Example Solution Homework Week 4 Mixed-Models Class 2014, Bernd Figner

Please note: I created a very detailed and commented R script with lots of explanations (and different variants, e.g., to get p values)

In this document here, I only pasted the whole code plus added the figures I created (because I couldn't paste the figures into the R script).

The R script is called "ValuationRatingsMixMod2014_ITC_8March2014a_bf.R" and also available on BlackBoard.

....

```
# Mixed-models R script for ITC dataset from mixed-models course 2014
# valuation ratings
# Bernd Figner, March 9, 2014


# NOTE: If you want to jump right to the part that was NOT part of the homework,
 search for the following: Below here is some additional stuff that was NOT part
 of the homework


# PREPARING

options(scipen=10) #so serials are shown in regular)

# set contrasts to sum-to-zero (aka deviation coding) for unordered factors and
 polynomial for ordered factors
options(contrasts=c("contr.sum", "contr.poly"))

# Load libraries

#install.packages('reshape') # to get data frame from wide to long format; you
 need only to run this if you haven't already installed reshape
# not all of them might be needed; it's better to load ONLY the ones that are
 needed, rather than always loading all

library(reshape)
library(car)
library(lattice)
library(lme4)
library(pbkrtest)

library(multcomp)
library(boot)

library(psych)
#library(lsmeans) not needed here
#library(ggplot2)
#library(plyr)
#library(gdata)
```

```
# set working directory to the folder where I have my data
setwd('~/Dropbox/Radboud/Teaching/MultilevelClass/2014/Data/ITC')



# loading and saving of work spaces


##################################################
##################################################



# load raw data file
v1 <- read.csv('additional_qs_8Feb2014a.csv')

# check whether the import went fine
head(v1)
tail(v1)

# this seems to be the column numbers with the actual ratings:
head(v1[, 11:ncol(v1)])

# create a new data frame with only the relevant variables (actually, we don't
 even need practice as all the entries are 'no' as it turns out)
v2 <- subset(v1, select = c('serial', 'practice', 'q1', 'q2', 'q3', 'q4', 'q5',
 'q6', 'q7', 'q8', 'q9', 'q10', 'q11', 'q12', 'q13', 'q14', 'q15'))

# check the new data frame
head(v2)
tail(v2)
v2 # one participant always rated 100...


# let's split up 'serial' in its components

# digits 9 to 10 are the actual pp_number
v2$pp_code <- as.numeric(substring(v2$serial,9,10)) # This is the actual
 pp_number

# check whether that worked as intended
v2$pp_code # looks good

v2$pp_code <- paste('pp', v2$pp_code, sep = '_')
#check:
v2$pp_code # looks good

# Order: always the same for everybody in that case of the intertemp choice
 task, but for the risky choice tasks, such a command will be handy
v2$Order <- as.numeric(substring(v2$serial,11,11))
check:
v2$Order # yep, always the same

# I could also do it like this:
```

```
unique(v2$Order) # yep, all entries are 2



# now let's turn the data frame into long format
head(v2)


# ok, now i need to load the reshape package

v3 <- melt(v2, id = c('pp_code', 'serial', 'practice'), variable_name = 'item')
v3 <- melt(v2, id = c('pp_code', 'serial', 'practice', 'Order'), variable_name =
 'item')

# check:
head(v3)
tail(v3)
v3

# looks good so far

# as a safety check, see how many rows and columns the data frame had before and
 after melt-ing
nrow(v2) #32 --> i.e., 32 participants
nrow(v3) #480

# if everything is correct v3 should be 32 (participants) * 15 (items) rows
 long: 32*15 is 480, so it seems everything went fine!


# OK, I want to change the name of the DV to rating

# just for safety I first save the old data frame under a new name
v4 <- v3

# have to find out which column is the one I want to rename:
head(v4) # ok, it's the 6th column

# just to double check whether this is indeed true:
names(v4)[6] # yep, it's ok

# so let's rename that column

names(v4)[6] <- "rating"

# check
head(v4) # yep, it's now not called value anymore, but rating




# OK, now I'm going to add the time/amount information for the 15 different
 items each participant did
```

```r
# read file with q1 to q15 information about times and money values
q <- read.csv('Dutch_ValuationRatings_Design_TimeMoney.csv')

# check
q

# ok, so the 'key variable' for merging is called 'item' in both data frames

v5 <- merge(v4, q, by = 'item')

# check
v5
# looks good



# recode exact monetary amounts into levels (I plan to use that only for the
 plots, not for the analyses); I use an ordered factor for that
# I need to load the package car for the command recode()

v5$o_money <- ordered(recode(v5$money, "10:30 = 'low'; 40:60 = 'medium'; 70:90 =
 'high'"), levels = c('low', 'medium', 'high'))

# check
v5$o_money # good

# do a quick interaction plot showing the effect of amount and time on the
 ratings
with(v5, interaction.plot(time, o_money, rating)) # looks reasonable, looks like
 both variables (time and money) have a clear effect.
# At first, I thought: "oh, the lines look very parallel, so there's not much
 evidence that there is an interaction." While this is still true, one has to be
 a bit careful, as the spacing on the x axis does not reflect the numerical
 values of the predictor: The difference between 0 and 3 is obviously only 3
 days, while the next step from 3 to 14 is 11 days, and the next two are each 14
 days. interaction.plot treats these values as categories and thus spaces them
 equally. Thus, in principle, this might obscure some effects (more likely it
 obscures non-linearity in time, rather than interaction effects, but I think it
 might also make interaction effects harder to detect)
```

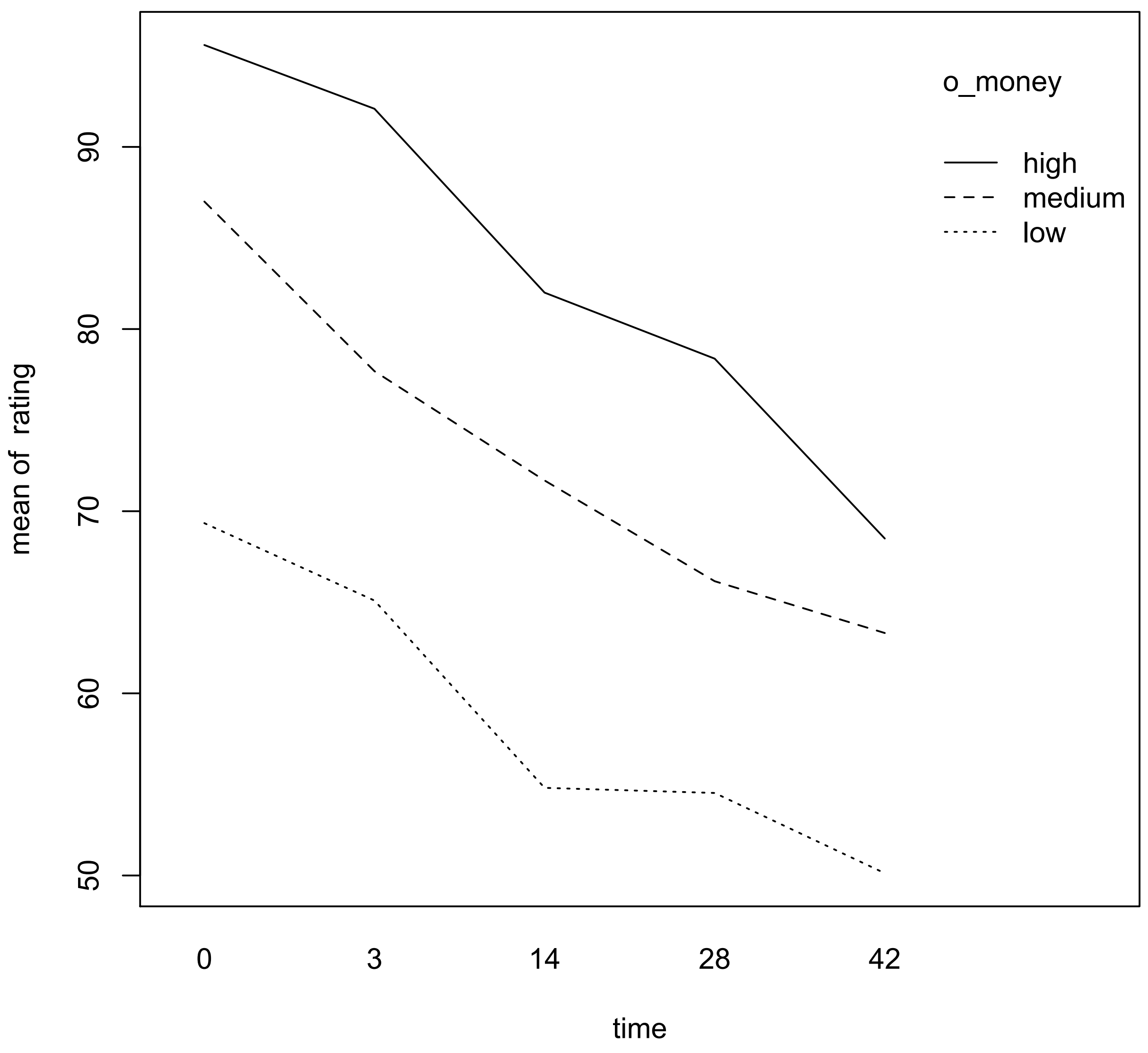

Figure 1: Valuation ratings (y axis) as a function of time (x axis) and amount level (low = approx. Euro 20; medium = approx. Euro 50; high = approx. Euro 80).
[NOTE: I'm sure you can do a much better and prettier job than me, as you are using ggplot2!]

```
# let's have a quick look whether the DV looks normally distributed
# I need to load the package lattice for the densityplot() command
with(v5, densityplot(rating))
# hmm, looks skewed, the values stack up at the upper end of the value range
 (i.e., 100)
# actually, my transformation attempts weren't really successful, so I decided
 to use the untransformed DV but then use bootstrapping for the significance
 tests, as they don't assume symmetric confidence intervals and should be robust
 against non-normal residuals
```

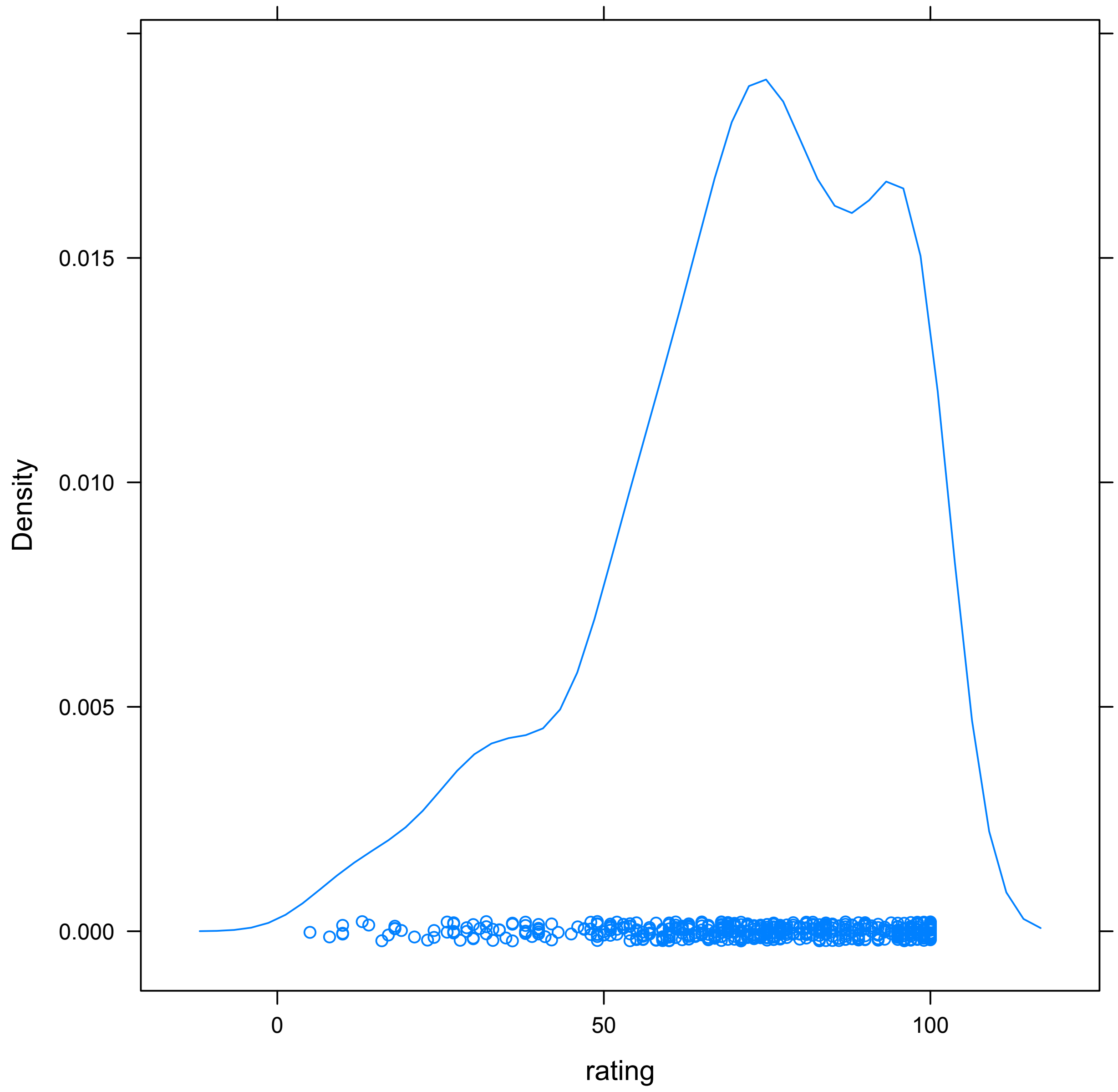

Figure 2: Distribution of the valuation ratings.
[NOTE: Same here, I'm sure your plots look prettier than mine.]

```
# before we can run our lmer models, I need to center the predictors (as it will
 turn out later, I actually need to scale them)
# centering

v5$c_time <- v5$time - mean(v5$time)
# is the mean now indeed 0 (that also checks at the same time whether there
 might be any NA entries, as the mean would then be NA)
mean(v5$c_time) # good!

# same for money
v5$c_money <- v5$money - mean(v5$money)
```

```
# as alternative, I could also have used the following commands to achieve the
 same
v5$c_time <- scale(v5$time, center = TRUE, scale = FALSE)
v5$c_money <- scale(v5$money, center = TRUE, scale = FALSE)



#####################################################################
#lmer models (i obviously need the package lme4 now)
#####################################################################

# run a first model with the centered predictors: i'm testing only main effects,
 no interaction


v_m1 <- lmer(rating ~ c_time + c_money + (1 + c_time + c_money | pp_code), data
 = v5)
# I got the following warning message:
# Warning message:
# In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,  :
  # Model failed to converge with max|grad| = 0.019193 (tol = 0.001)

# can I still have a look at the summary output?
summary(v_m1) # yep, I get the following output

# Linear mixed model fit by REML ['lmerMod']
# Formula: rating ~ c_time + c_money + (1 + c_time + c_money | pp_code)
   # Data: v5

# REML criterion at convergence: 3798.1

# Scaled residuals:
    # Min      1Q  Median      3Q     Max
# -3.4956 -0.4403  0.0859  0.4877  3.4912

# Random effects:
 # Groups   Name        Variance  Std.Dev. Corr
 # pp_code  (Intercept) 122.24718 11.0565
          # c_time        0.14195  0.3768   0.59
          # c_money       0.05827  0.2414  -0.66 -0.01
 # Residual             110.83084 10.5276
# Number of obs: 480, groups: pp_code, 32

# Fixed effects:
            # Estimate Std. Error t value
# (Intercept) 71.75417    2.01274   35.65
# c_time      -0.50540    0.07326   -6.90
# c_money      0.40868    0.04695    8.70

# Correlation of Fixed Effects:
        # (Intr) c_time
# c_time   0.520
# c_money -0.585 -0.006
# That looks reasonable to me, but we still would prefer NOT to have this
 warning, so I'm going to try a bunch of different things



# first, I try to increase the number of iterations: I use 100,000 (which is
 quite a lot)
v_m1b <- lmer(rating ~ c_time + c_money + (1 + c_time + c_money | pp_code), data
 = v5, control = lmerControl(optCtrl = list(maxfun = 100000)))
# I still get an warning (and the model didn't run for longer, so it doesn't
 seem like it used all iterations)
# Warning message:
# In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,  :
  # Model failed to converge with max|grad| = 0.019193 (tol = 0.001)

summary(v_m1b) # this looks the same as from the model before, so the number of
 iterations didn't help



# what if I scale the predictors? sometimes that can help
v5$s_time <- scale(v5$time, center = TRUE, scale = TRUE)
v5$s_money <- scale(v5$money, center = TRUE, scale = TRUE)
# check whether indeed the mean is 0 and sd 1
mean(v5$s_time) # yep
sd(v5$s_time)  # yep
mean(v5$s_money) # yep
sd(v5$s_money)  # yep

v_m1c <- lmer(rating ~ s_time + s_money + (1 + s_time + s_money | pp_code), data
 = v5)
# hey, wow, that worked now, great! no warning, so let's have a look
summary(v_m1c)
# Linear mixed model fit by REML ['lmerMod']
# Formula: rating ~ s_time + s_money + (1 + s_time + s_money | pp_code)
   # Data: v5

# REML criterion at convergence: 3786.1

# Scaled residuals:
    # Min      1Q  Median      3Q     Max
# -3.4956 -0.4403  0.0859  0.4877  3.4912

# Random effects:
 # Groups   Name        Variance Std.Dev. Corr
 # pp_code  (Intercept) 122.25   11.057
          # s_time       35.25    5.938    0.59
          # s_money      35.15    5.929   -0.66 -0.01
 # Residual             110.83   10.528
# Number of obs: 480, groups: pp_code, 32

# Fixed effects:
```

```
            # Estimate Std. Error t value
# (Intercept)   71.754      2.013   35.65
# s_time        -7.965      1.155   -6.90
# s_money       10.038      1.153    8.70

# Correlation of Fixed Effects:
        # (Intr) s_time
# s_time   0.520
# s_money -0.585 -0.006

# hmm, that looks actually the same as the output from the models that gave the
 warnings? (except for the coefficients of time and money, but that's not
 surprising that they are bigger now, because the scaling changed the range of
 the values)

# OK, so let's do some diagnostic plots
plot(v_m1c) # hmm, they really converge at the right end of the x axis (not
 surprising given that we already saw in the raw data DV that the values stacked
 up at 100)
```

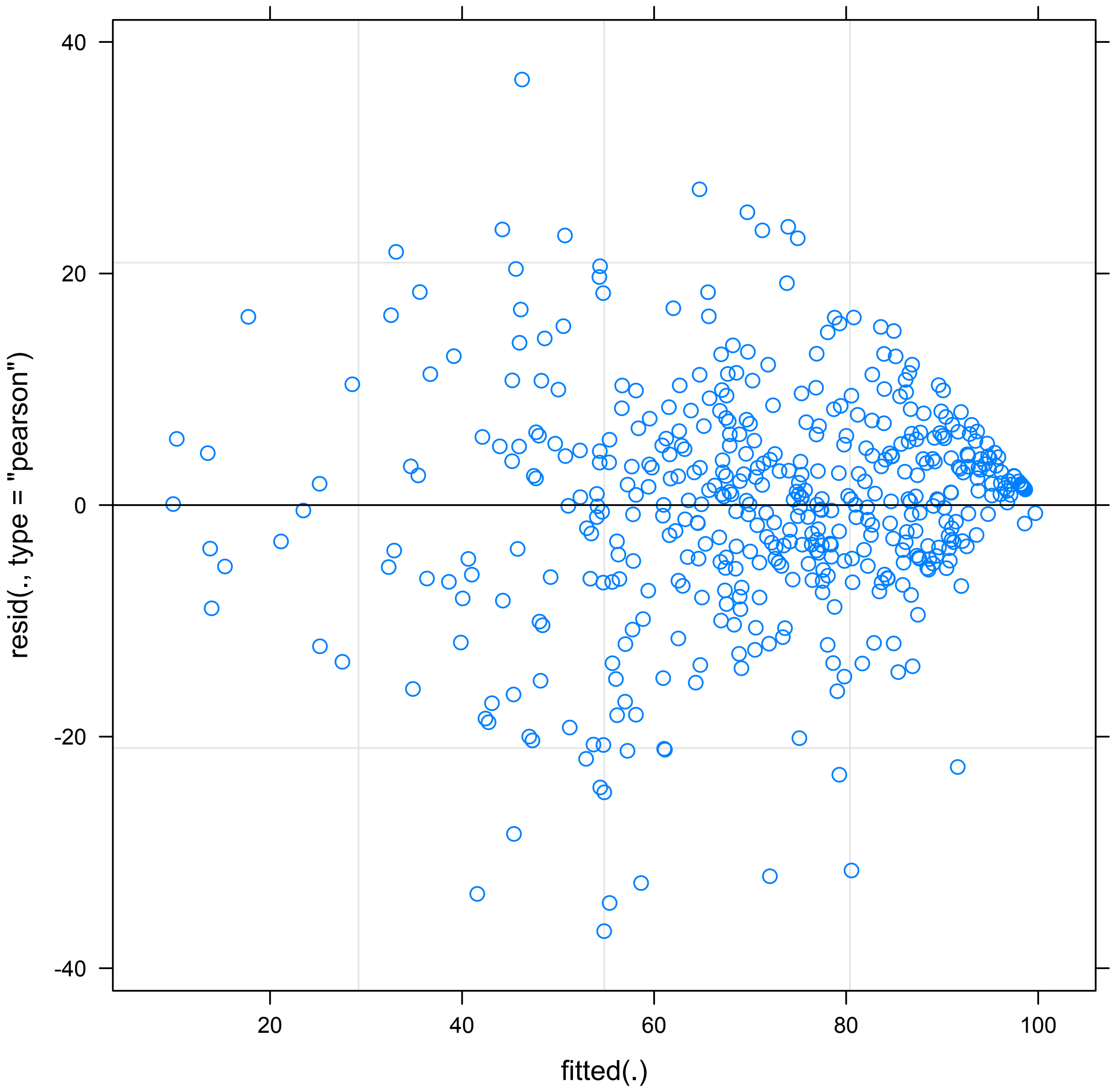

Figure 3: Diagnostic plot 1 for model `v_m1c`: Residuals (y axis) as a function of the fitted values (x axis). Towards the higher values (i.e., maximal ratings of 100), the residuals become smaller.

```
# let's look at the distribution of the residuals
densityplot(resid(v_m1c))
# that doesn't look too bad actually in my opinion, it looks pretty symmetric
 around 0; but it does look leptokurtic; so it might be still a good idea to at
 least verify the p values with a bootstrap approach (but, if we do that, we
 might as well then just use the bootstrap p values...)
```

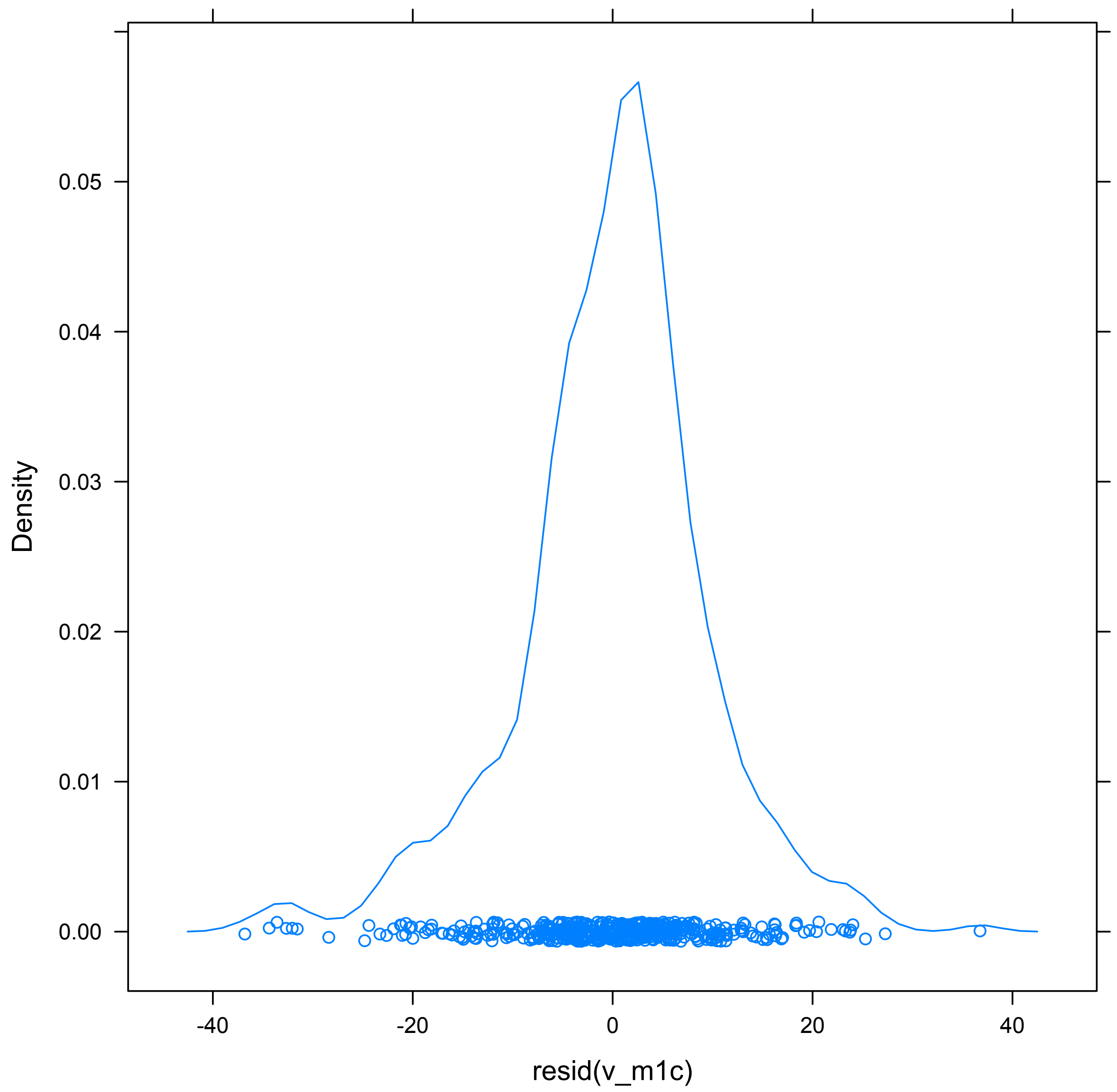

Figure 4: Diagnostic plot 2 for model `v_m1c`: Distribution of the residuals (y axis). The residuals are symmetrically distributed around 0, but appear to be leptokurtic.

```
# let's look at the relationship between observed and predicted values
plot(v5$rating, fitted(v_m1c)) # again the same impression in the upper range
 the model (correctly!) predicts a lot of values around 100. in the lower range,
 the model performs more poorly
```

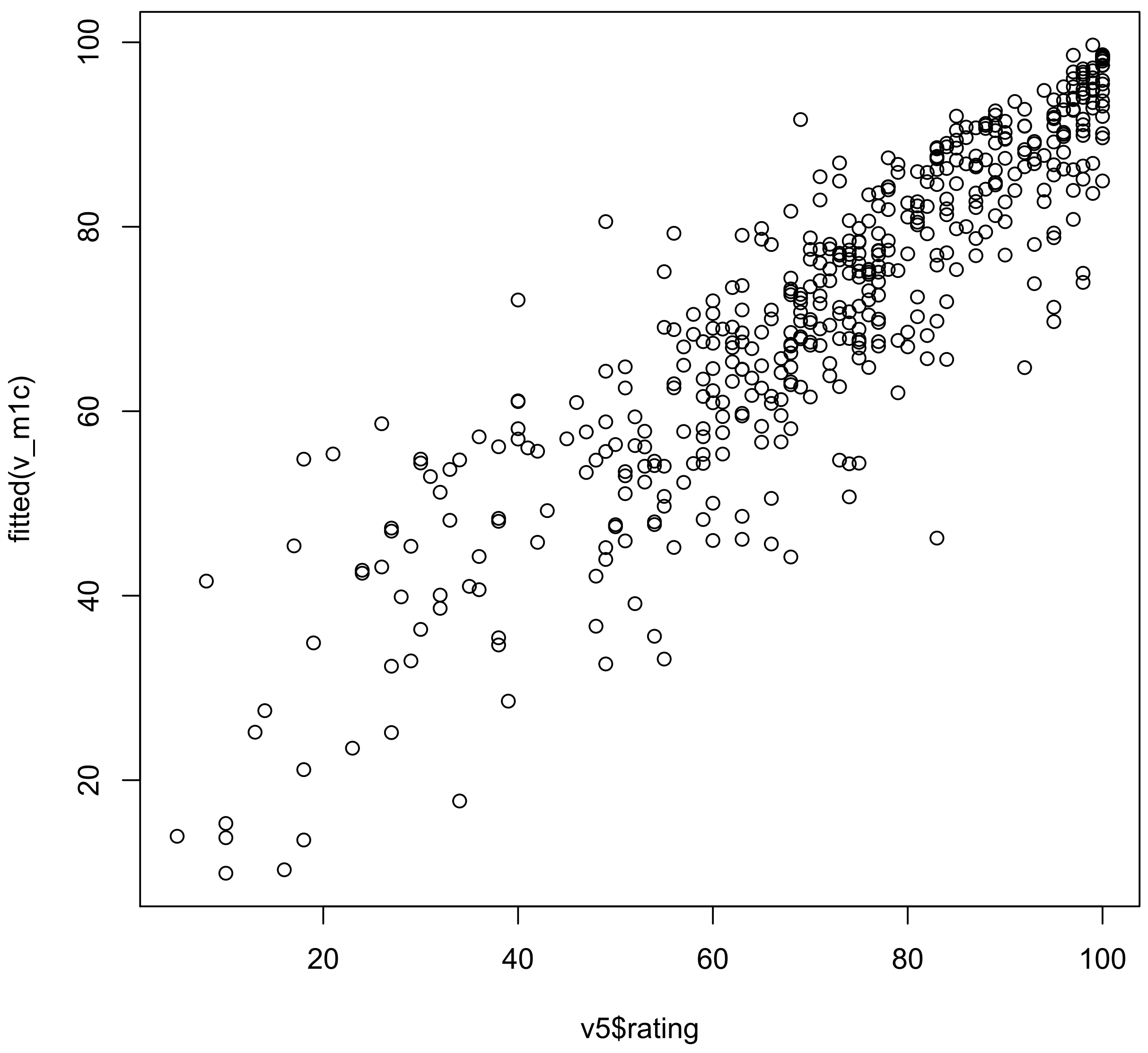

Figure 5: Diagnostic plot 3 for model `v_m1c`: Fitted values (y axis) as a function of the raw values of the dependent variable.

```
# ok, so let's try Anova to get p values

# first the conditional F tests with Kenward-Roger df correction

# I save the result of the command into a variable, so that it's saved when I
 save the workspace
# I add 3F to remind myself that this is type 3 test and is an F test
v_m1c_Anova_3F <- Anova(v_m1c, type = 3, test = 'F')
# I got the following 'Note' (i.e., not even a warning, so I assume it's safe to
 ignore; but we'll see whether we find the same or very different p values when
 using bootstrapping)

# so let's have a look:
```

```
v_m1c_Anova_3F
# Analysis of Deviance Table (Type III Wald F tests with Kenward-Roger df)

# Response: rating
                   # F Df Df.res          Pr(>F)
# (Intercept) 1270.924  1     31       < 2.2e-16 ***
# s_time        47.587  1     31 0.0000000980867 ***
# s_money       75.760  1     31 0.0000000007902 ***

# ok, but money and time are highly significant, consistent with the visual
 impression we got from the interaction plot


# just for curiosity, I try now also the Wald Chisquare test
v_m1c_Anova_3Chisq <- Anova(v_m1c, type = 3, test = 'Chisq')
v_m1c_Anova_3Chisq
# Analysis of Deviance Table (Type III Wald chisquare tests)

# Response: rating
               # Chisq Df       Pr(>Chisq)
# (Intercept) 1270.924  1        < 2.2e-16 ***
# s_time        47.587  1 0.00000000000526 ***
# s_money       75.760  1        < 2.2e-16 ***

# that supports the same conclusion, both time and money are highly significant

# hmm, what if we use drop1 to get p values from Likelihood Ratio Tests?
drop1_v_m1c <- drop1(v_m1c, ~., test = 'Chisq')
drop1_v_m1c
# Single term deletions

# Model:
# rating ~ s_time + s_money + (1 + s_time + s_money | pp_code)
        # Df    AIC    LRT         Pr(Chi)
# <none>     3812.6
# s_time   1 3840.4 29.767 0.0000000487165 ***
# s_money  1 3850.2 39.571 0.0000000003163 ***

# ok, once again the same conclusion is supported; this is all good news


# so let's use next KRmodcomp from the pbkrtest package
# for that, we need to run some new models, so we can do model comparisons

# first I take out the fixed effect of time (but, following Barr et al, I leave
 in the random slope for time!)
v_m1c_noTime <- lmer(rating ~ s_money + (1 + s_time + s_money | pp_code), data =
 v5)
summary(v_m1c_noTime)


# now, to get the significance for Time, we compare the full model (Time and
 Money) to the model that doesn't contain Time. If the full model provides a
 better fit than the model without Time, this means that Time has a significant
 effect

# there are different ways how to do this model comparison, we'll do all 3 of
 them here

# Barr et al recommend to use Likelihood Ratio Tests (LRT); this is done by
 using the anova() command
# the anova command requires first the "smaller" model in the parantheses,
 followed by the larger model (one can have also more than 2 models, but they
 need to be "nested," i.e., one model is a simplification of the other model)

anova_v_m1c_Time <- anova(v_m1c_noTime, v_m1c)
# when I start this command, I get the message:
#refitting model(s) with ML (instead of REML)
# as you have read in the papers, a LRT requires the models to be fit with ML,
 not REML. However, the lme4 default is to use REML; since we have used REML for
 our original model, R is so kind to recognize that we are lazy and does the
 refitting using ML for us.
# Similarly, if we put the models in the wrong order in the anova() command
 (first the large, then the small), it will also recognize that we are lazy
 and/or stupid and/or ignorant and re-arrange them for us (note however, that R
 does that only for mixed models, if you use normal lm regression models, it
 might not recognize it and do something non-interpretable)

# ok, so let's have a look at the result:
anova_v_m1c_Time # well, it's highly significant, consistent will all the other
 p values that we got using the approaches above




# now we want to test the effect of Money using the same approach:
# this second model is one that contains money, but not time; again, for the
 random effects specification, we leave both time and money in there, following
 Barr's advice
v_m1c_noMoney <- lmer(rating ~ s_time + (1 + s_time + s_money | pp_code), data =
 v5)
summary(v_m1c_noMoney)

# we compare again the reduced model to the full model
anova_v_m1c_Money <- anova(v_m1c_noMoney, v_m1c)

# again, R is so kind to refit both models for me using ML instead of REML;
 after it has done that, it does the actual model comparison
anova_v_m1c_Money # good to know, it's also highly significant, consistent with
 our other approaches to get p values


# OK, so there are other (perhaps more fancy ways) to also do this model
 comparisons. One is the Kenward-Roger F test approach (which is used by
 Anova(test = 'F)), the other is a parametric bootstrap model comparison. Both
 of them are available from the pbkrtest package. Thus we need to load that
 package
```

```
# IMPORTANT: in contrast to the anova() command, both approach require you to
 first put in the larger model, followed by the smaller model. I.e., it's exact
 the opposite order. And pbkrtest will NOT rearrange things for you.

# let's do first the conditional F test with Kenward-Roger df adjustment

# first for Time
KRmodcomp_v_m1c_Time <- KRmodcomp(v_m1c, v_m1c_noTime)
# let's have a look:
KRmodcomp_v_m1c_Time # again highly significant

# now the same for Money
KRmodcomp_v_m1c_Money <- KRmodcomp(v_m1c, v_m1c_noMoney)
KRmodcomp_v_m1c_Money # again, highly significant


# Now, as you probably remember, our residuals were not sooo nicely distributed,
 so it might be safer to use some bootstrap approach
# the pbkrtest also offers a model comparison based on parametric bootstrapping
# often, parametric bootstrapping takes a long time, as many many models are
 fitted first, and after that the results of this simulations are compared
# since this is computationally intensive, one can easily specifiy that more
 than one processor core can be used in parallel, to speed things up

# as part of pbkrtest, there is a command detectCores() which tells you how many
 cores your computer has
detectCores()
# the laptop I am working on right now has 4 cores

# WARNING: for the commands below, do not tell R to use ALL of your cores. I did
 that once and the problem is that no processing power is then left for other
 processes your computer might want to do (like, on a Mac, the Finder and other
 very important functions)

# Therefore, I always leave at least 1 core for other things than R

# OK, so let's first see how long it takes just using 1 core

# to be able to see how long the command takes (without me sitting in front of
 the computer with a stop watch), I use the command Sys.time(). It saves the
 current time into a variable. I take such a time before starting the command
 and right after the command has finished. I.e., I highlight and run all 3
 commands at the same time, to make sure as soon as one command has finished,
 the next one is started

# we can specify the number of simulations in the command using the argument
 nsim. The default is 1000, which seems reasonable. The more simulations, the
 longer the command takes, of course.

# When I am afraid that a command might take very long, I often measure how it
 long it takes for, say, 3 simulations. That way I can estimate how long it
 might take for 1000 simulations (that way I can for example decide to start
 that command before I go to bed)
```

```
# NOTE: there is a similar function called system.time(); it is a bit different
 as it tell you how long it took to run a command that is in the parantheses of
 system.time(). You you might use it for example for something like
 system.time(PBmodcomp(v_m1c, v_m1c_noTime, nsim = 3))




# BREAK -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK
 -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK -- BREAK
# OK, so now I'm taking a break. To save all the different models that I ran
 (and all the preprocessing steps), I save my work so far in a work space that I
 can later load.

# just so I know where it will be saved, I check what my current working
 directory is
getwd()

# ok, it's here:
 /Users/B.Figner/Dropbox/Radboud/Teaching/MultilevelClass/2014/Data/ITC


save.image("MixMod2014_ICT_ValuationRatings_8March2014.RData")


#
 @@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
 @@@
# BACK TO WORK!! BACK TO WORK!! BACK TO WORK!! BACK TO WORK!! BACK TO WORK!!
 BACK TO WORK!! BACK TO WORK!! BACK TO WORK!! BACK TO WORK!!

# OK, break is over, I now load the previously saved workspace

# I have to set my working directory again, so that it's the same it was before
 I had my break (during which I closed R). Of course it would be easier to just
 scroll up and run the command which is pretty close to the top. But to make
 sure you understand what I mean, I paste it here:
setwd('~/Dropbox/Radboud/Teaching/MultilevelClass/2014/Data/ITC')


# now I can load my work space
load("MixMod2014_ICT_ValuationRatings_8March2014.RData")

# NOTE: You will have to still load the libraries, that you want to use, as this
 is not saved with the work space. You also have to re-run your contrast
 settings and might want to check, in case you have ordered factors, whether
 they are still ordered or you have to re-run the command.

# Usually, I would scroll up and run the contrast setting commands and the
 loading of packages. But since this is an example script, I show you here which
 commands I typically re-run:


options(scipen=10) #so serials are shown in regular)
```

```
# set contrasts to sum-to-zero (aka deviation coding) for unordered factors and
 polynomial for ordered factors
options(contrasts=c("contr.sum", "contr.poly"))

# Load libraries (I'm loading here only the ones I already know I will be
 needing; perhaps I have to load some more later)

library(lme4)
library(pbkrtest)

# check
v5$o_money # good


t1 <- Sys.time()
PBmodcomp_v_m1c_Time_3sims <- PBmodcomp(v_m1c, v_m1c_noTime, nsim = 3)
t2 <- Sys.time()

t2 - t1 # ok, that was short: 0.677042 secs. So 1000 shouldn't take days either


# now let's do it with 1000 sims
t3 <- Sys.time()
PBmodcomp_v_m1c_Time_1000 <- PBmodcomp(v_m1c, v_m1c_noTime, nsim = 1000)
t4 <- Sys.time()

t4 - t3 # as expected, this was also not too bad: 2.461579 mins

# BUT: Note that if you have complicated models, this can take days! Thus, it's
 nearly always worth checking with only a few simulations

# OK, but I also wanted to show you how you can tell PBmodcomp to use more than
 1 processor core (not necessarily needed here, as it is relatively fast); but
 sometimes it can reduce the duration from, say, a few days to a few hours

# As my computer has 4 cores, I am going to use 3 cores for the parametric
 bootstrapping

# you can get information how to specify the number of processors by getting the
 respective help file:
?PBmodcomp

# Ok, so we have to run a few things; we have to specify a cluster; but luckily
 the helpfile tells us how to do this. This information is towards the bottom of
 the helpfile, in the examples section
# I copy/paste the relevant bit here:

# ## Do computations with multiple processors:
# ## Number of cores:
# (nc <- detectCores())
# ## Create clusters
# cl <- makeCluster(rep("localhost", nc))
# ## Then do:
# PBmodcomp(beet0, beet_no.harv, cl=cl)

# ## It is recommended to stop the clusters before quitting R:
# stopCluster(cl)


# OK, so let's try to do that

# first, save the number of cores in a variable
n_cores <- detectCores()
n_cores # ok, that's 4. NOTE: I don't want to use all my cores but only 3 of
 them (i.e., n_cores - 1)

# Now we create the clusters

clusters <- makeCluster(rep("localhost", n_cores - 1)) # note that I used
 n_cores - 1 (in contrast to the example in the help file); this way, I core
 will be left for other processes

# let's check what is in the variable clusters
clusters # ok, some funny text, but I guess R will know what to do with that...


t5 <- Sys.time()
PBmodcomp_v_m1c_Time_1000_3cores <- PBmodcomp(v_m1c, v_m1c_noTime, nsim = 1000,
 cl = clusters) # so in contrast to the previous, I only added "cl = clusters"
t6 <- Sys.time()

t6 - t5 # ok, so that took now 1.339562 mins, so we saved about 1 minute

# the fewer cores you have left for other processes, the more your computer
 might slow down if you do other things why are is doing the bootstrapping.
 Also, during the bootstrapping, R often becomes non-responsive, but that's ok
 and not a warning sign.



# OK, but actually we haven't looked at the results at all, so let's do that
 know!

PBmodcomp_v_m1c_Time_1000_3cores

# Parametric bootstrap test; time: 80.37 sec; samples: 999 extremes: 0;
# large : rating ~ s_time + s_money + (1 + s_time + s_money | pp_code)
# small : rating ~ s_money + (1 + s_time + s_money | pp_code)
         # stat df       p.value
# LRT    29.756  1 0.00000004899 ***
# PBtest 29.756            0.001 ***

# It gives the results for the parametric bootstrap test (that's why we did the
 whole thing), but also gives the results for an LRT. The result in the same
 conclusion, namely that Time is highly significant.
```

```
# Note also that it tells us how long it took and how many samples it used. It
 is based on 999 samples; this might seem odd that it's not 1000, but it's not
 uncommon that some simulations fail (due to non-convergence usually). If it's
 only such a small proportion (a few out of a 1000), this is no reason to worry
 at all.



# So now let's do the same for Money. I do only the 3-core 1000 simulation thing
 here

t7 <- Sys.time()
PBmodcomp_v_m1c_Money_1000_3cores <- PBmodcomp(v_m1c, v_m1c_noMoney, nsim =
 1000, cl = clusters)
t8 <- Sys.time()
t8 - t7 # that took about 1.4 min

PBmodcomp_v_m1c_Money_1000_3cores

# Parametric bootstrap test; time: 83.88 sec; samples: 999 extremes: 0;
# Requested samples: 999 Used samples: 997 Extremes: 0
# large : rating ~ s_time + s_money + (1 + s_time + s_money | pp_code)
# small : rating ~ s_time + (1 + s_time + s_money | pp_code)
        # stat df         p.value
# LRT    39.56  1 0.0000000003181 ***
# PBtest 39.56           0.001002 **

# NOTE: for the first time now here, we find a slightly different result: While
 the LRT suggests that Money is significant is with p < .001, the PBtest
 suggests p < .01. While it doesn't really matter that much (in both cases, we
 conclude that it's significant), for a paper, I might report the p < .01, just
 to be on the conservative (i.e., safe) side.

# ok and the helpfile recommends to 'stop the cluster' after one is finished
 with this PBmodcomp business, so let's do that
stopCluster(clusters)



# As final way to get p values, I will now use the bootMer function. In contrast
 to the model-comparison approaches above, the bootMer approach tests whether a
 coefficient is significantly different from zero, rather than testing whether a
 model with a predictor is a significantly better fit than a model without that
 predictor. Both approaches should lead to the same conclusions typically, but
 they sometimes might differ somewhat.



# As you already know, we first have to define a function that we'll afterwards
 need


FUN_bootMer <- function(fit) {
 return(fixef(fit))
}


# We then run our bootstrap command, bootMer
# NOTE: bootMer() can also take a long time to run and therefore it has a built-
 in easy way to use more than 1 core (it's even easier than in PBmodcomp)

# have a look at ?bootMer to see the details

# If you use only 1 core, you can ask for a simple 'progress bar' so that you
 see how R is making progress. Unfortunately that progress bar is not shown when
 you tell R to use more than 1 core.

# OK, so here first we just use 1 core, but tell R to show us a progress bar

# As for PBmodcomp, it typically makes sense to first let it run with only a
 small number of iterations (e.g., 3) to get a sense for how long the full
 number of simulations (1000 typically) will take.

# But as I already demonstrated for PBmodcomp how you can do that, I'm not going
 to show it here again. So we just do the 1000 simulations right away.


# OK, as I said, first using just 1 core. Accordingly, I can use the progress
 bar, which gives me some idea how far R is already.
# The code for the progress bar is this: .progress = "txt", PBargs = list(style
 = 3)
t9 <- Sys.time()
boot_v_m1c <- bootMer(v_m1c, FUN_bootMer, nsim = 1000, .progress = "txt", PBargs
 = list(style = 3), type = "parametric")
t10 <- Sys.time()

t10 - t9 # ok, so that was pretty fast actually: 1.164234 mins


# Just out of curiosity, we're trying it now with 3 cores
# note that I specify 2 things:
# parallel = "multicore" ---> this works for Mac; for PC, you might have to use
 instead parallel = "snow" (not sure, I don't have a PC handy, so try it out)
# ncpus = 3  ---> this tells R how many cores (ncpus = number of cpu's) it
 should use. as I have 4 in total but want to leave 1, i specify ncpus = 3

t11 <- Sys.time()
boot_v_m1c <- bootMer(v_m1c, FUN_bootMer, nsim = 1000, type = "parametric",
 parallel = "multicore", ncpus = 3)
t12 <- Sys.time()
t12 - t11 # that took 45.18787 secs


# Good, now to get p values with this approach, we need to compute the
 confidence intervals. for that, we need to load the package boot, to be able to
 use the command boot.ci

# first, let's look at we created and saved into boot_v_m1c
```

```
as.data.frame(boot_v_m1c)

# it is a data frame-like object with 3 columns (intercept, s_time, s_money;
 i.e., 1 column for each fixed effect in our model) and 1000 rows (1 row per
 simulation that we asked for)


# First, we want a 95% confidence interval (CI) for the intercept
boot_v_m1c_CI_intercept_95 <- boot.ci(boot_v_m1c, index = 1, conf = 0.95,
 type=c("norm", "basic", "perc")) #
boot_v_m1c_CI_intercept_95
# BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
# Based on 1000 bootstrap replicates

# CALL :
# boot.ci(boot.out = boot_v_m1c, conf = 0.95, type = c("norm",
    # "basic", "perc"), index = 1)

# Intervals :
# Level      Normal              Basic              Percentile
# 95%   (67.89, 75.56 )   (67.95, 75.86 )   (67.65, 75.56 )
# Calculations and Intervals on Original Scale

# So: 0 is not part of the 95% CI, thus I can conclude that the intercept is
 significant with p < .05


# If we want to check whether it's also significant with p < .01, we have to ask
 for the 99% CI. We can do that by specifying conf = 0.99
boot_v_m1c_CI_intercept_99 <- boot.ci(boot_v_m1c, index = 1, conf = 0.99,
 type=c("norm", "basic", "perc")) #
boot_v_m1c_CI_intercept_99
# BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
# Based on 1000 bootstrap replicates

# CALL :
# boot.ci(boot.out = boot_v_m1c, conf = 0.99, type = c("norm",
    # "basic", "perc"), index = 1)

# Intervals :
# Level      Normal              Basic              Percentile
# 99%   (66.68, 76.76 )   (66.72, 77.29 )   (66.22, 76.78 )
# Calculations and Intervals on Original Scale
# Some basic intervals may be unstable
# Some percentile intervals may be unstable

# OK, so it's significant with p < .01. But note that we get the information
 that some intervals might be 'unstable.' One reason is that we ask how many
 samples are outside of the 99% range. As we have asked for 1000 simulations, it
 is expected that these are 10 simulations. Which is just a small absolute
 number. So, if you want to be surer, you might ask for more simulations. But
 looking at the CIs, they are *really* not including 0, so it's probably safe to
 assume that it's significant at p < .01
```

```
# We could also now check for p < .001. But remember, this is now even more
 extreme: we have 1000 simulations and if we ask for a 99.9% CI, we will have
 only very few simulations (1 out of 1000), so with 1000 simulations, it's
 probably not trustworthy at all. Thus, if we want a trustworthy estimate for CI
 = 99.9%, we probably should include the number of simulations to 10,000

boot_v_m1c_CI_intercept_999 <- boot.ci(boot_v_m1c, index = 1, conf = 0.999,
 type=c("norm", "basic", "perc")) #
# And now we actually get a warning:
# Warning messages:
# 1: In norm.inter(t, (1 + c(conf, -conf))/2) :
  # extreme order statistics used as endpoints
# 2: In norm.inter(t, alpha) : extreme order statistics used as endpoints

# Thus, this suggests that we are asking for something that doesn't make too
 much sense (in other words, if we want to know the 99.9% CI, we should use more
 simulations)

boot_v_m1c_CI_intercept_999
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = boot_v_m1c, conf = 0.999, type = c("norm",
    "basic", "perc"), index = 1)

Intervals :
# Level      Normal              Basic              Percentile
# 99.9%   (65.29, 78.16 )   (65.29, 77.91 )   (65.59, 78.22 )
# Calculations and Intervals on Original Scale
# Warning : Basic Intervals used Extreme Quantiles
# Some basic intervals may be unstable
# Warning : Percentile Intervals used Extreme Quantiles
# Some percentile intervals may be unstable

# It still gives an answer, but the comment actually sound a bit more severe
 now, for example: Warning : Percentile Intervals used Extreme Quantiles


# OK, so let's ask for the CIs for time

# hmm, let me quickly check which column time was again (not sure whether it was
 the 2nd or third)

head(as.data.frame(boot_v_m1c))
# ok, it was the second column. Thus, I specify index = 2

boot_v_m1c_CI_Time_95 <- boot.ci(boot_v_m1c, index = 2, conf = 0.95,
 type=c("norm", "basic", "perc")) #
boot_v_m1c_CI_Time_95
# BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
# Based on 1000 bootstrap replicates
# CALL :
# boot.ci(boot.out = boot_v_m1c, conf = 0.95, type = c("norm",
    # "basic", "perc"), index = 2)

# Intervals :
# Level      Normal              Basic              Percentile
# 95%   (-10.270,  -5.744 )   (-10.254,  -5.747 )   (-10.182,  -5.675 )
# Calculations and Intervals on Original Scale

# significant at p < .05


# same for CI = 99%
boot_v_m1c_CI_Time_99 <- boot.ci(boot_v_m1c, index = 2, conf = 0.99,
 type=c("norm", "basic", "perc")) #
boot_v_m1c_CI_Time_99
# BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
# Based on 1000 bootstrap replicates

# CALL :
# boot.ci(boot.out = boot_v_m1c, conf = 0.99, type = c("norm",
    # "basic", "perc"), index = 2)

# Intervals :
# Level      Normal              Basic              Percentile
# 99%   (-10.981,  -5.033 )   (-11.168,  -4.692 )   (-11.237,  -4.762 )
# Calculations and Intervals on Original Scale
# Some basic intervals may be unstable
# Some percentile intervals may be unstable

# ok, significant at p < .01


# Thus, we can conclude that in all the many different approaches we used to get
 p values, both Time and Money were always significant. Usually (for a paper or
 your thesis), you would typically only use 1 approach to get all p values
 (except if you're unsure about, say, Anova(test = 'F') and want to verify using
 some bootstrapping approach).


# OK, basically we're done now. Below is one more model for the particularly
 curious ones



# ........................................................................
# test whether there is an interaction between time and money
# i included this here for 2 reasons: (1) pure curiosity (you didn't have to do
 that and actually not sure I would do that if it were a real study, as I don't
 have an hypothesis regarding an interaction term)
# (2) some of you might have included an interaction term and I want to point
 out that, if you do that, (a) it is even more important to have centered
 predictors and (b) you need to include the interaction term also in the random
 effects

v_m1c_interact <- lmer(rating ~ s_time * s_money + (1 + s_time * s_money |
 pp_code), data = v5)
summary(v_m1c_interact)
Anova(v_m1c_interact, type = 3, test = 'F') # as this was not part of the home
 work, I do only this way to get p values, but from the examples above, you can
 derive how the other approaches would work also in this case

# ..........................................................................




# OK, so now that everything is finished, I save the work space again, in case I
 want to do some more stuff later or look again at all the boostrapping results
 or whatever

save.image("MixMod2014_ICT_ValuationRatings_8March2014b.RData")

load('MixMod2014_ICT_ValuationRatings_8March2014b.RData')



#................................................................................
 ........................
#................................................................................
 ........................
#................................................................................
 ........................
# Below here is some additional stuff that was NOT part of the homework, but
 might still be of interest:

# 1. linear and quadratic continuous predictors
# 2. plotting linear and quadratic effects
# 3. pretty plots: (a) ggplot2 and (b) xyplot()
# 4. "extract" intercepts and slopes for each participant

# -------------------------------------------------------------------------------
 ---------------
# 1. linear and quadratic predictors

# there are different ways to test for non-linear effects
# one common approach is to use not only the continuous linear predictor, but to
 also a squared one
# again, there are different ways to produce a squared predictor, some ways are
 better than others

head(v5)

# the most typical approach is to simply square the centered continuous
 predictor
# that often leads to some collinearity (i.e., the linear and the quadratic
 predictors are somewhat correlated)
```

```
# if one doesn't center the linear predictor before squaring it, the correlation
 is much much higher typically

# below, I demonstrate that

# squaring the centered time and money predictors
v5$q_time <- v5$c_time ^ 2
v5$q_money <- v5$c_money ^ 2

# let's check the correlation between the linear and quadratic predictors
# for rcor.test(), I load the ltm library
library(ltm)

# here, I create squared predictors from the non-centered time predictors
v5$q_time_NonCentered <- v5$time ^ 2

# the correlation between the non-centered linear and quadratic predictors is
 extremely high!
with(v5, rcor.test(cbind(time, q_time_NonCentered)))

# the correlation between the centered linear and quadratic predictors is much
 lower (but there's still a correlation)
with(v5, rcor.test(cbind(c_time, q_time)))

# R's command poly() can create polynomial predictors that are uncorrelated
 (which is of course very cool for our purposes)
?poly # have a look to get the help file

# poly(v5$time, 2) tells R to create the linear and quadratic predictors. The
 [,1] tells are to use only the first column (which is the linear one)
v5$poly_lin_Time <- poly(v5$time, 2)[,1]

# the [,2] tells are to use the 2nd column (which is the quadratic one)
v5$poly_quadr_Time <- poly(v5$time, 2)[,2]

# check the correlation between these two: nice, it's 0!
with(v5, rcor.test(cbind(poly_lin_Time, poly_quadr_Time)))


mean(v5$poly_lin_Time) # 0, nice!
mean(v5$poly_quadr_Time) # 0, nice!

# NOTE: sometimes, the actual values in the predictors with poly are tiny tiny
 numbers; to get them into a more reasonable range, I sometimes just multiply
 everything with 100 or 1000 or whatever number gets them in a range without
 lots of decimals (I'm not doing this here)

# same for money
v5$poly_lin_Money <- poly(v5$money, 2)[,1]
v5$poly_quadr_Money <- poly(v5$money, 2)[,2]

# here, I run the same model as before, with time and money as continuous
 predictor, just to verify that I get the same results as with the centered or
 scaled predictors. As you will see, the size of the coefficients is different
 (which is not surprising), but the t values and p values that we get with
 Anova() etc are all the same
v_m1c_poly <- lmer(rating ~ poly_lin_Time + poly_lin_Money + (1 + poly_lin_Time
 + poly_lin_Money | pp_code), data = v5)
summary(v_m1c_poly)
Anova_v_m1c_poly_3F <- Anova(v_m1c_poly, type = 3, test = 'F')


# now I'm curious to test for both linear and quadratic effects at the same time
v_m1c_poly_lin_quad <- lmer(rating ~ poly_lin_Time + poly_quadr_Time +
 poly_lin_Money + poly_quadr_Money + (1 + poly_lin_Time + poly_quadr_Time +
 poly_lin_Money + poly_quadr_Money | pp_code), data = v5, control =
 lmerControl(optCtrl = list(maxfun = 10000)))
# I get some warning messages, but they actually don't seem to be that
 problematic (one has to carefully check the model plots, though!)
# Warning messages:
# 1: In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,
 :
  # Model failed to converge with max|grad| = 1.79679 (tol = 0.001)
# 2: In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,
 :
  # Model failed to converge: degenerate  Hessian with 1 negative eigenvalues

summary(v_m1c_poly_lin_quad) # this looks reasonable to me
# the coefficients for the quadratic terms have in both cases the opposite sign
 of the linear predictors, which suggests that the effects might indeed as
 Weber-Fechner would suggest (namely that sensitivity decreases with increasing
 value of the predictor). But one would want to verify that



# ---------------------------------------------------------------------------
 ---------------
# 2. plotting linear and quadratic effects

# one way to check for linear and non-linear effects in such cases is to create
 scatter plots that show a "smoothed" curve (sometimes in addition to a straight
 line, i.e., linear effect, so that the comparison is easier)

# the commands here below create a plot showing the linear (red) and non-linear
 (blue) relationship between rating and time
# as we can see, the blue line indeed first shows a more step drop that then
 flattens out, consistent with the Weber-Fechner idea
with(v5, plot(time, rating))
with(v5, abline(lm(rating ~ time), col = 'red'))
with(v5, lines(lowess(time, rating), col = 'blue'))


# the same for money
with(v5, plot(money, rating))
with(v5, abline(lm(rating ~ money), col = 'red'))
with(v5, lines(lowess(money, rating), col = 'blue'))

# for money, the quadratic effect looks smaller, but the difference between the
```

```
 red (linear) and blue (quadratic) line is biggest in the middle, suggesting
 also that the difference between approx Euro 20 and approx Euro 50 is larger
 than the difference between approx Euro 50 and approx Euro 89


# the library car has the function scatter.smooth() which creates a scatter plot
 with an added "smoothed" line
# one way to visualize non-linear relationships is to create a scatter plot that
 adds a smoothed line; e.g., the function scatter.smooth from car
# for time
with(v5,scatter.smooth(time, rating))

# for money
with(v5,scatter.smooth(money, rating))


plot(v_m1c_poly_lin_quad) # looks quite similar to our only-linear model above

densityplot(resid(v_m1c_poly_lin_quad)) # also looks ok-ish

plot(v5$rating, fitted(v_m1c_poly_lin_quad)) # that looks a bit better than the
 linear-only model (which is not surprising really, as this model here is more
 complex)

# some *relatively* quick ways to get p values: both the linear and quadratic
 terms are significant
Anova_v_m1c_poly_lin_quad_3F <- Anova(v_m1c_poly_lin_quad, type = 3, test = 'F')


drop1_v_m1c_poly_lin_quad_3F <- drop1(v_m1c_poly_lin_quad, ~., test = 'Chisq')



# same, but using centered linear and quadratic terms: works less well then with
 the poly predictors; most likely due to the correlation between linear and
 quadratic predictors; however, when I increase the number of iterations A LOT,
 I do get pretty similar estimates as with the poly model, which is a good sign

v_m1c_c_lin_quad <- lmer(rating ~ c_time + q_time + c_money + q_money + (1 +
 c_time + q_time + c_money + q_money | pp_code), data = v5, control =
 lmerControl(optCtrl = list(maxfun = 10000000)))
# --> seems to give more serious warnings, even with MUCH increased number of
 iterations
# Warning messages:
# 1: In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,
 :
  # Model failed to converge with max|grad| = 732.948 (tol = 0.001)
# 2: In if (resHess$code != 0) { :
  # the condition has length > 1 and only the first element will be used
# 3: In checkConv(attr(opt, "derivs"), opt$par, checkCtrl = control$checkConv,
 :
  # Model is nearly unidentifiable: very large eigenvalue
 # - Rescale variables?;Model is nearly unidentifiable: large eigenvalue ratio
 # - Rescale variables?
```

```
summary(v_m1c_c_lin_quad)

save.image("MixMod2014_ICT_ValuationRatings_9March2014a.RData")



# ------------------------------------------------------------------------------
 ---------------
# 3. pretty plots: ggplot2 and xyplot()

# ..................................
# (a) ggplot2
# Here's some code how to create a much prettier version of the interaction plot
library(ggplot2)

# very simple version
ggplot(data = v5, aes(x = time, y = rating, colour = o_money, group = o_money))
 + stat_summary(fun.y = mean, geom = "point") + stat_summary(fun.y = mean, geom
 = "line")

head(v5)



# making it a bit prettier (larger font sizes for the x and y axis)
ggplot(data = v5, aes(x = time, y = rating, colour = o_money, group = o_money))
 + stat_summary(fun.y = mean, geom = "point") + stat_summary(fun.y = mean, geom
 = "line") + theme(axis.text.x = element_text(size = 15), axis.title.x =
 element_text(size = 20), axis.text.y = element_text(size = 15), axis.title.y =
 element_text(size = 20))



#larger font sizes for the legend
ggplot(data = v5, aes(x = time, y = rating, colour = o_money, group = o_money))
 + stat_summary(fun.y = mean, geom = "point") + stat_summary(fun.y = mean, geom
 = "line") + theme(axis.text.x = element_text(size = 15), axis.title.x =
 element_text(size = 20), axis.text.y = element_text(size = 15), axis.title.y =
 element_text(size = 20), legend.title = element_text(size = 14), legend.text =
 element_text(size = 14))



# ............................................................................
# (b) xyplot()
# per-participant plots (to show how variable or similar participants are)
# use xyplot from the package lattice to plot a separate panel for each
 participant

?xyplot


# the effect of time
xyplot(rating ~ time | pp_code, groups = pp_code, data = v5, type = c('p', 'r'))
```

```
# same for money
xyplot(rating ~ money | pp_code, groups = pp_code, data = v5, type = c('p',
 'r'))

# Show Time and Money!
# without legend
xyplot(rating ~ money | pp_code, groups = time, data = v5, type = c('p', 'r'))

# same, but with legend
xyplot(rating ~ time | pp_code, groups = o_money, data = v5, type = c('p', 'r'),
 auto.key = TRUE)

# the other way around: money on x axis and time as separate lines
xyplot(rating ~ money | pp_code, groups = time, data = v5, type = c('p', 'r'),
 auto.key = TRUE)



# --------------------------------------------------------------------------------
 ---------------
# 4. "extract" intercepts and slopes for each participant

# fixef() gives you the fixed effect estimates
fixef(v_m1c)

# ranef() gives you the random intercept adjustments and the random slope
 adjustments per participant (or whatever your grouping variable is)
ranef(v_m1c)

# coef() gives you the fixed and random part added together: this is often the
 most useful information when we want, for example correlate these individual
 differences with some other measures that we might have; or plot the individual
 differences in these estimates
coef(v_m1c)

# If you want to save them (use row.names = TRUE to also save the first column
 with the participant codes!)
v_m1c_coefs <- coef(v_m1c)$pp_code
write.csv(v_m1c_coefs, row.names = TRUE, file = 'v_m1c_coefs.csv')
```