Guidelines for developing reproducible workflows

Shravan Vasishth[1]

[1] University of Potsdam

Abstract

These are some suggested guidelines on developing a reproducible workflow for carrying out research in any experiimental domain. The guidelines are not meant to be hard and fast rules, but rather are intended to be suggestions. The ultimate aim of these guidelines is to allow the researcher planning a study to have some informed basis for deciding what sample size to choose, to allow anyone in the field to examine and reanalyze your data and code, and to allow anyone to use your research results for carrying out further analyses, such as meta-analyses.

Keywords: data analysis; open science; transparency; power analysis
Word count: 5352 words

Contents

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – Project number 317633480 – SFB 1287, project Q.

Correspondence concerning this article should be addressed to Shravan Vasishth, University of Potsdam. E-mail: vasishth@uni-potsdam.de

Revision history and online repository

- First version: 8 April 2017 following comments from Nieuwland and others.
- Revision: 10 Feb 2020. Added power analyses.

All code and data are available from:

https://vasishth.github.io/ReproducibleWorkflows/

The motivation for these guidelines

As a PhD advisor, I often encounter the following situation: the student carries out a study, and presents their experimental results to me. The code is often not well organized, and almost never documented. Students writing their first paper will often hard-code numerical values such as model estimates in their papers; inevitably, what happens is that something in the data or modeling changes as a result of the reviewing process or due to some other reason (coding errors are common), which then necessitates recopying all the code and data into the paper again. This is very error-prone, not to mention painfully inefficient. It is possible to automate the entire process, as I show below.

As a reviewer and action editor for journals, one problem I face regularly is that I review a paper, but I can't understand what statistical models the authors actually fit. Often I can't even figure out what the estimates of the effect were, because people usually don't even reveal basic information about the effect (such as the estimated coefficient and the standard error of the estimate; or what the parameter estimates of the different variance components were).

As a researcher, I carry out a lot of meta-analyses. Whenever I ask a researcher for the data and code behind the paper, I am usually met with silence, or some reason why the data or code cannot be released. In my experience, only about 25% of researchers are able to release their data and code. **Not providing the data and code that lie behind a paper is a waste of research funds**. It leads to loss of information, and prevents us from incrementally building up our knowledge about a research problem. To see how valuable meta-analyses are, see some of our recent work: Nicenboim, Vasishth, & Rösler (2019), Jäger, Engelmann, & Vasishth (2017), Nicenboim, Roettger, & Vasishth (2018), Bürki, Elbuy, Madec, & Vasishth (2020). You can also watch a talk I recently gave that spells out the importance of meta-analysis in computational modeling: https://youtu.be/UfB6JYaIY9I.

This tutorial aims to provide some suggestions on how to develop a reasonably reproducible workflow, which will allow future you and other researchers to revisit your work, and to build on it. As an example, I will use a replication attempt by Nieuwland et al of a famous EEG study from Marta Kutas' lab.

Introduction: DeLong et al 2005 replication

Recently, Nieuwland et al. (2018) (see here) carried out an interesting series of replication attempts of the DeLong, Urbach, & Kutas (2005) study (see here) Nature Neuroscience paper.

DeLong and colleagues examined the effect of predicting an upcoming noun at the determiner region that precedes the noun. Consider the sentences below:

(1) a. The day was breezy so the boy went outside to fly **a** **kite**.
    b. The day was breezy so the boy went outside to fly **an** **airplane**.

Participants were shown sentences ending with a predictable noun phrase, such as *"a kite"* in (1a), or an unpredictable one, such as *"an airplane"* in (1b).

In both examples, the determiners preceding the critical noun have the same meaning, so there should be no difference between the fit of *"a"* and *"an"* to the semantic context (i.e., both determiners should incur the same integration costs). DeLong and colleagues showed that the amplitude of the negativity was smaller (becomes more positive) with increasing cloze probability at the determiner (and also at the noun; but that effect is expected given prior work, so not so surprising).

Nieuwland et al report a failure to replicate the original effect. Here, we focus only on the article data from the Nieuwland et al study.

Incidentally, DeLong et al never released their data, even though they have it available in a form that could be made available publicly. This sort of situation should never happen; data and code should be made available routinely with a published paper. I believe that leading journals like the *Journal of Memory and Language* now require data and code release, unless there is a compelling reason (e.g., privacy issues) not to make the data public.

Nieuwland et al's work is exemplary in this respect; they publicly released their data and code even before the paper was accepted. First, let's look at how Nieuwland et al made their data and code available.

Data and code from Nieuwland et al

Here are Nieuwland et al's data and code.

- All materials were made available on the Open Science Foundation (OSF) repository: https://osf.io/eyzaq/
- The tree structure of the directory was as follows when I downloaded it in 2017:

```
.
|- 500ms_baseline
      |- art_b5_P.txt
      |- art_b5_R.txt
      |- noun_b5_P.txt
      |- noun_b5_R.txt
|- BFs
      |- Art_Bayes_Factor_Replication_Delong.txt
      |- Nouns_Bayes_Factor_Replication_Delong.txt
      |- orig_art.txt
      |- orig_noun.txt
      |- rep_art.txt
      |- rep_noun.txt
|- Nieuwland_etal_elife_accepted.pdf
|- ReplicationFunctionsCorrelation.R
|- artfinalP.txt
|- artfinalR.txt
|- cor_data_art.txt
|- cor_data_noun.txt
|- nounfinalP.txt
|- nounfinalR.txt
|- public_article_data.txt
|- trial_level_data
    |- art_replication_analysis_500msbaseline.txt
    |- art_replication_analysis_original.txt
    |- nouns_replication_analysis_500msbaseline.txt
    |- nouns_replication_analysis_original.txt
```

What is missing here is a README file. Many of the filenames are essentially self-explanatory, but not all.

The R code file

The R code is presented as a file called public_script.txt.

Some suggestions for improvement:

- An R markdown file would have allowed the reader to see the precomputed output. One can always extract the R code from an R Markdown file called, say, yourfile.Rmd, by doing:

```
library(knitr)
purl("yourfile.Rmd")
```

- Running SessionInfo() provides some minimal information on the packages used and their versions; this can help in diagnosing problems if the code fails to run due to lack of backward compatibility.

- The call to the file has a hard-coded absolute path. This will fail to run on your computer:

```
#read article data
articles <-read.delim("~/Desktop/DELONGR/public_article_data.txt", quote="")
```

Suggestion: give relative path names (relative to current working directory).

```
articles <-read.delim("NieuwlandEtAl2018/public_article_data.txt",
                      quote="")
head(articles)
```

```
##   subject item cloze segment  lab base100 base200 base500 filt01
## 1  birm01  102     0       1 birm   -2.33 -3.4271 -7.0365  -1.06
## 2  birm01  113    36       2 birm    8.56  4.4114  1.7205   9.13
## 3  birm01  111    84       3 birm    7.40  7.9277  8.0056   7.30
## 4  birm01  103    55       4 birm    8.56 11.2289  6.7573  10.27
## 5  birm01  105     0       5 birm   10.62 11.0954  7.0479  12.23
## 6  birm01  101     0       6 birm   15.08 11.4245  9.4637  12.15
##   prestim
## 1  5.8780
## 2  8.5032
## 3 -0.8013
## 4  2.1753
## 5  4.4121
## 6  6.9001
```

- Avoid using T and F for TRUE and FALSE. This can play havoc with your code if T and F are bound variables. RStudio allows auto-completion, so one can just type T and hit tab, and one should get TRUE (same for F and FALSE).

```
clozemean <- mean( articles$cloze, na.rm=T )
clozesd <- sd( articles$cloze, na.rm=T )
```

- I could not find a column called n400, so this line doesn't run:

```
articles$base100 <- as.numeric(as.character(articles$n400))
```

Here is the code that I had in 2017 (it may have been updated since by Nieuwland et al):

```
# R script for the DeLong replication data.
# base100 is the N400 dependent measure with the pre-registered 100 ms baseline, which will
# base200 has the 200 ms baseline, base500 the 500 ms baseline
# filt01 has the 0.01 hz filtered data, and pre-stim the -500 to -100 ms data
#read article data: loaded above by SV.
#articles <-read.delim("~/Desktop/DELONGR/public_article_data.txt", quote="")

# make sure data is right format
articles$item <-as.factor(articles$item)
```

```
articles$cloze <- as.numeric(as.character(articles$cloze))
## no column called n400:
##articles$base100 <- ##as.numeric(as.character(articles$n400))

# Z-transform cloze and store the mean and SD
articles$zcloze <- scale(articles$cloze, center = TRUE, scale = TRUE)
clozemean <- mean( articles$cloze, na.rm=T )
clozesd <- sd( articles$cloze, na.rm=T )
```

Four linear mixed models are fit for the determiner. Most of the models fail to converge for me.

```
# article models
model1a <- lmer(base100 ~ lab*zcloze +
                  ( zcloze | subject) + (zcloze  | item),
                contrasts=list( lab=contr.sum(9) ), data = articles ,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )

model2a <- lmer(base100 ~ lab/zcloze +
                  ( zcloze | subject) + (zcloze  | item),
                contrasts=list( lab=contr.sum(9) ), data = articles ,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )

model3a <- lmer(base100 ~  zcloze + ( zcloze | subject) +
                  (zcloze  | item),
                data = articles,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| =
## 0.00959847 (tol = 0.002, component 1)
```

```
model4a <- lmer(base100 ~         ( zcloze | subject) +
                  (zcloze  | item),  data = articles ,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )

# compare models
anova(model1a,model2a)
```

```
## Data: articles
## Models:
## model1a: base100 ~ lab * zcloze + (zcloze | subject) + (zcloze | item)
## model2a: base100 ~ lab/zcloze + (zcloze | subject) + (zcloze | item)
##         Df    AIC    BIC logLik deviance Chisq Chi Df Pr(>Chisq)
## model1a 25 186855 187059 -93402   186805
## model2a 25 186855 187059 -93402   186805     0      0          1
```

Linear mixed models for the noun:

```
########################
#### NOUNS

#nouns <-read.delim("~/Desktop/DELONGR/public_noun_data.txt", quote="")
nouns <-read.delim("NieuwlandEtAl2018/public_noun_data.txt", quote="")

nouns$item <-as.factor(nouns$item)
nouns$cloze <- as.numeric(as.character(nouns$cloze))
nouns$n400 <- as.numeric(as.character(nouns$n400))

nouns$zcloze <- scale(nouns$cloze, center = TRUE, scale = TRUE)
clozemean <- mean( nouns$cloze, na.rm=T )
clozesd <- sd( nouns$cloze, na.rm=T )

model1n <- lmer(n400 ~  lab*zcloze +
                  ( zcloze | subject) + ( zcloze  | item), contrasts=list( lab=contr.sum(9)
                data = nouns ,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )
model2n <- lmer(n400 ~  lab + zcloze +
                  (zcloze | subject) + (zcloze  | item), contrasts=list( lab=contr.sum(9) )
                data = nouns ,
                control=lmerControl(optCtrl=list(maxfun=1e9)), REML = FALSE )
model3n <- lmer(n400 ~  zcloze +
                  (zcloze | subject) + (zcloze  | item), data = nouns ,
                control=lmerControl(optCtrl=list(maxfun=1e9)), REML = FALSE )
model4n <- lmer(n400 ~
                  (zcloze | subject) + (zcloze  | item), data = nouns ,
                control=lmerControl(optCtrl=list(maxfun=1e9)), REML = FALSE )

# compare models
anova(model1n,model2n)
```

Results for the determiner

Let's focus on just one of the models:

```
model3a <- lmer(base100 ~  zcloze +
                  ( zcloze | subject) + (zcloze  | item),
                data = articles,
                control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl =
## control$checkConv, : Model failed to converge with max|grad| =
## 0.00959847 (tol = 0.002, component 1)
```

```
model3aNULL <- lmer(base100 ~  1 +
                      ( zcloze | subject) + (zcloze  | item),
                    data = articles,
                    control=lmerControl(optCtrl=list(maxfun=1e5)), REML = FALSE )
anova(model3a,model3aNULL)
```

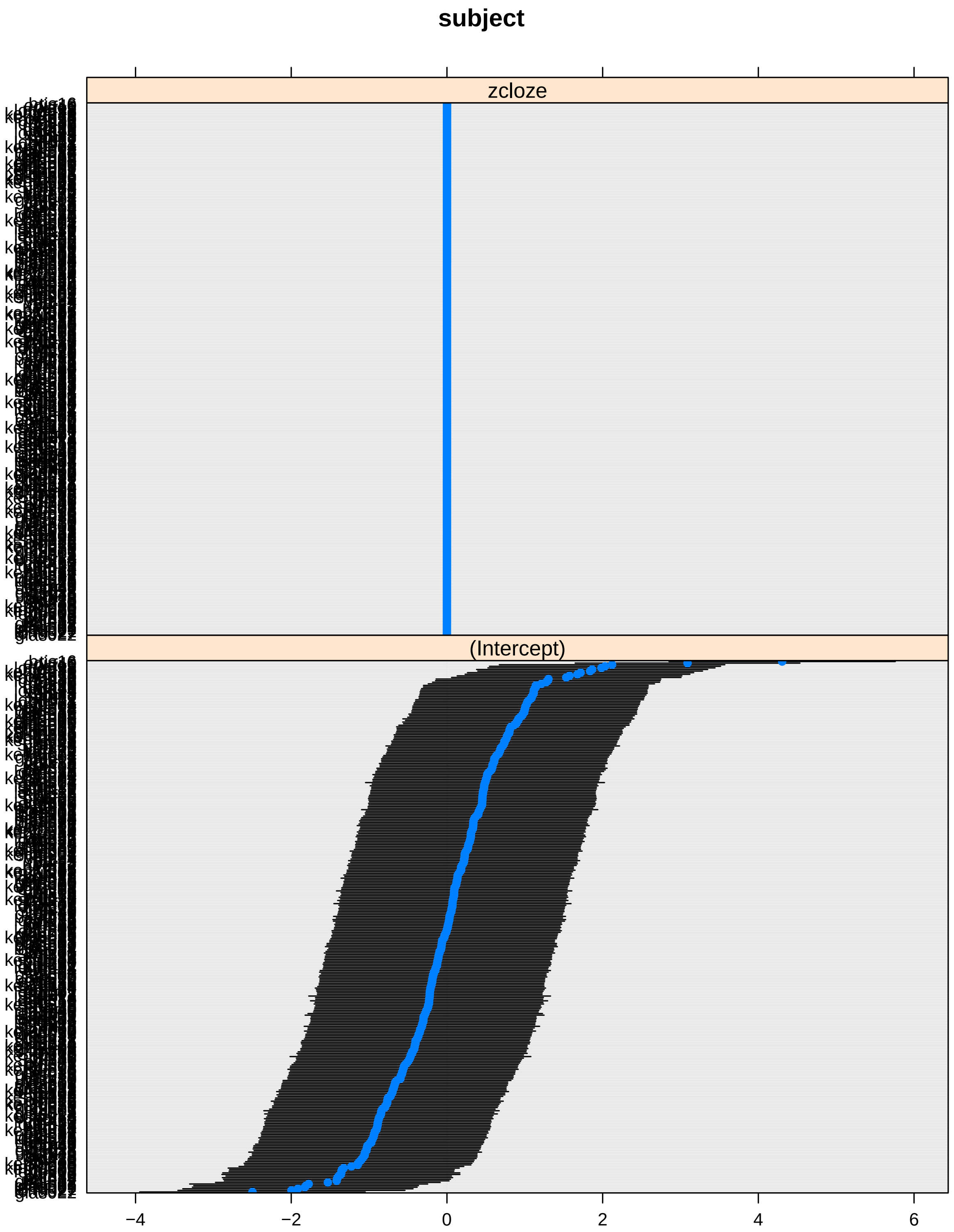
```
## Data: articles
## Models:
## model3aNULL: base100 ~ 1 + (zcloze | subject) + (zcloze | item)
## model3a: base100 ~ zcloze + (zcloze | subject) + (zcloze | item)
##             Df    AIC    BIC logLik deviance Chisq Chi Df
## model3aNULL  8 186840 186905 -93412   186824
## model3a      9 186840 186913 -93411   186822  2.31      1
##             Pr(>Chisq)
## model3aNULL
## model3a           0.13
```

There doesn't seem to be evidence for zcloze affecting the response.

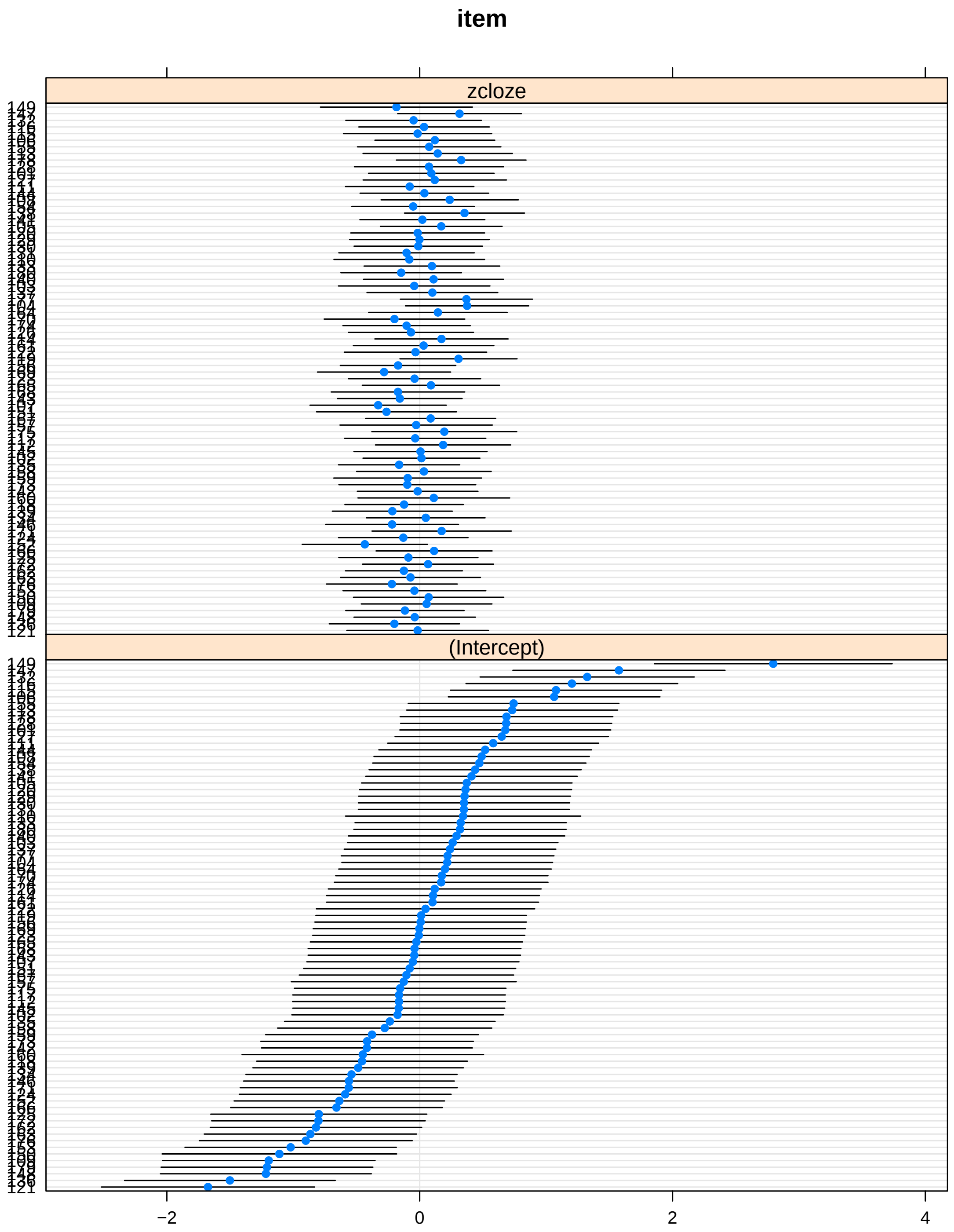
One useful thing to look at is the u0 and u1 values for each participant and item:

```
print(dotplot(ranef(model3a,condVar=TRUE)))
```

```
## $subject
```



```
##
## $item
```



There are more sophisticated ways to plot individual level effects. See our Bayesian modeling lecture notes here; see section 5.1.3.

Planning a future study based on existing data

This assumes a frequentist analysis.

We will:

- begin with the existing data from Nieuwland et al 2018
- first visualize the data
- then use all available data to compute an estimate of the effect (meta-analysis)
- compute sample size needed to achieve 80% power using simulation
- carry out new study
- an alternative Bayesian approach would be to use the estimate of the effect from the previous studies as an informative prior in a Bayesian analysis.

Simulating data is a very important skill

One skill we will learn here is how to simulate data that reflects the underlying generative process we are assuming from our experiment. Simulating data is a very important skill for the data analyst because:

- one can determine whether the model we have specified for data analysis is sensible (generates reasonable data)
- one can determine whether the model we have specified can in principle recover the parameters of interest under repeated sampling
- one can easily do power calculations (although this can take some time to compute for complex models)

Case study: The Niewland et al study data

Subset the existing data. First, choose the relevant columns:

```
dat<-dplyr::select(articles,subject,item,lab,zcloze,base100)
```

If we are using all the data, it might be nicer if we re-name all the subjects by numerical id's instead of lab id plus the subject id.

```
## get subject names
subjnames<-unique(dat$subject)
## convert to numerical values:
subjects_numerical<-data.frame(subjname=subjnames,subj=factor(1:length(subjnames)))

## create new column with numerical id's:
dat2<-merge(dat,subjects_numerical,by.x="subject",by.y="subjname")
dat<-dat2
head(dat)
```

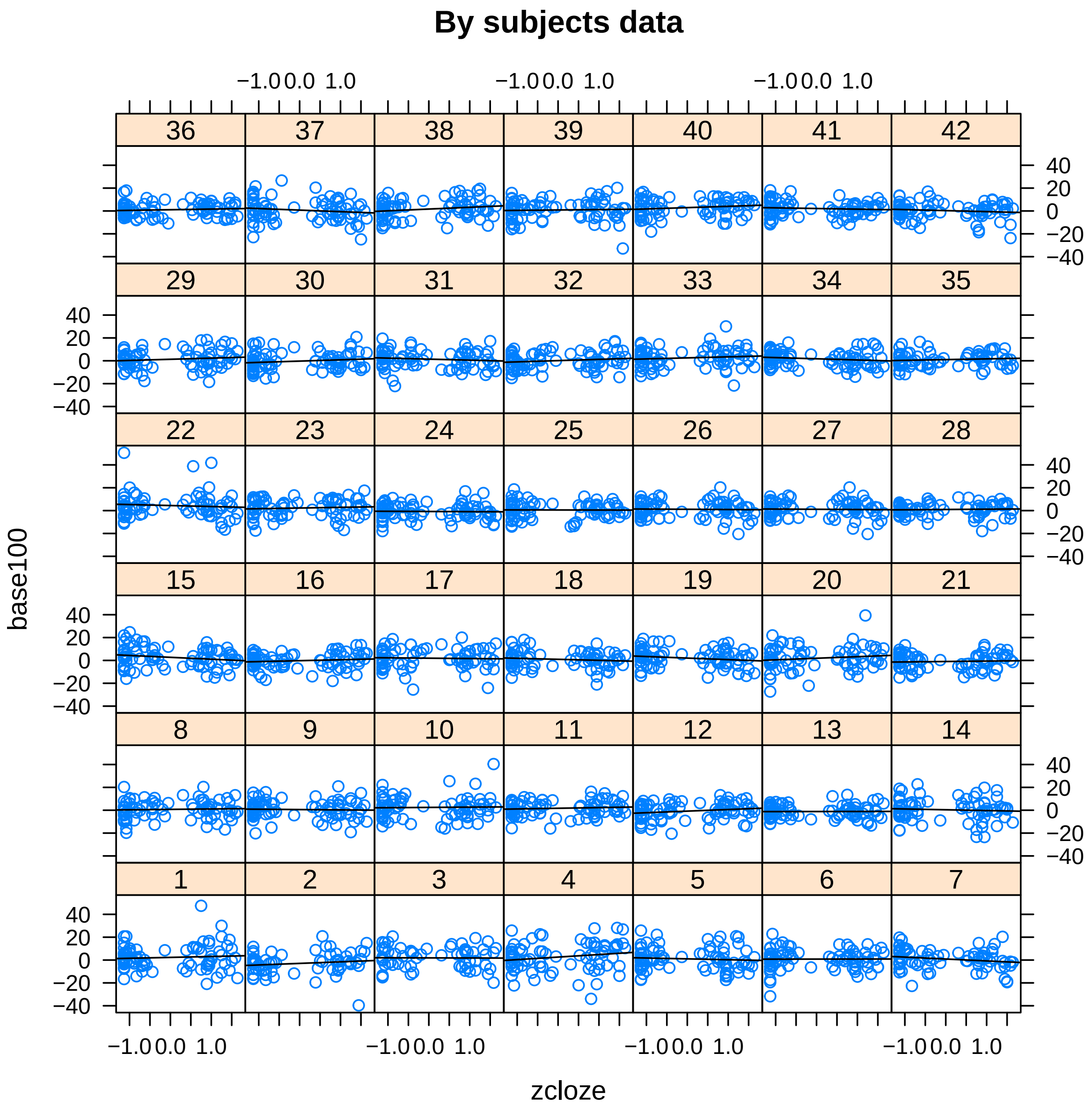
```
##   subject item  lab   zcloze base100 subj
## 1  birm01  102 birm -1.13318   -2.33    1
## 2  birm01  113 birm -0.13494    8.56    1
## 3  birm01  111 birm  1.19605    7.40    1
## 4  birm01  103 birm  0.39191    8.56    1
## 5  birm01  105 birm -1.13318   10.62    1
## 6  birm01  101 birm -1.13318   15.08    1
```

Visualize the data by subject and by item. There is data from 9 labs. We can look at the by-subjects data for each lab. For example, here is the data from Birmingham.

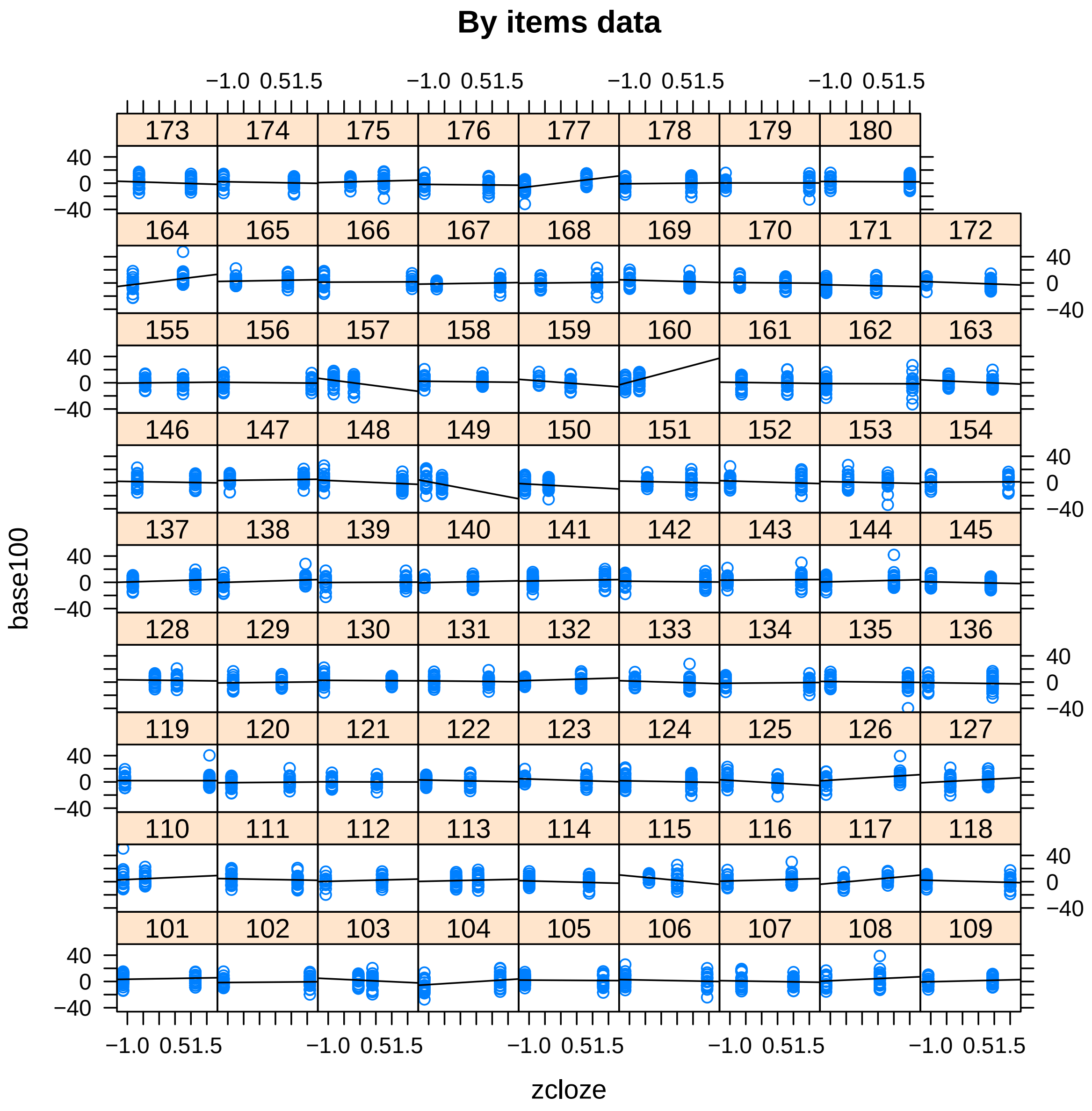
```
## save lab names:
labnames<-as.character(unique(dat$lab))

## Choose first lab's data:
current_lab<-filter(dat,lab==labnames[1])

xyplot(base100~zcloze|subj,current_lab,
       panel = function(x, y) {
         panel.xyplot(x, y)
         panel.abline(lm(y ~ x))
       },main="By subjects data")
```



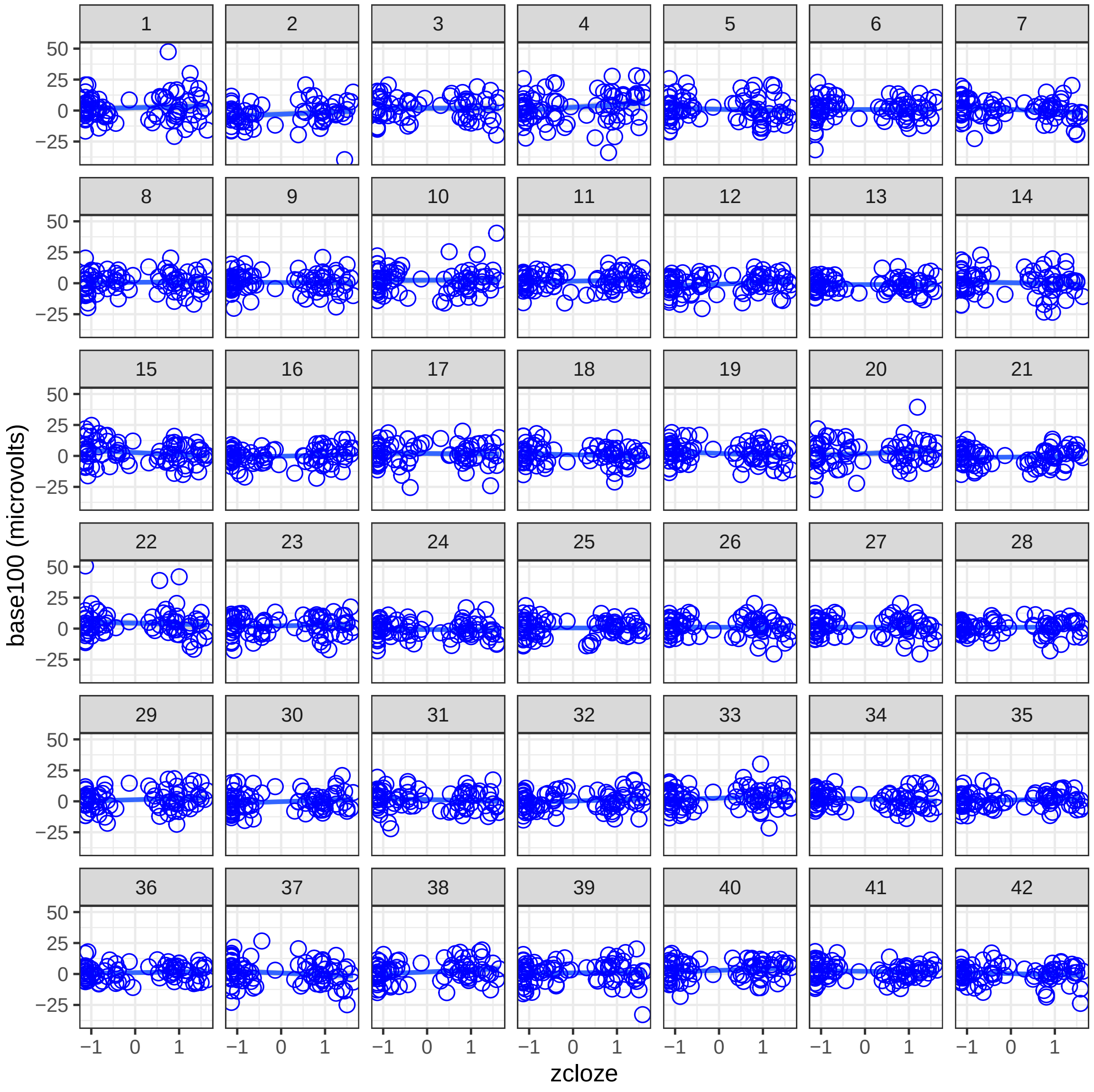
```
xyplot(base100~zcloze|item,
       current_lab,
       panel = function(x, y) {
         panel.xyplot(x, y)
         panel.abline(lm(y ~ x))
       },,main="By items data")
```



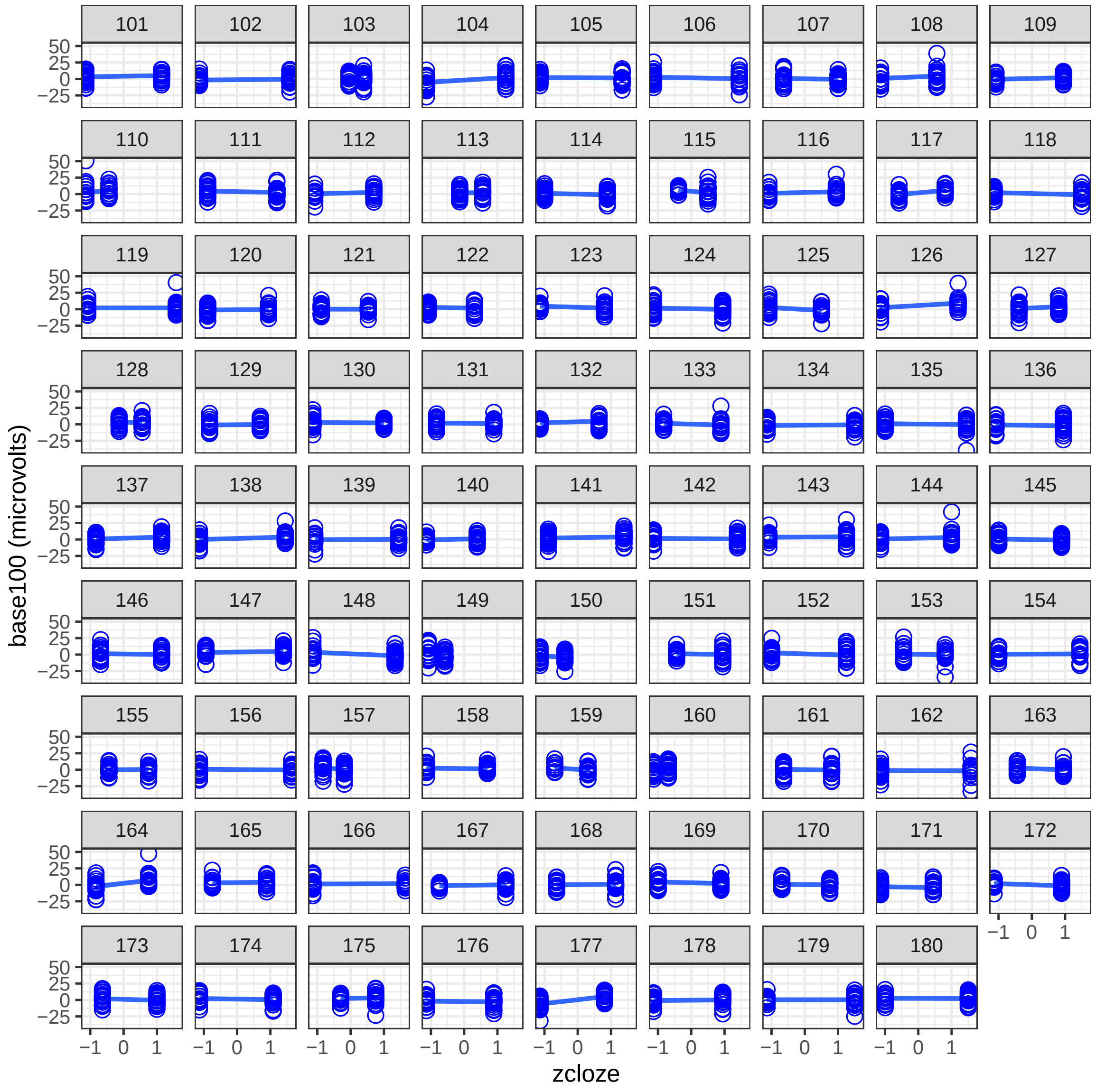
One can do this in ggplot as well. First, we define a function that mimics the lattice plot. I got this code from someone on twitter but I lost the source.

```
gg_xyplot <- function(x, y, formula, shape, size,
                      xlabel="zcloze",
                      ylabel="base100 (microvolts)",
                      data=current_lab){
    ggplot(data = data, aes(x = data[,x],
                            y = data[,y]))  +
    facet_wrap(formula) +
    geom_smooth(method="lm")+
    geom_point(color = "blue",
               shape = shape, size = size) +
    theme(panel.grid.minor = element_blank()) +
    theme_bw() +
    ylab(ylabel) +
    xlab(xlabel)
}
```

```
gg_xyplot(x = "zcloze", y = "base100",  ~ subj,
          shape = 1, size = 3,
          data = current_lab)
```



```
gg_xyplot(x = "zcloze", y = "base100",  ~ item,
          shape = 1, size = 3,
          data = current_lab)
```



One problem with the above visualizations is that we would have to plot nine labs' plots separately and not get an overall impression of the by-subject variability in the zcloze effect.

```
## fit separate linear models for each subject:
m_lmlist<-lmList(base100~zcloze|subj,dat)
lmlist_coef<-summary(m_lmlist)$coefficients
## extract by subject slopes:
slopes<-lmlist_coef[,,2]
means<-slopes[,1]
lower<-means-2*slopes[,2]
```

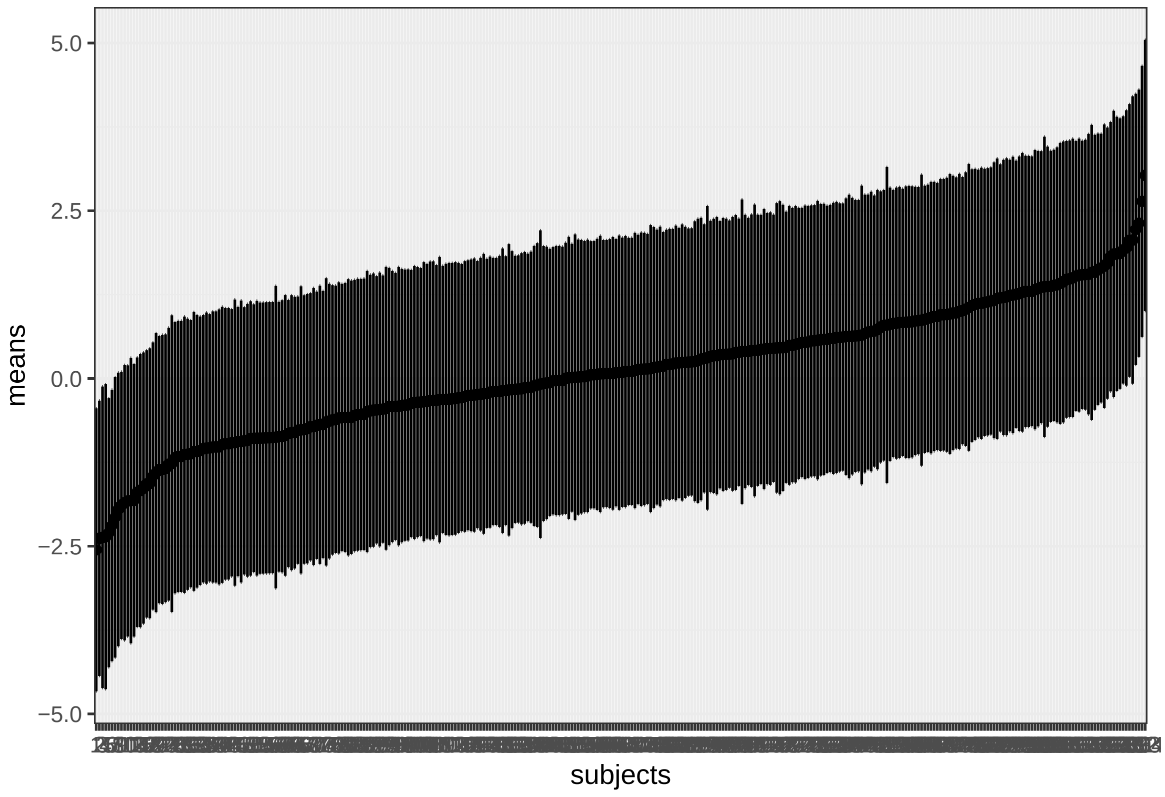
```
upper<-means+2*slopes[,2]

subjects<-unique(dat$subj)

slopes_summary<-data.frame(subj=subjects,
                           means=means,
                           lower=lower,upper=upper)

## reorder means by magnitude:
slopes_summary<-slopes_summary[order(slopes_summary$means),]

p<-ggplot(slopes_summary,aes(x=subjects, y=means)) +
  geom_line() +
  geom_point()+
  geom_errorbar(aes(ymin=lower, ymax=upper), width=.2,
                position=position_dodge(0.05))+theme_bw()
p
```



Notice that there is a lot of variability across the subjects (each subject's data is being analyzed separately). Some show negative effects, some show the expected positive effects. This kind of oscillation is characteristic of low power experiments; the overall experiment by Nieuwland et al might not be low power, but analyzing each subject separately definitely is going to lead to an underpowered design! See Gelman & Carlin (2014) for discussion on how this kind of oscillation of the effects around 0 happens.

Power analyses. In order to understand the simulation approach I take here, it is important to understand how linear mixed models work. Here, we are going to switch to the slides accompanying this file, on linear mixed models introduction.

Incidentally, in order to avoid messages about convergence failures, I am going to add the following to the lmer function:

```
control=lmerControl(calc.derivs=FALSE)
```

This is just for convenience; normally, we will have to worry about convergence problems. See Barr, Levy, Scheepers, & Tily (2013), Matuschek, Kliegl, Vasishth, Baayen, & Bates (2017), Bates, Kliegl, Vasishth, & Baayen (2015).

Computing power distribution for the current sample size.

Step 1: Extract estimates. First, we extract the estimated parameter values from the model that was fit to the whole data-set:

```
## extract estimates of fixed-effects parameters:
beta<-summary(model3a)$coefficients[,1]
## extract standard deviation estimate:
sigma_e<-attr(VarCorr(model3a),"sc")
## assemble variance covariance matrix for subjects:
subj_ranefsd<-attr(VarCorr(model3a)$subj,"stddev")
subj_ranefcorr<-attr(VarCorr(model3a)$subj,"corr")
Sigma_u<-round(SIN::sdcor2cov(stddev=subj_ranefsd,
                              corr=subj_ranefcorr),6)

## assemble variance covariance matrix for items:
item_ranefsd<-attr(VarCorr(model3a)$item,"stddev")
item_ranefcorr<-attr(VarCorr(model3a)$item,"corr")
Sigma_w<-SIN::sdcor2cov(stddev=item_ranefsd,
                              corr=item_ranefcorr)
```

Step 2: Generate simulated data using estimates, compute power (or Type I error): Load a function to generate simulated data. I explain the function in the appendix.

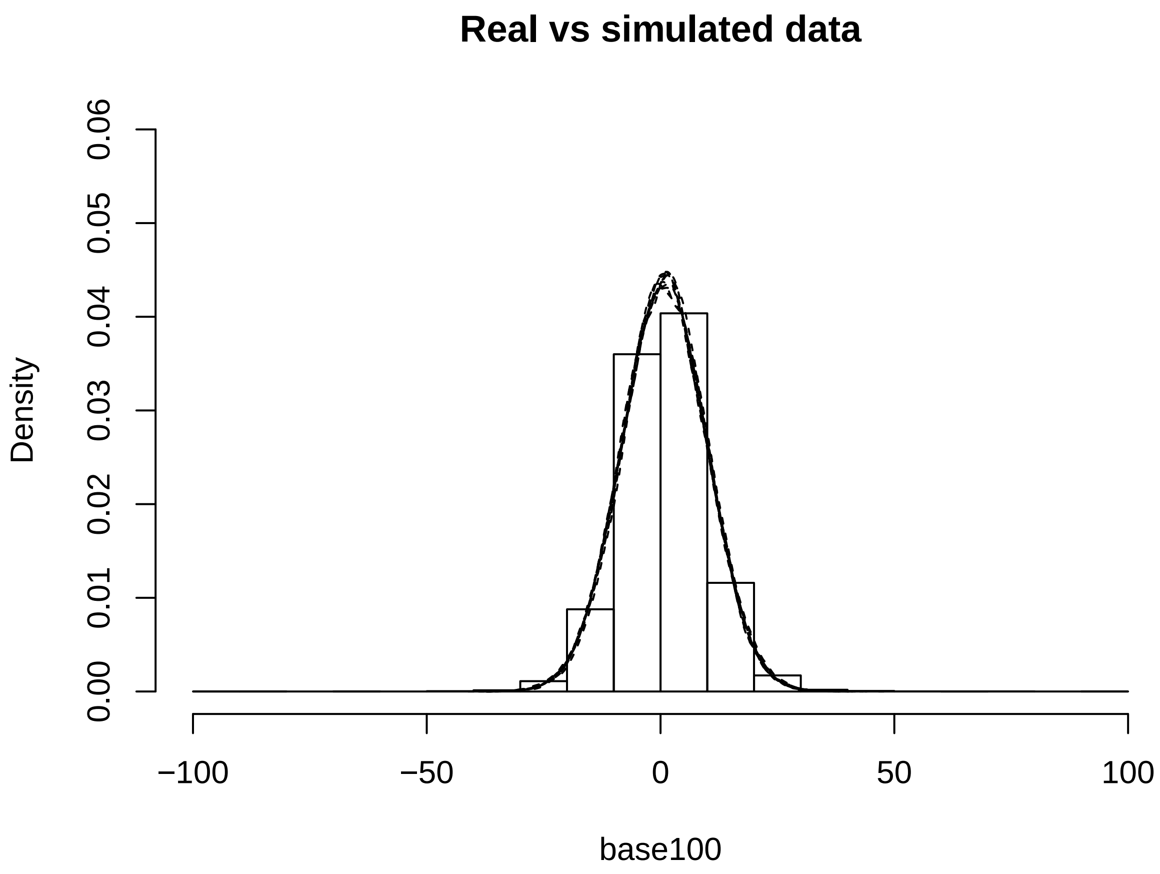
```
source("R/gen_sim_norm.R")
```

```
## Example simulated data, generated several times:
nsim<-10
sim_values<-matrix(rep(NA,nsim*dim(dat)[1]),ncol = nsim)
for(i in 1:nsim){
simdat<-gen_sim_norm(dat,
                     alpha=beta[1],beta=beta[2],
                     Sigma_u=Sigma_u,Sigma_w=Sigma_w,
                     sigma_e=sigma_e)
sim_values[,i]<-simdat$simbase100
}
```

```
head(sim_values)
```

It is useful to compare real and simulated data, to check if the generative model produces realistic data.

```
## Compare fake and simulated data:
## load precomputed simulated values:
load("data/sim_values.Rda")
## the observed data:
hist(dat$base100,freq=FALSE,
     ylim=c(0,0.06),
     main="Real vs simulated data",
     xlab="base100")
## simulated data:
for(i in 1:ncol(sim_values)){
lines(density(sim_values[,i]),lty=2)
}
```



Here, it looks like the model is producing realistic data.

A critical question here is: what should the estimate of the effect be? Normally, for the problems I study, the estimate would be derived from a computational model or from a meta-analysis of existing data (Nicenboim et al., 2019, Jäger et al. (2017), Nicenboim et al. (2018), Bürki et al. (2020)). For now, we will take the estimate (mean and SE) from the data; this is just a convenient starting point. We will improve on this later in an exercise.

```
b1<-round(summary(model3a)$coefficients[2,1],3)
b1se<-round(summary(model3a)$coefficients[2,2],3)
```

Given these estimates, we can compute the lower and upper bounds of the estimate:

```
b1-2*b1se; b1+2*b1se
```

```
## [1] -0.031
```

```
## [1] 0.245
```

We could repeatedly (100 times) simulate estimates from the distribution Normal(0.11,0.07). Each time that we take a sample, we generate simulated data 100 times, and then compute the proportion of times that the null hypothesis is rejected. This gives us 100 estimates of power; we can plot these estimates as a distribution.

But the above approach will take a lot of time, so for this demonstration, I would compute the power estimate based only on the estimated mean effect. One could also compute power based on the lower bound and upper bound of the estimated effect (i.e., mean $\pm$ 2$\times$SE).

```
## store for power:
nsim<-100
tvals<-c()
for(i in 1:nsim){
  #print(paste("i=",i,sep=""))
simdat<-gen_sim_norm(dat,
               alpha=beta[1],
               beta=b1, ## using the estimated slope
               Sigma_u=Sigma_u,Sigma_w=Sigma_w,
               sigma_e=sigma_e)
m<-lmer(simbase100~zcloze+(1+zcloze|subj)+
          (1+zcloze|item),simdat,
        control=lmerControl(calc.derivs=FALSE))
tvals[i]<-summary(m)$coefficients[2,3]
}
power_mean<-mean(abs(tvals)>2)
power_mean
```

```
## [1] 0.3
```

```
save(power_mean,file="data/power_mean.Rda")
```

Power for the estimated mean from the existing studies is 0.30.

I provide the code for computing power using the lower and upper bounds of the mean estimate, but I don't run the code here (by setting eval to FALSE), to save time.

```r
## store for power:
nsim<-100
tvals<-c()
for(i in 1:nsim){
  #print(paste("i=",i,sep=""))
simdat<-gen_sim_norm(dat,
              alpha=beta[1],
              beta=b1-2*b1se, ## using the lower bound
              Sigma_u=Sigma_u,Sigma_w=Sigma_w,
              sigma_e=sigma_e)
m<-lmer(simbase100~zcloze+(1+zcloze|subj)+(1+zcloze|item),
        simdat,
        control=lmerControl(calc.derivs=FALSE))
tvals[i]<-summary(m)$coefficients[2,3]
}
power_lower<-mean(abs(tvals)>2)
power_lower
```

```r
## store for power:
nsim<-100
tvals<-c()
for(i in 1:nsim){
  #print(paste("i=",i,sep=""))
simdat<-gen_sim_norm(dat,
              alpha=beta[1],
              beta=b1+2*b1se, ## using the upper bound
              Sigma_u=Sigma_u,Sigma_w=Sigma_w,
              sigma_e=sigma_e)
m<-lmer(simbase100~zcloze+(1+zcloze|subj)+(1+zcloze|item),
        simdat,
        control=lmerControl(calc.derivs=FALSE))
tvals[i]<-summary(m)$coefficients[2,3]
}
power_upper<-mean(abs(tvals)>2)
power_upper
```

In summary, the power estimate for the mean is 0.30. If you run the estimates for the lower and upper bound, you will find that there is a lot of uncertainty on the power estimate; this uncertainty is coming from the fact that we are unsure about the magnitude of the true effect.

Exercise. Write a function computepower that can produce a power estimate given a particular effect size. You should be able to write something like

computepower(b=0.10)

and it should return the power estimate for the effect size estimate 0.10 microvolts.

Computing sample size needed for 80% power. Next, we want to know how many subjects we will need to obtain 80% power. To do this, we will replicate our existing data frame n times, where n can 2, 3, 4, etc.

For illustration, suppose our data frame is only the Birmingham data.

We can use the data.table package to quickly replicate the data frame. Here is an example of how

```r
library(data.table)
df <- data.frame(a = c(1,2,3), b = c(1,2,3))
dt <- as.data.table(df)
n <- 3
dt[rep(dt[, .I], n)]
```

```
##    a b
## 1: 1 1
## 2: 2 2
## 3: 3 3
## 4: 1 1
## 5: 2 2
## 6: 3 3
## 7: 1 1
## 8: 2 2
## 9: 3 3
```

Select the relevant columns:

```r
birm_dat<-dplyr::select(current_lab,
                        subj,item,zcloze)
head(birm_dat)
```

```
##   subj item   zcloze
## 1    1  102 -1.13318
## 2    1  113 -0.13494
## 3    1  111  1.19605
## 4    1  103  0.39191
## 5    1  105 -1.13318
## 6    1  101 -1.13318
```

Generate 3 replicates of the data frame for generating a larger set of simulated data. Note that one has to update the subject ids.

```r
birm_dat<-as.data.table(birm_dat)
subjid<-birm_dat$subj
n <- 3
repl_dat<-birm_dat[rep(birm_dat[, .I], n)]
dim(birm_dat)[1]*3
```

```
## [1] 9858
```

```
dim(repl_dat)[1]
```

```
## [1] 9858
```

```
## create new subject vector extended n times
add_id<-rep(seq(100,100*n,by=100),each=dim(birm_dat)[1])
repl_dat$subj<-rep(as.numeric(as.character(birm_dat$subj)),n)+add_id
head(repl_dat)
```

```
##    subj item   zcloze
## 1:  101  102 -1.13318
## 2:  101  113 -0.13494
## 3:  101  111  1.19605
## 4:  101  103  0.39191
## 5:  101  105 -1.13318
## 6:  101  101 -1.13318
```

```
length(unique(birm_dat$subj))*3
```

```
## [1] 126
```

```
length(unique(repl_dat$subj))
```

```
## [1] 126
```

Then we simulate data to compute power for a given sample size:

```
nsim<-100
tvals<-c()
for(i in 1:nsim){
simdat<-gen_sim_norm(repl_dat,
               alpha=beta[1],
               beta=b1, ## using the estimated slope
               Sigma_u=Sigma_u,Sigma_w=Sigma_w,
               sigma_e=sigma_e)
m<-lmer(simbase100~zcloze+(1+zcloze|subj)+(1+zcloze|item),simdat,
        control=lmerControl(calc.derivs=FALSE))
tvals[i]<-summary(m)$coefficients[2,3]
}
## number of subjects:
length(unique(simdat$subj))
```

```
## [1] 126
```

```
power_mean<-mean(abs(tvals)>2)
power_mean
```

```
## [1] 0.16
```

Obviously, 126 subjects is too little here! We will need a *lot* more to achieve 80% power! One can now re-run the above code with increasing numbers of subjects to get a power curve as a function of the subject sample size.

We can write a function to simplify the process of computing power as a function of subject sample size.

```
compute_power<-function(dat=birm_dat,
                        replicates=3,nsims=100){
dat<-as.data.table(dat)
subjid<-dat$subj
nrep <- replicates
repl_dat<-dat[rep(dat[, .I], nrep)]

## create new subject vector extended n times
add_id<-rep(seq(100,100*nrep,by=100),each=dim(dat)[1])
repl_dat$subj<-rep(as.numeric(as.character(birm_dat$subj)),
                   nrep)+add_id
nsim<-nsims
tvals<-c()
for(i in 1:nsim){
simdat<-gen_sim_norm(repl_dat,
               alpha=beta[1],
               beta=b1, ## using the estimated slope
               Sigma_u=Sigma_u,Sigma_w=Sigma_w,
               sigma_e=sigma_e)
m<-lmer(simbase100~zcloze+(1+zcloze|subj)+(1+zcloze|item),
        simdat,
        control=lmerControl(calc.derivs=FALSE))
tvals[i]<-summary(m)$coefficients[2,3]
}
power_mean<-mean(abs(tvals)>2)
## return result with sample size:
res<-c(length(unique(simdat$subj)),
                power_mean)
res
}
```

```
repl4<-compute_power(dat=birm_dat,
              replicates = 4,
              nsims=10)
repl4
```

```
## [1] 168.0   0.2
```

```
repl6<-compute_power(dat=birm_dat,
              replicates = 6,
              nsims=10)
repl6
```

```
## [1] 252.0   0.3
```

Now, with the above code, in principle we can display the effect of increasing subject sample size on power. Here is a simplified version of such a table:

```
results<-rbind(repl4,repl6)
results<-data.frame(results)
colnames(results)<-c("nsubj","power estimate")
results
```

```
##       nsubj power estimate
## repl4   168            0.2
## repl6   252            0.3
```

Don't be surprised if the power doesn't go up much with sample size, or if it even goes down with increasing sample size. Because power is being computed using simulated data, we will see some variation in the power calculation from one simulation to another.

Exercise (to be done after the workshop)

This exercise should be done overnight! Compute power for 2,4,6,8,10 replicates of the Birmingham data, (use 100 simulations, not 10) and plot power against sample size.

Some comments on power

Note here that power computed from the data you already have is called **post-hoc power**, and is pointless to compute; this is because once the p-value is known, power can be computed analytically (Hoenig & Heisey, 2001). In other words, post-hoc power is just a transform of the observed p-value.

What we are doing here is using existing data to estimate **variance components**. Then, we use simulation to compute a power *distribution* for a range of plausible values of the effect of interest.

The key problem here is figuring out what a plausible range of values of the effect is. Here, we have simply used all the available data to get an estimate of the effect. This makes our power analysis look like a post-hoc power calculation. But I used the observed estimates of the effect here just as an example.

In realistic data, a better way to obtain estimates of the effect is by doing a meta-analysis. See Nicenboim et al. (2019) for an estimate based on a meta-analysis. The estimate based on all publicly available data is: 0.11 microvolts, 95% credible intervals [0.05,0.16]. One can use that to compute power.

Exercise. Use the existing data and the above meta-analysis estimates to work out estimates of power for a sample size of 334 subjects. Use the mean, lower, and upper bounds of the meta-analysis estimate to work out the power estimates.

Suggestions for making data and code public

- Develop and release a data management plan (required by the DFG). Example: https://osf.io/f9dqk/
- Use R Markdown to produce commented and compiled code output so that the reader doesn't need to re-run code.
- Once the analysis is complete, put all the code and data on OSF or github (or link github to osf). Example: https://github.com/vasishth/StatSigFilter
- Make sure you don't depend on data and functions loaded via a hidden .Rprofile file that is local to your computer.
- Avoid Excel files for storing data that will be loaded into R; save data in .csv or .txt files.
- The directory structure for the data release should have a structure that the outsider can easily work out where everything is. The structure can be based on R package structures (discussed below), but doesn't have to be.
- Create a README file that users can look at for guidance. Sometimes names of columns in data-frames are not easy to interpret. E.g., in eyetracking data, the user cnanot know whether RP is regression probability or regression path duration.

- At the end of the R Markdown file, provide session information using SessionInfo().
- Make sure that your R Markdown file runs! Ideally, download all the code and data onto another machine and check that it runs.
- If you have very large files that can't be saved on github, osf, etc., then store them on a (university) server and link to the files. Example: https://osf.io/reavs/
- Fitting complicated models can lead to problems when compiling an Rmd file. One way around that problem is to save the results (or a summary) of the resulting model offline and to load it when needed.
- One irritating problem with R Markdown is that the error messages on compilation failure are often not very informative. One solution to this is to incrementally build the file, using separate "child" files. The R Markdown online documentation explains how to do this. Despite the frustration that R Markdown causes, the long-term advantages of using it are great.

Appendix 1: Explanation of the function in gen_sim_norm.R

The way I generate data is by traversing the observed data's data frame row by row, and then generating simulated data for each row, using the following pieces of information:

- the subject id
- the item id
- the parameter estimates from the model that was fit on the whole data-set

This section presuppose knowledge of how a linear mixed model is "assembled". The underlying model is as follows.

i indexes subjects, j items.

$$y_{ij} = \alpha + u_{0i} + w_{0j} + (\beta + u_{1i} + w_{1j}) * zcloze_{ij} + \varepsilon_{ij} \tag{1}$$

where $\varepsilon_{ij} \sim Normal(0, \sigma)$ and

$$\Sigma_u = \begin{pmatrix} \sigma_{u0}^2 & \rho_u \sigma_{u0} \sigma_{u1} \\ \rho_u \sigma_{u0} \sigma_{u1} & \sigma_{u1}^2 \end{pmatrix} \quad \Sigma_w = \begin{pmatrix} \sigma_{w0}^2 & \rho_w \sigma_{w0} \sigma_{w1} \\ \rho_w \sigma_{w0} \sigma_{w1} & \sigma_{w1}^2 \end{pmatrix} \tag{2}$$

$$\begin{pmatrix} u_0 \\ u_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_u \right), \quad \begin{pmatrix} w_0 \\ w_1 \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \Sigma_w \right) \tag{3}$$

Given such a mathematical model, we can generate simulated data row by row, given subject and item ids, and given point value estimates for

- α
- β
- σ
- $\sigma_{u0}, \sigma_{u1}, \sigma_{w0}, \sigma_{w1}$
- ρ_u, ρ_w

The function first creates subject and item random effects:

```
nsubj<-length(unique(dat$subj))
nitem<-length(unique(dat$item))
u<-mvrnorm(n=nsubj, # number of subjects
             mu=c(0,0),Sigma=Sigma_u)
w<-mvrnorm(n=nitem, # number of items
             mu=c(0,0),Sigma=Sigma_w)

## add subject and item id's:
u<-data.frame(subjid=unique(dat$subj),u)
w<-data.frame(itemid=unique(dat$item),w)
```

Each row comtains randomly generated intercept and slope adjustments for each of the subjects (items):

```
head(u)
```

```
##   subjid X.Intercept.      zcloze
## 1      1      0.98346 -8.9290e-04
## 2      2      0.94825  1.4206e-03
## 3      3      0.20257  8.6194e-05
## 4      4      1.95450 -1.1303e-03
## 5      5     -1.16006 -5.4569e-05
## 6      6     -0.19090  1.1947e-03
```

```
head(w)
```

```
##   itemid X.Intercept.    zcloze
## 1    102    -1.107248 -0.203715
## 2    113    -0.792191  0.246479
## 3    111    -0.137405  0.294321
## 4    103     0.609060 -0.349255
## 5    105     2.019478  0.391643
## 6    101     0.063671 -0.043894
```

If we want to know what subject 1's intercept adjustment is, we write:

```
u[1,2]
```

```
## [1] 0.98346
```

If we want to know what subject 1's slope adjustment is, we write:

```
u[1,3]
```

```
## [1] -0.0008929
```

Now, for the first row of the data frame `dat`:

```
dat[1,]
```

```
##   subject item  lab  zcloze base100 subj
## 1  birm01  102 birm -1.1332   -2.33    1
```

we can produce randomly generated data as follows. In this row, we have subject id 1, and item id 102. This can be extracted from the data frames u and w.

```
## intercept adjustment:
u[u$subjid==1,2]
```

```
## [1] 0.98346
```

```
## slope adjustment:
u[u$subjid==1,3]
```

```
## [1] -0.0008929
```

These simply need to be added to the fixed effects intercept and slope values:

```
beta[1] + u[u$subjid==1,2] +
  w[w$itemid==102,2]+
  (beta[2] +
     u[u$subjid==1,3] +
     w[w$itemid==102,3])*dat[1,]$zcloze +
  rnorm(1,0,sigma_e)
```

```
##       [,1]
## [1,] 7.072
```

Now, if we repeat this for every row of the data-frame, we can produce simulated data. Obviously, we cannot write the subject and item id by hand for each row. But we can automatically extract it from the data frame `dat`:

```
dat[1,]$subj
```

```
## [1] 1
## 334 Levels: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ... 334
```

```
dat[1,]$item
```

```
## [1] 102
## 80 Levels: 101 102 103 104 105 106 107 108 109 110 111 ... 180
```

Thus, for each row, we can extract the subject and item id by indexing the row id with i ranging from 1 to the total number of rows N, and then plugging in that value into the equation below. I show what happens with i=1:

```
i<-1
current_subjid<-dat[i,]$subj
current_itemid<-dat[i,]$item

beta[1] + u[u$subjid==current_subjid,2] +
  w[w$itemid==current_itemid,2]+
  (beta[2] + u[u$subjid==current_subjid,3] +
      w[w$itemid==current_itemid,3])*dat[i,]$zcloze + rnorm(1,0,sigma_e)
```

```
##       [,1]
## [1,] 17.07
```

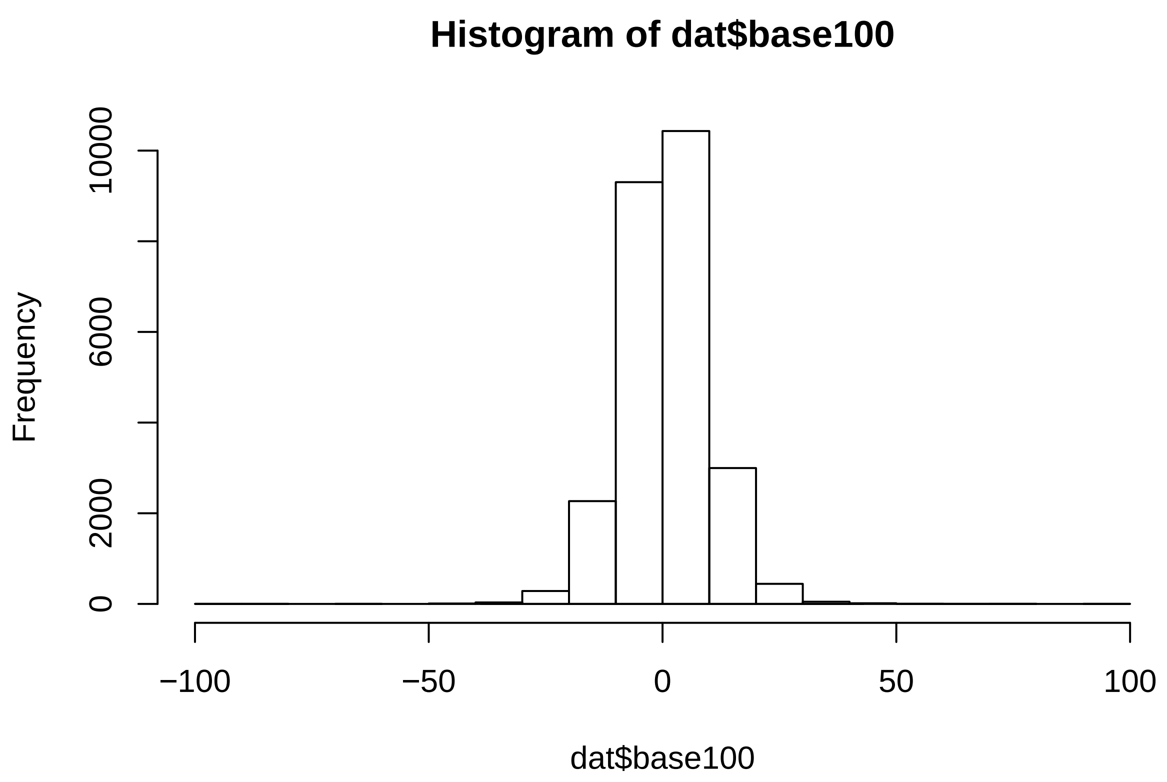
Now, if we change the index i to 2 (this is an exercise for you), we get the simulated value for row 2.

One can now generalize this, and write a for-loop that traverses every row of the data frame:

```
## number of rows
N<-dim(dat)[1]
# save vector of simulated data:
simbase100 <- rep(NA,N)
for(i in 1:N){
current_subjid<-dat[i,]$subj
current_itemid<-dat[i,]$item
simbase100[i] <-  beta[1] + u[u$subjid==current_subjid,2] +
  w[w$itemid==current_itemid,2]+
  (beta[2] + u[u$subjid==current_subjid,3] +
      w[w$itemid==current_itemid,3])*dat[i,]$zcloze + rnorm(1,0,sigma_e)
}
```

These data look pretty similar to the observed data:

```
hist(dat$base100)
lines(density(simbase100))
```



Why I don't use ready-made packages for computing power. There are ready-made packages in R that generate simulated data from a fitted lmer model. However, I never teach how to use those packages, because the procedure I show you above helps the researcher appreciate the underlying generative process that produced the data. It also allows for more flexible variants on the above approach. For example, if we think like a Bayesian and have a prior distribution on each parameter, we can sample from a prior distribution for the effect size or indeed for any of the variance components. This only takes a small modification of the above code and data. With a ready-made package, you only get what the package developer decided to allow you to do.

Session information

```
sessionInfo()
```

```
## R version 3.6.2 (2019-12-12)
## Platform: x86_64-apple-darwin15.6.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/3.6/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods
## [7] base
##
## other attached packages:
##  [1] data.table_1.12.8 MASS_7.3-51.5     lattice_0.20-38
##  [4] forcats_0.4.0     stringr_1.4.0     dplyr_0.8.3
##  [7] purrr_0.3.3       readr_1.3.1       tidyr_1.0.0
## [10] tibble_2.1.3      tidyverse_1.3.0   lme4_1.1-21
## [13] Matrix_1.2-18     brms_2.10.0       Rcpp_1.0.3
## [16] papaja_0.1.0.9842 ggplot2_3.2.1
##
## loaded via a namespace (and not attached):
##   [1] minqa_1.2.4          colorspace_1.4-1
##   [3] ggridges_0.5.1       rsconnect_0.8.16
##   [5] markdown_1.1         fs_1.3.1
##   [7] base64enc_0.1-3      rstudioapi_0.10
##   [9] farver_2.0.1         rstan_2.19.2
##  [11] DT_0.11              fansi_0.4.0
##  [13] lubridate_1.7.4      xml2_1.2.2
##  [15] bridgesampling_0.7-2 splines_3.6.2
##  [17] knitr_1.26           shinythemes_1.1.2
##  [19] zeallot_0.1.0        bayesplot_1.7.1
##  [21] jsonlite_1.6         nloptr_1.2.1
##  [23] broom_0.5.3.9000     dbplyr_1.4.2
##  [25] shiny_1.4.0          compiler_3.6.2
##  [27] httr_1.4.1           backports_1.1.5
##  [29] assertthat_0.2.1     fastmap_1.0.1
##  [31] lazyeval_0.2.2       cli_2.0.0
##  [33] later_1.0.0          htmltools_0.4.0
##  [35] prettyunits_1.0.2    tools_3.6.2
##  [37] igraph_1.2.4.2       coda_0.19-3
##  [39] gtable_0.3.0         glue_1.3.1
##  [41] reshape2_1.4.3       cellranger_1.1.0
##  [43] vctrs_0.2.1          nlme_3.1-143
##  [45] crosstalk_1.0.0      xfun_0.11
##  [47] ps_1.3.0             rvest_0.3.5
##  [49] mime_0.8             miniUI_0.1.1.1
##  [51] lifecycle_0.1.0      gtools_3.8.1
##  [53] zoo_1.8-6            scales_1.1.0
##  [55] colourpicker_1.0    hms_0.5.3
##  [57] promises_1.1.0      Brobdingnag_1.2-6
##  [59] parallel_3.6.2      inline_0.3.15
##  [61] shinystan_2.5.0     yaml_2.2.0
##  [63] gridExtra_2.3       loo_2.2.0
##  [65] StanHeaders_2.19.0  stringi_1.4.3
##  [67] dygraphs_1.1.1.6    boot_1.3-24
##  [69] pkgbuild_1.0.6      rlang_0.4.2
##  [71] pkgconfig_2.0.3     matrixStats_0.55.0
##  [73] evaluate_0.14       labeling_0.3
##  [75] rstantools_2.0.0    htmlwidgets_1.5.1
##  [77] processx_3.4.1      tidyselect_0.2.5
##  [79] plyr_1.8.5          magrittr_1.5
##  [81] bookdown_0.16       R6_2.4.1
##  [83] generics_0.0.2      DBI_1.1.0
##  [85] pillar_1.4.3        haven_2.2.0
##  [87] withr_2.1.2         xts_0.11-2
##  [89] abind_1.4-5         modelr_0.1.5
##  [91] crayon_1.3.4        rmarkdown_2.0
##  [93] grid_3.6.2          readxl_1.3.1
##  [95] SIN_0.6             callr_3.4.0
##  [97] threejs_0.3.1       reprex_0.3.0
##  [99] digest_0.6.23       xtable_1.8-4
## [101] httpuv_1.5.2        stats4_3.6.2
## [103] munsell_0.5.0       shinyjs_1.0
```

References

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278.

Bates, D. M., Kliegl, R., Vasishth, S., & Baayen, H. (2015). *Parsimonious mixed models.*

Bürki, A., Elbuy, S., Madec, S., & Vasishth, S. (2020). *Picture naming in the context of semantically related distractors: A Bayesian meta-analysis 40 years after Lupker's first reaction time study.*

DeLong, K., Urbach, T., & Kutas, M. (2005). Probabilistic word pre-activation during language comprehension inferred from electrical brain activity. *Nature Neuroscience*, *8*(8), 1117–1121.

Gelman, A., & Carlin, J. (2014). Beyond power calculations assessing type S (sign) and type M (magnitude) errors. *Perspectives on Psychological Science*, *9*(6), 641–651.

Hoenig, J. M., & Heisey, D. M. (2001). The abuse of power: The pervasive fallacy of power calculations for data analysis. *The American Statistician*, *55*(1), 19–24.

Jäger, L. A., Engelmann, F., & Vasishth, S. (2017). Similarity-based interference in sentence comprehension: Literature review and Bayesian meta-analysis. *Journal of Memory and Language*, *94*, 316–339. https://doi.org/10.1016/j.jml.2017.01.004

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, R. H., & Bates, D. M. (2017). Balancing Type I Error and Power in Linear Mixed Models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Nicenboim, B., Roettger, T. B., & Vasishth, S. (2018). Using meta-analysis for evidence synthesis: The case of incomplete neutralization in German. *Journal of Phonetics*, *70*, 39–55. https://doi.org/https://doi.org/10.1016/j.wocn.2018.06.001

Nicenboim, B., Vasishth, S., & Rösler, F. (2019). *Are words pre-activated probabilistically during sentence comprehension? Evidence from new data and a Bayesian random-effects meta-analysis using publicly available data.*

Nieuwland, M. S., Politzer-Ahles, S., Heyselaar, E., Segaert, K., Darley, E., Kazanina, N., . . . others. (2018). Large-scale replication study reveals a limit on probabilistic prediction in language comprehension. *eLife*, *7*, e33468.

Barr, D. J., Levy, R., Scheepers, C., & Tily, H. J. (2013). Random effects structure for confirmatory hypothesis testing: Keep it maximal. *Journal of Memory and Language*, *68*(3), 255–278.

Bates, D. M., Kliegl, R., Vasishth, S., & Baayen, H. (2015). *Parsimonious mixed models.*

Bürki, A., Elbuy, S., Madec, S., & Vasishth, S. (2020). *Picture naming in the context of semantically related distractors: A Bayesian meta-analysis 40 years after Lupker's first reaction time study.*

DeLong, K., Urbach, T., & Kutas, M. (2005). Probabilistic word pre-activation during language comprehension inferred from electrical brain activity. *Nature Neuroscience*, *8*(8), 1117–1121.

Gelman, A., & Carlin, J. (2014). Beyond power calculations assessing type S (sign) and type M (magnitude) errors. *Perspectives on Psychological Science*, *9*(6), 641–651.

Hoenig, J. M., & Heisey, D. M. (2001). The abuse of power: The pervasive fallacy of power calculations for data analysis. *The American Statistician*, *55*(1), 19–24.

Jäger, L. A., Engelmann, F., & Vasishth, S. (2017). Similarity-based interference in sentence comprehension: Literature review and Bayesian meta-analysis. *Journal of Memory and Language*, *94*, 316–339. https://doi.org/10.1016/j.jml.2017.01.004

Matuschek, H., Kliegl, R., Vasishth, S., Baayen, R. H., & Bates, D. M. (2017). Balancing Type I Error and Power in Linear Mixed Models. *Journal of Memory and Language*, *94*, 305–315. https://doi.org/10.1016/j.jml.2017.01.001

Nicenboim, B., Roettger, T. B., & Vasishth, S. (2018). Using meta-analysis for evidence synthesis: The case of incomplete neutralization in German. *Journal of Phonetics*,

70, 39–55. https://doi.org/https://doi.org/10.1016/j.wocn.2018.06.001

Nicenboim, B., Vasishth, S., & Rösler, F. (2019). *Are words pre-activated probabilistically during sentence comprehension? Evidence from new data and a Bayesian random-effects meta-analysis using publicly available data.*

Nieuwland, M. S., Politzer-Ahles, S., Heyselaar, E., Segaert, K., Darley, E., Kazanina, N., ... others. (2018). Large-scale replication study reveals a limit on probabilistic prediction in language comprehension. *eLife*, *7*, e33468.