# 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 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). 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 of the measures provided. Depending on the experimental design, this can 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. In the first part of this manuscript, we will compare different statistical approaches to Null Hypothesis Testing for typical psychophysical data. We furthermore provide a step-by-step tutorial to simulating psychophysical data that R users can easily adapt to match their own experiments. We will use these simulated data to illustrate the different approaches in significance testing for psychophysical data and give concrete recommendations for a variety of use cases.

In the second part of this paper, we will show how we can leverage simulations to plan sample sizes more efficiently. While power analyses are becoming more mainstream in other areas, they still are the exception in the typical psychophysical study. Some tutorials 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 rely heavily on fitting 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 the following, we thus first discuss the two main methods for analyzing data from Two-Alternative Forced-Choice Tasks. We then elaborate on how to simulate datasets that resemble the data expected for a task, and provide a sample analysis for this fictional dataset. Then, we give recommendations on how to obtain the power of an experimental setup given certain assumptions, and compare the two approaches of analysis in terms of their power. We accompany this with example implementations in R and Julia and show ways to determine the most efficient trials-per-participant ratio.

## Different Approaches to Null Hypothesis Testing in Psychophysics

There are different approaches to Null Hypothesis Testing in Psychophysics. Classically, psychometric functions (Cumulative Gaussian or Weibull functions) are fitted for each condition and participant to obtain the Points of Subjective Equality (PSEs) and Just Noticeable Differences (JNDs). This yields one data point per subject and condition, over which a t test or an ANOVA is performed to test for statistical significance. This approach, however, neglects that each PSE and JND is based on a large number of trials and thus fails to account for the added reliability of the measures provided. Depending on the experimental design, this can 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. In the following, we will estimate power for both types of analyses, to quantify how both approaches compare in terms of sensitivity to detect effects.

Classically, psychometric functions (Cumulative Gaussian functions, Logistic functions or Weibull functions) are fitted for each condition and participant to obtain the Points of Subjective Equality (PSEs) and Just Noticeable Differences (JNDs). The mean of the Cumulative Gaussian corresponds to the PSE and its standard deviation corresponds to the JND. This yields one data point per subject and condition, over which a t-test or an ANOVA are performed to test for statistically significant differences between conditions.



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.

This approach neglects that each PSE and JND is based on a large number of trials: it treats each PSE and JND as a single number, while neglecting that the more trials we use in our staircase, the more precise and reliable our estimates of PSEs and JNDs are. To avoid the resulting loss of statistical power, Moscatelli et al. (Moscatelli et al., 2012) have suggested the use of General Linear Mixed Modelling (GLMM). GLMM allows to obtain population-wide parameters for PSEs and JNDs, while still accounting for inter-subject variability in responses. Mixed Modelling is a more flexible form of linear regression. It allows to fit regression coefficients across the whole population for some parameters, while allowing the coefficients for other parameters to vary within subgroups of the dataset. A classic example is the modelling of the efficacy of a learning intervention on population of students from different classes in one school. Mixed models can account for inherent performance differences between classes, but fit a population-wide coefficient for the efficacy of the intervention. General Linear Mixed Modelling extends this principle by allowing to fit not only linear regression lines, but also other functions – such as cumulative Gaussians, which are commonly used as approximations for psychometric functions. Moscatelli et al. (2012) provide a more thorough explanation.

One of improvements that have been demanded in the wake of the reproducibility crisis is a more thorough and meticulous study planning. Researchers need to be more aware of underlying theoretical considerations, specify hypotheses before analyzing the data, make precise predictions of how their hypotheses should manifest in their data and formulate statistical models to test these hypotheses. One important step in this process is to make sure that the experiment has sufficient statistical power to detect the postulated effects. We will first show that many experiments using the two-level approach of hypothesis testing described above often lacks power and indicate how much power could be gained for the same experimental designs by using Moscatelli et al.’s GLMM approach. In the second part of this paper, we will demonstrate how to plan the sample size in a psychophysical experiment, levering the advantages of the GLMM approach.

# Power analyses for the GLMM approach using simulations in R

In the following we will provide an example of how to compute the power for common psychophysical designs, using the GLMM approach for analysis. Further below, we will also compare the power we obtain for the same designs when using the Two-Level approach. Please note that we advise strongly to read this document together with the R script available in the GitHub Repository. 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 datasets. To this end, we will use pilot data from our lab.

## Assumptions

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 1 visualizes 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")

We then fit the psychometric functions using the “quickpsy” package. 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 1 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.

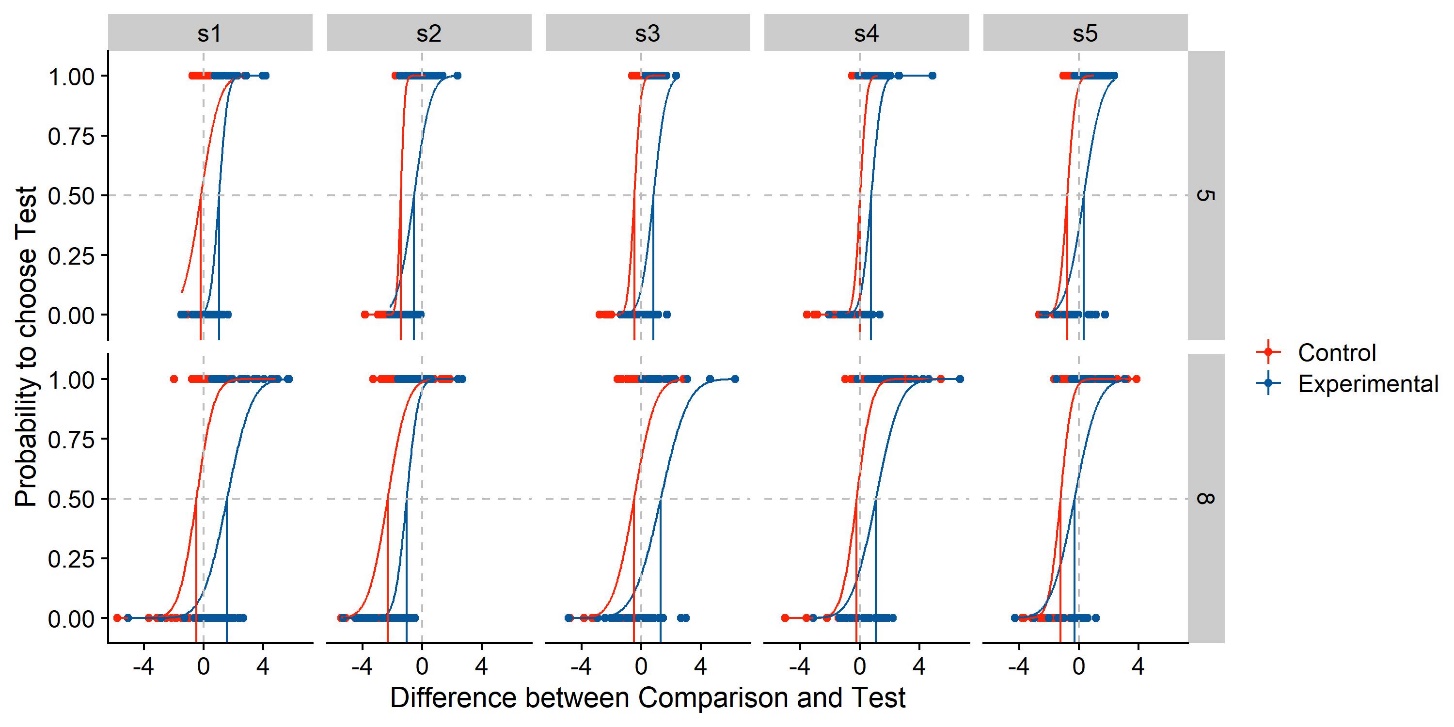


Figure : Simulated psychometric functions based on the example values chosen above. We plot the difference in stimulus intensity between test and standard stimulus (x axis) against the participants’ probability to choose the test stimulus as more intense (y axis). Different panels are the psychometric 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.

## Estimating population parameters of the psychometric functions with the GLMM approach

### Accuracy

Next, we establish the statistical models we use to test our hypotheses. Following (Moscatelli et al., 2012), we use Generalized Linear Mixed Models for this purpose. For differences in PSEs in our simulated data set, the GLMM could have “**Condition Of Interest**” (a binary categorical variable with the values “**1**” for “**Test Condition**” and “**0**” for “**Standard Condition**”) as fixed effect, and random intercepts and slopes for the Difference between test stimulus and standard stimulus (“**Diff**”) per Participant (“**ID**”) and value of the standard stimulus (“**Standard Value**”).

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

family = binomial(link = "probit"),

data = Psychometric,

nAGQ = 0,

control = glmerControl(optimizer = "nloptwrap"))

Note that there are different ways of specifying the model depending on assumptions about the data and the hypotheses. This sample specification:

* Assumes
  + That we are interested in a population-wide estimate of the the impact of the condition of interest on PSEs (fixed effect of Condition of Interest)
  + That we are not interested in its population-wide impact on JNDs (no interaction between "**Condition of Interest**" and "**Difference**")
* Allows intercepts and slopes to vary per participant. Intercepts correspond to PSEs, while slopes correspond to JNDs. That is, we acommodate individual differences in sensitivity and accuracy.
* Allows intercepts and slopes to vary per standard value. It thus acommodates that different standard values might lead to differences in PSEs and JNDs. For example, higher standard values should lead to more shallow slopes (in absolute terms) if Weber Fractions hold for the stimulus type under investigation.

Applying the summary() function to the statistical model yields estimates for the coefficients, along with standard errors. Furthermore, the lmerTest package (Kuznetsova, Brockhoff, & Christensen, 2017) provides the possibility to compute p values using the Satterthwaite degrees of freedom method (Gaylor & Hopper, 1969). The authors of the lme4 package (Bates, Mächler, Bolker, & Walker, 2015) recommend to rely on coefficients and their standard errors alone to estimate the impact of the Condition of Interest. p values from the lmerTest package should thus be regarded as complimentary rather than essential tool. Nonetheless, we believe that p values for the variables of interest are an appropriate, simple proxy for detection of differences between conditions, and allow a quick-and-dirty judgement of whether the ConditionOfInterest has a significant impact in a simulated dataset. After loading lmerTest, we can inspect coefficient, standard errors and p values with the summary() function from R core (R Core Team, 2017).

### Precision

Keep in mind that, in the above model, the coefficient of "**Difference**" corresponds to the slope of the psychometric function. To capture to what extent a manipulation impacts the slope (as a proxy for precision), one needs to determine how much the slope differs between two conditions. This corresponds to the interaction between "**Condition of Interest**" and "**Difference**" in our model. We thus add an interaction between “**Condition Of Interest**” and “**Difference**” to the GLMM we used above to test for PSEs differences.

|  |  |
| --- | --- |
|  | (1) |

As above, we can recover the p value for the interaction between “ConditionOfInterest” and “Difference” with the summary() function after loading the lmerTest package. Importantly, the same disclaimers about the interpretation of p values apply as above.

Computing the Power from Simulations

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; a much faster Julia implementation can be found under <https://github.com/b-jorges/Power-Analyses-Psychophysics>), 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 functions containing the above procedure. Then, we determine a range of subject counts for which we want to compute the power. Typically, participant counts between 10 and 20 should allow to detect most relevant effects. Fewer are generally not recommended unless there is strong evidence that between-participant variability is really low. Then, we execute thiss method "nIterations" times for each number of subjects specified above, both for precision and accuracy. Once the simulation process is finished, we can plot the data for each number of subjects, which might ressemble our example illustrated in Figure 3.

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

Moscatelli & Lacquaniti (2012) argue that power is lost when using the Two-Level 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. We thus use the above power simulations to quantify just how much power is lost when using the Two-Level approach in comparison to the GLMM approach. We will use the above procedure to simulate power for the GLMM approach.

## Estimating population parameters of the psychometric functions with the Two-Level approach

For the Two-Level approach, one would first fit psychometric functions for each condition and participant. Then, one would conduct a t.test or an anova to test whether they are different. 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. Then, we extract the parameters and bring the output of quickpsy into the adequate format for ANOVA analysis. Finally, we perform ANOVAs over means and standard deviations of the fitted psychometric functions, with Condition of Interest and Standard Values as main effects. We are mainly interest in the main effect of Condition of Interest, so we extract the p value for this main effect for both means and standard deviations of the psychometric function.

If the aim is not to

## Comparing the statistical power of each approach

We pack these computations into functions and follow the same approach as above to compute the power for PSE and JND differences with the GLMM approach and the Two-Level approach. We use the same values we chose above. Figure XX visualizes the simulated power for each approach (GLMM/Two-Level) and measure (PSE/JND). We can see that the power for PSEs is generally so high that even the Two-Level approach achieves a power of nearly 1 for the whole range of participant counts. For the JNDs, the panorama is quite different. While the GLMM approach obtains a satisfactory power almost from 10 participants on out, the Two-Level approach fails to detect population differences for many cases. To quantify by how much power differs across a more diverse range of power levels, we repeat this process another three times with different parameters. We then obtain the false negative rate (1-power) for each approach, measure and participant count, and obtain the ratio as a measure of how much power is lost when using the Two-Level approach with regards to the GLMM approach. Figure XX illustrates that this relationship is, …

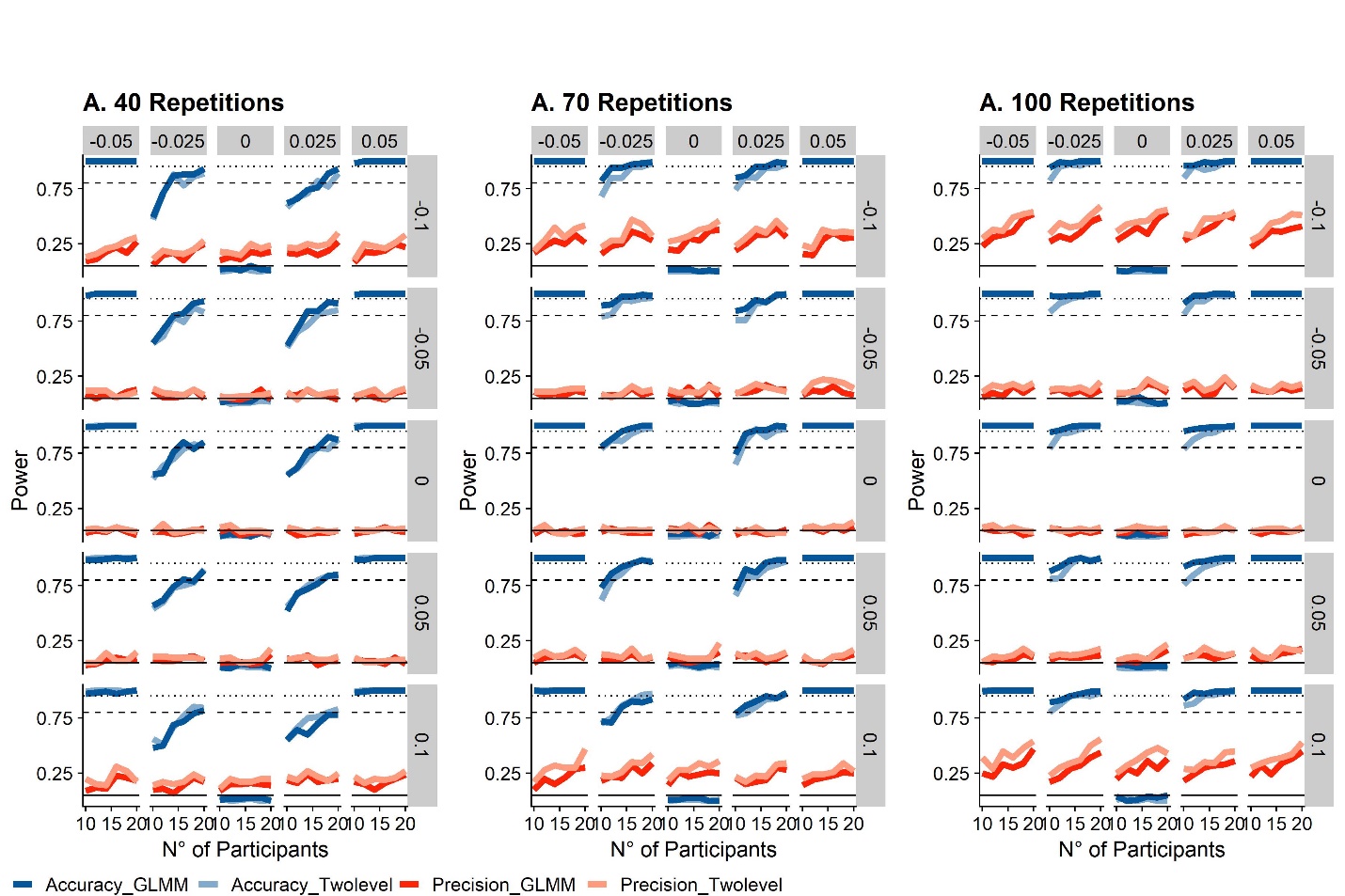


Figure : The solid lines indicate the power level across the range of participant numbers, red for accuracy and blue for precision. The panels represent different number of repetitions per condition. The intermittent horizontal lines indicate power levels of 0.8 (bare minimum), 0.9 (acceptable) and 0.95 (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 worst scoring languages in benchmarks. Fortunately, packages for the fitting of (Generalized) Mixed Models are already available for Julia. Thus, it 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 to ability to call R. 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 fucntion 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

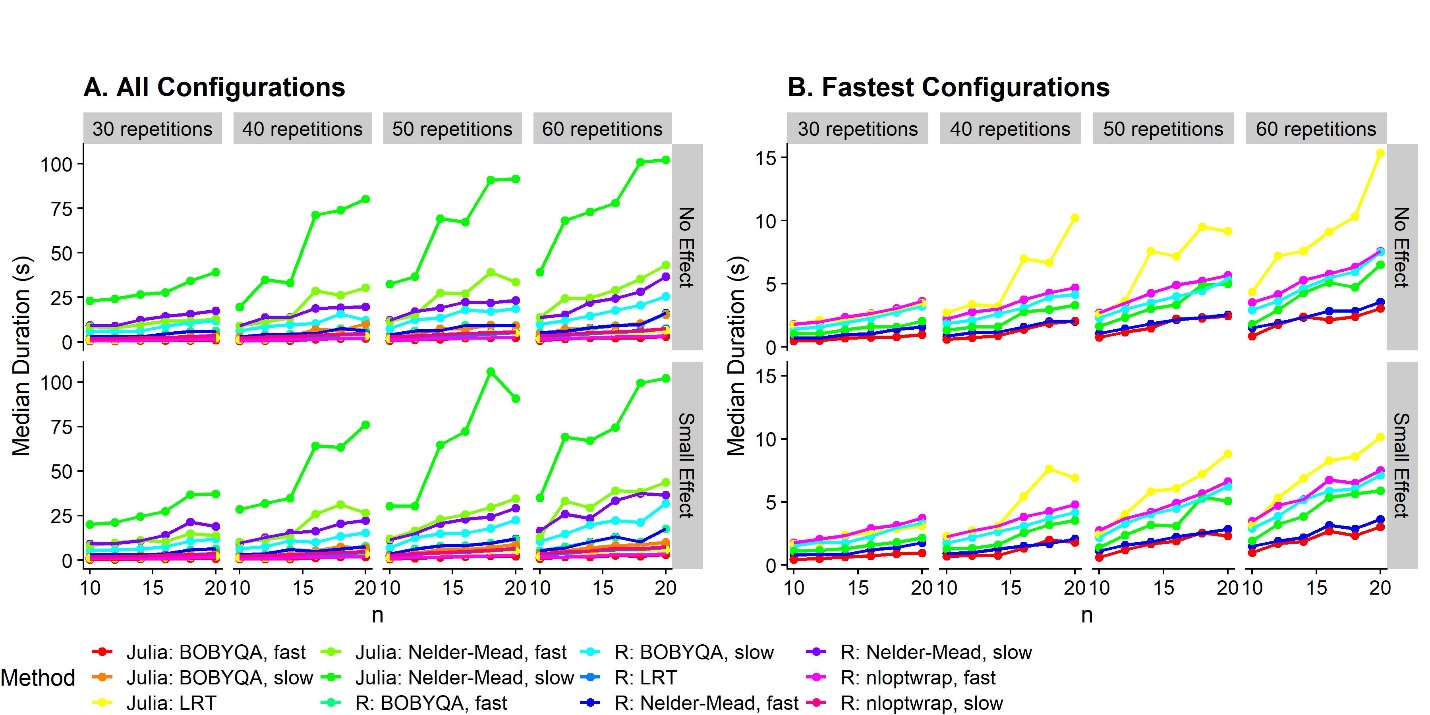
There are different approaches to significance testing in Mixed Modelling. Common approaches are Wald Z Tests, Likelihood Ratio Testing and bootstrapped confidence intervals. Bootstrapped confidence intervals are too computationally costly for the purpose of power simulations. We will therefore test 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.

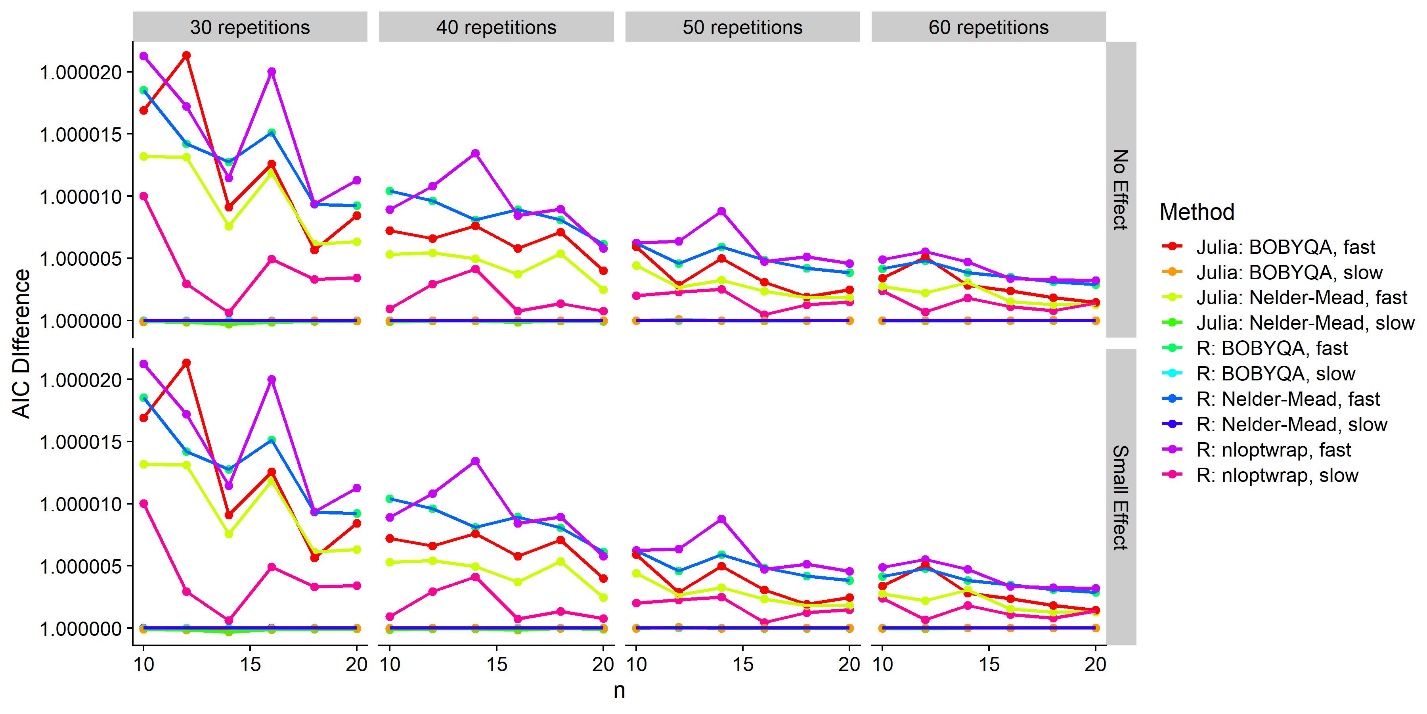


Figure : Median difference between each combination of configurations (color-coded) and the “R: Nelder-Mead, slow” combination for 10-20 participants (x axis) and 30-60 repetitions per staircase (panels).

#### 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.

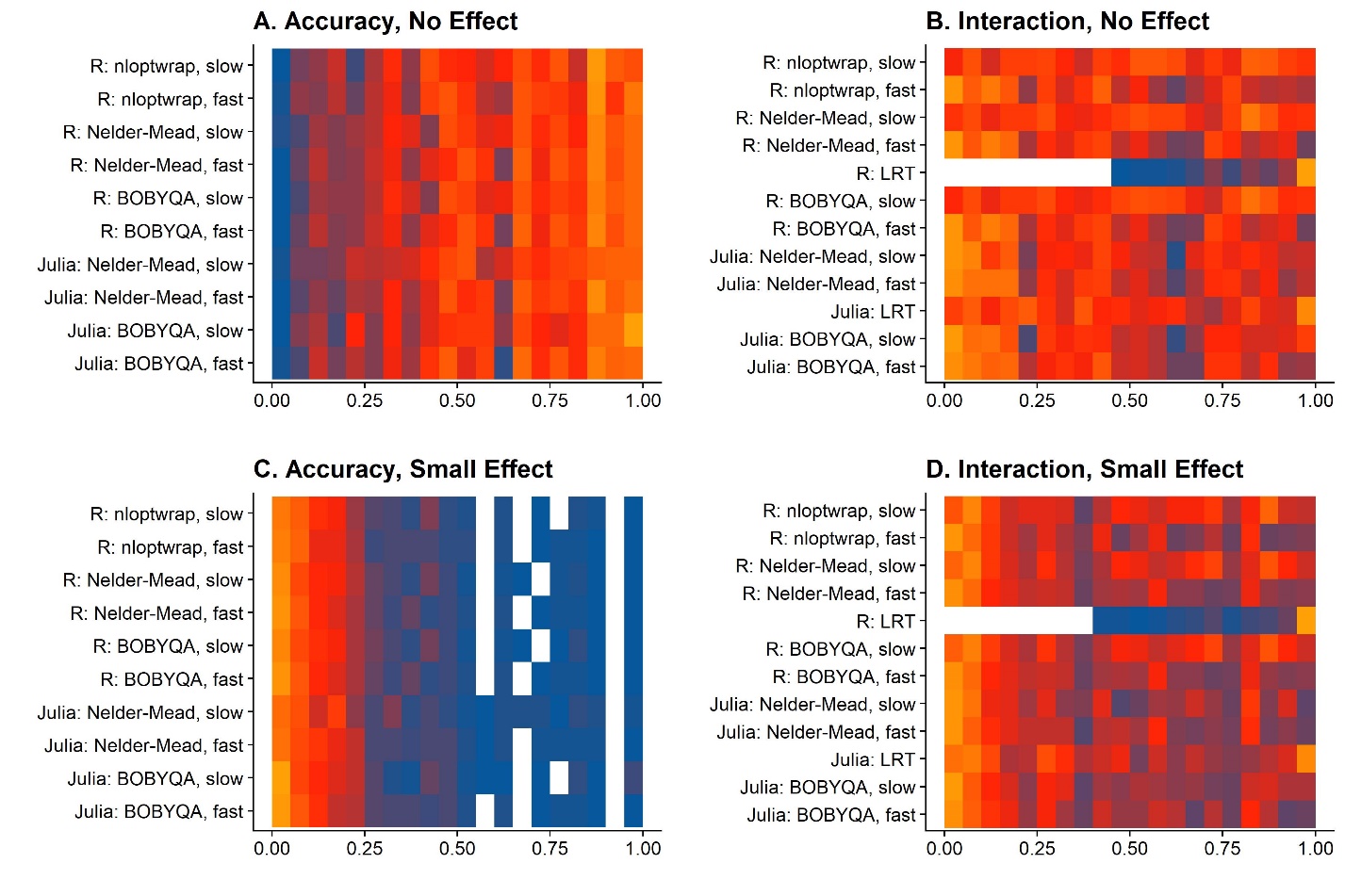


Figure : Frequency of p values per bin of 0.05 for different fitting methods. The gradient from blue over red to orange indicates the frequency. A. P values for PSE differences when there is no effect (PSE\_Difference = 0). B. P values for JND differences when there is no effect (JND\_Difference = 0).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