Introduction to statistics: Simulating data

Shravan Vasishth

Universität Potsdam
vasishth@uni-potsdam.de
http://www.ling.uni-potsdam.de/∼vasishth

April 9, 2020

Why simulation is important

We will need to simulate data to

- Understand the power properties of our experiment design. This is important for
  - deciding on sample sizes needed
  - deciding whether we are in a Type M error situation when we get a significant result (can we take the sig. result seriously?).
- Understand which parameters can in principle be recovered under repeated sampling (model selection)

Two simulation scenarios

We will consider two scenarios:

- Between-subject designs: This is just to get you used to simulation. We rarely use this in practice.
- Within-subject designs: This is where all the interesting action is for us, as we will almost always run repeated measures designs.

Between-subjects example

- Suppose we have reading time data in milliseconds from 10 subjects who see subject relative clause sentences, and a *different* set of 10 subjects who see object relative clauses.
- Assume that (a) the standard deviation is 300 ms, (b) the true subject relative reading time is 700 ms and (c) the true object relative reading time is 750 ms.
- Thus, the true difference between subject relatives and object relatives is 50 ms.

How to simulate such data?

Between-subjects example

Create a simulated data-set like this:
Take 10 samples from $Normal(\mu = 700, \sigma = 300)$
Take 10 samples from $Normal(\mu = 750, \sigma = 300)$
Create a data frame by adding a subject id.

Between-subjects example

```
sr<-rnorm(10,mean=700,sd=300)
or<-rnorm(10,mean=750,sd=300)
subj<-1:20
cond<-rep(c("sr","or"),each=10)
sim_dat<-data.frame(subj=subj,cond=cond,rt=c(sr,or))
head(sim_dat)
```

```
##    subj cond       rt
## 1     1   sr 1188.937
## 2     2   sr   56.974
## 3     3   sr  798.427
## 4     4   sr  522.021
## 5     5   sr  735.388
## 6     6   sr  877.154
```

Between-subjects example

Question: How can we repeatedly generate such data?
Answer: write a function (I set some default values here).

```
betwsubj_simdat<-function(n=10,## no. subjs in each cond.
                          mean1=700,stddev=300,
                          mean2=750){
sr<-rnorm(n,mean=mean1,sd=stddev)
or<-rnorm(n,mean=mean2,sd=stddev)
subj<-1:(2*n)
cond<-factor(rep(c("sr","or"),each=n))
sim_dat<-data.frame(subj=subj,cond=cond,rt=c(sr,or))
sim_dat
}
```

Between-subjects example

Now you can repeatedly generate simulated data:

```
for(i in 1:100){
  sim_dat<-betwsubj_simdat()
}
```

Why would you want to do that? For power analysis, and to check if our model recovers the parameters correctly under repeated runs of the experiment (with new simulated subjects each time).

Between-subjects example of power analysis

Analytically:

```
round(power.t.test(delta=50,n=10,
                   sd=300,type="two.sample",
                   alternative="two.sided",
                   strict=TRUE)$power,3)

## [1] 0.064
```

Between-subjects example of power analysis

Using simulation:

```
nsim<-100000
## critical t-value:
crit_t<- qt(0.975,df=18)
## observed t-values:
obs_t<-rep(NA,nsim)

for(i in 1:nsim){
sim_dat<-betwsubj_simdat()
obs_t[i]<-t.test(rt~cond,paired=FALSE,sim_dat)$statistic
}
```

Between-subjects example of power analysis

```
pow<-mean(abs(obs_t)>abs(crit_t))
round(pow,2)

## [1] 0.06
```

Lesson learnt: for simple between-subject designs, you can use the analytical approach or simulation.

Between-subjects example using linear models

Next, we will learn how to compute power using simulation in **between**-subjects designs using the linear modeling framework (as opposed to the t.test)
After that, will learn how to compute power using simulation in **within**-subjects designs using t.test and lmer.

Between-subjects example using linear models

Notice that the default contrast coding is 0,1 treatment coding:

```
sim_dat<-betwsubj_simdat()
contrasts(sim_dat$cond)

##    sr
## or  0
## sr  1
```

Between-subjects example using linear models

Recode to sum contrasts:

```
## Method 1:
contrasts(sim_dat$cond)<-contr.sum(2)
contrasts(sim_dat$cond)

##    [,1]
## or    1
## sr   -1

##  Method 2: (my personal preference)
sim_dat$rctyp<-ifelse(sim_dat$cond=="or",1,-1)
```

Between-subjects example using linear models

Recall that the two-sample t-test (and ANOVA) and the linear model are exactly the same thing:

```
##  these are all the same test:
t.test(rt~cond,paired=FALSE,sim_dat)$statistic

##      t
## -2.444
```

Between-subjects example using linear models

```
anova(m0<-lm(rt~cond,sim_dat))

## Analysis of Variance Table
##
## Response: rt
##           Df  Sum Sq Mean Sq F value Pr(>F)
## cond       1  392195  392195    5.97  0.025
## Residuals 18 1181828   65657
```

Between-subjects example using linear models

```
summary(m0<-lm(rt~cond,sim_dat))

##
## Call:
## lm(formula = rt ~ cond, data = sim_dat)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -485.6 -139.8  -14.7  120.7  571.7
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)    765.6       57.3   13.36  8.8e-11
## cond1         -140.0       57.3   -2.44    0.025
##
```

Between-subjects example using linear models

We can write the linear model as follows.
For every row i in the data frame dat above, the reading time rt is generated from a model where the contrast coding for the condition column in the i-th row of the data is either -1 or $+1$, depending on whether we are looking at subject or object relatives.
$\varepsilon \sim Normal(0, 300)$.

$$rt_i = 725 + 25 \times cond_i + \varepsilon_i \quad (1)$$

Now, for the first 10 rows of the data frame, we have only SRs, which are coded -1, and for the second 10 rows, we have only ORs, which are coded $+1$.

Between-subjects example using linear models

So, the above definition of the linear model is basically expressing how the reading time rt in each row in the data frame is assumed to be generated:

row	Intercept	slope + error
1	725× 1	+25× (-1) + ε_1
2	725× 1	+25×(-1) + ε_2
3	725× 1	+25×(-1) + ε_3
4	725× 1	+25×(-1) + ε_4
5	725× 1	+25×(-1) + ε_5
6	725× 1	+25×(-1) + ε_6
7	725× 1	+25×(-1) + ε_7
8	725× 1	+25×(-1) + ε_8
9	725× 1	+25×(-1) + ε_9
10	725× 1	+25×(-1) + ε_{10}
11	725× 1	+25×(+1) + ε_{11}
12	725× 1	+25×(+1) + ε_{12}
13	725× 1	+25×(+1) + ε_{13}
14	725× 1	+25×(+1) + ε_{14}
15	725× 1	+25×(+1) + ε_{15}
16	725× 1	+25×(+1) + ε_{16}
17	725× 1	+25×(+1) + ε_{17}
18	725× 1	+25×(+1) + ε_{18}
19	725× 1	+25×(+1) + ε_{19}
20	725× 1	+25×(+1) + ε_{20}

Between-subjects example using linear models

The model m0 is giving us an estimate of the adjustment to the grand mean needed to obtain OR or SR processing cost (700$\pm$ 25). ε is a random variable with pdf Normal(0,300), and is responsible for the noisiness (variability) in the data.
You can decide whether to reject the null hypothesis in model m0 by looking at the p-value. Recall that the null hypothesis is that OR and SR processing cost has no difference:
$H_0 : \mu_{SR} - \mu_{OR} = 0$. Here is how you extract the p-value:

```
summary(m0)$coefficients

##              Estimate Std. Error t value   Pr(>|t|)
## (Intercept)    765.57     57.296  13.362 8.7794e-11
## cond1         -140.03     57.296  -2.444 2.5049e-02

## get the second row and fourth column of the output:
```

Between-subjects example using linear models

You can now do a power analysis of the above experiment design using the linear modeling framework instead of the t-test. Here is how to do it:

```
nsim<-10000
pvals<-rep(NA,nsim)
for(i in 1:nsim){
## create random data
## fit linear model
## extract p-value and save in pvals vector
}
## check proportion of times the pvals are less
## than 0.05. That's your power.
```

Stop now and try to do this yourself before moving to the next slide.

Between-subjects example using linear models

```
nsim<-10000
pvals<-rep(NA,nsim)
for(i in 1:nsim){
 sim_dat<-betwsubj_simdat()
 m<-lm(rt~cond,sim_dat)
 pvals[i]<-summary(m)$coefficients[2,4]
}
round(mean(pvals<0.05),2)

## [1] 0.06
```

Between-subjects parameter recovery

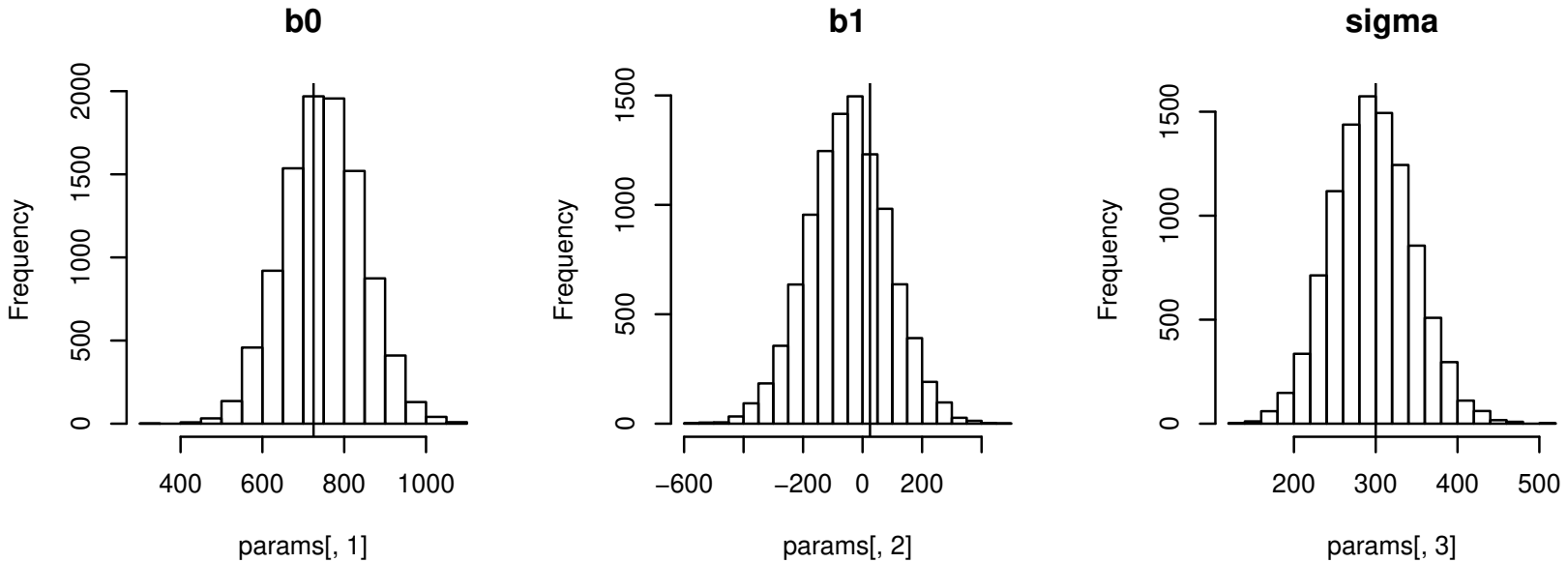
We can also check whether the model can recover the true parameterrs under repeated sampling.
This is an important check of the model's validity; this isn't normally done in frequentist courses, but it's a very important and useful tool for model selection, as you will soon see.

```
nsim<-10000
params<-matrix(rep(NA,nsim*3),ncol = 3)

for(i in 1:nsim){
  sim_dat<-betwsubj_simdat()
  m<-lm(rt~cond,sim_dat)
  params[i,c(1,2)]<-summary(m)$coefficients[,1]
  params[i,3]<-summary(m)$sigma
}
```

Between-subjects parameter recovery

Parameter recovery is pretty good:



Between-subjects parameter recovery

What this good parameter recovery is telling us is that the model can in principle provide accurate estimates of the true parameter values, under repeated sampling.

Later you will see examples of models that fail to achieve this, for some abstract parameters.

Generating within-subjects data

Now suppose we have a **total** of 10 subjects, and we show each subject SRs and ORs once each. Now the subject id's become important!

Generating within-subjects data

This is how we could try to generate data now:

```
## number of subjects
n<-10
sr<-round(rnorm(n,mean=700,sd=300))
or<-round(rnorm(n,mean=750,sd=300))
subj<-rep(1:n,2)
cond<-factor(rep(c("sr","or"),each=n))
sim_dat<-data.frame(subj=subj,cond=cond,rt=c(sr,or))
contrasts(sim_dat$cond)<-contr.sum(2)
```

Generating within-subjects data

```
head(sim_dat)

##   subj cond  rt
## 1    1   sr 719
## 2    2   sr 833
## 3    3   sr 794
## 4    4   sr 668
## 5    5   sr 527
## 6    6   sr 952
```

Generating within-subjects data

Notice that the subject column now repeats the subject id from 1 to n (whatever n is) for SRs and ORs. The total number of subjects is a total of n now, not $2 \times n$ like before.
But these generated data do not reflect the fact that we have within-subjects data! The reason: we are incorrectly assuming that SR and OR reading times are independent! The SR and OR data are paired and therefore dependent because they involve the same subjects.
How to induce dependency in the SR and OR data?

Generating within-subjects data

Let us start by assuming that **each subject has a different true grand mean processing time**. We will use this assumption to induce a dependency.
One way to encode the assumption that each subject has a different processing time is to assume that

- some subjects have mean processing time that is greater than the grand mean 725
- other subjects have a mean processing time that is smaller than 725
- some subjects have a mean processing time that is the same as 725

Generating within-subjects data

We can generate random adjustments to the grand mean of 725 by subject by creating a vector of n scores from a random variable that has mean 0 and some standard deviation (I assume 200 here for illustration):

```
## grand mean processing cost of each subject:
subject_adj<-round(rnorm(n,mean=0,sd=200))
subject_adj

##  [1] -228  -83 -179  161   23  349   76  121 -129 -128
```

Here, if a subject has a value of 0 in the by-subject adjustments, this implies that the true grand mean reading time of the subject is 725 ms.

Generating within-subjects data

Now we see that some subjects are faster than 725 ms and some are slower:

```
round(725+subject_adj)
```

```
##   [1]  497  642  546  886  748 1074  801  846  596  597
```

Generating within-subjects data

Now we can generate within subjects data as follows:

```
## create sum coded condition vector:
cond<-rep(c(-1,1),each=n)
rt <- round(725 + rep(subject_adj,2) +
                25 * cond + rnorm(n*2,0,200))
```

Generating within-subjects data

Next, we replace the rt column in our incorrect data frame sim_dat with this new corrected rt data, which is now coming from a within-subjects design.

```
## put in new RTs into data frame:
sim_dat$rt<-rt
head(sim_dat)

##   subj cond   rt
## 1    1   sr  414
## 2    2   sr  522
## 3    3   sr  250
## 4    4   sr 1119
## 5    5   sr  834
## 6    6   sr  887
```

Generating within-subjects data

This data can be (in fact, it *must* be) analyzed using the paired t-test!

Generating within-subjects data

```
t.test(rt~cond,sim_dat,paired=TRUE,var.equal=TRUE)

##
##  Paired t-test
##
## data:  rt by cond
## t = 1.69, df = 9, p-value = 0.13
## alternative hypothesis: true difference in means is not
## 95 percent confidence interval:
##  -37.348 257.948
## sample estimates:
## mean of the differences
##                   110.3
```

Generating within-subjects data

Recall that the same model can be fit using the linear mixed model with varying intercepts:

```
library(lme4)

## Loading required package:  Matrix

m1<-lmer(rt~cond+(1|subj),sim_dat,
control=lmerControl(calc.derivs=FALSE))
```

Generating within-subjects data

```
summary(m1)

## Linear mixed model fit by REML ['lmerMod']
## Formula: rt ~ cond + (1 | subj)
##    Data: sim_dat
## Control: lmerControl(calc.derivs = FALSE)
##
## REML criterion at convergence: 254.1
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -1.336 -0.374 -0.149  0.437  1.539
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subj     (Intercept) 65026    255
##  Residual             21300    146
## Number of obs: 20, groups:  subj, 10
##
## Fixed effects:
##             Estimate Std. Error t value
## (Intercept)    719.7       87.0    8.27
## cond1           55.1       32.6    1.69
##
## Correlation of Fixed Effects:
##       (Intr)
## cond1 0.000
```

Generating within-subjects data

The term $(1|subj)$ in the lmer call above refers to the by-subject adjustment to the grand mean that we did above when generating the data.

Generating within-subjects data

Now compute power for this within-subjects design using the paired t-test.
Hint: you can keep creating a new vector of data like this, and overwrite the rt column in the data frame sim_dat:

```
sim_dat$rt <- round(725 + rep(subject_adj,2) +
                      25 * cond +
                      rnorm(2*n,0,200))
```

Then you can do a paired t-test:

Generating within-subjects data

```
t.test(rt~cond,sim_dat,paired=TRUE,var.equal=TRUE)

##
##  Paired t-test
##
## data:  rt by cond
## t = 0.32, df = 9, p-value = 0.76
## alternative hypothesis: true difference in means is not
## 95 percent confidence interval:
##  -213.79  284.19
## sample estimates:
## mean of the differences
##                    35.2
```

Generating within-subjects data

```
nsim<-1000
pvals<-tvals_lmer<-rep(NA,nsim)
for(i in 1:nsim){
  subject_adj<-round(rnorm(n,mean=0,sd=200))
  sim_dat$rt <- round(725 + rep(subject_adj,2) +
                        25 * cond + rnorm(2*n,0,200))
pvals[i]<-t.test(rt~cond,sim_dat,paired=TRUE)$p.value
tvals_lmer<-summary(lmer(rt~cond+(1|subj),sim_dat,
                         control=lmerControl(calc.derivs=F
## power using paired t-test:
mean(pvals<0.05)

## [1] 0.085

## using lmer:
```

Within-subjects parameter recovery

Let's write a function for generating repeated measures data:

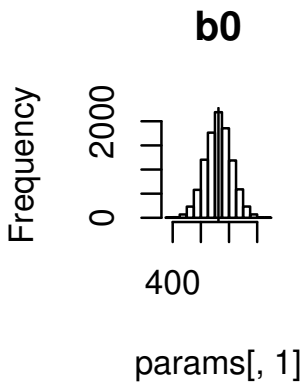
```
withinsubj_simdat<-function(n=10,## no. subjs in each cond
                            b0=725,stddev=300,
                            b1=25,sigma_u0=200){
  u0<-round(rnorm(n,mean=0,sd=200))
  cond<-rep(c(-1,1),each=n)
  subj<-rep(1:n,2)
  rt <- round(b0 + rep(u0,2) +
                          b1 * cond + rnorm(2*n,0,stddev))
sim_dat<-data.frame(subj=subj,cond=cond,rt=rt)
sim_dat
}
```

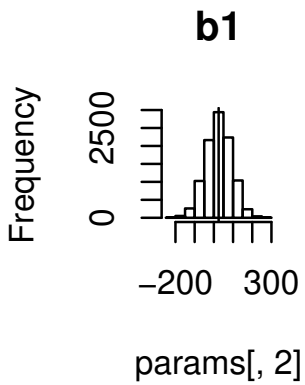
Within-subjects parameter recovery

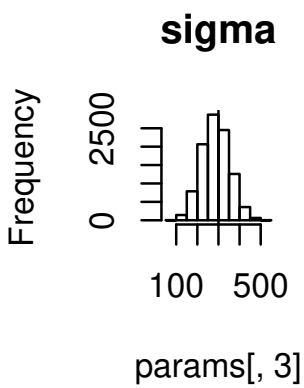
Now check if the model recovers the four parameters:

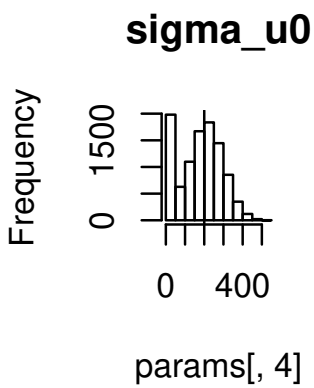
```
nsim<-10000
params<-matrix(rep(NA,nsim*4),ncol=4)
for(i in 1:nsim){
  sim_dat<-withinsubj_simdat()
  m<-lmer(rt~cond+(1|subj),sim_dat,
          control=lmerControl(calc.derivs=FALSE))
  params[i,1]<-summary(m)$coefficients[1,1]
  params[i,2]<-summary(m)$coefficients[2,1]
  params[i,3]<-sigma_e<-attr(VarCorr(m),"sc")
  params[i,4]<-sigma_e<-attr(VarCorr(m)$subj,"stddev")
}
```

Within-subjects parameter recovery









Within-subjects parameter recovery

So this linear mixed model is looking pretty OK in terms of parameter recovery.
However, in some cases the σ_{u0} parameter is not estimated accurately.

Replicates from each subject

Above, we had only one data point from each subject for each RC type.

```
xtabs(~subj + cond,sim_dat)

##      cond
## subj -1 1
##    1  1 1
##    2  1 1
##    3  1 1
##    4  1 1
##    5  1 1
##    6  1 1
##    7  1 1
##    8  1 1
##    9  1 1
```

Replicates from each subject

Normally, we collect more than one data point for each condition, from each subject.
We now generate 20 data points from each subject for SRs, and 20 data points from each subject for ORs. The multiple data points from each subjected in each condition is technically called a **replicate**.

```
## create data frame:
## num subjs
n<-10
## num replicates
k<-20
## the cond vector has to be made n*k times long for each
cond<-rep(c(-1,1),each=n*k)
## subject vector:
## each of the n subject ids has to be repeated k times, a
```

Replicates from each subject

```
head(sim_dat)

##   subj cond
## 1    1   -1
## 2    1   -1
## 3    1   -1
## 4    1   -1
## 5    1   -1
## 6    1   -1
```

Replicates from each subject

```
## generate repeated measures data:
sim_dat$rt<-725+
            rep(rep(round(rnorm(n,0,200)),each=k),2)+
            25*cond+
            rnorm(n*k*2,0,300)
```

Replicates from each subject

The only challenging thing here is the adjustment to the intercept:

- Each of the n subjects' adjustments has to be repeated k times.
- Then, we need to repeat that vector of adjustments for SRs and ORs.
- That's how the line of code below comes about.

```
rep(rep(round(rnorm(n,0,100)),each=k),2)
```

```
##   [1]  -19  -19  -19  -19  -19  -19  -19  -19  -19  -19
##  [15]  -19  -19  -19  -19  -19  -19  -11  -11  -11  -11
##  [29]  -11  -11  -11  -11  -11  -11  -11  -11  -11  -11
##  [43]  -59  -59  -59  -59  -59  -59  -59  -59  -59  -59
##  [57]  -59  -59  -59  -59   60   60   60   60   60   60
##  [71]   60   60   60   60   60   60   60   60   60   60
##  [85]   97   97   97   97   97   97   97   97   97   97
```


51/86

Replicates from each subject

Break this up step by step to understand it:

```
## For n=10 subjects, create an adjustment:
u0<-round(rnorm(n,0,100))
```

Replicates from each subject

Repeat each of these adjustments k=20 times for the first condition:

```
rep(round(u0),each=k)
```

```
##   [1]   11   11   11   11   11   11   11   11   11   11
##  [15]   11   11   11   11   11   11 -220 -220 -220 -220
##  [29] -220 -220 -220 -220 -220 -220 -220 -220 -220 -220
##  [43]  -65  -65  -65  -65  -65  -65  -65  -65  -65  -65
##  [57]  -65  -65  -65  -65  179  179  179  179  179  179
##  [71]  179  179  179  179  179  179  179  179  179  179
##  [85]  115  115  115  115  115  115  115  115  115  115
##  [99]  115  115   63   63   63   63   63   63   63   63
## [113]   63   63   63   63   63   63   63   63  116  116
## [127]  116  116  116  116  116  116  116  116  116  116
## [141] -112 -112 -112 -112 -112 -112 -112 -112 -112 -112
```

Replicates from each subject

We now have $10 \times 20 = 200$ data points for subject relatives and 200 for object relatives:

```
## sanity check time!
dim(sim_dat)

## [1] 400   3

xtabs(~subj+cond,sim_dat)

##      cond
## subj -1  1
##    1  20 20
##    2  20 20
##    3  20 20
##    4  20 20
##    5  20 20
```

Replicates from each subject

One can fit a varying intercepts model:

```
m2<-lmer(rt~cond+(1|subj),sim_dat,
         control=lmerControl(calc.derivs=FALSE))
```

We can also fit a varying intercepts + varying slopes model:

```
m3<-lmer(rt~cond+(1+cond|subj),sim_dat,
         control=lmerControl(calc.derivs=FALSE))
```

Replicates from each subject

You should get output like this (obviously, the numbers will be different for you as we are randomly generating data):

```
summary(m3)

## Linear mixed model fit by REML ['lmerMod']
## Formula: rt ~ cond + (1 + cond | subj)
##    Data: sim_dat
## Control: lmerControl(calc.derivs = FALSE)
##
## REML criterion at convergence: 5737.4
##
## Scaled residuals:
##    Min     1Q Median     3Q    Max
## -2.365 -0.654 -0.043  0.686  3.619
##
```

Replicates from each subject

By-subject adjustments to mean slope

Now, we add the assumption that some subjects have larger relative clause effects, and some smaller effects. Some have the same as same RC effect as the mean RC effect. This generative process corresponds to the varying intercepts and varying slopes model.

Let's assume that subjects variation about the mean slope can be modeled as:

$Normal(0, 50)$

I just chose an sd of 50 arbitrarily.

```
(u1<- round(rnorm(n,mean=0,sd=50)))

##  [1]  61  18 -48 -52 -50  -6 -45 -35  69  77
```

Replicates from each subject

By-subject adjustments to mean slope

Since the slope in the model is assumed to be 25 ms, what we see is that some subjects have a larger effect and some have a smaller effect than this 25 ms effect.
So, for object relatives:

```
(25+u1)*1
```

```
##   [1]   86   43 -23 -27 -25   19 -20 -10   94 102
```

and for subject relatives:

```
(25+u1)*(-1)
```

```
##   [1]  -86  -43   23   27   25  -19   20   10  -94 -102
```

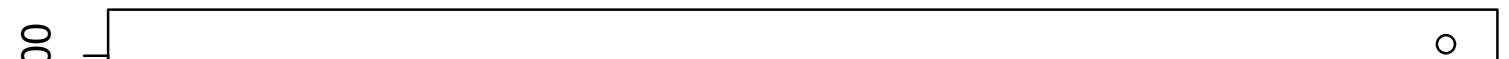
Replicates from each subject

By-subject adjustments to mean slope

We can also plot the variability in the mean SR and and the mean OR effect. In this plot, the circles refer to object relatives, and the square with the cross inside it to subject relatives.

```
## ORs
plot(1:10,(25+u1)*1,
     xlab="subject",
     ylab="estimate (ms)",ylim=c(-100,100))
## SRs:
points(1:10,(20+u1)*(-1),pch=12)
abline(h=25)
```


59 / 86



Replicates from each subject

By-subject adjustments to mean slope

```
## generate data for SR and OR conditions row by row:
rt <- 725+rep(rep(round(rnorm(n,0,100)),each=k),2)+
    (25 + rep(rep(round(rnorm(n,0,50)),each=k),2))*cond +
  rnorm(n*k*2,0,200)
## add to data frame:
sim_dat$rt<-rt
```

Replicates from each subject

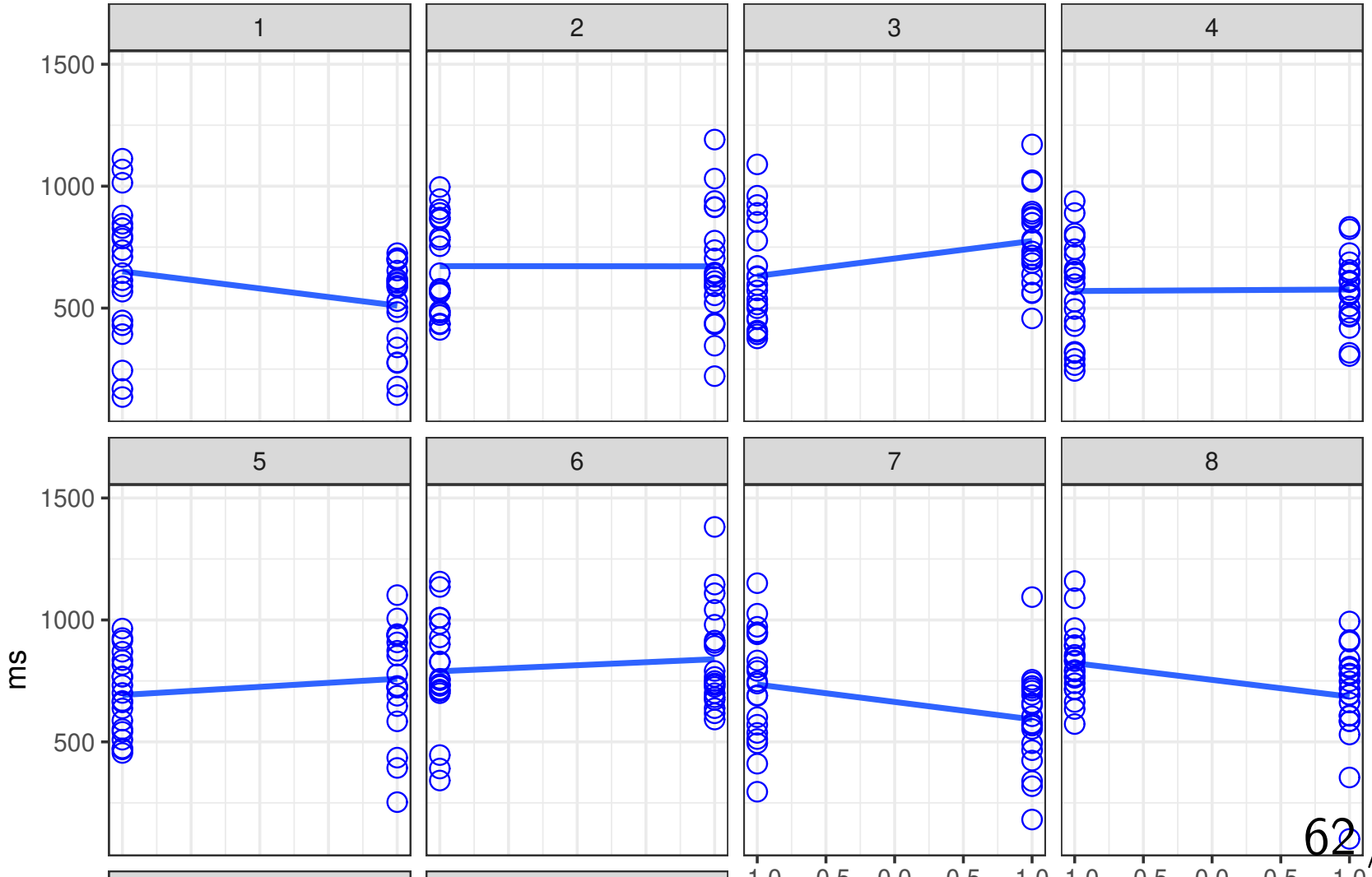
By-subject adjustments to mean slope

We can visualize the RC by-subject effects. First, I define a plotting function:

```
library(ggplot2)
gg_xyplot <- function(x, y, formula, shape, size,
                      xlabel="RC Type",
                      ylabel="ms",
                      data=sim_dat){
    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()) +
```

Replicates from each subject

By-subject adjustments to mean slope



62/ 86

Replicates from each subject

By-subject adjustments to mean slope

Fit the following model, which assumes that each subject has a different grand mean (intercept) and different slope. **Notice that I am using double vertical bars in the code below**.

```
m3<-lmer(rt~cond + (1+cond||subj),sim_dat)
```

Replicates from each subject

By-subject adjustments to mean slope

The output should look something like this (the numbers will vary for each of you).

```
summary(m3)

## Linear mixed model fit by REML ['lmerMod']
## Formula: rt ~ cond + ((1 | subj) + (0 + cond | subj))
##    Data: sim_dat
##
## REML criterion at convergence: 5408.2
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.9910 -0.6175 -0.0305  0.6491  2.7280
##
```

Replicates from each subject

By-subject adjustments to mean slope

This model can be written as follows:

$$rt \sim Normal(\beta_0 + u_0 + (\beta_1 + u_1) * cond, 200)$$

where

$$u_0 \sim Normal(0, 100)$$

$$u_1 \sim Normal(0, 50)$$

Replicates from each subject

By-subject adjustments to mean slope

Notice that there are **three** variance components:

- Normal(0,100): the adjustments by subject to the grand mean
- Normal(0,50): the adjustments by subjects to the mean slope
- Normal(0,200): the residual error

Replicates from each subject

By-subject adjustments to mean slope

The model estimates these standard deviations. Here is what I got (the three standard deviations' estimates are shown in order):

```
Random effects:
 Groups    Name        Variance Std.Dev.
 subject   (Intercept) 10547    102.7    <- Estimate of u0 s
 subject.1 cond        12432    111.5    <- Estimate of u1 s
 Residual              41881    204.6    <- estimate of resi
```

The first and last estimates are accurate, the slope standard deviation is an overestimate.

Replicates from each subject

A more sophisticated method for generating random slopes

The data generation approach used above can be made easier to understand by using the following approach:

```
## number of subjects
n<-10
## number of replicates
k<-20
## generate data frame with subject and condition column:
cond<-rep(c(-1,1),each=n*k)
sim_dat<-data.frame(sub=rep(rep(1:n,each=k),2),
                    cond=cond)
head(sim_dat)

##   sub cond
## 1   1   -1
```

Replicates from each subject

A more sophisticated method for generating random slopes

Now generate the by-subject intercept adjustments and by subject slope adjustments and create another data frame with these:

```
sigma_u0<-200
sigma_u1<-50
u<-data.frame(subject=1:n,
              u0=round(rnorm(n,0,sigma_u0)),
              u1=round(rnorm(n,0,sigma_u1)))
```

Replicates from each subject

A more sophisticated method for generating random slopes

```
head(u)

##   subject   u0  u1
## 1       1   83  26
## 2       2   83  50
## 3       3  -80 -47
## 4       4  304  68
## 5       5   13  71
## 6       6 -145 -15
```

Replicates from each subject

A more sophisticated method for generating random slopes

Now, what we will do is that **for each row** i in the sim_dat data frame we created above, we will

- look up the subject id s in the data frame dat's row i
- find out what the by-subject intercept adjustment (u0) is for that subject s
- find out what the by-subject slope adjustment (u1) is for that subject s
- generate the reading time for that row in the data frame sim_dat using the equation we saw above:
  $rt \sim b0 + u0 + (b1 + u1) * cond + rnorm(1, 0, 300)$

Replicates from each subject

A more sophisticated method for generating random slopes

```
## record the number of rows in the data frame dat:
nrows<-dim(sim_dat)[1]

## create a vector to store reading times generated row by
rt<-rep(NA,nrows)

b0<-725
b1<-25
sigma<-300
## run a for-loop to carry out the above steps:
for(i in 1:nrows){
rt[i] <- b0 + u[sim_dat[i,]$subj,]$u0 +
   (b1 + u[sim_dat[i,]$subj,]$u1)*sim_dat[i,]$cond + rnorm
}
```

Replicates from each subject

A more sophisticated method for generating random slopes

- The critical new development here is that in the for loop, we are using the row id to figure out which subject we need to get the adjustment to the intercept and slope for.
- For example, when i=1, i.e., when the row id is 1, we can figure out the subject id in the data frame as follows:

```
sim_dat[1,]$subj
```

```
## NULL
```

Replicates from each subject

A more sophisticated method for generating random slopes

Once you know this subject id, you can look up appropriate subject id in the data frame u to get the adjustment to the intercept and the slope. For example, how do I get the first subject's intercept and slope adjustment?

```
u[1,]$u0
```

```
## [1] 83
```

```
u[1,]$u1
```

```
## [1] 26
```

Replicates from each subject

A more sophisticated method for generating random slopes

Instead of writing 1 manually, we could have looked up the subject id we need from the i-th row of the data frame sim_dat!

```
u[sim_dat[1,]$subject,]$u0

## numeric(0)

u[sim_dat[1,]$subject,]$u1

## numeric(0)
```

That is what the above code that generates the simulated rt data is doing.

Replicates from each subject

A more sophisticated method for generating random slopes

- And that's it. Once you have generated the data, fit the same model as m3 to this data.
- Print out the summary of the model, and check whether the model's parameter estimates roughly match the numbers you used to generate the data. This is just a sanity check for you.

Replicates from each subject

A more sophisticated method for generating random slopes

Now, using the simpler/more elegant approach I showed above, write a function called gendat that generates data, with input the sample size, the number of repetitions per condition k, the grand mean, the RC effect, and the three standard deviations.
The function should work as follows. When we type:

```
sim_dat<-gendat(n=10,k=20,
                b0=725,b1=25,
                sigma_u0=200,sigma_u1=50,
                sigma=300)
```

Replicates from each subject

A more sophisticated method for generating random slopes

We should get the appropriate data frame. For example:

```
sim_dat<-gendat(n=5,k=5,b0=725,b1=50,
                sigma_u0=200,
          sigma_u1=50,
          sigma=300)
head(sim_dat)

##    subj cond     rt
## 1     1   -1 263.52
## 2     1   -1 811.40
## 3     1   -1 786.08
## 4     1   -1 832.92
## 5     1   -1 267.60
## 6     2   -1 417.03
```

Replicates from each subject

A more sophisticated method for generating random slopes

Using the gendat function, write a for-loop (100 iterations) that computes the proportion of times that the absolute value of the t-score for the slope (the RC effect) is greater than 2.
This is the estimated power of the experiment; the estimated probability that we would detect the effect (assuming our belief about the true effect is being whatever we set it to be is correct).

Generating data with correlated intercepts and slopes

Here is how we can generate bivariate correlated data.
Let's assume that the correlation between the varying intercepts and slopes is 0.5.

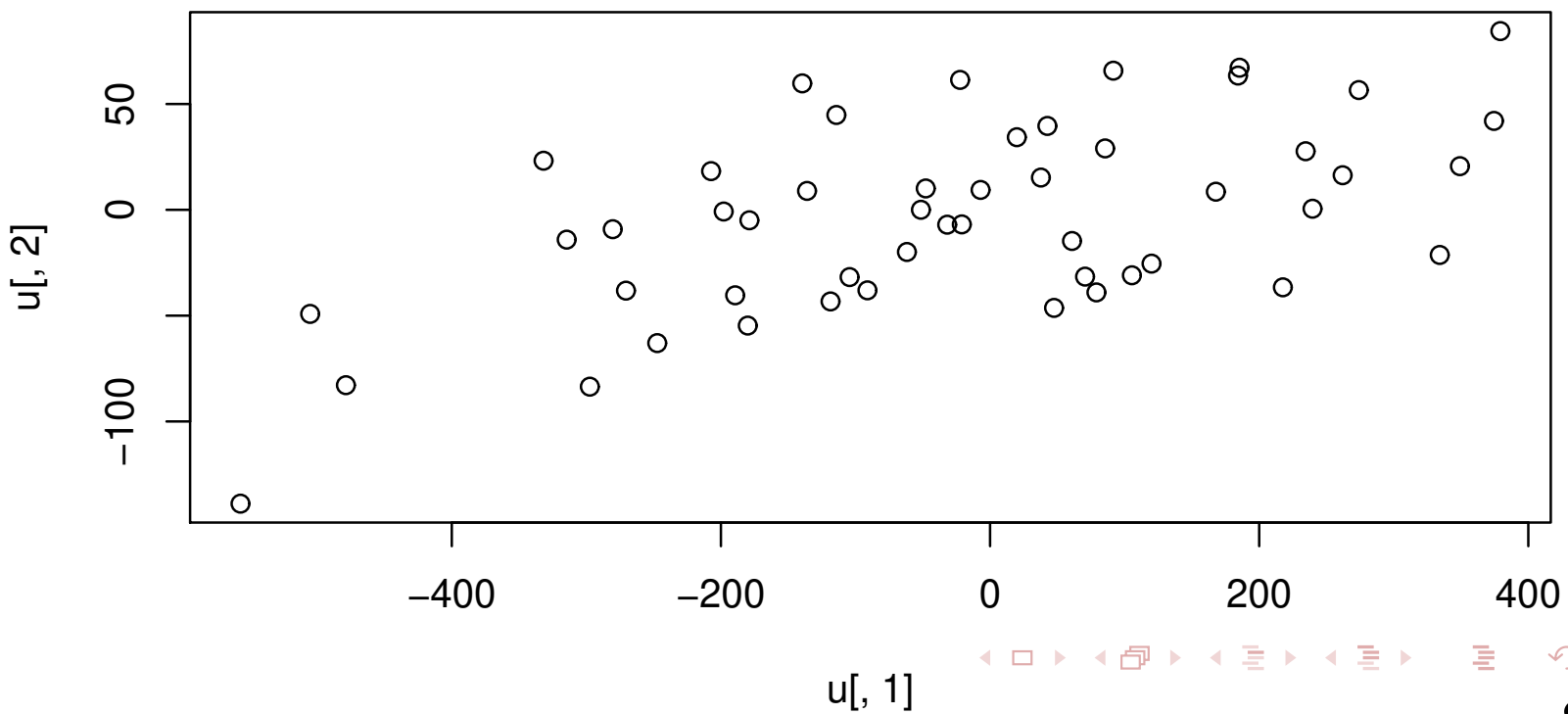
```
library(MASS)
Sigma<-matrix(c(200^2,200*50*0.5,
                200*50*0.5,50^2),ncol=2)

u<-mvrnorm(50,mu=c(0,0),Sigma=Sigma)
head(u)
```

```
##          [,1]    [,2]
## [1,] -91.106 -38.049
## [2,] 262.028  16.343
## [3,] 119.996 -25.426
## [4,]  19.894  34.306
```

Generating data with correlated intercepts and slopes

```
plot(u[,1],u[,2])
```



Generating data with correlated intercepts and slopes

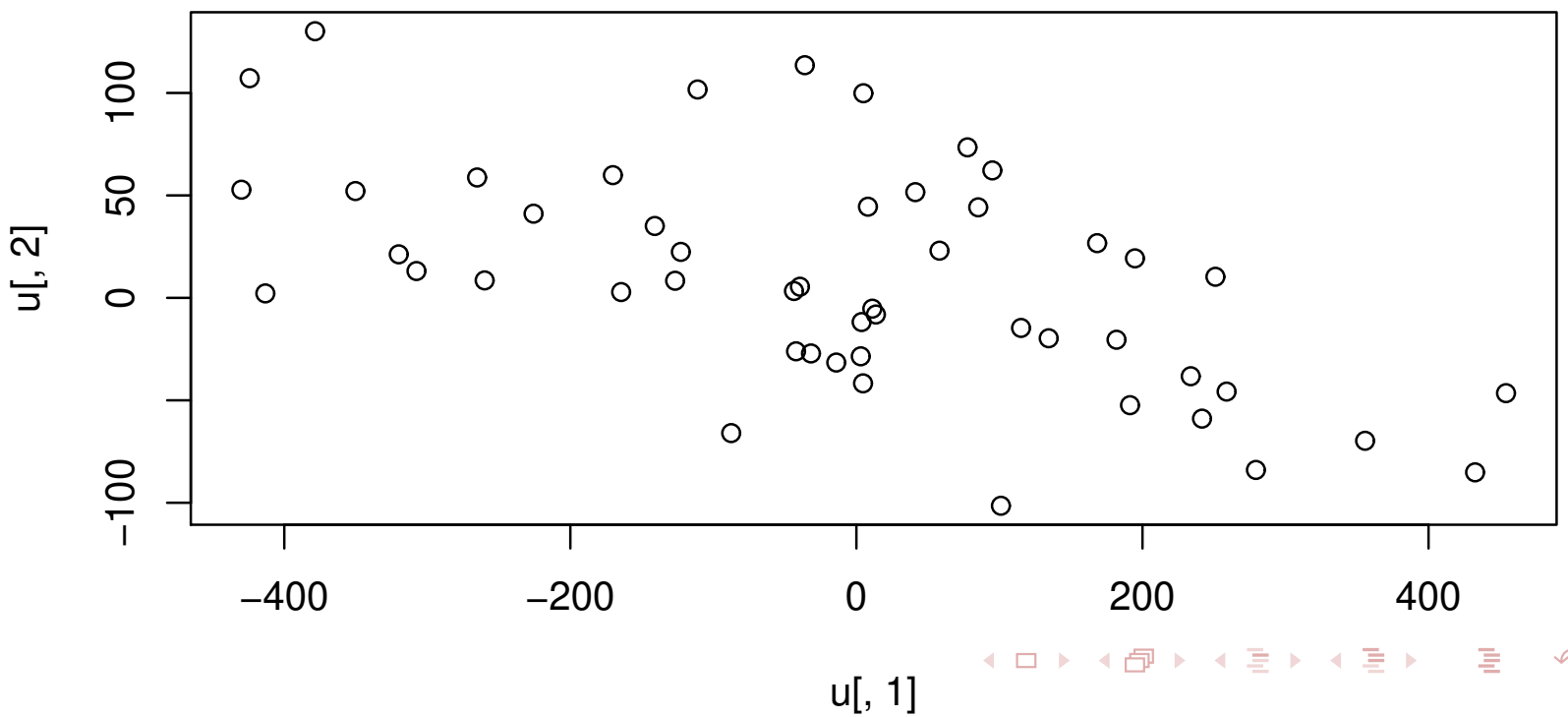
Or assume that the correlation between the varying intercepts and slopes is -0.5.

```
Sigma<-matrix(c(200^2,200*50*-0.5,
                200*50*-0.5,50^2),ncol=2)

u<-mvrnorm(50,mu=c(0,0),Sigma=Sigma)
```

Generating data with correlated intercepts and slopes

```
plot(u[,1],u[,2])
```



Generating data with correlated intercepts and slopes

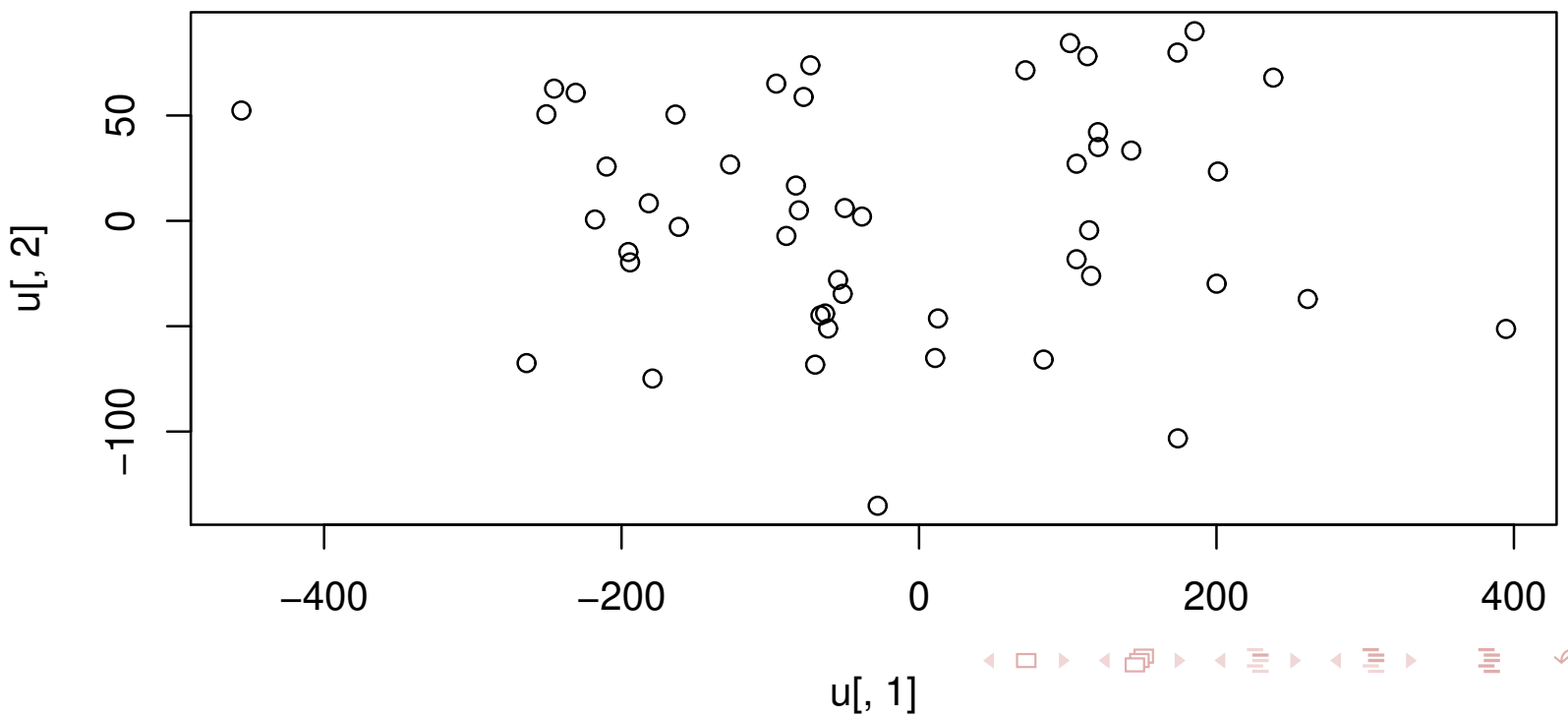
Or assume that the correlation between the varying intercepts and slopes is 0.

```
Sigma<-matrix(c(200^2,200*50*0,
                200*50*0,50^2),ncol=2)

u<-mvrnorm(50,mu=c(0,0),Sigma=Sigma)
```

Generating data with uncorrelated intercepts and slopes

```
plot(u[,1],u[,2])
```



Generating data with correlated intercepts and slopes

Now you should be able to:

- extend the gendat function to generate correlated intercepts and slopes
- compute power using this function
- check whether parameters can be recovered under repeated sampling in the so-called maximal model.

These things will be part of the homework assignments for simulating data.