# Data Analysis and Power Simulations with General Linear Mixed Modelling for Psychophysical Data – A Practical R- and Julia-Based Guide

# Introduction

Data from psychophysical experimental designs such as Two-Alternative Forced-Choice tasks are challenging to analyze. Dependent variables are often binary (“yes”/”no”) and the data are often generated in a nested fashion, with multiple participants completing many trials that are often arranged in blocks, with manipulations occurring between participants and/or blocks or within participants and/or blocks. Simple statistical tools for Null Hypothesis Significance Testing like Student’s *t* tests or Analyses of Variance (ANOVAs) are not equipped to deal with these specific properties. For these reasons, usually Psychometric Functions (Cumulative Gaussian or Weibull functions) are fitted for each condition, block and participant to obtain the Points of Subjective Equality (PSEs) and Just Noticeable Differences (JNDs); see Figure 1. This yields one data point per subject, condition, and block, over which a t test or an ANOVA is performed to test for statistical significance. While this can be a valid approach, this generally neglects that each PSE and JND is based on a large number of trials and thus fails to account for the added reliability this provides. Depending on the experimental design, this may lead to a loss of statistical power. As a solution, Moscatelli at al. (Moscatelli, Mezzetti, & Lacquaniti, 2012) have suggested the use of General Linear Mixed Modelling (GLMM). GLMM allows to fit population parameters across all data, while still taking into account that responses within each condition and participant are correlated more strongly than across conditions and participants.

Power analyses are a crucial part of study planning: they provide an idea of how many participants have to be tested in how many trials obtain an adequate probability of detecting an effect, under the assumption that it is a true effect of a given strength. Psychophysical studies generally rely on heuristics when it comes to sample size planning. While this practice may not have the equally pernicious consequences in Cognitive Psychology as in other areas of psychology, replicability rates are still only about 50% in our area (Aarts et al., 2015). A more rigorous and principled approach to sample size planning, together with other efforts to increase openness and replicability, could be useful to increase this metric. Power analyses are slowly becoming more mainstream in other areas, but they still are the exception in the typical psychophysical study. Some general tutorials for power analyses through simulations have been brought forward that are quite easily adaptable to many different designs (Debruine & Barr, 2019; Kumle, Võ, & Draschkow, 2020). Based on Linear Mixed Modelling, these take into account complex data structures where often several participants complete a large number of trials in several conditions. However, very common psychophysical designs such as two-alternative forced-choice tasks, which typically generate psychometric curves, require additional considerations. Among these additional considerations are the fact that responses are often binary, and that relationships between dependent and independent variables are usually not linear. The present paper thus aims to give advice on how to conduct power analyses for common psychophysical designs that investigate the effect of a categorical experimental variable on precision and accuracy in two-alternative forced-choice paradigms.

In this manuscript, we thus first provide a step-by-step tutorial to simulate 2AFC psychophysical datasets. We then use these data to compare different statistical approaches to Null Hypothesis Significance Testing for one common psychophysical design. And finally, we will provide an implementation for power analyses based on the two analyses we identified as most robust in the previous chapter. This paper includes both theoretical considerations and practical implementations. We therefore provide code for the most practical parts (extraction of parameters from preexisting datasets, simulation of datasets, data analysis and power analysis) throughout the manuscript, and will provide GitHub links for the code we use to compare different analyses, model specifications and fitting methods.



Figure : Sample psychometric function for a two-alternative forced choice task. We plot the difference in stimulus intensity (x axis) against the probability to judge that the test stimulus had the higher intensity (black curve). The JND (Just Noticeable Difference), a measure of sensitivity/precision, is that difference in stimulus intensity that leads to a 25%/75% response probability (yellow); 0.7 in this example. The PSE (Point of Subjective Equality), a measure of biases/accuracy, is that stimulus intensity that leads to 50 % correct responses (red); 0 in this example.

# Simulating Psychophysical Data

We will first discuss the specifics of simulating psychophysical data for one typical case of psychophysical study: A Two-Alternative Forced-Choice task with a within-participant manipulation, where the presented stimuli are chosen according to a staircase procedure. Words in bold and between quotation marks refer to variables in the script. For some of the variables, we demonstrate how to derive them from existing (pilot) datasets. To this end, we will use pilot data from our lab.

## Required values

This method requires estimates of all relevant parameters. Some pertain to the stimuli, some can be taken from the literature, and some must be guessed (educatedly).

“**ID**” is a vector containing one ID for each subject we want to simulate.

“**ConditionOfInterest**” is a vector containing IDs for a binary categorical variable related to the main hypothesis of the experiment. For example: Is there a pictorial background scene?

“**StandardValues**” is a vector containing values for a categorial variable that serves as comparison stimuli. It can contain one value if you want to determine PSEs/JNDs for only one stimulus intensity, but typically you will have several, e. g. when you want to diversify your stimuli to show that a certain effect is not tied to one specific stimulus strength.

“**reps**” is a vector containing an ID for each trial, the maximum number being the average number of trials we expect for any given staircase.

“**PSE\_Difference**” is a value that indicates the percentage to which the PSEs differ between test and standard condition. It can be zero if the condition of interest is not expected to influence PSEs.

“**JND\_Difference**” is a value that indicates the percentage to which the JNDs differ between test and standard condition. It can be zero if the condition of interest is not expected to influence JNDs.

“**Mean\_Standard**” is the Mean of the psychometric function expected for the standard condition. In many cases, this is the stimulus strength of the comparison stimulus.

“**Multiplicator\_SD\_Standard**” is the Standard Deviation of the psychometric function expected for the standard condition, normalized to a mean of 1. We later multiply this normalized standard deviation by the Mean of the psychometric function we aim to simulate. That is, we assume that Weber fractions are constant across the tested stimulus range, which is generally assumed to hold for many cases. While this has been put into doubt (Krueger, 1989) and we recommend to verify to what extent Weber’s law holds for the stimulus in question, we believe this to be a reasonable simplification.

The standard deviation is thus proportional to the relevant Weber fraction and JNDs, which are available in the literature for many different stimulation types. Weber fractions and JNDs can be converted into standard deviations of psychometric functions and vice-versa. The JND is that difference in stimulus intensity that leads the participant to choose the correct stimulus in 75 % of the cases. Weber fractions are normalized versions of this value. Normalization is achieved by dividing it by the intensity of the standard stimulus. To obtain the standard deviation, convert JNDs first into Weber fractions. The Weber fraction is that distance to the mean where the psychometric function yields 25% or 75% correct responses. With the Weber Fraction given, we thus need to determine the appropriate standard deviation given these constraints .

“**SD\_Standard**” is then the standard deviation of the psychometric function for each stimulus intensity (**Multiplicator\_SD\_Standard** \* **Mean\_Standard**).

“**Type\_ResponseFunction**” describes the function the stimulus strengths are chosen from by the method. It can take the values "normal", "Cauchy" and "uniform". "Normal" and "Cauchy" are recommended when you are using a staircase procedure, while "uniform" corresponds to methods of constant stimuli. For a comparison between the three options, see further below. Figure 2visualizes different response distributions. A Gaussian distribution with an adequate standard deviation should be accurate enough for most intents and purposes when staircase procedures are used. The Cauchy distribution has more heavy tails and could be used if the starting values are relatively far away from the expected PSEs, and the initial step sizes are small. For the Method of Constant Stimuli, no randomness is involved in how the presented stimulus strengths are chosen. In this case, we use the values chosen for your stimulus.



Figure : Two sample distributions of stimulus strengths, representative of stimulus intensities presented when using a staircase procedure. The red distribution corresponds to stimulus strengths drawn from a Cauchy function with a mode of 1 and a scale of 0.05. The blue distribution are responses drawn from a Gaussian distribution with a mean of 1 and a standard deviation of 0.1.

“**SD\_ResponseFunction**” further describes the function the stimulus strengths are chosen from. For normal distributions, this value corresponds to its standard deviation; for Cauchy distributions, this corresponds to its scale; and for uniform distributions, this corresponds to a vector with the values tested.

We assume that there is between-participant variability in the means of the psychometric functions. “**Mean\_Variability\_Between**” sets the standard deviation of the normal distribution these PSEs are drawn from. This normal distribution has a mean of 1, that is, the standard deviation needs to be set accordingly.

We assume that there is between-participant variability in the standard deviations of the psychometric functions. “**SD\_Variability\_Between**” sets the standard deviation of the normal distribution these standard deviations are drawn from. This normal distribution has a mean of 1, that is, the standard deviation has to be set accordingly.

## Extracting the parameters from existing data

In the following, we will show with example pilot data from our lab how to extract the above values from an existing dataset. We collected these data in a velocity estimation task: Participants were shown two intervals of object motion in a 3D environment. One interval consisted in one big ball moving horizontally in front of the observer. The other consisted in a cloud of smaller balls moving in the same direction as the big target. The big ball moved at one of two speeds to the right (horizontal velocity signed positive) or to the left (horizontal velocity signed negative). The velocity of the ball cloud was controlled by a PEST staircase (Taylor & Creelman, 1967), with a slight adjustment: to get a more robust estimate of the JNDs, the stepsize did not change during the first 10 trials of each PEST. During the big target motion interval, the participant experienced visual self-motion in the same direction as the target (“congruent”), in the opposite direction of the target (“incongruent”) or no self-motion at all (“no motion”). Participants then judged by button press which of the motions was faster.

In the following, we provide the code we use to compute the needed values to accurately simulate datasets. We first load the necessary packages: “dplyr” for data manipulation, “quickpsy” to fit psychometric functions and “MASS” for an way to determine response functions and the parameters of these response functions. When then define and use the “Where\_Am\_I” function to set the working directory to the location of the script and read the Pilotdata.csv dataset, which should be located in the same directory as the script.

require(dplyr)

require(quickpsy)

require(MASS)

Where\_Am\_I <- function(path=T){

if (path == T){

dirname(rstudioapi::getSourceEditorContext()$path)

}

else {

rstudioapi::getSourceEditorContext()$path

}

}

setwd(Where\_Am\_I())

Dataframe <- read.csv(header=TRUE,"PilotData.csv")

We then bring the data into the format needed for quickpsy: First, we indicate whether for each trial the participant judged the test stimulus to be faster or slower than the comparison stimulus (“Pest\_Bigger”). We also compute the difference between test and comparison stimulus (“Difference”) and mark trials as “incongruent” (target and observer motion in opposite directions), “congruent” (target and observer motion in the same direction) and “no motion” (no observer motion). Then, we apply a very crude exclusion criterion by excluding all those trials where the test stimulus motion was more than two times faster than the comparison stimulus. We will furthermore only compare “incongruent” and “no motion” trials, as we limit these guidelines to comparing one baseline and one test condition.

Dataframe = Dataframe %>%

mutate(

Pest\_Bigger = case\_when(

Response\_Interval == Pest\_Interval ~ 1,

Response\_Interval != Pest\_Interval ~ 0),

Difference = abs(velH\_Pest)-abs(velH),

Congruent = case\_when(

velH\*velH\_Subject < 0 ~ "incongruent",

velH\*velH\_Subject > 0 ~ "congruent",

velH\*velH\_Subject == 0 ~ "1no motion")) %>%

filter(abs(velH\_Pest) < abs(velH)\*2 & Congruent != "1no motion")

While there are different methods to fit psychometric functions that each have their own benefits, we use a direction likelihood maximization method (Knoblauch & Maloney, 2012; Prins & Kingdom, 2016), implemented in the R package quicksy (Linares & López-Moliner, 2016). The bootstrap option is used to compute confidence intervals, which allow for statistical comparisons. However, the quickpsy package currently does not include an option to estimate population-wide parameters. We thus deactivate the bootstrap option, which speeds up the fitting process significantly. We fit separate psychometric functions for each self-motion condition, participant and object velocity. Then, we can use the plot() function from base R to plot the psychometric functions.

PsychometricFunctions = quickpsy(Dataframe,Difference,Pest\_Bigger,

grouping = .(Congruent,participant,velH),

bootstrap = "none")

plot(PsychometricFunctions)

From the quickpsy object, we can extract the estimates for means and standard deviations of the fitted cumulative Gaussians. We save means and standard deviations in separate tibbles.

PSEs = PsychometricFunctions$par %>%

filter(parn == "p1" & Congruent != "congruent")

SDs = PsychometricFunctions$par %>%

filter(parn == "p2" & Congruent != "congruent")

The PSE corresponds to the means of the fitted cumulative Gaussian functions. To get an estimate for PSE\_Difference, we normalize the estimated mean for each condition by dividing it by the velocity of the comparison stimulus. We then take the mean of these values for “congruent” and “no motion” conditions and subtract one from the other.

PSEs\_Condition1\_Absolute = (PSEs %>% filter(Congruent == "incongruent"))$par

velHs\_Condition1 = abs((PSEs %>% filter(Congruent == "incongruent"))$velH)

PSEs\_Condition1\_Percentage = PSEs\_Condition1\_Absolute/velHs\_Condition1

Mean\_PSE\_Condition1\_Percentage = mean(PSEs\_Condition1\_Percentage)

PSEs\_Condition2\_Absolute = (PSEs %>% filter(Congruent == "1no motion"))$par

velHs\_Condition2 = abs((PSEs %>% filter(Congruent == "1no motion"))$velH)

PSEs\_Condition2\_Percentage = PSEs\_Condition2\_Absolute/velHs\_Condition2

Mean\_PSE\_Condition2\_Percentage = mean(PSEs\_Condition2\_Percentage)

PSE\_Difference = Mean\_PSE\_Condition1\_Percentage-Mean\_PSE\_Condition2\_Percentage

We follow the same procedure for JND\_Difference. While the standard deviation of the fitted Cummulative Gaussian is not the same as the JND, they are proportional. Since JND\_Difference is expressed as a percentage, the difference between standard deviation and JND in absolute values is not a problem.

SDs\_Condition1\_Absolute = (SDs %>% filter(Congruent == "incongruent"))$par

velHs\_Condition1 = abs((SDs %>% filter(Congruent == "incongruent"))$velH)

SDs\_Condition1\_Percentage = SDs\_Condition1\_Absolute/velHs\_Condition1

Mean\_SD\_Condition1\_Percentage = mean(SDs\_Condition1\_Percentage)

SDs\_Condition2\_Absolute = (SDs %>% filter(Congruent == "1no motion"))$par

velHs\_Condition2 = abs((SDs %>% filter(Congruent == "1no motion"))$velH)

SDs\_Condition2\_Percentage = SDs\_Condition2\_Absolute/velHs\_Condition2

Mean\_SD\_Condition2\_Percentage = mean(SDs\_Condition2\_Percentage)

JND\_Difference = Mean\_SD\_Condition1\_Percentage-Mean\_SD\_Condition2\_Percentage

Mean\_Standard is the mean PSE across participants for the “no motion” condition, after normalizing it by adding the mean target velocity and dividing this sum by the mean target velocity, and “Multiplicator\_SD\_Standard” is the mean standard deviation across participants for the “no motion” condition, again after normalizing. We already computed these values above:

Mean\_Standard = mean(PSEs\_Condition2\_Percentage)

Multiplicator\_SD\_Standard = mean(SDs\_Condition2\_Percentage)

Similarly, we can use the values from above to get the between-participant variability for PSEs and standard deviations of the psychometric functions:

Mean\_Variability\_Between = sd(PSEs\_Condition2\_Percentage)

SD\_Variability\_Between = sd(SDs\_Condition2\_Percentage)

To choose whether a Gaussian or a Cauchy function is more appropriate for “ResponseFunction” and determine their standard deviation or scale, respectively, we can use the fitdistr() function from the MASS package to determine the best fit for each PEST. To get the normalized value, we use the function to fit Gaussian and Cauchy functions to the quotient velH\_Pest/velH, separately for each congruency condition, participant and target velocity. We furthermore extract the loglikelihood from the fit as a measure of modelfit. We subtract the loglikelihood for the Normal distribution from the loglikelihood for the Cauchy distribution. Higher loglikelihoods signify a better model fit. When this difference is positive, the Cauchy distribution makes for the better fit, and if it is negative, the Normal distribution makes for the better fit. We then take the median for each of these parameters across all conditions as final values for **SD\_ResponseFunction.**

Dataframe %>%

group\_by(participant) %>%

mutate(Scale\_Cauchy = fitdistr(velH\_Pest/velH,"cauchy")$estimate[2],

SD\_Normal = fitdistr(velH\_Pest/velH,"normal")$estimate[2],

loglikelihood\_Cauchy = fitdistr(velH\_Pest/velH,"cauchy")$loglik,

loglikelihood\_Normal = fitdistr(velH\_Pest/velH,"normal")$loglik,

loglikelihood\_Difference = loglikelihood\_Cauchy-loglikelihood\_Normal) %>%

dplyr::select(participant,Scale\_Cauchy,loglikelihood\_Cauchy,SD\_Normal,loglikelihood\_Normal, loglikelihood\_Difference) %>%

slice(1) %>%

ungroup() %>%

summarise(median\_Scale\_Cauchy = median(Scale\_Cauchy),

median\_SD\_Normal = median(SD\_Normal),

median\_loglike\_CauchyMinusNormal = median(loglikelihood\_Difference))

if (ResponseDistribution[3] > 0){

SD\_ResponseFunction = ResponseDistribution[1]

} else {

SD\_ResponseFunction = ResponseDistribution[1]

}

Please note that this procedure yields the same variability parameter for both conditions. That is, we assume that the precision is not vastly different between the conditions. For JND differences bigger than 25 %, it might be advisable to use different variability parameters for the baseline and the test condition.

The whole package, including code and sample data, is available on [GitHub](https://github.com/b-jorges/Power-Analyses-Psychophysics/tree/master/Get%20parameters%20from%20existing%20data).

## Simulating the Data

We can then proceed to simulating the dataset. You can use either the above procedure to extract the values from a dataset or make estimated guesses about each value based on the literature.

ID = paste0("s",1:15)

ConditionOfInterest = c(0,1)

StandardValues = c(5,8)

reps = 1:100

PSE\_Difference = -0.1

JND\_Difference = 0.25

Multiplicator\_PSE\_Standard = 0

Multiplicator\_SD\_Standard = 0.15

Type\_ResponseFunction = "Normal"

SD\_ResponseFunction = 0.1

Mean\_Variability\_Between = 0.1

SD\_Variability\_Between = 0.1

Next, we simulate one whole data set based on the above values. We first create a data frame with one row for each trial.

Psychometric = expand.grid(ID=ID, ConditionOfInterest=ConditionOfInterest, StandardValues=StandardValues, reps = reps)

Then, we draw multiplicators for PSEs and JNDs per subject, accounting for between-subject differences in biases and precision.

Psychometric = Psychometric %>%

group\_by(ID) %>%#

mutate(PSE\_Factor\_ID = rnorm(1,1,Mean\_Variability\_Between),

SD\_Factor\_ID = rnorm(1,1,SD\_Variability\_Between))

Omitting this step amounts to the assumption that the effect of interest is equally strong in each participant. This can be a valid assumption, but it should not be the default. Rather, the value chosen here should be justified, independently of whether it is zero or above zero. Next, we simulate means and standard deviations of the psychometric functions for each condition and we add between-subject variability, and factor between-subject variability in.

Psychometric = Psychometric %>%

mutate(

Mean\_Standard = StandardValues+StandardValues\*Multiplicator\_PSE\_Standard,

SD\_Standard = StandardValues\*Multiplicator\_SD\_Standard,

Mean = (Mean\_Standard + (ConditionOfInterest==1)\*Mean\_Standard\*PSE\_Difference),

SD = abs(SD\_Standard + (ConditionOfInterest==1)\*SD\_Standard\*JND\_Difference))

Psychometric = Psychometric %>%

mutate(

Mean = Mean\*PSE\_Factor\_ID,

SD = SD\*SD\_Factor\_ID)

Then, we draw the stimulus strengths likely to be presented in our experiment. As mentioned above, this varies depending on the way the experiment is controlled. For staircase procedures, the responses are more akin to normal distributions with relatively low standard deviations or Cauchy distributions with low scales. A good way to determine the most appropriate function would be to plot the distribution of presented stimulus strengths for pilot data and compare them to different distributions. For the method of constant stimuli, the responses are typically uniformly distributed across 5 to 9 values around the standard stimulus strength.

if (Type\_ResponseFunction == "normal"){

Psychometric = Psychometric %>%

mutate(

staircase\_factor = pnorm(length(reps),1,SD\_ResponseFunction\*(1+ConditionOfInterest\*JND\_Difference)))

} else if (Type\_ResponseFunction == "Cauchy"){

Psychometric = Psychometric %>%

mutate(

staircase\_factor = rcauchy(length(reps),1,SD\_ResponseFunction\*(1+ConditionOfInterest\*JND\_Difference)))}

We then use these multipliers ("**staircase\_facto**r") to compute the test stimulus strengths presented in the experiment ("**Presented\_TestStimulusStrength**"). Lastly, we compute the difference between test stimulus and standard stimulus for each trial ("**Difference**").

Psychometric = Psychometric %>%

mutate(

staircase\_factor = rcauchy(length(reps),1,SD\_ResponseFunction),

Presented\_TestStimulusStrength = Mean\*staircase\_factor,

Difference = Presented\_TestStimulusStrength - StandardValues)

Then, we compute the probability on each trial to judge the test stimulus intensity as higher (e. g. the test stimulus was faster, brighter, longer, ...) by feeding the simulated test stimulus strengths in a cummulative Gaussian with the mean and the standard deviations calculated above. We then use this value ("**AnswerProbability**") to simulate binary answers ("**Answer**") by drawing responses from a Bernoulli distribution. Figure 3 illustrates the stimulated data set for five subjects, where both PSE and JND differ between conditions.

Psychometric = Psychometric %>%

mutate(

AnswerProbability = pnorm(Presented\_TestStimulusStrength,Mean,SD),

Answer = as.numeric(rbernoulli(length(AnswerProbability),AnswerProbability))

)

As a next step, we bring the data into the format necessary for the glmer() function: We first remove extreme outliers (e.g., by a simple criterion such as excluding trials in which the difference between test and standard stimulus was higher than half the standard stimulus strength), which are likely occur to some extent when the cauchy function is used. Then, we compute the number of "Test stimulus intensity was higher" responses for each Condition and difference between test and comparison stimulus strength and the number of total observerations for each condition and difference in intensities.

Psychometric = Psychometric %>%

filter(abs(staircase\_factor-1) < 0.75) %>%

group\_by(ID,ConditionOfInterest,StandardValues,Difference) %>%

mutate(Yes = sum(Answer==1),

Total = length(ConditionOfInterest))

Now, we can inspect these psychometric functions visually to verify whether the values chosen above give rise to the expected psychometric functions in terms of PSE and slopes. We use the quickpsy package (Linares & López-Moliner, 2016) to fit the psychometric functions and plot them with the ggplot2 package.

PsychometricFunctions = quickpsy(Psychometric,Difference,Answer,grouping = .(ConditionOfInterest,ID,StandardValues), bootstrap = "none")

plot(PsychometricFunctions) +

scale\_color\_manual(name = "",

values = c(Red,BlauUB),

labels = c("Control","Experimental")) +

xlab("Difference between Comparison and Test") +

ylab("Probability to choose Test") +

geom\_vline(linetype = 2, xintercept = 0, color = "grey") +

geom\_hline(linetype = 2, yintercept = 0.5, color = "grey")

Figure 3 illustrates the simulated psychometric functions for the above values. The vertical lines indicate the PSE for each participant and stimulus strength. We can see that the PSEs for Condition of Interest: 1 are shifted towards the right. Furthermore, the curves for Condition of Interest: 1 are more shallow, indicating higher JNDs.

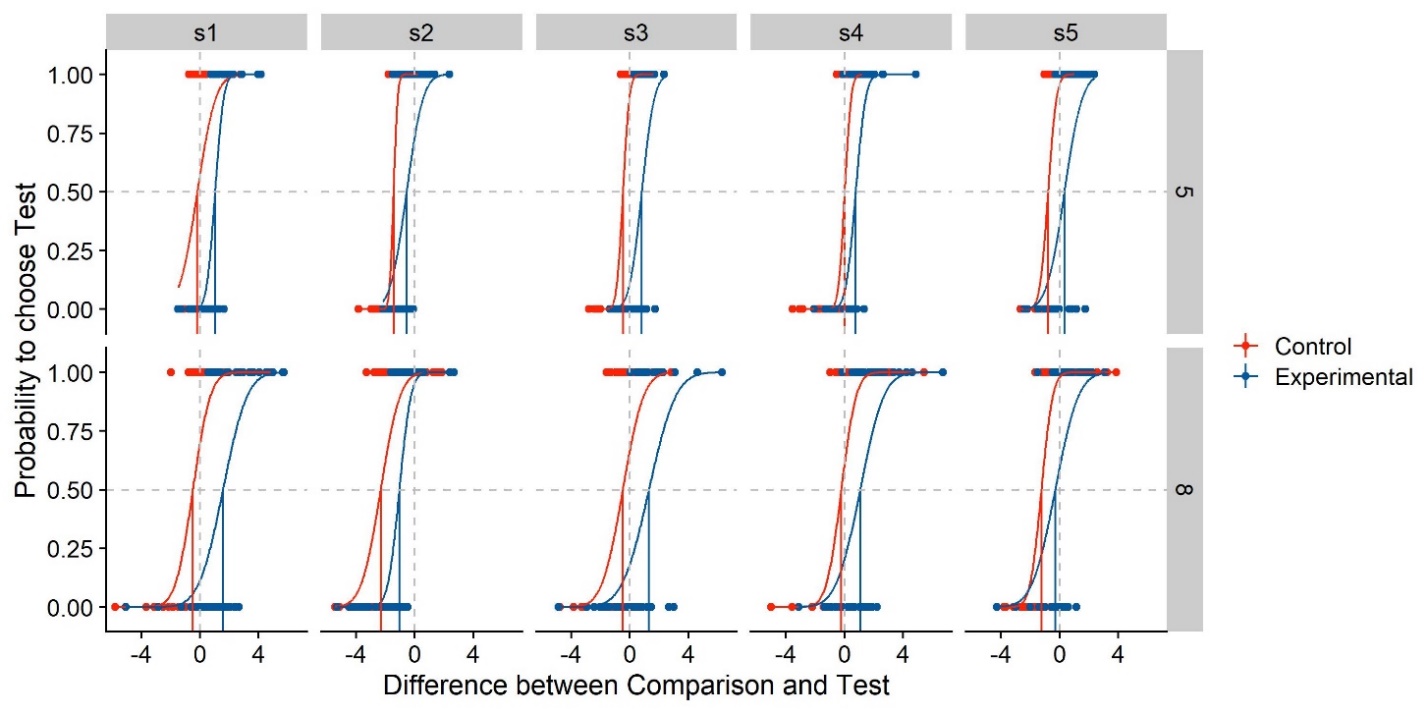


Figure : Simulated psychometric functions based on the example values chosen above. We plot the difference in stimulus intensity between test and standard stimulus (x axis) against the participants’ probability to choose the test stimulus as more intense (y axis). Different panels are the psychometric functions per participant (columns) and per standard stimulus intensity (rows). The psychometric functions are color-coded blue for the experimental Condition of Interest, and red for the control condition without manipulation. The red and blue vertical lines indicate the Points of Subjective Equality, while the vertical and horizontal grey dashed lines denote a difference between test and comparison of 0, and a probability of 0.5 to choose either stimulus. Their intersection thus indicates perfect accuracy, with a PSE of 0. The curves are cumulative Gaussians fitted to the data, while the dots indicate the answer (0 or 1) for each trial.

# Analyzing Psychophysical Data

In the introduction, we have briefly touched upon two different ways of applying Null Hypothesis Significance Testing to psychophysical data that are usually illustrated as psychometric functions:

* Fitting Psychometric functions, extracting their means and standard deviations as proxies for PSEs and JNDs and conducting the appropriate statistical test over these values. This is usually a *t* test or an ANOVA, but for more accurate results, one can use Linear Mixed Models, which we will discuss in the following. We will call this the “Two-Step Approach”.
* Following Moscatelli et al. (2012), one can use Generalized Linear Mixed Models (GLMM) to extract population parameters directly without the intermediate step of fitting psychometric functions. This approach should, according to the authors, lead to a higher power for detecting these effects. We will call this the “GLMM Approach”.

In the following, we will discuss the advantages and limitations of both methods and illustrate them with the data simulated above.

## Two-Step Approach

**How to** – The first step for Two-Step Approaches is the estimation of means and standard deviations of those psychometric functions with the best fits for the observed (or simulated) data. We can extract means and standard deviations from the quickpsy fit we used above:

Parameters = PsychometricFunctions$par

Parameters2 = Parameters %>%

filter(parn == "p1") %>%

select(ID,ConditionOfInterest,Mean=par, StandardValues)

Parameters2$SD = Parameters$par[Parameters$parn == "p2"]

Parameters = Parameters2

Then, we load the “lmerTest” package which computes p values for ANOVAs and includes these in the regular summary() output and conduct 2x2 ANOVAs (StandardValues x ConditionOfInterest) for both means and standard deviations.

require(lmerTest)

ANOVA\_Mean = lm(Mean ~ as.factor(ConditionOfInterest)\*StandardValues,Parameters)

ANOVA\_SD = lm(SD ~ StandardValues\*as.factor(ConditionOfInterest),Parameters)

Then, we can display the output of the ANOVAs by calling the summary() function on the fitted ANOVA objects:

summary(ANOVA\_Mean)

summary(ANOVA\_SD)

**Discussion** – This simple approach has three important pitfalls. First, statistical power is lost, especially when we have many trials for each condition, because this approach disregards that each PSE and standard deviation estimate is based on several tens of trials and treats them as one measurement each. In our example script, each staircase is measured in 100 trials. With the Two-Step Approach, each of these chunks of 100 trials are reduced to one mean and one standard deviation each, while there is no way to adjust for the added confidence in these values in a simple t test or ANOVA. Second, when participants and/or conditions contain different numbers of trials, this method still treats the means and standard deviations as equally reliable. In staircase procedures, participants or conditions tend to have unequal numbers of trials because, often, the staircases terminate once the step size has fallen under a certain threshold. Some participants may achieve this threshold substantially sooner or later than others. Therefore, each PSE and standard deviation is treated equally, despite one being more reliable than the other. Third, one of the assumptions of an ANOVA is the independence of the dependant variables. This assumption is often violated in these analyses: mean and standard deviation estimates from one participant are likely more similar to other mean and standard deviation estimates from the same participant than to estimates from other participants. This may not always be fatal, but a more careful modelling of the data is recommended. Mixed Effect Models allow to model these data structures more closely and provide a means to account for their partial dependence.

**Linear Mixed Models to the rescue** – Mixed Effect Models are an extension of Linear Regression Models. Just like regular regression models, they fit straight lines through the data. Additionally, they can account for particular properties of sub-populations of the data. Effects that are examined across the whole population are called “fixed effects”, while these sub-group properties are called “random effects”. LMMs can accommodate a whole range of diferent random effects. For example, in our case, LMM can account for the fact that the psychometric functions may be shifted consistently to the left or to the right (indicating higher or lower PSEs) for some participants, but not for others, or for that the fact that some participants are generally more sensitive than others, which in turn results in steeper psychometric functions and thus lower standard deviations/JNDs. In such cases of mean differences between groups, we speak of “random intercepts”. With the inclusion of “random slopes” we can accommodate for between-participant differences in the effect of ConditionOfInterest or StandardValues, i.e., that participant s01 may be affected differently by ConditionOfInterest than participant s02. There is no clear consensus on which random effects (random intercepts and/or random slopes) should be included and it generally regarded as situational decision that depends on domain knowledge about the data (Barr, Levy, Scheepers, & Tily, 2013; Bates, Mächler, Bolker, & Walker, 2015; Hodges, 2016; Matuschek, Kliegl, Vasishth, Baayen, & Bates, 2017). We recommend to include at least random intercepts per participant. Figure 4 illustrates the difference between a regular ANOVA and an LMM with random interceptions per participant graphically: a regular 2x4 ANOVA (2 Conditions x 4 Standard Values) does not distinguish between participants (see Figure 4A), which leads to a higher variability overall. A Linear Mixed Model with Condition and Standard Values and their interaction as fixed effects and random intercepts per participant (See Figure 4B) allocates some of the variability in the data to idiosyncrasies of each participant, which makes it easier to identify an overall consistent effect of ConditionOfInterest. The lme4 package (Bates et al., 2015) for R provides a user-friendly interface for the fitting of Linear Mixed Models. In lme4 syntax, the model specification looks like the following:

LMM\_Mean = lmer(Mean ~ ConditionOfInterest\*StandardValues + (1 | ID),

data = Parameters)

summary(LMM\_Mean)

LMM\_SD = lmer(SD ~ ConditionOfInterest\*StandardValues + (1 | ID),

data = Parameters)

summary(LMM\_SD)

This adjudicates some of the variability to differences between each participant and lowers the standard errors associated with the main effects, thus raising power. For example, for the data presented in Figure 4, an ANOVA (**ConditionOfInterest** x **StandardValues**) yields a regression coefficient of -0.4, an associated standard error of 2.38 and a p value of 0.86 for the effect of **ConditionOfInterest** on the PSE. A Linear Mixed Model that also adds random intercepts per participant (**LMM\_Mean**), in turn, yields a regression coefficient of -0.41 and an associated standard error of 0.47. When the lmerTest package (Kuznetsova, Brockhoff, & Christensen, 2017) is loaded, calling the summary() function on the LMM object also yields an p value, based on degrees of freedom approximated with the Sattherwaite method. We will discuss another method for significance testing, Likelihood Ratio Tests, below. For our synthetic dataset, this estimated p value is 0.38. The Mixed Model analysis thus provides the same coefficient estimate, but a much lower standard error and even a significant difference between **Baseline** and **ConditionOfInterest** that wasn’t captured by the ANOVA. It has to be noted that the coefficients are likely only the same across both methods because none of the participants shows an outlier-like behaviour. The coefficient estimate would be biased unduely by such outliers, while the LMM estimates should remain largely unaffected. As for the random effects, it yields intercepts of -0.06 for S01, 0 for S02, -0.1 for S03, 0.15 for S04 and 0.025 for S05, which – as expected – corresponds roughly to these participants average baseline performance (see Figure 4B). We find similar differences for the standard deviations: The Condition x StandardValues ANOVA yields a regression coefficient of 0.84, a standard error of 0.69 and a p value of 0.22 for the effect of ConditionOfInterest on the standard deviation. A Linear Mixed Model, in turn, returns a coeffcient of 0.84, a standard error of 0.46 and a p value of 0.08.

It has to be noted that a repeated measures ANOVA with participants as grouping factor is roughly equivalent to the Linear Mixed Model (LMM) established above. However, Mixed Models have some advantages with regards to repeated measures ANOVAs: There are fewer issues with missing data and they are more flexible. For example, they allow the inclusion of more than one grouping factor, and more than one random effect per grouping factor, which can come in handy. We therefore recommend using Linear Mixed Models over repeated measures ANOVAs. For further information and an R-based introduction to Linear Mixed Models, please see Brown’s guide (Brown, 2020).

In fact, the specification of the Mixed Model requires further discussion. Since we are generally not interested in the population-wide effect of **StandardValues**, it might be more appropriate to include them as random effect grouping factor, for example with random intercepts.

LMM\_Mean = lmer(Mean ~ ConditionOfInterest + (1 | ID) + (1 | StandardValues),

data = Parameters)

LMM\_SD = lmer(SD ~ ConditionOfInterest + (1 | ID) + (1 | StandardValues),

data = Parameters)

summary(LMM\_Mean)

summary(LMM\_SD)

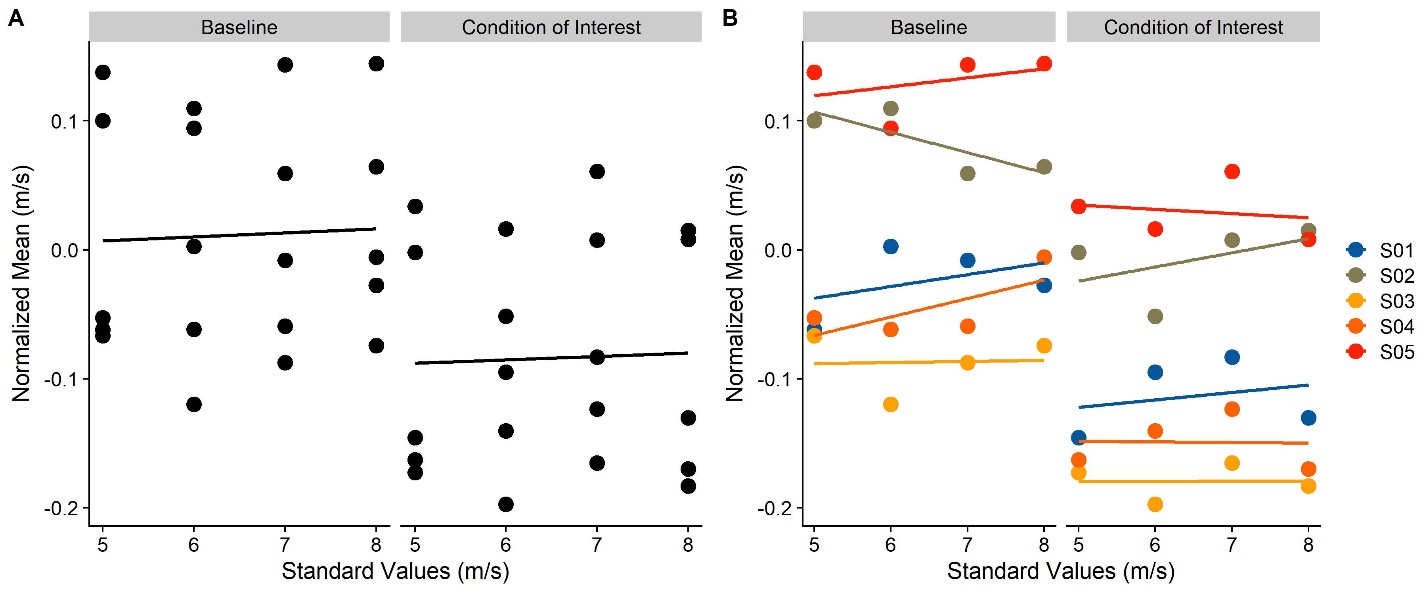


Figure : Means of simulated psychometric functions for five subjects (S01, S02, S03, S04, S05) and four different standard values (5, 6, 7, 8 m/s), for the baseline stimulus (left panel) and the condition of interest (right panel). In A, different participants are not color coded, mimicking the statistical behavior of a regular ANOVA. In B, the participants are color-coded, mimicking the statistical behavior of a Linear Mixed Model with random intercepts per participants or a Repeated Measures ANOVA.

We recommend (Brown, 2020) as an approachable, yet thorough introduction to Mixed Effect Models in R. However, when using this more sophisticated version of the Two-Step approach, we might still lose power by disregarding that each PSE and JND estimate is based on a large number of trials (e.g., 100 in our case). Moscatelli et al. (Moscatelli et al., 2012) have suggested the use of Generalized Linear Mixed Models to avoid this loss of power.

## Generalized Linear Mixed Modelling (GLMM)

**How to** – We recommend Moscatelli et al.’s paper (Moscatelli et al., 2012) for a more thorough discussion of the specifics of this method, both in terms of rationale and process. In this manuscript, we will only outline the basic idea. Our goal is to extract population coefficient estimates directly from the proportional or binomial data obtained in the experiment (i.e., the proportion of “Test Stimulus more intense” responses for each presented stimulus strength). Linear Mixed Models, however, (a) require continuous dependent variables and (b) can only fit straight lines. (a) is problematic because responses from psychophysical paradigms are generally either proportional or binomial. (b) is problematic because it is generally assumed that performance in such experiments is depicted accurately by psychometric functions, which are typically non-linear functions like Weibull, Cummulative Gaussian or logistic functions. Generalized Linear Mixed Modelling circumvents these two problems by introducing a so called “link function” that transforms proportional/binomial data with a non-linear relation between independent and dependent variable(s) such that they can be captured by a linear fit. The relevant link functions that lme4 offers are “probit” and “logit” links. Both allow to transform the typical psychometric function types, Weibull, Cummulative Gaussian or logistic functions, into the linear space. We follow Moscatelli et al. in choosing a probit link, but a logit link yields practically the same results.

We first filter for potentially presented outlier values; e.g., two mistakes early in a staircase can lead to the presentation of extreme values that might influence the analysis unduely. Then, we bring the data into a suitable format for the glmer() function from the lme4 package (Bates et al., 2015): it requires the responses for each stimulus value to be represented as a fraction of “Yes” responses and the total number of trials where this stimulus strength was presented.

Psychometric = Psychometric %>%

filter(abs(staircase\_factor-1) < 0.75) %>% group\_by(ID,ConditionOfInterest,StandardValues,Difference) %>%

mutate(Yes = sum(Answer==1),

Total = length(ConditionOfInterest))

We can then proceed to fitting the Generalized Mixed Model. Here, we have to make several decisions with regards to the specification of the model, particularly about the random effects. Moscatelli et al. (Moscatelli et al., 2012) use two different models in their examples: A model with ConditionOfInterest, Difference and their interaction as fixed effects, and random intercepts and random slopes for Difference per participant, and the same model, but with only random intercepts per participant. These are specified as follows in lme4 syntax:

GLMM\_RandomIntercepts\_JND = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest\*Difference + (1| ID),

family = binomial(link = "probit"),

data = Psychometric)

GLMM2\_RandomInterceptsAndSlopes\_JND = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest\*Difference + (1 + Difference| ID),

family = binomial(link = "probit"),

data = Psychometric)

summary(RandomIntercepts)

summary(RandomInterceptsAndSlopes)

**Difference** indicates to what extent the proportion of responses changes in response to the difference in stimulus strength between stimulus 1 and stimulus 2. The coefficient for this variable is thus proportional to the standard deviation of the psychometric function and the JND. **ConditionOfInterest** indicates to what extent the whole psychometric function is shifted to the left or to the right. The coefficient for **ConditionOfInterest** thus corresponds to the extent to which the presence of the manipulation shifts the mean of the psychometric function, that is, the PSE. The interaction between **ConditionofInterest** and **Difference** indicates to what extent the manipulation **ConditionOfInterest** changes the coefficient for Difference. This corresponds to the influence of the manipulation on the standard deviation of the psychometric function, and with that, its JND. The random effects (Intercepts per **ID** in the first model, and Intercepts and Slopes for **Difference** per **ID** in the second model) allow to account for individual differences in PSE and JND per participant.

When the lmerTest package (Kuznetsova et al., 2017) is loaded, calling the summary() function on the GLMER object automatically provides p values for each fixed effect coefficient; in this case, for the Intercept, **ConditionOfInterest**, **Difference** and their interaction. These p values are based on degrees of freedom approximated with the Sattherwaite method, which is generally considered a computationally cheap, reliable-enough, but imperfect procedure (Gaylor & Hopper, 1969). An alternative, which can be more accurate, are Likelihood Ratio Tests (Luke, 2017). A Likelihood Ratio Test compares a test model to the next simpler model that doesn’t include the variable of interest in terms of their respective log-likelihoods, with a penalty for the additional parameter in the test model. The p value is then based on a model fit criterion. If we want to test for the impact of ConditionOfInterest on the JND, this null model would be a model that is equal to the test model, but lacks the interaction of ConditionOfInterest and Difference. The lme4 syntax would be:

GLMM\_RandomIntercepts\_Null\_JND = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest + Difference + (1| ID),

family = binomial(link = "probit"),

data = Psychometric)

GLMM2\_RandomInterceptsAndSlopes\_Null\_JND = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest + Difference + (Difference| ID),

family = binomial(link = "probit"),

data = Psychometric)

The Likelihood Ratio Test is implemented in the anova() function from base R.

anova(GLMM\_RandomIntercepts\_JND,GLMM\_RandomIntercepts\_Null\_JND)

anova(GLMM2\_RandomInterceptsAndSlopes\_JND,GLMM2\_RandomInterceptsAndSlopes\_Null\_JND)

This Likelihood Ratio Test provides a p value which allows to judge whether the test model is significantly better than the null model. If it is, this is evidence that **ConditionOfInterest** has a significant influence on the slope of the psychometric function, that is, on the JND.

Testing for PSE differences requires a slightly different approach in model specification. Since Likelihood Ratio Testing requires isolating the effect to be tested in the test model, in order to drop it in the null model, the interaction between the fixed effects **ConditionOfInterest** and **Difference** can’t be included:

GLMM\_RandomIntercepts\_PSE = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest + Difference + (1| ID),

family = binomial(link = "probit"),

data = Psychometric)

GLMM2\_RandomInterceptsAndSlopes\_PSE = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest + Difference + (Difference| ID),

family = binomial(link = "probit"),

data = Psychometric)

The corresponding null models should be specified as follows:

GLMM\_RandomIntercepts\_Null\_PSE = glmer(cbind(Yes, Total - Yes) ~ Difference + (1| ID),

family = binomial(link = "probit"),

data = Psychometric)

GLMM2\_RandomInterceptsAndSlopes\_Null\_PSE = glmer(cbind(Yes, Total - Yes) ~ Difference + (Difference| ID),

family = binomial(link = "probit"),

data = Psychometric)

And finally, a Likelihood Ratio Test is performed over both models, which yields a p value.

anova(GLMM\_RandomIntercepts\_PSE,GLMM\_RandomIntercepts\_Null\_PSE)

anova(GLMM2\_RandomInterceptsAndSlopes\_PSE,GLMM2\_RandomInterceptsAndSlopes\_Null\_PSE)

A challenge when using Generalized Linear Mixed Models is the model specification, in this case particularly the decision about the random effects to be included into the model. Moscatelli et al.’s (Moscatelli et al., 2012) analyses suggest that including (a) only random intercepts per participant or (b) both random intercepts and random slopes for Difference per participant doesn’t lead to vastly different results. We can test this also for our simulated dataset:

anova(GLMM\_RandomIntercepts\_JND,GLMM2\_RandomInterceptsAndSlopes\_JND)

anova(GLMM\_RandomIntercepts\_PSE,GLMM2\_RandomInterceptsAndSlopes\_PSE)

While the model fit was superior for (b), the fixed effect coefficient estimates were roughly equal across both options. In our simulated dataset, a Likelihood Ratio Test yielded no significant difference between (a) and (b), for neither of the models specified above, in which case the model with fewer parameters (**GLMM\_RandomIntercepts\_JND** and **GLMM\_RandomIntercepts\_PSE**, respectively), is preferrable.

### A deep dive into model specifications

A further method to determine the correct model specification is to study to what extent each meets the assumptions of Generalized Linear Mixed Models: Homoskedasticity, and roughly uniformly distributed residuals. *Homoskedasticity* refers to the postulate that variance in responses should not depend in any lawful way on any experimental variables. For Linear Models, homoskedasticity is generally tested for by visual examination of plots where different conditions are plotted against the corresponding residuals. A statistical test for difference in variances, such as Levene’s test (Levene, 1960), Bartlett’s test (Bartlett, 1937) or the Brown-Forsythe test (Brown & Forsythe, 1974) can be applied, but visual inspection is usually considered sufficient. However, it is near impossible to visually ascertain heteroskedasticity from the raw residuals of Generalized Linear (Mixed) Models, as the expected pattern in the residuals depends on the fitted values. The R package DHARMa (Hartig, 2020) implements a method to standardize residuals for Generalized Linear Mixed Models (Dunn & Smyth, 1996; Gelman & Hill, 2008). These standardized residuals allow for visual analysis just like residuals of Linear Models; for a well specified model, one would expect these standardized residuals to be uniformly distributed. To judge whether heteroscedasticity might be a problem in the analysis of psychophysical data with GLMMs, we use the above procedure to produce extreme data, with StandardValues of 5, 10 and 50 m/s, and (high) inter-participant variability both for PSE and JND of 0.5, and plot the DHARMa-standardized residuals against each of the sub-groups in our data (ConditionOfInterest, StandardValues, ID; see Figure 5), for the simplest reasonable model GLMM\_RandomIntercepts\_JND and for the most complex reasonable model, including random intercepts and slopes for **Difference** and **ConditionOfInterest** both for **ID** and **StandardValues** (**GLMM2\_ThreeRandomEffectsPerIDAndStandardValues**, see below).

require(DHARMa)

Sim\_Simple = simulateResiduals(GLMM\_RandomIntercepts\_JND)

GLMM2\_ThreeRandomEffectsPerIDAndStandardValues = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest\*Difference +

(Difference + ConditionOfInterest| ID) +

(Difference + ConditionOfInterest| StandardValues),

family = binomial(link = "probit"),

data = Psychometric)

Sim\_Complex = simulateResiduals(GLMM\_RandomIntercepts\_JND)

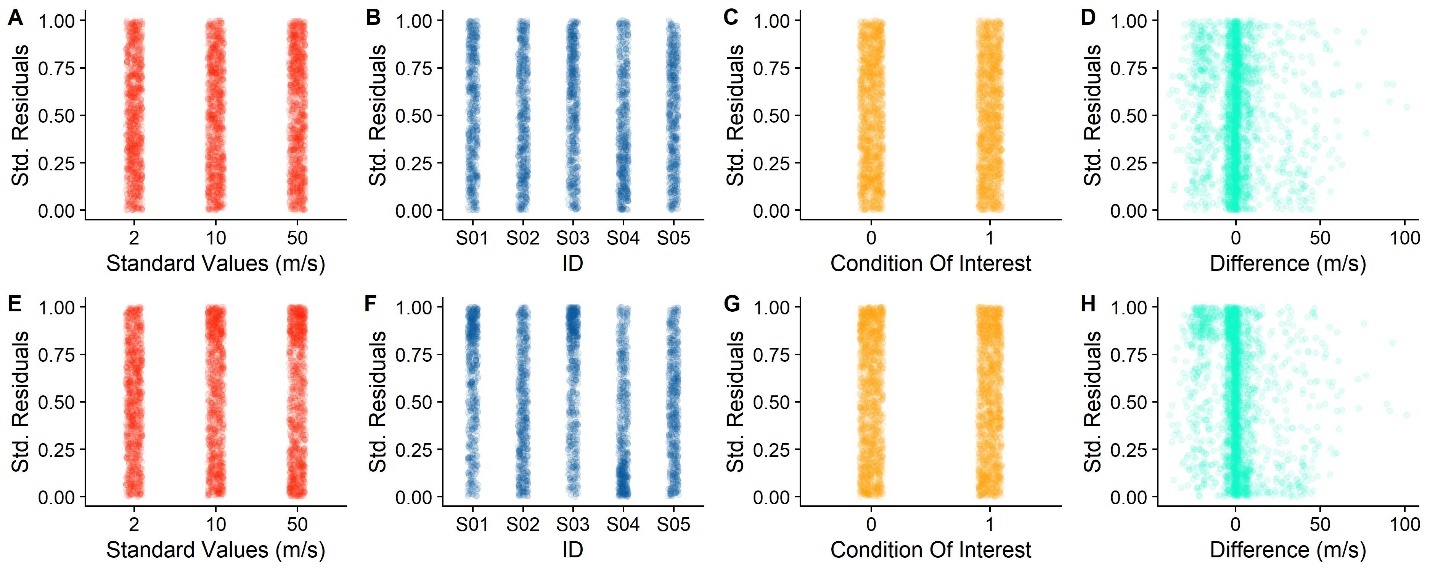


Figure . A, B, C, D. Standardized Residuals for the simple model, divided up by Standard Values (A), participant (B), Baseline versus Condition of Interest (C) and Differences (D). E, F, G, H. Same, but for the more complex model.

As evident from Figure 5, there are no notable patterns for any of factors. For the continuous variable Difference, there seems to be a slight pattern of negative values being connected to higher residuals. This pattern, however, is quite weak for the simplest model, and, contrary to the expectancy that a more complete model should reduce residual patterns, this pattern is stronger for the more complex model. However, the variability in residuals seems to be evenly distributed across **StandardValues**, **ID** and **ConditionOfInterest**.

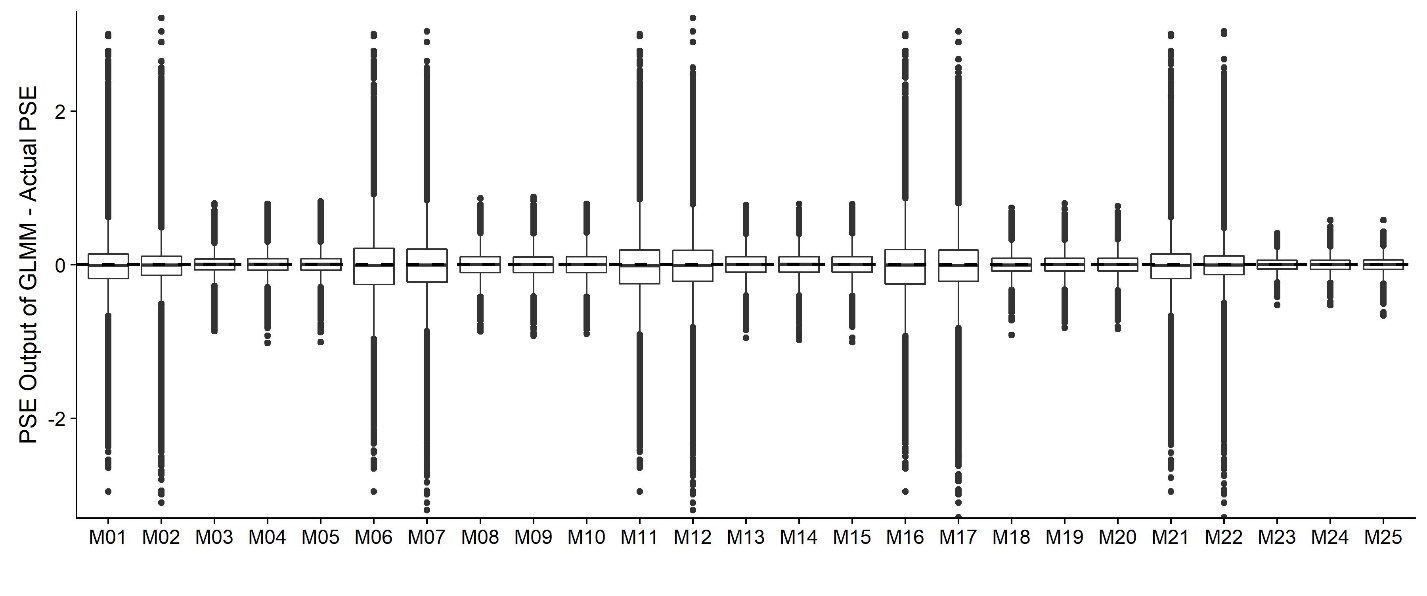
Linear Models can furthermore be unreliable if the underlying formula is not specified correctly. This can be detected by plotting the values predicted by the models against the observed values, as a quantile-quantile (QQ) plot. Alternatively, the residuals should present no apreciable patterns. Statistical tests, like a Kolmogorov-Smirnoff test (Massey, 1951), can be performed to detect whether the predicted values match the observed values closely enough, but are not generally deemed necessary. To verify whether this criterion is satisfied by some model specifications, but not by others, we simulate five average datasets with 10 participants each (using the above procedure), analyze them with the 25 reasonable model specifications described in Table 1 and verify visually for each if the residuals are distributed uniformly. For computation and visualization of the standardized residuals, we again rely on the DHARMa package (Hartig, 2020). You can find residuals plots in the supplementary materials.

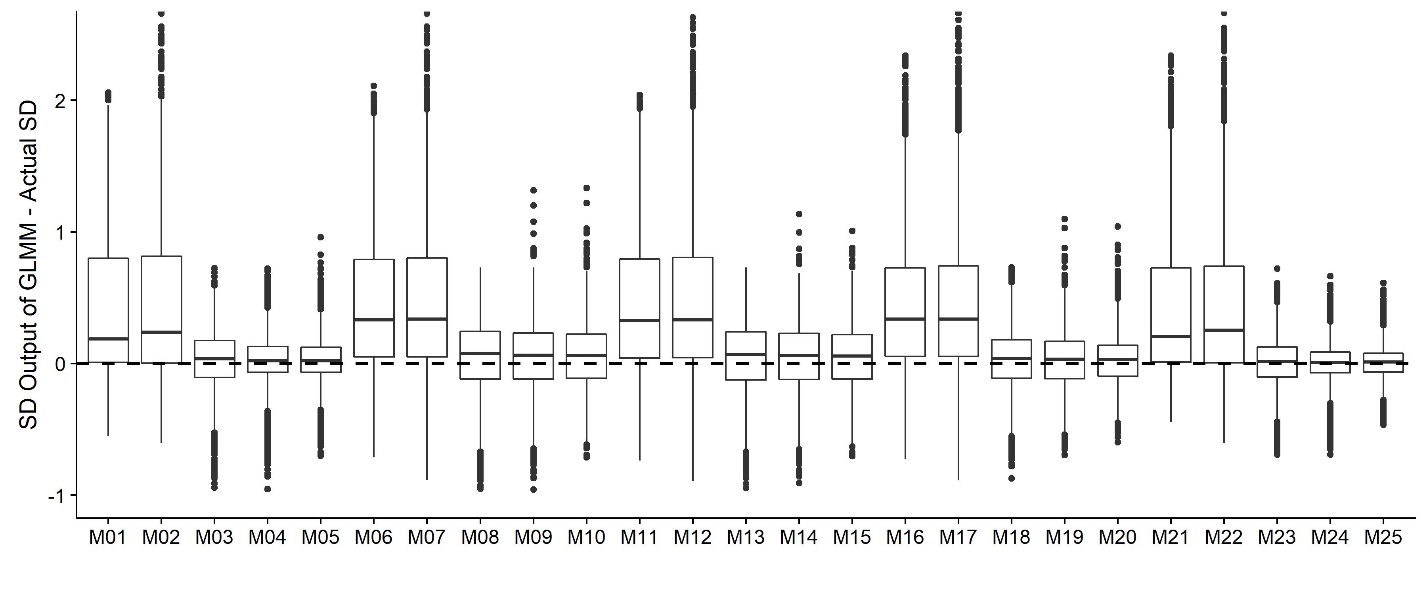
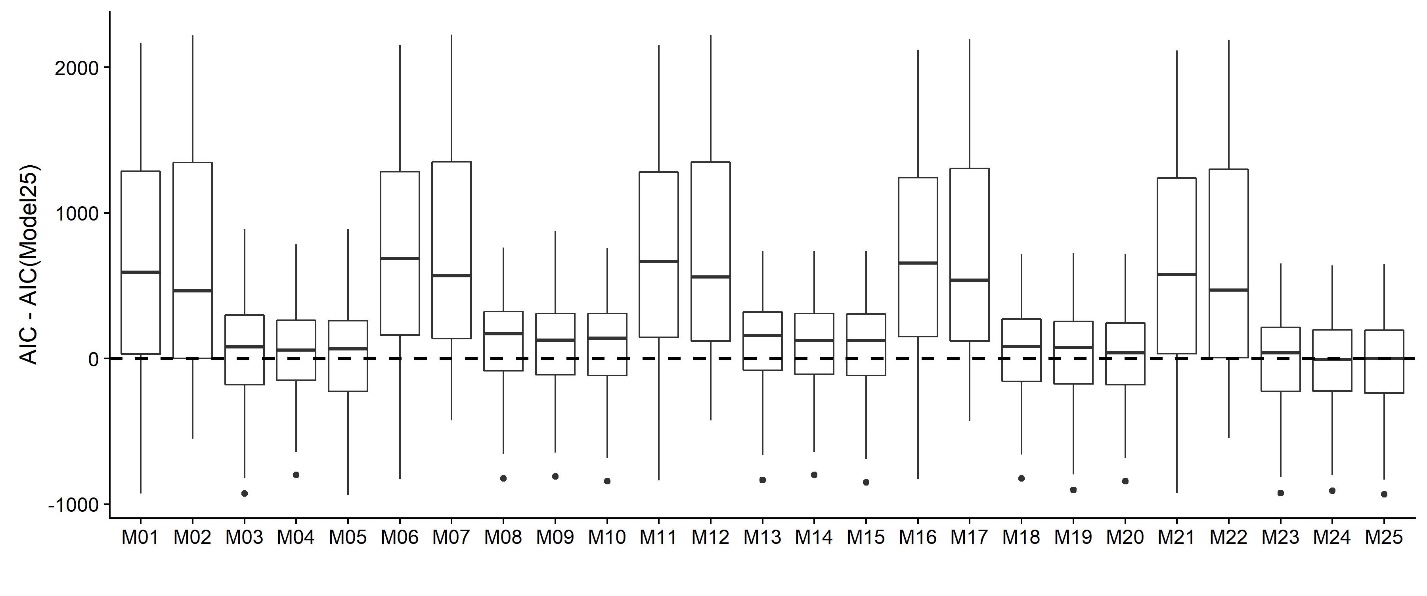
Furthermore, and maybe most importantly, we can verify to what extent each model specification recovers the correct population parameters. Since we simulate the datasets, we know its underlying properties and can compare them to the output of each model specification. To this end, we simulate 200 datasets and analyze them with each of the 25 model specifications in Table 1. We then extract the most relevant features (PSEs, standard deviations of the psychometric functions, AICs and p values) and compare them to the values choosen for the simulations. Figure XX illustrates the results of this process: on average, all models recover the correct PSE. However, the models M01, M02, M06, M07, M11, M12, M16, M17, M21 and M22 display vastly higher variability in recovered PSEs than the other models. These models have no random effects or only the intercept as random effect per ID, and lack both Difference and ConditionOfInterest as random effects per ID. The absence of random effects per StandardValues seems to have a very limited effect on the reliability of the PSE estimates. As for the estimates of the standard deviation of the psychometric functions, leaving out Difference and ConditionOfInterest as random effects per ID leads both to a decrease in reliability and biases. We can furthermore compare the AICs for each model as an indicator of model fit, as depicted in Figure XX. We find a similar picture: models that have Difference and/or ConditionOfInterest as random effects per ID yield the best model fits, while any random effects per StandardValues seem to yield marginal gains in model fit.

XXXXX finalize stuff INTERPRETATION

|  |  |  |
| --- | --- | --- |
| *Model name* | **Random Effects** | |
| **Grouping Variable: StandardValues** | **Grouping Variable: ID** |
| *M01* | - | - |
| *M02* | - | Intercept |
| *M03* | - | Intercept, Difference |
| *M04* | - | Intercept, ConditionOfInterest |
| *M05* | - | Intercept, Difference, ConditionOfInterest |
| *M06* | Intercept | - |
| *M07* | Intercept | Intercept |
| *M08* | Intercept, Difference | Intercept, Difference |
| *M09* | Intercept, ConditionOfInterest | Intercept, ConditionOfInterest |
| *M10* | Intercept, ConditionOfInterest | Intercept, Difference, ConditionOfInterest |
| *M11* | Intercept, Difference | - |
| *M12* | Intercept, Difference | Intercept |
| *M13* | Intercept, Difference | Intercept, Difference |
| *M14* | Intercept, Difference | Intercept, ConditionOfInterest |
| *M15* | Intercept, Difference | Intercept, Difference, ConditionOfInterest |
| *M16* | Intercept, ConditionOfInterest | - |
| *M17* | Intercept, ConditionOfInterest | Intercept |
| *M18* | Intercept, ConditionOfInterest | Intercept, Difference |
| *M19* | Intercept, ConditionOfInterest | Intercept, ConditionOfInterest |
| *M20* | Intercept, ConditionOfInterest | Intercept, Difference, ConditionOfInterest |
| *M21* | Intercept, Difference, ConditionOfInterest | - |
| *M22* | Intercept, Difference, ConditionOfInterest | Intercept |
| *M23* | Intercept, Difference, ConditionOfInterest | Intercept, Difference |
| *M24* | Intercept, Difference, ConditionOfInterest | Intercept, ConditionOfInterest |
| *M25* | Intercept, Difference, ConditionOfInterest | Intercept, Difference, ConditionOfInterest |

Table : Specification of different models. The first column indicates the model names, the second column indicates the random effects per StandardValues and the third column shows the random effects per ID.





**Conclusions** – INTERPRETATION

## Bayesian Linear Mixed Modelling

Finally, there are Bayesian methods of estimating JND and PSE that rely on Monte Carlo Markov Chain (MCMC) modelling. Particularly the brms package (Bürkner, 2018) for R is an interesting alternative that is due to its similar syntax very user-friendly for lme4 users. brms offers an interface for the package rstan (Stan Development Team, 2016), which is based on the probabilistic programming language Stan. The usage of brms is very similar to lme4, but users are advised to be familiar with Bayesian statistics before analyzing data with brms. The brms documentation provides an excellent starting point (https://paul-buerkner.github.io/brms/articles/brms\_distreg.html), especially if you already have some general knowledge about Bayesian analysis, but it focusses on applying the package correctly and is therefore by no means complete. For a more exhaustive reading, we recommend, for example, Spiegelhalter & Reis’ book on Bayesian Statistics (Spiegelhalter & Rice, 2009).

To fit a Bayesian Generalized Linear Mixed Model that resembles the ones discussed above in structure, we first install and load the packages rstan (Stan Development Team, 2016) and brms (Bürkner, 2018), and optimize the system for the fitting of Bayesian GLMMs.

require(brms)

require(rstan)

options(mc.cores = parallel::detectCores())

rstan\_options(auto\_write = TRUE)

Sys.setenv(LOCAL\_CPPFLAGS = '-march=corei7')

brms allows Bayesian analysis based on the same principles as the GLMM approach described above. We can fit a model with ConditionOfInterest, Difference and their interaction as main effects, and random intercepts and slopes for Difference per participant ID and StandardValues as random effects. Please note that brms requires “bernoulli()” as family, while the command for lme4 is “binomial(link = “probit”)” or “binomial(link = “logit”)”. This is roughly equivalent for our purposes.

BayesianGLMM = brm(bf(Yes ~ ConditionOfInterest\*Difference + (ConditionOfInterest + Difference | ID) + (ConditionOfInterest + Difference | StandardValues)),

data = Psychometric,

family = bernoulli())

We can call the fitted object and examine the results.

summary(BayesianGLMM)

Finally, we can use the hypothesis() function to test which of two mutually exclusive hypotheses the data support and to what extent; for example whether the coefficient for ConditionOfInterest is larger than zero:

hypothesis(BayesianGLMM,c("ConditionOfInterest > 0"))

While Monte Carlo Markov-Chain modelling, the principle on which brms is based, is often considered the most accurate means to estimate parameters (Bolker et al., 2009), it also comes with a Bayesian flavour that requires a different lense for interpretation and may thus not be suitable for everyone. Nonetheless, we recommend that readers keep this option in mind.

# Conducting Power Analyses for Psychophysical Data

To simulate the power with a given set of parameters, we need to execute the above procedure sufficient times (we recommend at least 1000 times, although this might be too time consuming in R for studies with a high count of subjects and/or trials; we will discuss a potentially faster Julia implementation below), and calculate the ratio of simulations in which the test model is significantly better than the test model, given a certain false positive rate (typically 0.05). To this end, we establish a function to simulate representative datasets according to the above procedure.

SimulatePsychometricData = function(nParticipants,

ConditionOfInterest,

StandardValues,

reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

Type\_ResponseFunction,

SD\_ResponseFunction,

Mean\_Variability\_Between,

SD\_Variability\_Between){

ID = paste0("S0",1:nParticipants)

Psychometric = expand.grid(ID=ID, ConditionOfInterest=ConditionOfInterest, StandardValues=StandardValues, reps = 1:reps)

Psychometric = Psychometric %>%

group\_by(ID) %>%#

mutate(PSE\_Factor\_ID = rnorm(1,1,Mean\_Variability\_Between), #how much variability is in the means of the psychometric functions between subjects?

SD\_Factor\_ID = rnorm(1,1,SD\_Variability\_Between)) #how much variability is in the standard deviations of the psychometric functions between subjects?

Psychometric = Psychometric %>%

mutate(

Mean\_Standard = StandardValues+StandardValues\*Multiplicator\_PSE\_Standard,

SD\_Standard = StandardValues\*Multiplicator\_SD\_Standard,

Mean = (Mean\_Standard + (ConditionOfInterest==1)\*Mean\_Standard\*PSE\_Difference),

SD = abs(SD\_Standard + (ConditionOfInterest==1)\*SD\_Standard\*JND\_Difference))

Psychometric = Psychometric %>%

mutate(

Mean = Mean\*PSE\_Factor\_ID,

SD = SD\*SD\_Factor\_ID)

if (Type\_ResponseFunction == "normal"){

Psychometric = Psychometric %>%

mutate(

staircase\_factor = rnorm(length(reps),1,SD\_ResponseFunction\*(1+ConditionOfInterest\*JND\_Difference)))

} else if (Type\_ResponseFunction == "Cauchy"){

Psychometric = Psychometric %>%

mutate(

staircase\_factor = rcauchy(length(reps),1,SD\_ResponseFunction\*(1+ConditionOfInterest\*JND\_Difference)))

} else{

print("distribution not valid")

}

Psychometric = Psychometric %>%

mutate(Presented\_TestStimulusStrength = Mean\*staircase\_factor,

Difference = Presented\_TestStimulusStrength - StandardValues)

Psychometric = Psychometric %>%

mutate(

AnswerProbability = pnorm(Presented\_TestStimulusStrength,Mean,SD),

Answer = as.numeric(rbernoulli(length(AnswerProbability),AnswerProbability))

)

Psychometric = Psychometric %>%

filter(abs(staircase\_factor-1) < 0.75) %>%

group\_by(ID,ConditionOfInterest,StandardValues,Difference) %>%

mutate(Yes = sum(Answer==1),

Total = length(ConditionOfInterest))

Psychometric

}

Then we set the relevant parameters. RangeNs is a vector with the number of participants for which we want to simulate the power, and RangeRepetitions is a vector with the number of repetitions per staircase for which we want to simulate the power. Varying both allows us to find an optimal tradeoff between trials per participant and number of participants.

RangeNs = c(10,12,14,16,18,20)

RangeRepetitions = c(40,70,100)

ConditionOfInterest = c(0,1)

StandardValues = c(5,6,7,8)

PSE\_Difference = 0.1

JND\_Difference = 0.25

Multiplicator\_PSE\_Standard = 0

Multiplicator\_SD\_Standard = 0.15

Type\_ResponseFunction = "normal"

SD\_ResponseFunction = 0.1

Mean\_Variability\_Between = 0.2

SD\_Variability\_Between = 0.2

We then choose a number of iterations per combination of number of repetitions and number of participants. We recommend a high number, but even with a relatively low number of 200 iterations, this process can take several hours.

nIterations = 200

Then, we simulate the data once for each number of participants, number of trials and iterations, fit a GLMM for each and save the p values for the influence of ConditionOfInterest on PSEs and JNDs in a dataframe. We choose “**nAGQ = 1**” and “**glmerControl(optimizer = "nloptwrap"))**” in the glmer() function because these settings make for reliable p values, while a reasonable fitting duration. We will discuss these option in more detail below when comparing different optimizers. We also add in a timer to keep you updated about how far through the power simulations you are.

PowerfulDataframe = data.frame()

for (nParticipants in RangeNs){

for (reps in RangeRepetitions){

TimeStartTrial = Sys.time() #get time at beginning of trial

for(i in 1:nIterations){

print(nParticipants)

print(reps)

print(nIterations)

Psychometric = SimulatePsychometricData(nParticipants,

ConditionOfInterest,

StandardValues,

reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

Type\_ResponseFunction,

SD\_ResponseFunction,

Mean\_Variability\_Between,

SD\_Variability\_Between)

GLMM = glmer(cbind(Yes, Total - Yes) ~ ConditionOfInterest\*Difference + (ConditionOfInterest+Difference| ID) + (ConditionOfInterest+Difference| StandardValues),

family = binomial(link = "logit"),

data = Psychometric,

nAGQ = 1,

glmerControl(optimizer = "nloptwrap"))

print(RDuration\_nloptwrap\_nAGQ0)

summary(GLMM)$coefficients[14]

summary(GLMM)$coefficients[16]

PowerfulDataframe = rbind(PowerfulDataframe,c(nParticipants=nParticipants,

reps=reps,

pvalue\_PSE = summary(GLMM)$coefficients[14],

pvalue\_JND = summary(GLMM)$coefficients[16],

iteration = i))

}

print(paste0("200 iterations took ", round(Sys.time() - TimeStartTrial), " seconds. The power for the current run through (",nParticipants," Participants, ", reps, " Repetitions) is ",mean(PowerfulDataframe$pvalue\_PSE[PowerfulDataframe$nParticipants == nParticipants & PowerfulDataframe$reps == reps] < 0.05)))

}

}

colnames(PowerfulDataframe) = c("nParticipants","reps","pvalue\_PSE","pvalue\_JND","iteration")

Finally, we can print the power for each combination of number of repetitions and number of participants by setting an alpha level (typically 0.05) and counting in how many iterations out of nIterations we found a pvalue below alpha.

alpha = 0.05

PowerfulDataframe = PowerfulDataframe %>% group\_by(nParticipants,reps) %>%

mutate(Power\_PSE = mean(pvalue\_PSE < alpha),

Power\_JND = mean(pvalue\_JND < alpha))

PowerfulDataframe %>% group\_by(nParticipants,reps) %>%

slice(1)

We can also plot the computed power with the following lines.

ggplot(PowerfulDataframe, aes(reps,Power\_PSE, color = as.factor(nParticipants))) +

geom\_line() +

xlab("Number of trials per staircase") +

ylab("Power") +

scale\_color\_discrete(name = "Number of\nParticipants")

Figure XXXXX shows this plot for the values chosen above.

# Comparing the power for the GLMM and the Two-Level approach

Following Moscatelli & Lacquaniti (2012), we have argued above that power is lost when using the Two-Step approach. While this is an intuitive notion, it has, to our knowledge, only been confirmed for PSEs, and only for one combination of dataset parameters. In the following, we thus use the above procedure to assess just how much power is lost when using the Two-Step Approach in comparison to the GLMM Approach, for the influence of **ConditionOfInterest** on both PSEs and JNDs.

We use the same parameters as above, with 10, 12, 14, 16, 18, 20 participants, and 40, 70 and 100 trials per staircase. We furthermore compare different effect sizes, with JND differences between baseline and Condition of Interest of -10, -5, 0, 5 and 10% and PSE difference of -5, -2.5, 0, 2.5 and 5%. We furthermore include different ways of analyzing the data: Two-Step analyses using simple ANOVAs, the more sophisticated Linear Mixed Modelling approach to Two-Step analyses, and two different GLMM specifications; a full GLMM specification (corresponding to Model25 in Table 1) and a reduced GLMM specification (Model13 in Table 1).

INTERPRETATION of results XXX

[POWER FIGURE]

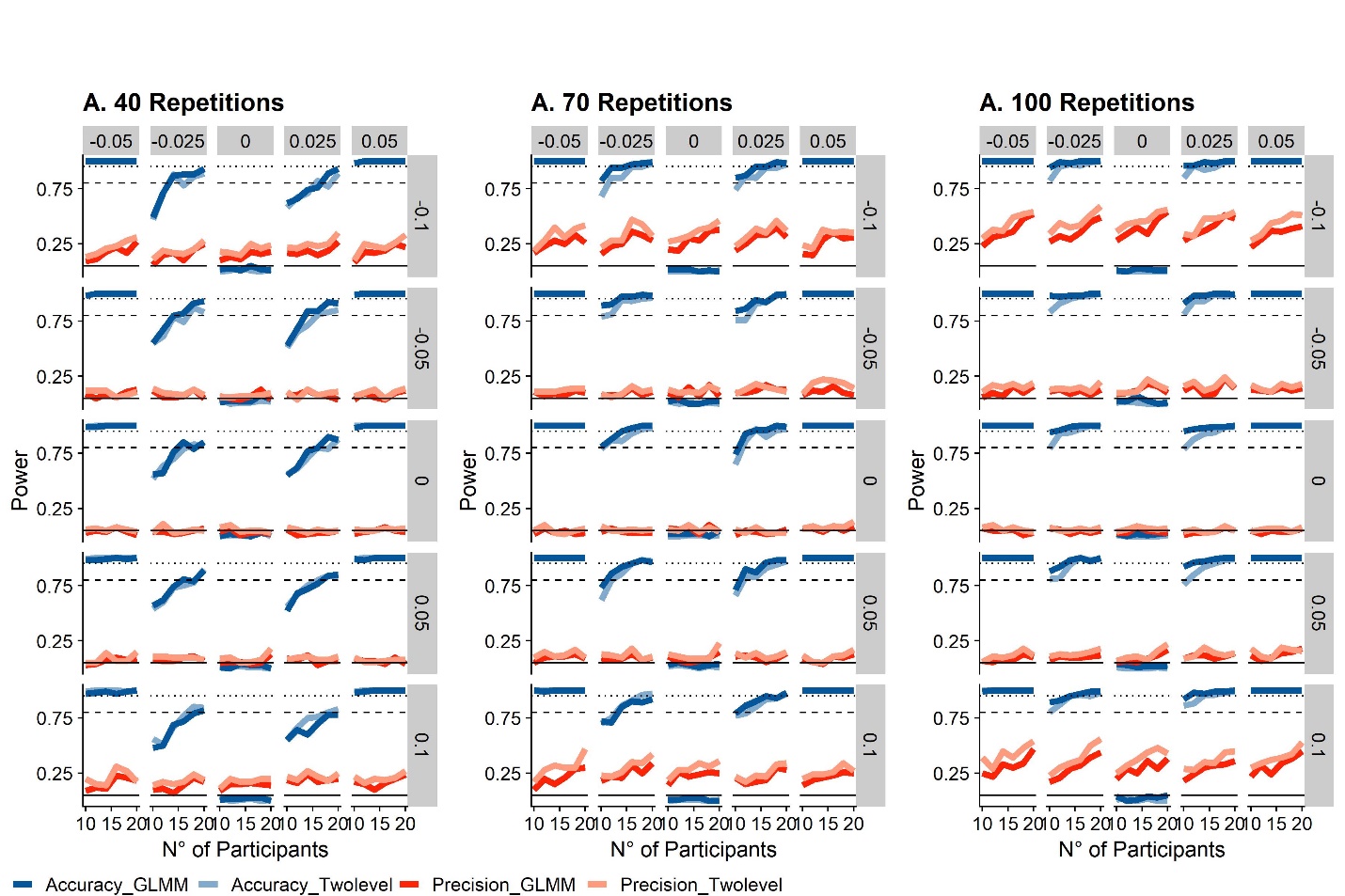


Figure : The solid lines indicate the power level across the range of participant numbers, red for accuracy and blue for precision. The panels represent different number of repetitions per condition. The intermittent horizontal lines indicate power levels of 0.8 (bare minimum), 0.9 (acceptable) and 0.95 (quite good).

## Power Analyses in Julia

The relatively new programming language Julia is advertised as a up-and-coming, faster alternative to R. Developped with a focus on speed, it can achieve a performance similar to C, while R is routinely among the slowest languages in benchmarks. Fortunately, packages for the fitting of (Generalized) Mixed Models are already available for Julia. It thus provides a means to speed up the simulation process, which, as mentioned above, can take one to several hours in R. Julia is also relatively intuitive and user-friendly for usar with R experience and offers the ability to call R code from within Julia. This enables us to generate the datasets in R, export them and conduct the time-consuming fitting of the GLMMs in Julia. For the present paper, we expect readers to have already installed Julia. Short instructions for installing Julia and the necessary packages can be found here.

After calling the required packages (Pkg, MixedModels, RCall, Dataframes, CSV, RData, CategoricalArrays, Statistics and Dates), we use the packages RCall to call the R function we used above to simulate datasets.

using Pkg

Pkg.activate()

Pkg.instantiate()

using MixedModels

using RCall

using DataFrames, Tables

using Random

using CSV

using RData

using CategoricalArrays

using Statistics

using Dates

R"""

require(dplyr, quietly = TRUE) # for data wrangling

require(tidyverse, quietly = TRUE) # for data wrangling

require(lme4)

require(lmerTest)

require(quickpsy)

SimulatePsychometricFunction\_Staircase = function(ID,

ConditionOfInterest,

StandardValues,

reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

SD\_ResponseFunction,

Mean\_Variability\_Between = 0.1,

SD\_Variability\_Between = 0.1){

Psychometric = expand.grid(ID=ID, ConditionOfInterest=ConditionOfInterest, StandardValues=StandardValues, reps = reps)

Psychometric = Psychometric %>%

group\_by(ID) %>%#

mutate(PSE\_Factor\_ID = rnorm(1,1,Mean\_Variability\_Between),

SD\_Factor\_ID = rnorm(1,1,SD\_Variability\_Between))

Psychometric = Psychometric %>%

mutate(

Mean\_Standard = StandardValues+StandardValues\*Multiplicator\_PSE\_Standard,

SD\_Standard = StandardValues\*Multiplicator\_SD\_Standard,

Mean = (Mean\_Standard + (ConditionOfInterest==ConditionOfInterest[2])\*StandardValues\*PSE\_Difference)\*PSE\_Factor\_ID,

SD = abs((SD\_Standard + (ConditionOfInterest==ConditionOfInterest[2])\*SD\_Standard\*JND\_Difference)\*SD\_Factor\_ID),

staircase\_factor = rcauchy(length(reps),1,SD\_ResponseFunction),

Presented\_TestStimulusStrength = Mean\*staircase\_factor,

Difference = Presented\_TestStimulusStrength - StandardValues,

AnswerProbability = pnorm(Presented\_TestStimulusStrength,Mean,SD),

Answer = as.numeric(rbernoulli(length(AnswerProbability),AnswerProbability))

)

Psychometric = Psychometric %>%

filter(abs(staircase\_factor-1) < 0.75) %>%

group\_by(ID,ConditionOfInterest,StandardValues,Difference) %>%

mutate(Yes = sum(Answer==1),

Total = length(ConditionOfInterest))

Psychometric

}

""";

Then, we establish a Julia function that calls the above R function and prepares the data for analysis with the MixedModels.jl package. First, we send the arguments of the Julia function to R with the “@rput” command. Then, we call the R function to simulate the dataframe and send it from R back to Julia with the “@rget” command. Since MixedModels.jl doesn’t recognize floats as factors when used as random effects, we then convert the standard values into categorical variables with help of the CategoricalArray package. The output of this function is thus a Julia dataframe that the MixedModels.jl package can work with.

function SimulateDataframe(n,

ConditionOfInterest,

StandardValues,

reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

SD\_ResponseFunction,

Mean\_Variability\_Between,

SD\_Variability\_Between)

@rput n ConditionOfInterest StandardValues reps PSE\_Difference JND\_Difference Multiplicator\_PSE\_Standard Multiplicator\_SD\_Standard SD\_ResponseFunction Mean\_Variability\_Between SD\_Variability\_Between

R"""

ID = paste0("s",1:n)

Psychometric = SimulatePsychometricFunction\_Staircase(ID,

ConditionOfInterest,

StandardValues,

1:reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

SD\_ResponseFunction,

Mean\_Variability\_Between,

SD\_Variability\_Between

"""

@rget Psychometric

Psychometric[:StandardValuesAsFactor] = "placeholder"

for i = 1:length(Psychometric[!,:StandardValues])

Psychometric[i,:StandardValuesAsFactor] = string(Psychometric[i,:StandardValues])

end

Psychometric[!,:StandardValuesAsFactor] = CategoricalArray(Psychometric[!,:StandardValuesAsFactor])

end

We then establish two functions that take the output of the previous one, fit the respective GLMM and output the respective p values.

function GLMM\_Accuracy(Psychometric)

formulaAccuracy = @formula(Answer ~ 1 + ConditionOfInterest + (1 + Difference|ID) + (1 + Difference|StandardValuesAsFactor));

modelAccuracy = GeneralizedLinearMixedModel(formulaAccuracy, Psychometric, Bernoulli())

(coeftable(GLMM)).cols[4][2]

end

Function GLMM\_Precision(Psychometric)

formula1 = @formula(Answer ~ Difference\*ConditionOfInterest + (Difference|ID) + (Difference|StandardValuesAsFactor));

modelPrecision = fit!(GeneralizedLinearMixedModel(formula1, Psychometric, Binomial()), fast=true)

(coeftable(GLMM)).cols[4][4]

end

Finally, we choose the values of interest. We then use the above functions to simulate simulate 1000 datasets and perform the GLMM analysis over each dataset. We also save the power for each combination of repetition and subject number in the dataframe **PowerfulDataframe**.

ConditionOfInterest = [0;1]

StandardValues = [5;8]

Range\_reps = [60]

PSE\_Difference = 0.1

JND\_Difference = 0.3

Multiplicator\_PSE\_Standard = 0

Multiplicator\_SD\_Standard = 0.108

SD\_ResponseFunction = 0.1

Mean\_Variability\_Between = 0.1

SD\_Variability\_Between = 0.1

nIterations = 100

Range\_Participants = [10,12,14,16,18,20]

nIterations = 1000

TotalNumber = length(Range\_reps)\*length(Range\_Participants)

CurrentRunthrough = 0

rightnow = Dates.now()

for reps in Range\_reps

for n in Range\_Participants

TimeStartTrial = Dates.now()

Pvalues\_Accuracy = []

Pvalues\_Precision = []

Pvalues\_Accuracy\_TwoLevel = []

Pvalues\_Precision\_TwoLevel = []

for j in 1:nIterations

Pvalues = SimulateDataframe(n,

ConditionOfInterest,

StandardValues,

reps,

PSE\_Difference,

JND\_Difference,

Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard,

SD\_ResponseFunction,

Mean\_Variability\_Between,

SD\_Variability\_Between)

Pvalues\_Accuracy = [Pvalues\_Accuracy;Pvalues[1]]

Pvalues\_Precision = [Pvalues\_Precision;Pvalues[2]]

Pvalues\_Accuracy\_TwoLevel = [Pvalues\_Accuracy\_TwoLevel;Pvalues[3]]

Pvalues\_Precision\_TwoLevel = [Pvalues\_Precision\_TwoLevel;Pvalues[4]]

end

CurrentRunthrough = CurrentRunthrough + 1

if CurrentRunthrough == 1

global PowerfulDataframe = DataFrame(n=n,

ConditionsOfInterest=length(ConditionOfInterest),

StandardValue1=StandardValues[1],

StandardValue2=StandardValues[2], reps=reps,

PSE\_Difference=PSE\_Difference,

JND\_Difference=JND\_Difference,

Multiplicator\_PSE\_Standard=Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard=Multiplicator\_SD\_Standard,

SD\_ResponseFunction=SD\_ResponseFunction,

Mean\_Variability\_Between=Mean\_Variability\_Between,

SD\_Variability\_Between=SD\_Variability\_Between,

power\_Accuracy = mean(Pvalues\_Accuracy .< 0.05),

power\_Precision = mean(Pvalues\_Precision .< 0.05),

power\_Accuracy\_Twolevel = mean(Pvalues\_Accuracy\_TwoLevel .< 0.05),

power\_Precision\_Twolevel = mean(Pvalues\_Precision\_TwoLevel .< 0.05),

Duration = ((Dates.now()) - TimeStartTrial))

else

row = DataFrame(n=n,

ConditionsOfInterest=length(ConditionOfInterest),

StandardValue1=StandardValues[1],StandardValue2=StandardValues[2],

reps=reps,

PSE\_Difference=PSE\_Difference,

JND\_Difference=JND\_Difference,

Multiplicator\_PSE\_Standard=Multiplicator\_PSE\_Standard,

Multiplicator\_SD\_Standard=Multiplicator\_SD\_Standard,

SD\_ResponseFunction=SD\_ResponseFunction,

Mean\_Variability\_Between=Mean\_Variability\_Between,

SD\_Variability\_Between=SD\_Variability\_Between,

power\_Accuracy = mean(Pvalues\_Accuracy .< 0.05),

power\_Precision = mean(Pvalues\_Precision .< 0.05),

power\_Accuracy\_Twolevel = mean(Pvalues\_Accuracy\_TwoLevel .< 0.05),

power\_Precision\_Twolevel = mean(Pvalues\_Precision\_TwoLevel .< 0.05),

Duration = ((Dates.now()) - TimeStartTrial))

PowerfulDataframe = append!(PowerfulDataframe,row)

end

print("RUNTHROUGH ", CurrentRunthrough, " out of ", TotalNumber,": ", n, " ", reps, " ",

PSE\_Difference, " ", JND\_Difference, " ", mean(Pvalues\_Accuracy .< 0.05), " ",

mean(Pvalues\_Precision .< 0.05), " ", PowerfulDataframe[!,:Duration][CurrentRunthrough], " END. ")

end

end

Finally, we can write the computed powers for each number of repetitions and participants in a table.

CSV.write(join([reps,"\_", PSE\_Difference, "\_", JND\_Difference, ".csv"]),PowerfulDataframe)

We can also plot this table with a call to ggplot:

@rput PowerfulDataframe

R"""

plot = ggplot(PowerfulDataframe) +

geom\_line(aes(n,power\_Precision),color="red") +

geom\_line(aes(n,power\_Accuracy),color="blue") +

facet\_grid(.~reps) +

geom\_hline(yintercept = 0.8, linetype=5) +

geom\_hline(yintercept = 0.9, linetype=2) +

geom\_hline(yintercept = 0.95, linetype=3)

plot

"""

### R vs. Julia – A comparison

We used the above procedures to measure the speed for the same operations in Julia and R. We used a Julia script and use the RCall implementation to obtain lme4 fits. We also tried using lme4 natively in R and found by-and-large the same fitting durations as when calling lme4 through RCall from the Julia script. Using RCall allows us to compare fitting durations across the same datasets, which helps to eliminate variability due to differences in the simulated datasets. We perform the procedure 20 times for each combination of Optimizer Configuration (see below), number of participants, number of trials and Effect condition (No Effect, PSE\_Difference = JND\_Difference = 0; and “Small Effect”, PSE\_Difference = 0.025 and JND\_Difference = 0.05) and compare the median fitting duration across these four variables. We also compare median AICs to assess whether slower fits might yield increases in model fit, which might justified the increased duration. There are different implementations and configurations available for both Julia and R. We are going to evaluate them based on fitting duration and model fit. Since p values for (Generalized) Linear Mixed Models are generally approximations, we are also going to evaluate the false positive rates.

#### Optimizers

lme4 for R supports implementations of the Nelder-Mead method (Nelder & Mead, 1965) and the BOBYQA method (Powell, 2009). It furthermore supports the (generally faster) implementations of optimizing algorithms from the package nloptwrap; we thus also add the BOBYQA instantiation from this package for comparison.

#### P value approximation

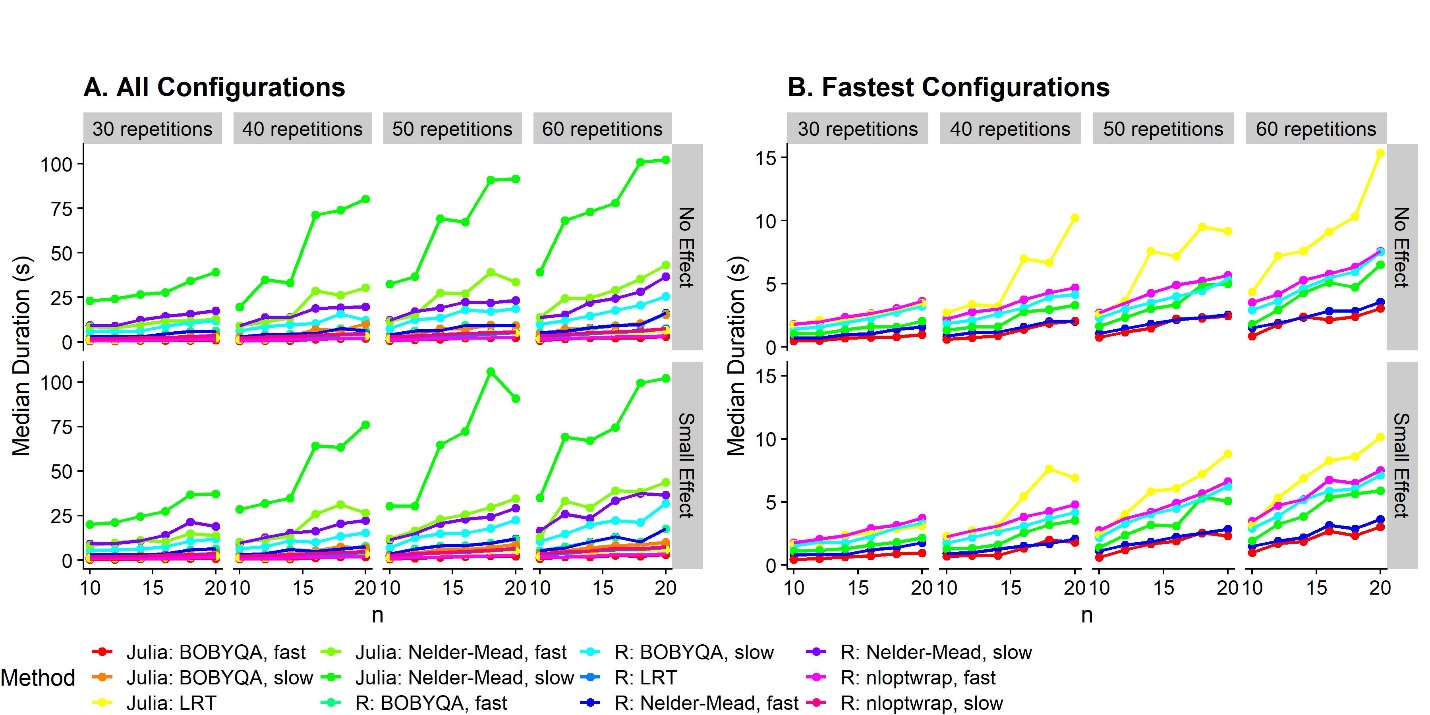
There are different approaches to significance testing in Mixed Modelling. Common approaches are Wald Z Tests, Likelihood Ratio Testing and bootstrapped confidence intervals. Bootstrapped confidence intervals are too computationally costly for the purpose of power simulations. We will therefore test the Wald Z Test (implemented with the Satterthwaite degrees of freedom Method in the R package lmerTest, and natively in the MixedModels.jl package in Julia) and Likelihood Ratio Tests.

#### nAGQ = 0/fast = true

Furthermore, both the lme4 implementation in R and the MixedModels.jl implementation in Julia offer the possibility to trade-off accuracy for higher speed (nAGP = 0 argument in R, and fast = true in Julia). We will test whether the gains in model fit for the slower, more accurate version are worth the increased fitting duration.

#### Speed

Overall, the fastest implementations are by far the BOBYQA implementation in R from the “nloptwrap” package and the BOBYQA implementation in Julia (see Figure 6) in their nAGQ=0/fast=true versions. All other implementations (the nAGQ=1/fast=false versions of the BOBYQA implementations, the default BOBYQA implementations from the lme4 package, and all Nelder-Mead implementations) are much slower, taking between three and ten times longer than the two fastest implementations. We display the median fitting durations for each combination of Optimizer Configuration, number of subjects, number of trials and Effect strength in Figure 6A. Figure 6B represents a close-up of the fastest four optimizers: Julia: BOYQA, fast; R: BOBYQA (nloptwrap), fast; R: BOBYQA (nloptwrap), slow; Julia: BOBYQA, slow.



**Figure 6**: Median durations to fit GLMMs in different languages and with different configurations (color-coded), for datasets of different sizes. We illustrate the durations for 10, 12, 14, 16, 18 and 20 participants (x axis), 30, 40, 50 and 60 trials per staircase (columns of panels) and no effect (PSE\_Difference = JND\_Difference = 0) and a small effect (PSE\_Difference = 0.025 and JND\_Difference = 0.05) in rows of panels. We plot the median values across 20 repetitions per combination of repetition number, participant number, effect (none, small) and GLMM fitting configuration. **A**. All optimizer configurations. **B**. Only the fastest four configurations, from fastest to slowest: Julia: BOYQA, fast; R: nloptwrap, fast; R: nloptwrap, slow; Julia: BOBYQA, slow.

#### Model fits

To assess whether the slower algorithms yield a better model fit, we subtract the AICs for the other optimizer configurations by the AIC for the slowest optimizer configuration (“R: Nelder-Mead, nAGQ=1”). This method yields a ratio where a value of below 1 indicates that the optimizer configuration in question makes for better fits than “R: Nelder-Mead, slow”, and values above 1 indicate that the optimizer makes for worse fits. As evident from Figure 7, the AIC is indeed lowest for this optimizer, along with “Julia: Nelder-Mead, slow” and “Julia: BOBYQA, slow”. However, the differences with regards to the other optimizer configurations, including the fastest ones, are miniscule, with ratio differences below 0.00005.

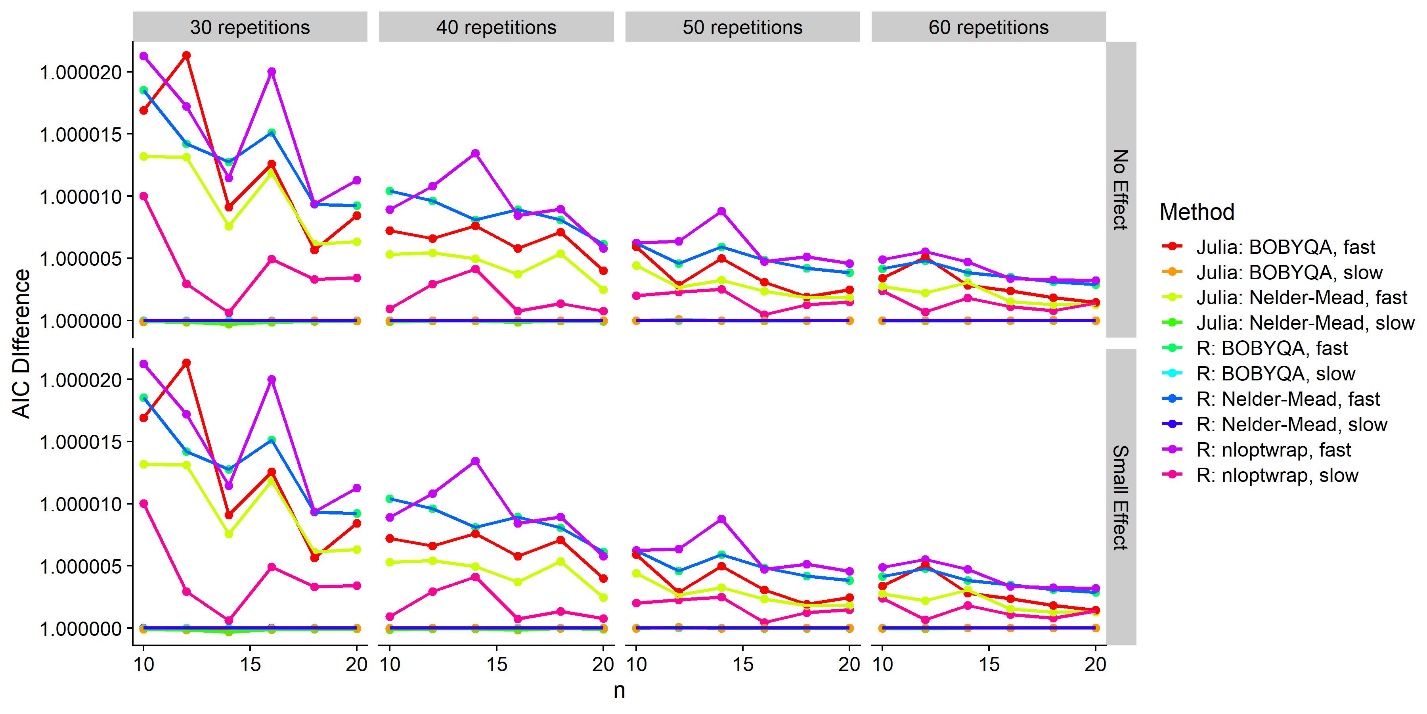


Figure : Median difference between each combination of configurations (color-coded) and the “R: Nelder-Mead, slow” combination for 10-20 participants (x axis) and 30-60 repetitions per staircase (panels).

#### False positive rates

We illustrate the distribution of p values in Figure 5A for accuracy and in Figure 5B for precision. For accuracy, that is, PSE differences, there are very small differences between the different fits and ways of obtaining p values. For precision, that is, for JND differences, however, several optimizer configurations and p value approximations yield inflated false positive rates. All nAQP = 0 R implementations and all Julia implementations give false positive rates of 10 to 20 %, in comparison to the expected 5 %. Only the three nAQP = 1 R implementations yield acceptable false positive rates. Notably, these inflated false positive rates seem to translate also to a higher rate of “true positives” in the presence of a very small effect. However, a comparison with the those optimizer configurations that yield an acceptable false positive rate reveals a higher “true positive rate”, indicating that using these implementations might overestimate power. With the exception of “R: nloptwrap, slow”, the less false positive-prone optimizer configurations are all extremely slow (see Figure 6) – so slow, in fact, that they are hardly suitable for power simulations. When using Likelihood Ratio Tests instead of the default Z Wald tests, the elevated false positive rates disappear. However, Likelihood Ratio Tests require fitting two models: a test model that contains the variable of interest (in our case the interaction between “ConditionOfInterest” and “Difference”), and the next simplest model that doesn’t contain it (ConditionOfInterest and Difference as main effects, but not their interaction) as null model. That is, two models need to fitted to obtain one p value instead of just one. We used the fast method of fitting GLMMs for each program (“R: BOBYQA (nloptwrap), fast” and “Julia: BOBYQA, fast”) and compared them with the Likelihood Ratio Test implemented in the stats::anova() function in R, and the MixedModels::LikelihoodRatioTest() implementation for Julia. As evident from Figure 5D, the R version of this procedure (“R: LRT”) might *underestimate* power vastly, while the Julia implementation (“Julia: LRT”) yields roughly the same ratio of true positives as the slower, possibly more accurate optimizer configurations in R.

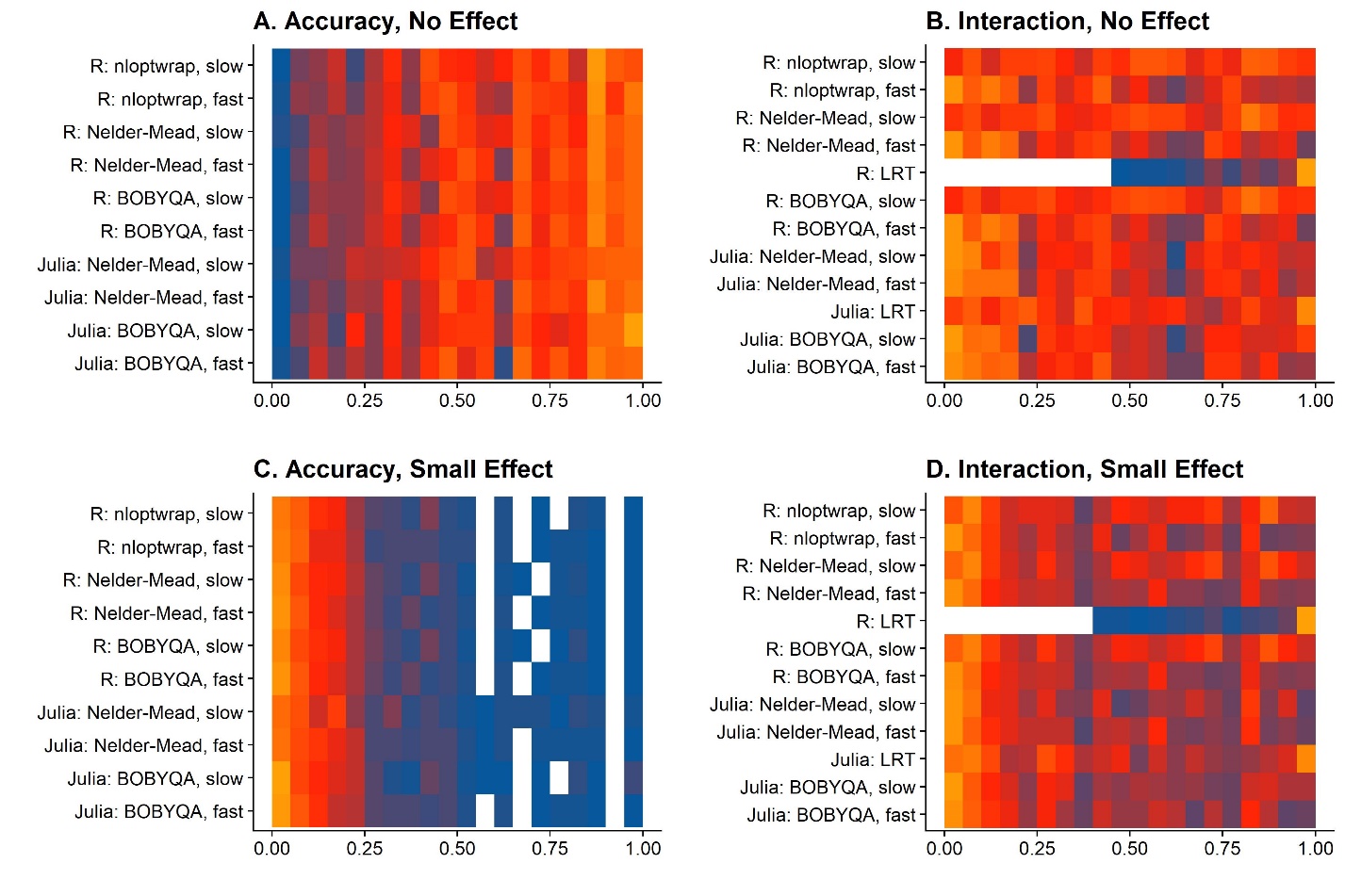


Figure : Frequency of p values per bin of 0.05 for different fitting methods. The gradient from blue over red to orange indicates the frequency. A. P values for PSE differences when there is no effect (PSE\_Difference = 0). B. P values for JND differences when there is no effect (JND\_Difference = 0). C. P values for PSE differences when there is a small effect (PSE\_difference = 0.025). C. P values for JND differences when there is a small effect (JND\_Difference = 0.05).

#### Concluding recommendations

For PSE differences, the best ways to implement power simulations are:

* 1. Fitting GLMMs in Julia with the “BOBYQA, fast” implementation.
  2. Fitting GLMMs in R with the nloptwrap implementation of the BOBYQA algorithm, with nAQP = 1

For JND differences, there seem to be two good ways of implementing power analyses with regards to the different optimizer configurations that are both fast enough and produce reliable-enough p values:

2.1 Fitting GLMMs in Julia with the “BOBYQA, fast” implementation and comparing Test and Null models with the MixedModels::LikelihoodRatioTest() function. This variant is a bit faster, but the speed benefit is dampened by the need to fit two models (Test and Null).

2.2 Fitting GLMMs in R with the BOBYQA implementation from the “nloptwrap” package and the slower nAQP = 1 option. This version is slightly slower than version (1), but still fast enough, and it doesn’t require R users to code in Julia.

Overall, variants 1.2 and 2.2 will probably be more appealing to most users. While Julia is quite intuitive for experienced R users, the speed benefit is probably not worth the effort of learning it. However, there might be cases, especially when larger datasets are involved (as necessary for, for example, between-subject designs), where the effort may be worth it.