We then output the summary of the model to the console.

```r
summary(logistic_MLM0)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: effort_choice ~ 1 + (1 | PID)
##    Data: data.Ctl
##
##      AIC      BIC   logLik deviance df.resid
##  14234.5  14249.2  -7115.3  14230.5    11202
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.6890 -0.9050 -0.3631  1.0096  4.7586
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  PID    (Intercept) 0.7554   0.8691
## Number of obs: 11204, groups:  PID, 38
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.3338     0.1426  -2.341   0.0192 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As described in the text, the grand average intercept (γ_{00}) is -0.34, which represents the average log-odds of choosing the high-demand option. This can be converted into a probability as follows:

$P(\text{Choose High Demand}) = \frac{\exp(-0.34)}{1+\exp(-0.34)} = 0.42$.

In R, we can do this computation this as follows:

Extract the fixed effects from `logistic_MLM0` using the `fixef` function from `lme4`. Since this is a null model, the only fixed effect is the grand intercept (γ_{00}).

```
gamma_00 = fixef(logistic_MLM0)
```

Convert this to a probability using Eq. 2-3 in the main paper.

```
pChooseHighEffort <- exp(gamma_00)/(1+exp(gamma_00))
cat('estimated group average probability of choosing the high-effort cue = ', pChooseHighEffort, '\n')
```

```
## estimated group average probability of choosing the high-effort cue =  0.4173215
```

Print this estimate to the console.

```
cat('estimated group average probability of choosing the high-effort cue = ', pChooseHighEffort, '\n')
```

```
## estimated group average probability of choosing the high-effort cue =  0.4173215
```

3. Reproducing Figure 2 in the paper

To reproduce Figure 2 in the paper, we will extract the estimated demand preferences (in log odds) per participant in the DST. These estimates are also known as "empirical Bayes' estimates", hence we will store these in a variable called `eb`.

```
eb  <- coef(logistic_MLM0)$PID
head(eb)
```

```
##   (Intercept)
## 1 -1.35734356
## 2 -0.28074761
## 3 -0.03893175
## 4 -0.03199605
## 5  0.77900510
## 6 -0.42353287
```

Next, we extract the variance around the intercept (τ_0^2) using the `VarCorr` function from `lme4$`. The syntax here is a little convoluted. The `VarCorr` command extracts the random variance components from a fitted model (here `logistic_MLM0`). Alone, this yields a variance component per grouping variable and per number of random variance parameters specified in the model (τ_0^2, τ_1^2, etc.). In our case, we have one grouping variable, `PID`, and so we specify that we are interested in this the random intercept variance for this group using `$PID[1]`.

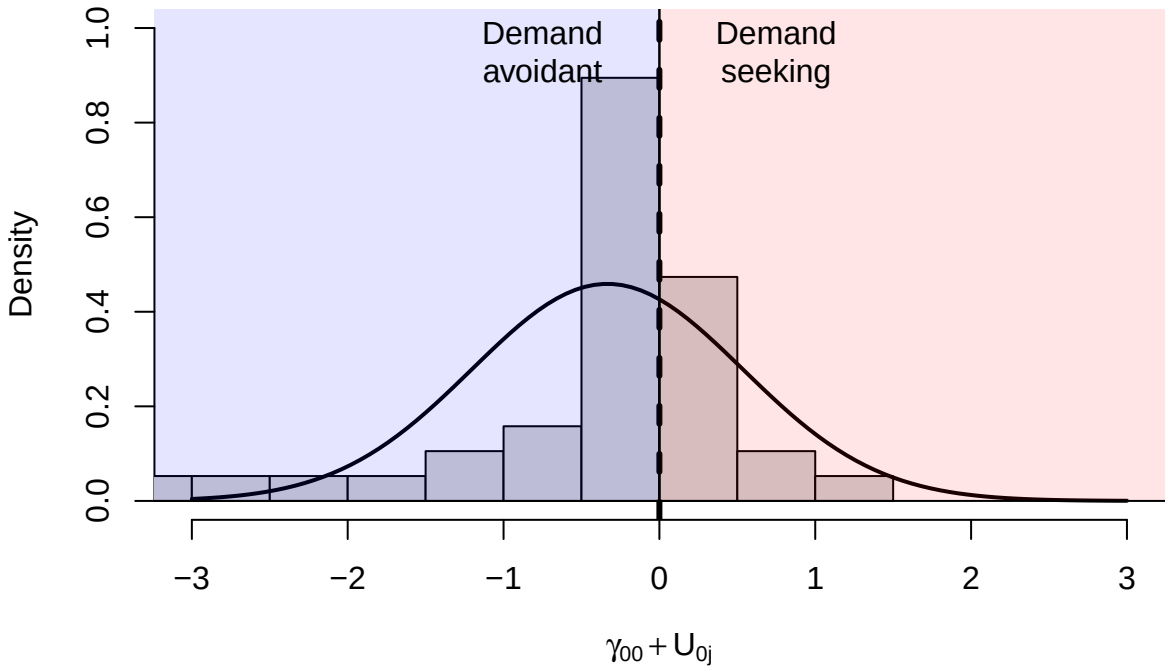
```
tau <- VarCorr(logistic_MLM0)$PID[1]
```

Finally, we visualize this distribution in a histogram and overall an estimated normal density curve with $\mu = \gamma_{00}$ and $\sigma = \sqrt{\tau_0^2}$. The specific steps taken here are not of critical importance to the topic of this tutorial, as they mainly involve plotting commands in base R. If this section is confusing for readers and they wish to be able to follow along better, we suggest this introductory website that explains basic visualization techniques inherent to R: https://bookdown.org/rdpeng/exdata/the-base-plotting-system-1.html.

```
# plot the histogram of the random intercepts and overlay an estimated normal density curve with
# mu = gamma00 and sd = sqrt(tau^2_0)
hist(eb[[1]], main='', xlab=expression(gamma[`00`] + U[`0j`]), freq = F, xlim=c(-3,3), ylim=c(0,1))
```

```r
curve(dnorm(x, fixef(logistic_MLM0), sqrt(tau)), add=T, lwd=2)

# finally, shade in areas of demand-avoidance (b0 < 0) and demand-seeking respectively (b0 > 0)
abline(v=0, lty='dashed', lwd=3)
rect(0, 0, 10, 100, col = scales::alpha('red', .1))
rect(0, 0, -10, 100, col = scales::alpha('blue', .1))
text(-.75, .95, labels = 'Demand\navoidant')
text(.75, .95, labels = 'Demand\nseeking')
```



4. Computing the ICC

4a. Latent Threshold Method

First, We extract the random intercept variance (τ_0^2) using the `VarCorr` command (see explanation above).

```r
tau20 <- VarCorr(logistic_MLM0)$PID[1]
```

Then, we specify the within-subjects (residual) variance to be $\sigma^2 = \frac{\pi^2}{3}$.

```r
sigma2    <- pi^2/3
```

Finally, we compute the ICC using the following equation $\frac{\tau_0^2}{\tau_0^2+\sigma^2}$ and print this output to the console.

```r
threshICC <- tau20/(tau20+sigma2)
cat('ICC using the Latent Threshold Approach = ',threshICC,'\n')
```

```
## ICC using the Latent Threshold Approach =  0.1867374
```

4b. Simulation Method

The simulation method requires multiple steps.

First, we set a seed for reproducibility. This ensures that the subsequent results will replicate on any device that runs this code.

```
set.seed(2022)
```

Next, we extract the the random intercept variance using `VarCorr`.

```
tau20 <- VarCorr(logistic_MLM0)$PID[1]
```

We extract the grand intercept (γ_{00}) using `fixef` (see explanation above).

```
gamma00 <- fixef(logistic_MLM0)[[1]]
```

We set a large number of simulations to execute (here `1e5`or 100 000). We call this variable `M`.

```
M <- 1e5
```

Next, we simulate the random effects from `M` draws and store these values in a variable called `U0j`. Note. we take the square root of τ_0^2 because `rnorm()` requires the standard deviation, not the variance.

```
U0j <- rnorm(M, 0, sqrt(tau20))
```

We then compute the expected probability (in log odds) of choosing the high demand option on each draw. Because `logisticMLM_0` has no predictors, this is computed simply as the grand intercept (γ_{00}) plus the random deviations from this intercept we simulated in the previous step.

```
logit_p_hat <- gamma00+U0j
```

This is then converted into a probability (using Eq. 2-3 from the paper).

```
p_hat <- exp(logit_p_hat)/(1+exp(logit_p_hat))
```

We can then compute the level 1 variance using the equation for the Bernoulli variance: $\text{var} = \hat{p} * (1 - \hat{p})$ (see main paper).

```
var_L1 <- p_hat*(1-p_hat)
```

Finally, we compute the ICC in the traditional way, taking the average level 1 variance as σ^2 and the variance of the predicted probabilities from each simulation $\hat{p}$ as τ_0^2

```
sigma2 <- mean(var_L1)
simICC <- var(p_hat)/(var(p_hat) + sigma2)
```

We then print the result to the console.

```
cat('ICC using the Simulation Model = ',simICC,'\n')
```

```
## ICC using the Simulation Model =  0.1390484
```

4c. Linearization Method

The linearization method also proceeds in a series of steps.

First, τ_0^2 and γ_{00} are extracted from `logistic_MLM0` using `VarCorr` and `fixef` as we have done above.

```
tau20   <- VarCorr(logistic_MLM0)$PID[1]
gamma00 <- fixef(logistic_MLM0)[[1]]
```

Next, we evaluate the probability of success at the mean of random effects (i.e., at the fixed effect, γ_{00}).

```r
p <- exp(gamma00)/(1+exp(gamma00))
```

We then compute the Bernouilli variance of this fixed estimate and store this value in a variable called `var1`.

```r
var1 <- p*(1-p)
```

Next, we compute the variance in the level-1 outcome using the linearized equation provided in the main text. We store this in a variable called `var2`.

```r
var2  <- tau20*p^2*(1 + exp(gamma00))^(-2)
```

Finally, with these values in hand, we compute the ICC, taking `var2` as our measure of between-person variance, τ_0^2, and `var1` as our measure of within-person variance, σ^2, and print the result of this computation to the console.

```r
linICC  <- var2/(var1+var2)

# print result
cat('ICC using the Linearization method = ',linICC,'\n')
```

```
## ICC using the Linearization method =  0.1551821
```

4d. Median Odds Ratio

The MOR is straightforward to compute.

First, we extract the random intercept variance from the model using `VarCorr`.

```r
tau20 <- VarCorr(logistic_MLM0)$PID[1]
```

Next, compute the 75th percentile of the cumulative distribution function of a standard normal distribution. In R, this can be done using the `qnorm` function and specifying the percentile (here 0.75).

```r
phi75 <- qnorm(.75) # 75th percentile of normal CDF
```

Now we can compute the MOR as: $MOR = \exp\left(\sqrt{2\tau_0^2}\phi(0.75)\right)$ and print this value to the console.

```r
MOR <- exp(sqrt(2*tau20)*phi75)
cat('Median Odds Ratio = ',MOR,'\n')
```

```
## Median Odds Ratio =  2.291137
```

5. Bootstrapping

Below we provide two bootstrapping techniques. First, we describe how to implement the bootstrapping procedure using the functions we provide alongside the paper. These functions are meant to be simple to implement, without the need for complex code that involves loops and storage. That being said, some readers may be interested in understanding this process. Accordingly, we provide a fully-worked example that goes through such a process.

5a. Bootstrapping using provided functions

To utilize the function calls stored in `fx.R`, we first have to source them using the `source()` function from R.

```r
source('fx.R')
```

Once this is done, we should have access to all of the custom functions provided in `fx.R`. Of interest to us here are those used for bootstrapping. Specifically, `bootstrap_icc`. This function takes the following arguments:

- `fit`: The fitted model
- `gr` : he grouping variable (in our case, `PID`)
- `method`: The method to use: `icc_thr` (Threshold method), `icc_sim` (Simulation method), `icc_lin` (Linearization method), `MOR` (MOR).
- `B`: The number of samples to produce.
- `seed`: The random seed to use.

As an example, we can compute 100 bootstraps of the ICC using the latent threshold method as follows. We can the print out these bootstrapped values.

Note: This can take a few moments.

```
bootstrap_thr <- bootstrap_icc(logistic_MLM0, gr='PID', method='icc_thr', B = 100, seed = 2022)
```

```
## ================================================================================
```

```
bootstrap_thr
```

```
##   [1] 0.16830697 0.15787487 0.23262770 0.19854612 0.17972198 0.16440196
##   [7] 0.17151428 0.17047279 0.15203957 0.14435074 0.20494568 0.22442203
##  [13] 0.27947478 0.15180074 0.16084417 0.11891926 0.12382954 0.15670157
##  [19] 0.21180850 0.26129530 0.09869003 0.12523149 0.18097583 0.18429135
##  [25] 0.24291502 0.18805121 0.27131427 0.18858676 0.19876350 0.19104779
##  [31] 0.17898273 0.19081998 0.21721753 0.19838683 0.13443018 0.16345277
##  [37] 0.17571961 0.21163977 0.25371038 0.18071584 0.19019566 0.22854664
##  [43] 0.22408069 0.16930695 0.20692029 0.13554932 0.19115447 0.15771066
##  [49] 0.17324681 0.15087385 0.21250849 0.22208254 0.21865074 0.21256008
##  [55] 0.18931581 0.21051402 0.15417479 0.19446001 0.10452172 0.22578986
##  [61] 0.21902593 0.17891327 0.16474834 0.16896050 0.23221552 0.18091489
##  [67] 0.22636177 0.17349255 0.14743257 0.09563111 0.23128899 0.15047848
##  [73] 0.14955724 0.15063867 0.15099533 0.19727423 0.16227090 0.19690115
##  [79] 0.17141104 0.23512419 0.16254044 0.11387893 0.22550295 0.17085960
##  [85] 0.14860604 0.22323194 0.16042631 0.21371768 0.17335656 0.19818017
##  [91] 0.23903077 0.21792764 0.16886654 0.22587631 0.20353066 0.14575272
##  [97] 0.14468275 0.14297011 0.21971101 0.18426185
```

Using the built-in `quantile` function in R, we can then compute 95% confidence intervals based on these bootstrapped estimates.

```
quantile(bootstrap_thr, c(.025, .975))
```

```
##      2.5%     97.5%
## 0.1089664 0.2576925
```

For completeness, we illustrate the same syntax for bootstrapping estimates using the other methods described above, but do not run this code for time purposes.

```
bootstrap_sim <- bootstrap_icc(logistic_MLM0, gr='PID', method='icc_sim', B = 100, seed = 2022)
quantile(bootstrap_sim, c(.025, .975))

bootstrap_lin <- bootstrap_icc(logistic_MLM0, gr='PID', method='icc_lin', B = 100, seed = 2022)
quantile(bootstrap_lin, c(.025, .975))

bootstrap_MOR <- bootstrap_icc(logistic_MLM0, gr='PID', method='MOR', B = 100, seed = 2022)
quantile(bootstrap_MOR, c(.025, .975))
```

5b. Bootstrapping "by-hand"

Below we provide commented code to estimate bootstrapped samples "by-hand" for readers who are interested. We will not comment on this in depth, as it is outside the scope of the tutorial and we provide functions to accomplish the same effect, but interested readers are welcome to explore the code below.

```
# 0. Set constants for bootstrapping procedure
B       <- 1000                    # number of bootstraps
ids     <- logistic_MLM0@frame$PID # extract id vector from model data
K       <- length(unique(ids))     # number of clusters (subjects)
nTrials <- table(ids)              # number of trials per subject
tau20   <- VarCorr(logistic_MLM0)$PID[1]
g00     <- fixef(logistic_MLM0)[[1]]
output  <- matrix(NA, B, 7, dimnames = list(NULL, c('iteration', 'threshICC', 'simICC', 'linICC', 'MOR'

# 1. Cycle through iterations (i) of bootstrapped samples
for(i in 1:B) {
  if(i%%100==0) cat('bootstrapping is', round(i/B,2)*100, '% complete.')
  # 1.1 Simulate data
  U0j           <- rnorm(K, 0, tau20)        # random deviations per subject
  LOR_j         <- g00 + U0j                 # log odds of response==1 per subject
  p_j           <- exp(LOR_j)/(1+exp(LOR_j)) # convert LOR to probability
  y_ij          <- sapply(1:K, function(k) rbinom(nTrials[k], 1, p_j[k]))
  y_ij          <- unlist(y_ij)              # break out of a list format

  # 1.2 Fit new model and compute values of interest
  thisMLM       <- glmer(y_ij ~ 1 + (1|ids), family='binomial')
  thisG00       <- fixef(thisMLM)[[1]]
  thistau20     <- VarCorr(thisMLM)$ids[1]

  # 1.2.1. Latent Threshold ICC
  thisthreshICC <- thistau20/(thistau20+pi^2/3)

  # 1.2.2. Simulation ICC
  thisU0j       <- rnorm(1e5, 0, sqrt(thistau20))
  logit_p_hat   <- thisG00+U0j
  p_hat         <- exp(logit_p_hat)/(1+exp(logit_p_hat))
  var_L1        <- p_hat*(1-p_hat)
  sigma2        <- mean(var_L1)
  thissimICC    <- var(p_hat)/(var(p_hat) + sigma2)

  # 1.2.3. Linearlization
  p             <- exp(thisG00)/(1+exp(thisG00))
  var1          <- p*(1-p)
  var2          <- thistau20*p^2*(1 + exp(gamma00))^(-2)
  thislinICC    <- var2/(var1+var2)

  # 1.2.4. MOR
  thisMOR       <- exp(sqrt(2*thistau20)*qnorm(.75))

  # 1.3. Save output
  output[i,] = c(i, thisthreshICC, thissimICC, thislinICC, thisMOR, thisG00, thistau20)

}
```

6. Computing the ICC with a model that contains predictors

We will fit a model that predicts effort choice as a function of trial number (see main text for more details).

First, we will compute a block-wise trial number estimate, since one was not provided in the main text. The details here are not very important for readers unfamiliar with loops in R. All you need to know is that the final output is a vector of increasing integers that represent the trial number within a block of the DST, which is called `trial` in the dataframe, `data.Ctl`.

```
data.Ctl = data.Ctl[order(c(data.Ctl$PID, data.Ctl$block)), ]
data.Ctl = data.Ctl[!is.na(data.Ctl$PID),]

trial = c()
for(i in unique(data.Ctl$PID)) {
  for(j in unique(data.Ctl$block)) {
    trial = c(trial, 1:nrow(data.Ctl[data.Ctl$PID==i & data.Ctl$block==j, ]))
  }
}

data.Ctl$trial=trial
```

Once these values are computed, we can model these data using the same `glmer` syntax described at the beginning of this document.

First, we centre trial number at its median. This is accomplished by simply substracting the median trial number (37) from each trial number in the vector. This guarantees that the intercept of the resultant model will represent the average probability (in log odds) of choosing the high-effort option at the middle trial of a block. We store these centred trial numbers in a vector called `trial_c`.

```
data.Ctl$trial_c <- data.Ctl$trial - median(data.Ctl$trial)
```

We will also scale this newly centred trial numbers, `trial_c` to be between 0 and 1. This is necessary because `glmer` will sometimes run into issues when predictors are too large in magnitude.

```
data.Ctl$trial0 <- data.Ctl$trial_c/max(data.Ctl$trial_c)
```

Having done this, we can fit the model below. Again, if this syntax is confusing to readers, visit https://www.learn-mlms.com/.

```
logistic_MLM1 = glmer(effort_choice ~ trial0 + (1|PID), data=data.Ctl, family='binomial')
summary(logistic_MLM1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: effort_choice ~ trial0 + (1 | PID)
##    Data: data.Ctl
##
##      AIC      BIC   logLik deviance df.resid
##  14206.7  14228.7  -7100.3  14200.7    11201
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.8579 -0.8999 -0.3562  0.9815  4.4082
##
## Random effects:
##  Groups Name        Variance Std.Dev.
##  PID    (Intercept) 0.7595   0.8715
```

```
## Number of obs: 11204, groups:  PID, 38
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -0.3322     0.1431  -2.322   0.0202 *
## trial0       -0.1953     0.0358  -5.455 4.89e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##        (Intr)
## trial0 -0.001
```

From here, the same code as described throughout this document could be used to estimate within- and between-person variability, but below we simply provide the functions to do so, available from `fx.R`. Readers are also welcome to compare these estimates to those obtained from `logistic_MLM0` above.

```
icc_thr(logistic_MLM1, 'PID')
```

```
## [1] 0.1875599
```

```
icc_sim(logistic_MLM1, 'PID')
```

```
## [1] 0.1367671
```

```
icc_lin(logistic_MLM1, 'PID')
```

```
## [1] 0.1558737
```

```
MOR(logistic_MLM1, 'PID')
```

```
## [1] 2.296285
```

7. Supplemental Materials

For the examples here, we will predict participants' response times during the DST (`TS_RT`) from whether the previous trial was the same decision rule as the previous trial (`Switch`).

a. Linear regression of response times

In R, we can compute a linear regression using the `lm()` command.

We predict the time it takes to respond from `Switch`. The outcome variable, `TS_RT` is placed before the `~` and the predictor, `Switch` is placed after. We then print the summary of the model to the console.

```
linear_regression <- lm(TS_RT ~ Switch, data=data.Ctl)
print(summary(linear_regression))
```

```
##
## Call:
## lm(formula = TS_RT ~ Switch, data = data.Ctl)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2348 -0.4562 -0.2658  0.0910 27.1693
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  1.00780    0.01307   77.12   <2e-16 ***
## SwitchTRUE   0.27067    0.01940   13.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.022 on 11202 degrees of freedom
## Multiple R-squared:  0.01708,    Adjusted R-squared:  0.01699
## F-statistic: 194.6 on 1 and 11202 DF,  p-value: < 2.2e-16
```

We can also print the residual variance, σ^2 of this model to the console using the `sigma` function with the model as the argument. Note: the `sigma` function return the standard deviation, which we then square to obtain σ^2.

```
cat('The residual variance (sigma^2) of the linear regession equals', sigma(linear_regression)^2, '\n')
```

```
## The residual variance (sigma^2) of the linear regession equals 1.045302
```

b. Multilevel linear regression of response times

To fit a multilevel form of the regression above, we use the `lmer` function from the `lme4` package.

We fit the multilevel model using similar syntax to `lm`. The only difference is that we include a term for the random effects, where we specify we want random intercepts (`1`) per (`|`) participant (`PID`).

```
linear_MLM <- lmer(TS_RT ~ Switch + (1|PID), data=data.Ctl)
```

We then extract the random intercept variance, τ_0^2 using `VarCorr` (see above for explanation). Unlike a logistic multilevel model, linear multilevel models have an estimated residual standard deviation term, which can be extracted using the `sigma` function and squared to produce the residual variance, σ^2.

```
tau20  <- VarCorr(linear_MLM)$PID[1]
sigma2 <- sigma(linear_MLM)^2
```

The ICC can be computed as: $ICC = \frac{\tau_0^2}{\tau_0^2+\sigma^2}$. The summary of the model can then printed, along with the ICC.

```
ICC <- tau20/(tau20+sigma2)
print(summary(linear_MLM))
```

```
## Linear mixed model fit by REML ['lmerMod']
## Formula: TS_RT ~ Switch + (1 | PID)
##    Data: data.Ctl
##
## REML criterion at convergence: 31097.5
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -1.7431 -0.4136 -0.1702  0.1363 27.4372
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  PID      (Intercept) 0.1235   0.3514
##  Residual             0.9272   0.9629
## Number of obs: 11204, groups:  PID, 38
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)  0.99589    0.05836   17.06
```

```
## SwitchTRUE   0.30480    0.01886   16.16
##
## Correlation of Fixed Effects:
##             (Intr)
## SwitchTRUE -0.146
```

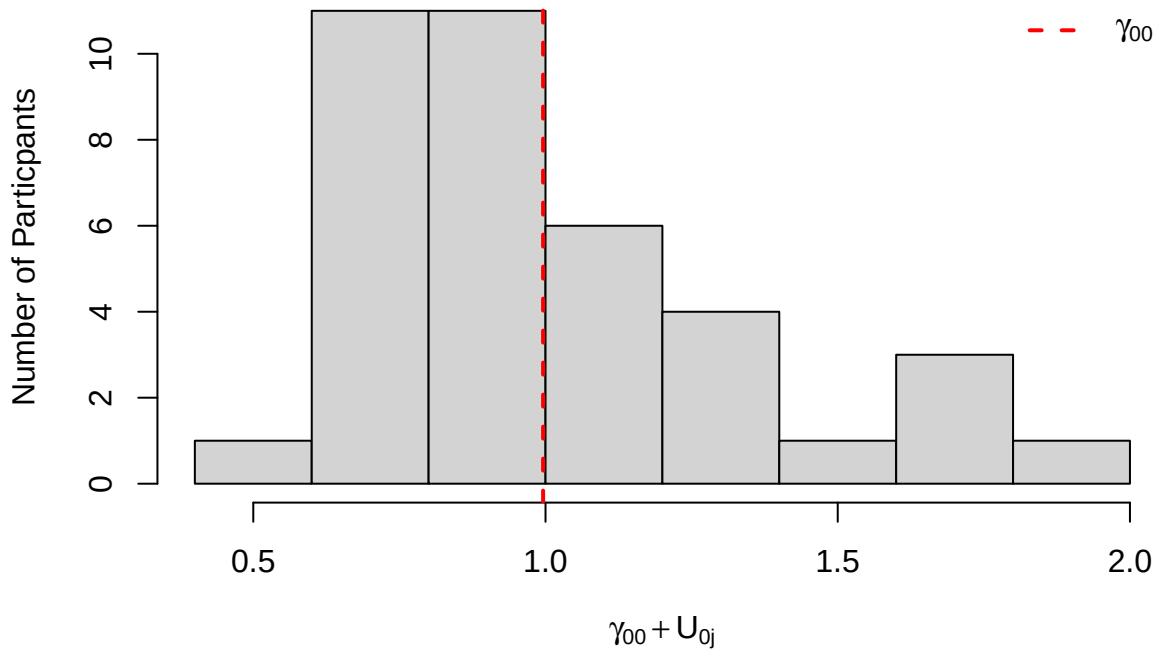
```
cat('The ICC for the linear MLM is', ICC, '\n')
```

```
## The ICC for the linear MLM is 0.1175482
```

c. Reproduce Figure S1

The steps to reproduce Figure S1 are more or less the same as to reproduce Figure 2. As such, we suggest readers read that section again to understand below. Again, if any of the visualization steps are unclear, we point readers to https://bookdown.org/rdpeng/exdata/the-base-plotting-system-1.html.

```
U0js <- coef(linear_MLM)$PID[,1]
g00  <- fixef(linear_MLM)[1]
hist(U0js, xlab = expression(gamma['00']+U['0j']), ylab='Number of Particpants', main='')
abline(v=g00, col='red', lty=2, lwd=2)
legend('topright', bty='n', col='red', lty=2, lwd=2, legend=expression(gamma['00']))
```



References

Bogdanov, M., Nitschke, J. P., LoParco, S., Bartz, J. A., & Otto, A. R. (2021). Acute psychosocial stress increases cognitive-effort avoidance. Psychological Science, 32(9), 1463-1475.