Portfolio 1, Methods 3, 2021, autumn semester

Sara Krejberg, Niels A. Krogsgaard, Sigurd F. Sørensen, Laura W. Paaby

29/9 - 2021

This assignment has been made in collaboration in Studygroup 9. At each exercise initials will indicate, who has made that particular exercise. However, we must emphasise that almost everything have been done as a group making it hard to distinguish, who has made what specific contributions. The members of the studygroup have the following initials (Name, initials, studynumber): - Laura Wulff Paaby (LWP), 202006161 - Niels Aalund Krogsgaard (NAK), 202008114 - Sara Engsig Krejberg (SEK), 202007949 - Sigurd Fyhn Sørensen (SFS), 202006317

Assignment 1: Using mixed effects modelling to model hierarchical data

In this assignment we will be investigating the *politeness* dataset of Winter and Grawunder (2012) and apply basic methods of multilevel modelling.

Dataset

The dataset has been shared on GitHub, so make sure that the csv-file is on your current path. Otherwise you can supply the full path.

```
politeness <- read.csv('/Users/laura/Desktop/GitHub methods 3/github_methods_3/week_02/politeness.csv') ## read in data
```

Exercises and objectives

The objectives of the exercises of this assignment are:
1) Learning to recognize hierarchical structures within datasets and describing them
2) Creating simple multilevel models and assessing their fitness
3) Write up a report about the findings of the study

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below
REMEMBER: This assignment will be part of your final portfolio

Exercise 1 - describing the dataset and making some initial plots

1, (LWP)) Describe the dataset, such that someone who happened upon this dataset could understand the variables and what they contain

The dataset is a result of a study, which investigated the properties of formal and informal speech register. To do so different variables were measured, to enlighten what might characterize the register. The variables are what we see in the dataset: ***f0mn****: the mean frequency of the pitch of the sentence uttered i Hz.* ***scenarios****: the number indicates what specific scenario the subject has been presented with in that observation, e.g. "You are in the professor's office and want to ask for a letter of recommendation" (Grawunder & Winter et al., 2011, p. 2) is an example of a scenario. I must add that this specific scenario was aimed at producing formal speech, while a scenario much the same was aimed at producing informal speech.* ***gender****: the gender of the participant (f = female, m = male)* ***total_duration****: duration of response in seconds ,* ***hiss_count****: the amount of loud hissing breath intake (hiss_count). T* ***attitude****: is either polite or informal, which are variables the scenarios are categorized by. The subjects are the participants of the study - F females whereas M is male.*

```
i. Also consider whether any of the variables in _politeness_ should be encoded as fa
ctors or have the factor encoding removed. Hint: ```?factor```
```

```
#investigating the data
#ls.str(politeness)
glimpse(politeness)
```

```
## Rows: 224
## Columns: 7
## $ subject        <chr> "F1", "F1", "F1", "F1", "F1", "F1", "F1", "F1", "F1", …
## $ gender         <chr> "F", "F", "F", "F", "F", "F", "F", "F", "F", "F", "F",…
## $ scenario       <int> 1, 1, 2, 2, 3, 3, 4, 4, 5, 5, 6, 6, 7, 7, 1, 1, 2, 2, …
## $ attitude       <chr> "pol", "inf", "pol", "inf", "pol", "inf", "pol", "inf"…
## $ total_duration <dbl> 18.392, 13.551, 5.217, 4.247, 6.791, 4.126, 6.244, 3.2…
## $ f0mn           <dbl> 214.6, 210.9, 284.7, 265.6, 210.6, 285.6, 251.5, 281.5…
## $ hiss_count     <int> 2, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 2, 0, 0, 0, 0, 0, …
```

```
#making gender, attitude and scenaruo into facter and adding them to the dataframe:
attitude.f = as.factor(politeness$attitude)
gender.f = as.factor(politeness$gender)
scenario.f = as.factor(politeness$scenario)

politeness <- politeness %>%
  mutate(attitude.f, gender.f, scenario.f)
```

2, (NAK)) Create a new data frame that just contains the subject *F1* and run two linear models; one that expresses *f0mn* as dependent on *scenario* as an integer; and one that expresses *f0mn* as dependent on *scenario* encoded as a factor

```
#making a dataframe only for the first subject (F1)
F1_df <- politeness %>%
  filter(subject == 'F1')


## Running the two linear models
#model 1 with scenario as integer:
F1_model1 <- lm(f0mn ~ scenario, data = F1_df)
#model 2 with scenario as factor
F1_model2 <- lm(f0mn ~ scenario.f, data = F1_df)

summary(F1_model1)
```

```
##
## Call:
## lm(formula = f0mn ~ scenario, data = F1_df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -44.836 -36.807   6.686  20.918  46.421
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  262.621     20.616  12.738 2.48e-08 ***
## scenario      -6.886      4.610  -1.494    0.161
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 34.5 on 12 degrees of freedom
## Multiple R-squared:  0.1568, Adjusted R-squared:  0.0865
## F-statistic: 2.231 on 1 and 12 DF,  p-value: 0.1611
```

```
summary(F1_model2)
```

```
##
## Call:
## lm(formula = f0mn ~ scenario.f, data = F1_df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -37.50 -13.86   0.00  13.86  37.50
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   212.75      20.35  10.453  1.6e-05 ***
## scenario.f2    62.40      28.78   2.168   0.0668 .
## scenario.f3    35.35      28.78   1.228   0.2591
## scenario.f4    53.75      28.78   1.867   0.1041
## scenario.f5    27.30      28.78   0.948   0.3745
## scenario.f6    -7.55      28.78  -0.262   0.8006
## scenario.f7   -14.95      28.78  -0.519   0.6195
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 28.78 on 7 degrees of freedom
## Multiple R-squared:  0.6576, Adjusted R-squared:  0.364
## F-statistic:  2.24 on 6 and 7 DF,  p-value: 0.1576
```

```
i. Include the model matrices, $X$ from the General Linear Model, for these two model
s in your report and describe the different interpretations of _scenario_ that these
entail
```

```
#making a model matrix for each model:

X1 <- model.matrix(F1_model1) #integer model
X2 <- model.matrix(F1_model2) #factor model
```

The design matrix for the model with scenario as an integer take scenario as a continuous variable where going from 2 to 4 is some meaningful doubling. We therefore not only take the scenarios as having some kind of meaningful order, but also take scenario 6 is being double the amount of scenario 3, all in all treating it as a continuous variable (which is of course wrong, since we have no expectation that f0mn will change systematically with increasing scenario number).

The design matrix for the model with scenario as a factor take scenario to be a categorical variable. In the design matrix we can see all the different observations of scenario coded as dummy variables, so every factor level has its own beta-value connected to it. Scenario 1 is "excluded" since that will be the intercept.

Description of both models and matrixes: *the factored model:* The design matrix is a [14x7] matrix, so we will get the following β_{0-6}. This is also shown by the summary of a our linear regression model. *A simple regression f0mn ~ scenario was conducted. Scenario seemed to account for 36.4% of the variance in f0mn

following adjusted R^2. F(1,6) = 2.24, p >0.5) all beta values were insignificant. We only have 14 observations spread out over 7 different levels. So the high p-value is most likely due to sample-size. A further power-analysis could show the required sample size required.

the integer model: Now that scenario is encoded as an integer the design matrix will be a [14x2] matrix. Our model will therefore only give us β_{0-1} and not a β for each level of scenario as done in the previous model. This model assumes that there is a constant increment of f0mn following a "increase" in scenario (if you can even talk about a unit increase of scenario). This would only make sense if scenarios were ordered as getting harder and harder. The model is again f0mn ~ scenario F(1,12) = 2.231, p>0.5) with an adjusted R^2 = 0.0865 showing an explained variance of 8.65% (β_1 = -6.886, SE = 4.6, t = -1.5, p>0.16.) Again such a small sample size might be tricky to work with.

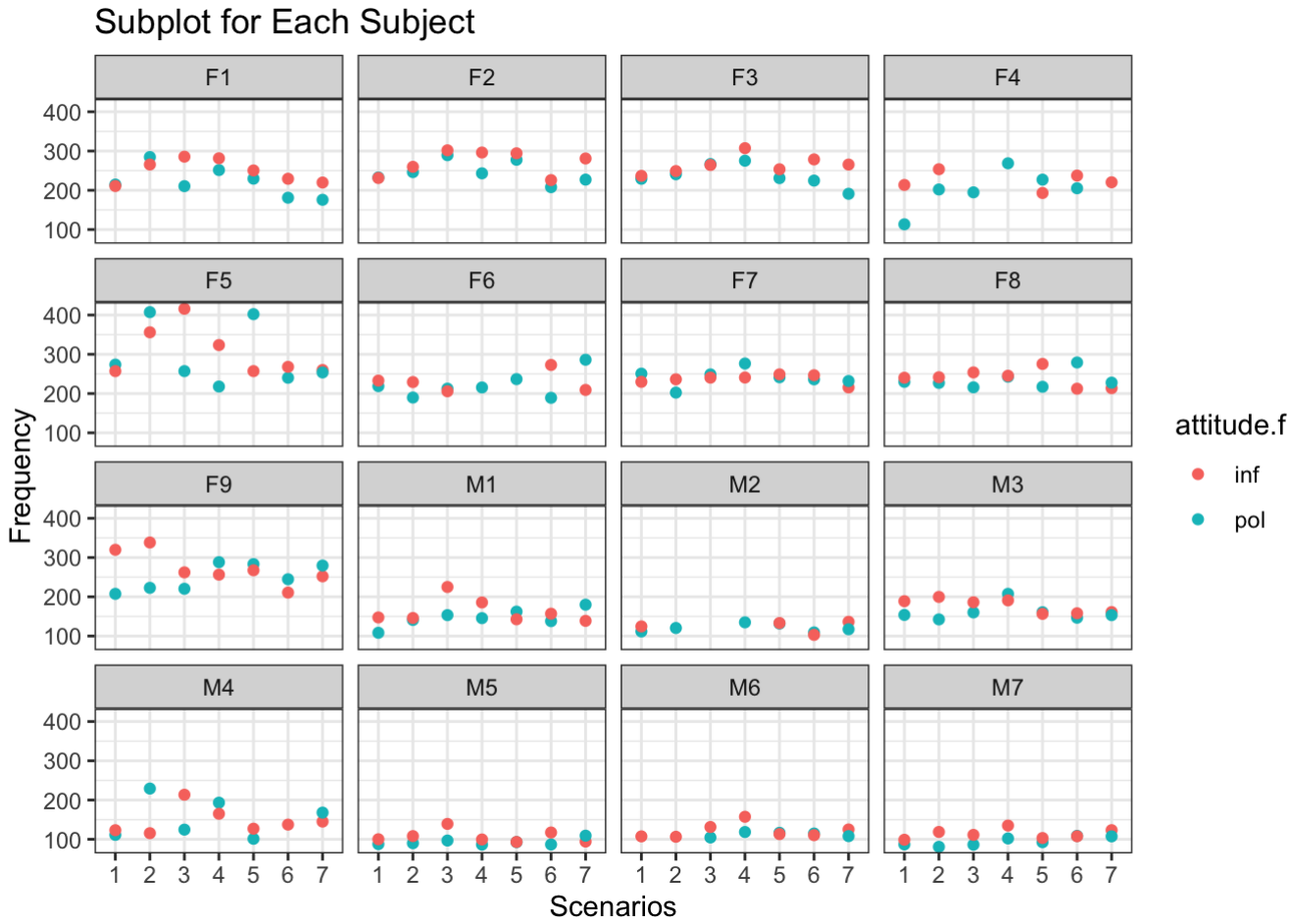
```
ii. Which coding of _scenario_, as a factor or not, is more fitting?
```

In this context it is only appropriate to code scenario as a factor. The reasons are given in the previous exercise.

3, (SEK)) Make a plot that includes a subplot for each subject that has *scenario* on the x-axis and *f0mn* on the y-axis and where points are colour coded according to *attitude* i. Describe the differences between subjects

```
politeness %>%
    ggplot(aes(scenario.f, f0mn, color = attitude.f)) + geom_point() +
    facet_wrap(~subject) +
  theme_bw() +
  xlab("Scenarios") +
  ylab("Frequency") +
  ggtitle("Subplot for Each Subject")
```

```
## Warning: Removed 12 rows containing missing values (geom_point).
```



There seem to be a lower baseline/intercept given that you're a male. Attitude doesn't seem to have an large effect on f0mn. So an idea could be to add Gender as a fixed effect and subject as a random intercept as there is also individual variance within the gender category.

Exercise 2 - comparison of models

1, (SFS)) Build four models and do some comparisons i. a single level model that models *f0mn* as dependent on *gender* ii. a two-level model that adds a second level on top of i. where unique intercepts are modelled for each *scenario* iii. a two-level model that only has *subject* as an intercept iv. a two-level model that models intercepts for both *scenario* and *subject* v. which of the models has the lowest residual standard deviation, also compare the Akaike Information Criterion `AIC` ? vi. which of the second-level effects explains the most variance?

```
#i
model1 <- lm(f0mn ~ gender.f, data = politeness)

#ii
model2 <- lmer(f0mn ~ gender.f + (1 | scenario.f), data = politeness, REML = FALSE)

#iii
model3 <- lmer(f0mn ~ gender.f + (1 | subject), data = politeness, REML = FALSE)

#iv
model4 <- lmer(f0mn ~ gender.f + (1 | scenario.f) + (1|subject), data = politeness, R
EML = FALSE)
```

Comparison of models by the Akaike Information Criterion:

```
# v
AIC(model1, model2, model3, model4)
```

	df <dbl>	AIC <dbl>
model1	3	2163.971
model2	4	2162.257
model3	4	2112.048
model4	5	2105.176

4 rows

Comparing the residual standard deviation of the models:

```
#v maybe gather everything in a table
sigma(model1)
```

```
## [1] 39.46268
```

```
sigma(model2)
```

```
## [1] 38.3546
```

```
sigma(model3)
```

```
## [1] 32.04227
```

```
sigma(model4)
```

```
## [1] 30.66355
```

Looking at both the standard deviation and the information criterion, we find that the model4 is the best performing model, since it has the smallest value both in AIC and RSD.

```
#vi the most variance explained by the effects (scenario or subject):
pacman::p_load(MuMIn)

r.squaredGLMM(model2)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##            R2m       R2c
## [1,] 0.6817304 0.6965456
```

```
r.squaredGLMM(model3)
```

```
##            R2m       R2c
## [1,] 0.6798832 0.7862932
```

```
r.squaredGLMM(model4)
```

```
##            R2m       R2c
## [1,] 0.6787423 0.8045921
```

model2 showed the best variance explained purely by fixed effects, 68,17%, with scenario as a random intercept. We can conclude in model3 that adding subject as random intercept rather than scenario explains more of the variance but also has more shared variance with our fixed effect gender. Model4 (f0mn ~ gender + (1|scenario) + (1|subject)) showed most explained variance with 80% of the variance being accounted for by both fixed and random effects.

2, (LWP)) Why is our single-level model bad? *(the single level model is bad, since it violates the most important assumption of independence)*

```
i. create a new data frame that has three variables, _subject_, _gender_ and _f0mn_,
where _f0mn_ is the average of all responses of each subject, i.e. averaging across _
attitude_ and_scenario_
```

```
#making a new dataframe with the selected variables:
politeness_sel <- politeness %>%
  filter(!is.na(f0mn)) %>% #making sure there is no NA in the new df
  select(f0mn,attitude,subject) %>%
  group_by(subject) %>%
  summarise(f0mn_mean = mean(f0mn))
```

```
## `summarise()` ungrouping output (override with `.groups` argument)
```

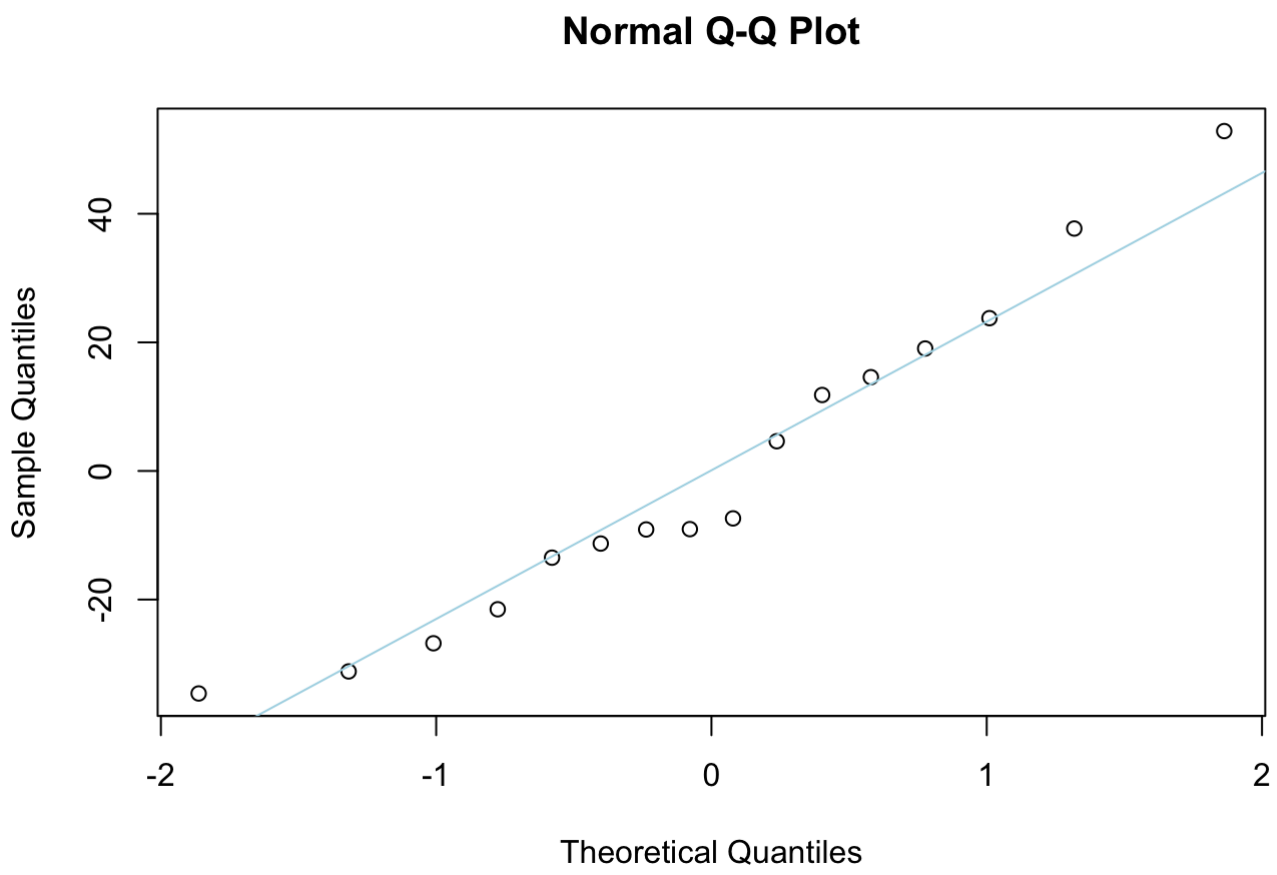
```
politeness_sel <- politeness_sel %>% #adding the gender to the dataframe
  mutate(gender = if_else(grepl("F", politeness_sel$subject, ignore.case = T),"F","M"
)) %>%
  mutate(gender = as.factor(gender))
```

ii. build a single-level model that models _f0mn_ as dependent on _gender_ using this new dataset

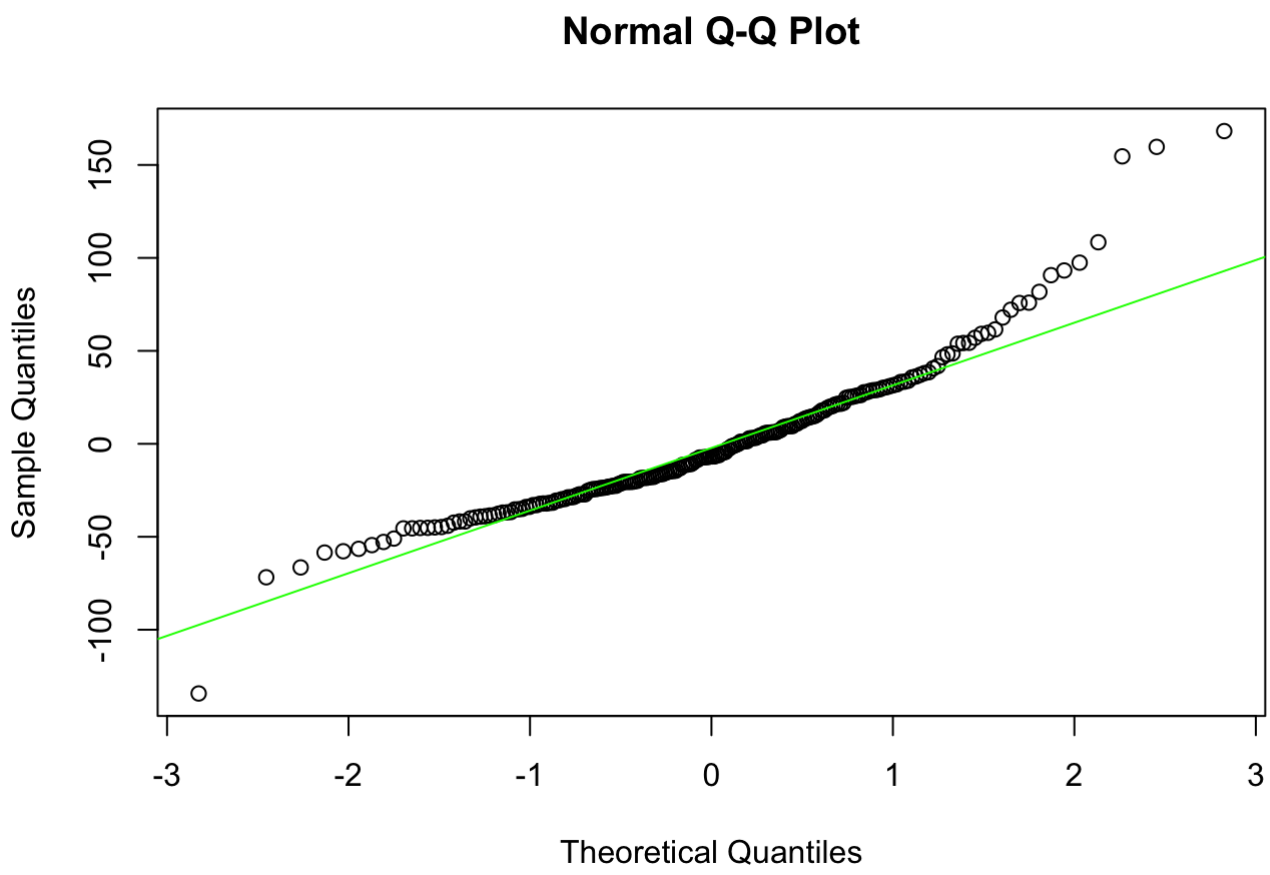
```
#builing single-level model
ms <- lm(f0mn_mean ~ gender, data = politeness_sel)
```

iii. make Quantile-Quantile plots, comparing theoretical quantiles to the sample quantiles) using `qqnorm` and `qqline` for the new single-level model and compare it to the old single-level model (from 1).i). Which model's residuals (ϵ) fulfil the assumptions of the General Linear Model better?)

```
#the new single model
qqnorm(resid(ms))
qqline(resid(ms), col = 'lightblue')
```



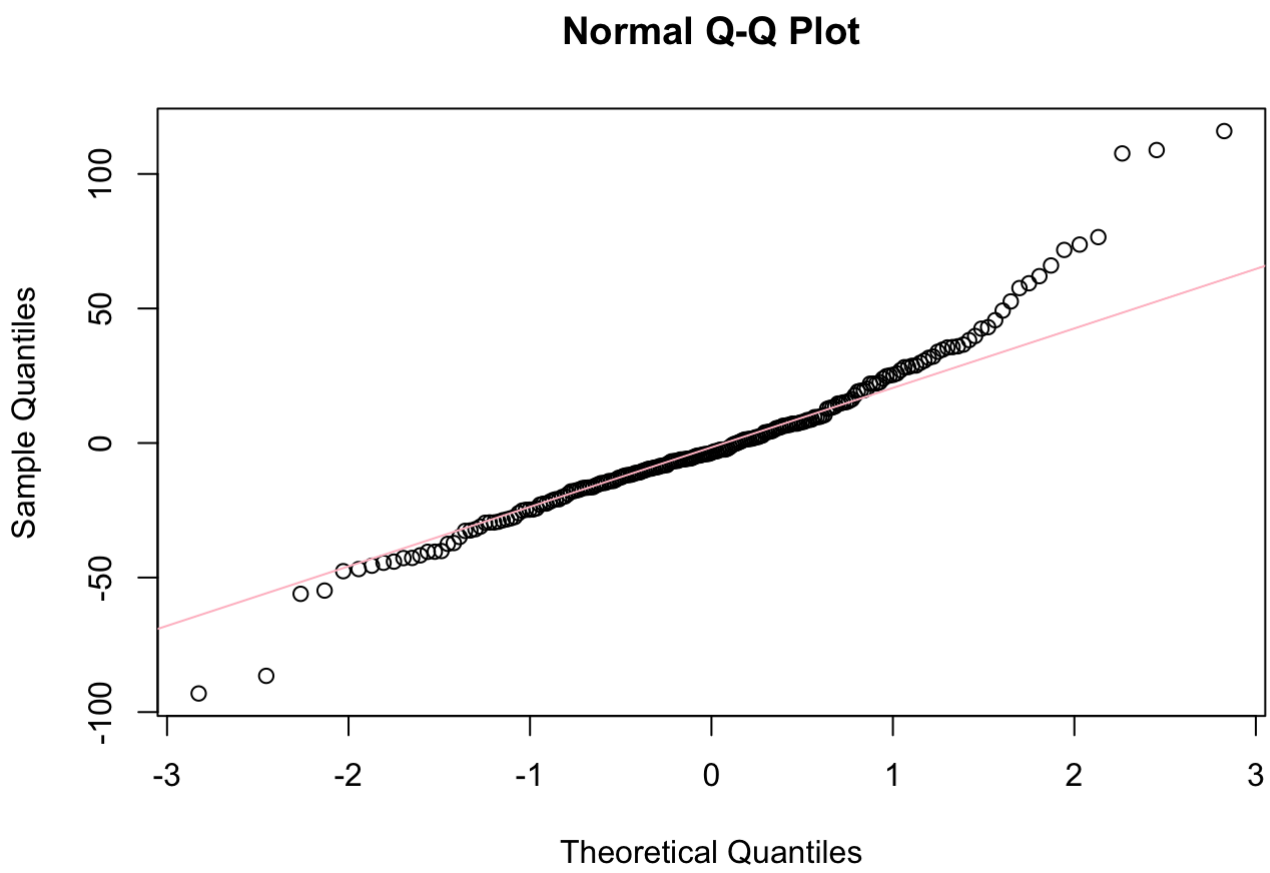

```
#The old single model
qqnorm(resid(model1))
qqline(resid(model1), col = 'green')
```



Looking at the data we how the ms model doesn't fit the line very well, however it does not seemed skewed. The model1 seems a bit skewed, and fits the line worse. This could properly have been fixed by trimming the data/remove outliers.

```
iv. Also make a quantile-quantile plot for the residuals of the  multilevel model wit
h two intercepts. Does it look alright?
```

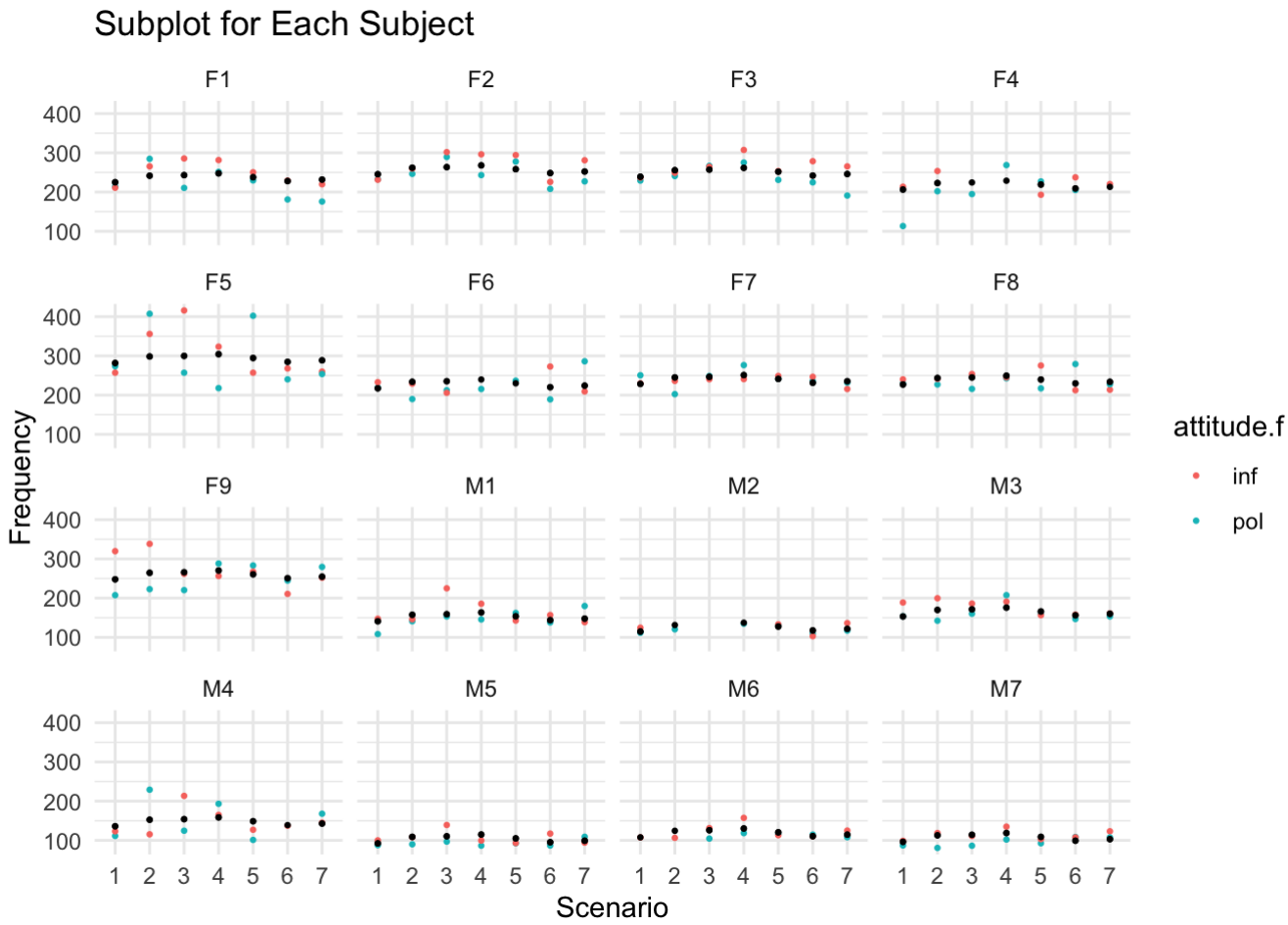
```
#The multilevel model (model 4)
qqnorm(resid(model4))
qqline(resid(model4), col = 'pink')
```



In a perfect world, this model would have made the datapoints fit the line better. This doesn't seem to be the case, and the residuals are still right skewed. They don't follow the normal distribution perfectly. However this is the least important of the assumptions, (normality of residuals).

3, (NAK)) Plotting the two-intercepts model i. Create a plot for each subject, (similar to part 3 in Exercise 1), this time also indicating the fitted value for each of the subjects for each for the scenarios (hint use `fixef` to get the "grand effects" for each gender and `ranef` to get the subject- and scenario-specific effects)

```
fitted <- fitted(model4) #making the fitted values

politeness_una <- politeness %>%
  filter(!is.na(f0mn)) %>%  #making sure we have no NA's
  mutate(fitted) #adding the fitted values to the dataset

politeness_una %>%
  ggplot(aes(scenario.f, f0mn, color = attitude.f))+
  geom_point(size = 0.5)+
  geom_point(aes(y = fitted), colour = 'black', size = 0.5)+
  facet_wrap(~subject) +
  theme_minimal()+
  xlab("Scenario")+
  ylab('Frequency') +
  ggtitle("Subplot for Each Subject")
```



Exercise 3 - now with attitude

1, (SEK)) Carry on with the model with the two unique intercepts fitted (*scenario* and *subject*). i. now build a model that has *attitude* as a main effect besides *gender*

```
# the model to carry on with: model4 <- lmer(f0mn ~ gender.f + (1 | scenario.f) + (1|
subject), data = politeness)
#the new model with both gender and attitude:
model5 <- lmer(f0mn ~ gender.f + attitude.f + (1|scenario.f)+(1|subject), data = poli
teness, REML = FALSE)
```

```
ii. make a separate model that besides the main effects of _attitude_ and _gender_ al
so include their interaction
```

```
model6 <- lmer(f0mn ~ gender.f*attitude.f + (1|scenario.f)+(1|subject), data = polite
ness, REML = FALSE)
summary(model6)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f * attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2096.0   2119.5  -1041.0   2082.0      205
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8460 -0.5893 -0.0685  0.3946  3.9518
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.09   22.674
##  scenario.f (Intercept)  99.08    9.954
##  Residual               876.46   29.605
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##                         Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)              255.632      9.289   23.556  27.521  < 2e-16 ***
## gender.fM               -118.251     12.841   19.922  -9.209 1.28e-08 ***
## attitude.fpol            -17.198      5.395  190.331  -3.188  0.00168 **
## gender.fM:attitude.fpol    5.563      8.241  190.388   0.675  0.50049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M atttd.
## gender.fM   -0.605
## attitud.fpl -0.299  0.216
## gndr.fM:tt.  0.195 -0.323 -0.654
```

```
iii. describe what the interaction term in the model says about Korean men's pitch wh
en they are polite relative to Korean women's pitch when they are polite (you don't h
ave to judge whether it is interesting)
```

Understanding the output of the model:

When males are asked to be polite, their pitch will according to this be higher. The intercepts is for the female, when uttering the statement informal, where they here have the average pitch of 255 hz. GenderfM is then when we go from female to male on the x ax, we see how the average pitch decrease with 118 hz. attitudef.pol, when we go from informal to polite does the average (of both females and males) pitch decrease with 17 hz. This is why need the interaction, so we can consider more than just the average genderM:attitudepol: this is the interaction between gender and attitude, and it indicates that it decreases 5,5hz less for men than women. This means that the change in pitch for men are on -17.192+5.54 = -11.652, whereas the womens changes with -17.192 hz. Summarizingly, both men

and women decrease their pitch when going from informal to polite, but the male pitch does not decrease as much the women. (for the reader: the f just means that it is factors, a bit confusing considering the females - but this is not the case!)

Reporting the model: *The model f0mn ~ attitude:gender + (1|subject)+ (1|scenario)has an R^2c 0.81 both attitude and gender showed a significant effect on f0mn (β_1 (attitude_pol) = -17.2, SE = 5.4, p>0.05) and (β_2 (genderM) = -119, SE = 12.8, p>0.05). Being polite and male lowers your frequency. Being both Male and Polite has an interaction effect of (β_3 = 5.5, SE = 8.24, p<0.05). Hereby concluding that there is a small positive insignificant interaction effect of being male and polite. The SE being proportional large compared to the effect size makes it very difficult to say anything meaningful.*

2, (SFS)) Compare the three models (1. gender as a main effect; 2. gender and attitude as main effects; 3. gender and attitude as main effects and the interaction between them. For all three models model unique intercepts for *subject* and *scenario*) using residual variance, residual standard deviation and AIC.

```
#model4: gender as main effect
summary(model4)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2105.2   2122.0  -1047.6   2095.2      207
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.0357 -0.5384 -0.1177  0.4346  3.7808
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 516.19   22.720
##  scenario.f (Intercept)  89.36    9.453
##  Residual               940.25   30.664
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  246.778      8.829   19.248  27.952  < 2e-16 ***
## gender.fM   -115.186     12.223   16.011  -9.424 6.19e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##           (Intr)
## gender.fM -0.604
```

```
#model5: gender and attitudes as main effects
summary(model5)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f + attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2094.5   2114.6  -1041.2   2082.5      206
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8791 -0.5968 -0.0569  0.4260  3.9068
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.92   22.692
##  scenario.f (Intercept)  99.22    9.961
##  Residual               878.39   29.638
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##               Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)    254.408      9.117   21.800  27.904  < 2e-16 ***
## gender.fM     -115.447     12.161   16.000  -9.494 5.63e-08 ***
## attitude.fpol  -14.817      4.086  190.559  -3.626 0.000369 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M
## gender.fM   -0.583
## attitud.fpl -0.231  0.006
```

```
#model6: gender and attitude as main effects and with an interaction between them
summary(model6)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender.f * attitude.f + (1 | scenario.f) + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2096.0   2119.5  -1041.0   2082.0      205
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.8460 -0.5893 -0.0685  0.3946  3.9518
##
## Random effects:
##  Groups     Name        Variance Std.Dev.
##  subject    (Intercept) 514.09   22.674
##  scenario.f (Intercept)  99.08    9.954
##  Residual               876.46   29.605
## Number of obs: 212, groups:  subject, 16; scenario.f, 7
##
## Fixed effects:
##                         Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)              255.632      9.289   23.556  27.521  < 2e-16 ***
## gender.fM               -118.251     12.841   19.922  -9.209 1.28e-08 ***
## attitude.fpol            -17.198      5.395  190.331  -3.188  0.00168 **
## gender.fM:attitude.fpol    5.563      8.241  190.388   0.675  0.50049
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) gndr.M atttd.
## gender.fM   -0.605
## attitud.fpl -0.299  0.216
## gndr.fM:tt.  0.195 -0.323 -0.654
```

```
#comparison by AIC:
AIC(model4, model5, model6)
```

	df	AIC
	<dbl>	<dbl>
model4	5	2105.176
model5	6	2094.489
model6	7	2096.034

3 rows

```
#comparing by standard deviation of residuals
sigma(model4)
```

```
## [1] 30.66355
```

```
sigma(model5)
```

```
## [1] 29.63771
```

```
sigma(model6)
```

```
## [1] 29.60505
```

```
#comparing by the residual variance:
sum(residuals(model4)^2)
```

```
## [1] 181913
```

```
sum(residuals(model5)^2)
```

```
## [1] 169681.1
```

```
sum(residuals(model6)^2)
```

```
## [1] 169305.6
```

Considering the output of the comparisons, we suggest model 5: it is the simpler model and adding the interaction effect (model 6) makes almost no explanatory power, while being more complex.

3, (LWP, NAK, SEK, SFS)) Choose the model that you think describe the data the best - and write a short report on the main findings based on this model. At least include the following: i. describe what the dataset consists of
The dataset used in this model consists of subject id, binary gender indication (F or M), scenario index (from 1 to 7 depending on what the scenario was), a variable indicating whether the text should be spoken in an formal/polite or informal tone, and a variable called f0mn basically stating the average frequency of the utterance in Hz. Besides these the data also consisted of total duration of utterances in seconds and count of hissing sounds but these are not relevant for the optimal model.

ii. what can you conclude about the effect of gender and attitude on pitch (if anything)?
f0mn was found to be significantly modulated by gender.
$\beta_2 = -115, SE = 12.16, p < 0.05$ *Attitude also showed a significant modulating of f0mn* $\beta_1 = -14.8, SE = 4, p < 0.05$

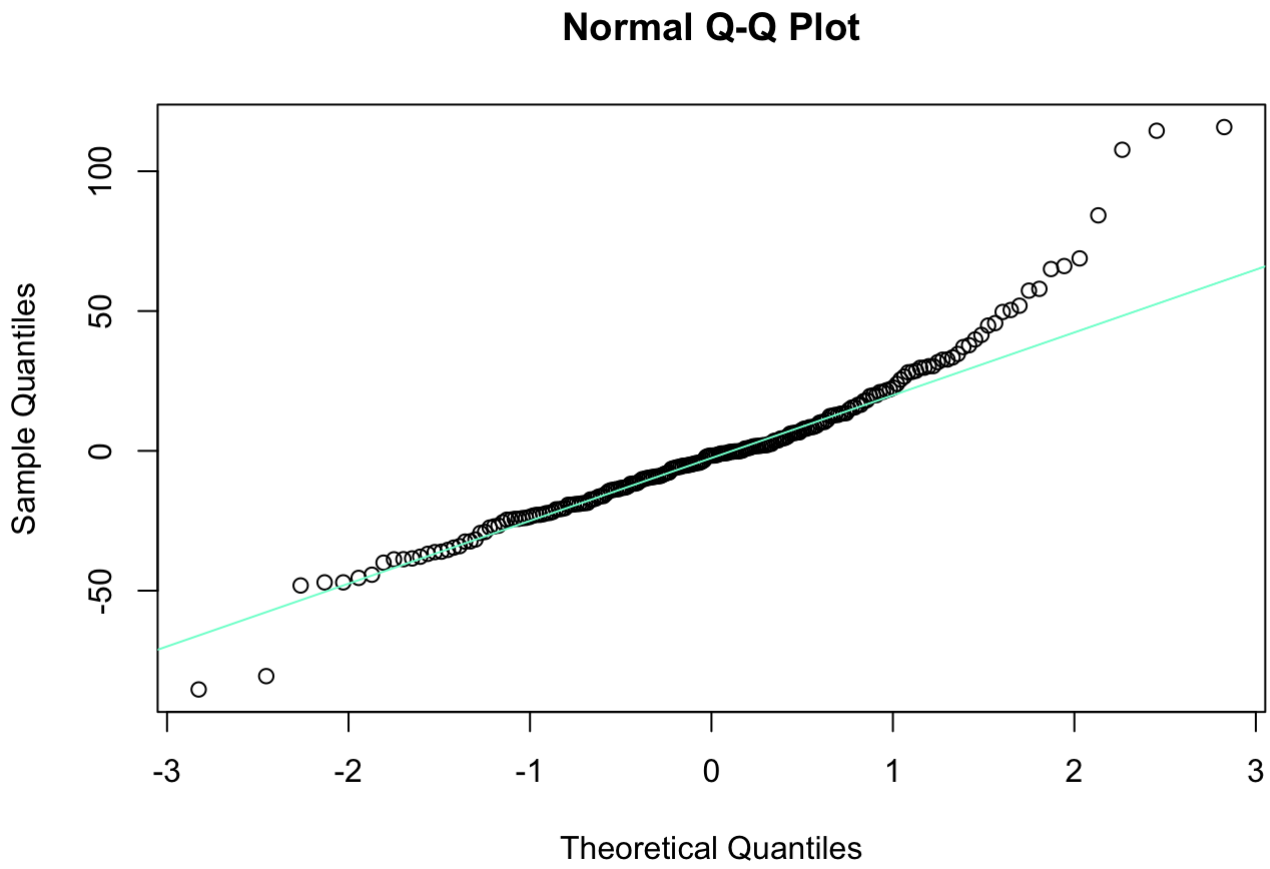
iii. motivate why you would include separate intercepts for subjects and scenarios (if you think they should be included)
Subjects: *these are only a sample of the total population. Because subject does not exhaust the population of interest (e.g. the whole Korean population) it should be modeled as a random effect. Also, each subject will express random variation caused by individual baselines and individual effects of formal vs. informal situation.*

Scenario: *Again, these scenarios does not exhaust the number of formal or informal scenarios that exist. It should be modeled as a random effect since we have no expectation of how the individual scenario will affect the pitch compared to the other scenarios. There are no preconceptions about any systematic differences between the scenarios, making them have idiosyncratic and random effects on pitch.*

iv. describe the variance components of the second level (if any)
Both fixed and random effects accounted for roughly 82% of the variance in the f0mn variable with random effects proportion being 12.7%. Visual inspection shows that both the qqplot and histogram violates the assumption of a mixed effect linear model. The more robust generalized mixed effect model with a link function would be preferred. But as it was not the task such model was not constructed.

v. include a Quantile-Quantile plot of your chosen model

```
qqnorm(resid(model5))
qqline(resid(model5), col = 'aquamarine')
```



We used R (R Core Team, 2019) and lmerTest (Kuznetsova, Brockhoff and Christensen, 2017) to perform a linear mixed effects analysis of the relationship between f0mn, gender and attitude. As random effects, we had intercepts for subjects, and scenario.

Portfolio 2.1, Methods 3, 2021, autumn semester

Sigurd Fyhn Sørensen

04/10/2021

Exercises and objectives

The objectives of the exercises of this assignment are:
1) Download and organise the data and model and plot staircase responses based on fits of logistic functions
2) Fit multilevel models for response times
3) Fit multilevel models for count data

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This assignment will be part of your final portfolio

Exercise 1

Go to https://osf.io/ecxsj/files/ (https://osf.io/ecxsj/files/) and download the files associated with Experiment 2 (there should be 29).
The data is associated with Experiment 2 of the article at the following DOI https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007)

1. Put the data from all subjects into a single data frame
2. Describe the data and construct extra variables from the existing variables
    i. add a variable to the data frame and call it *correct* (have it be a *logical* variable). Assign a 1 to each row where the subject indicated the correct answer and a 0 to each row where the subject indicated the incorrect answer (**Hint:** the variable *obj.resp* indicates whether the subject answered “even”, *e* or “odd”, *o*, and the variable *target_type* indicates what was actually presented.

```
df_exp <- df_exp %>%
  mutate(right_answer = if_else(target.type == "odd" & obj.resp == "o" | target.type
 == "even" & obj.resp == "e",1,0)) %>%
  mutate(right_answer = as.numeric(right_answer))

df_exp <- df_exp %>%
  mutate(right_answer = as.factor(right_answer)) %>%
  mutate(subject = as.factor(subject)) %>%
  mutate(task = as.factor(task)) %>%
  mutate(target.type = as.factor (target.type))

sum(is.na(df_exp) == TRUE)
```

```
## [1] 0
```

```
ii. describe what the following variables in the data frame contain, _trial.type_, _p
as_, _trial_, _target.contrast_, _cue_, _task_, _target_type_, _rt.subj_, _rt.obj_, _
obj.resp_, _subject_ and _correct_. (That means you can ignore the rest of the variab
les in your description). For each of them, indicate and argue for what `class` they
should be classified into, e.g. _factor_, _numeric_ etc.
```

** trial.type **: Contains two levels - staircase and experiment. The staircase trials are made as a way of adjusting the target.contrast. A 75% accuracy was aimed for. The experiment trials are what the actual experiment consists of (factor)** pas **: Perceptual awareness scale. Subjective rating of the experience of awareness of the stimulus.Ranging from 1 (no experience) to 4 (clear experience) (numeric)** trial **: Trial number, resets when trial.type changes (numeric)** target.contrast **: The contrast of the target stimulus relative to the background (numeric)** cue **: An indicator of which set of cue stimuli was used ranging from 0 to 35 (factor)** task **: Indication of if the cue was shown as singles, pairs or quadruplets (factor)** target_type **: Showing if the target variable is odd or even (factor)** rt.subj **: Reaction time on subjective rating (numeric)** rt.obj **: Task reaction time in milliseconds (numeric)** obj.resp **: The response given by the participant, e.g. even or odd (factor)** subject **: Participant ID (factor)** correct **: Indicating whether participants answered correctly (factor)

```
iii. for the staircasing part __only__, create a plot for each subject where you plot
the estimated function (on the _target.contrast_ range from 0-1) based on the fitted
values of a model (use `glm`) that models _correct_ as dependent on _target.contrast
_. These plots will be our _no-pooling_ model. Comment on the fits - do we have enoug
h data to plot the logistic functions?
```

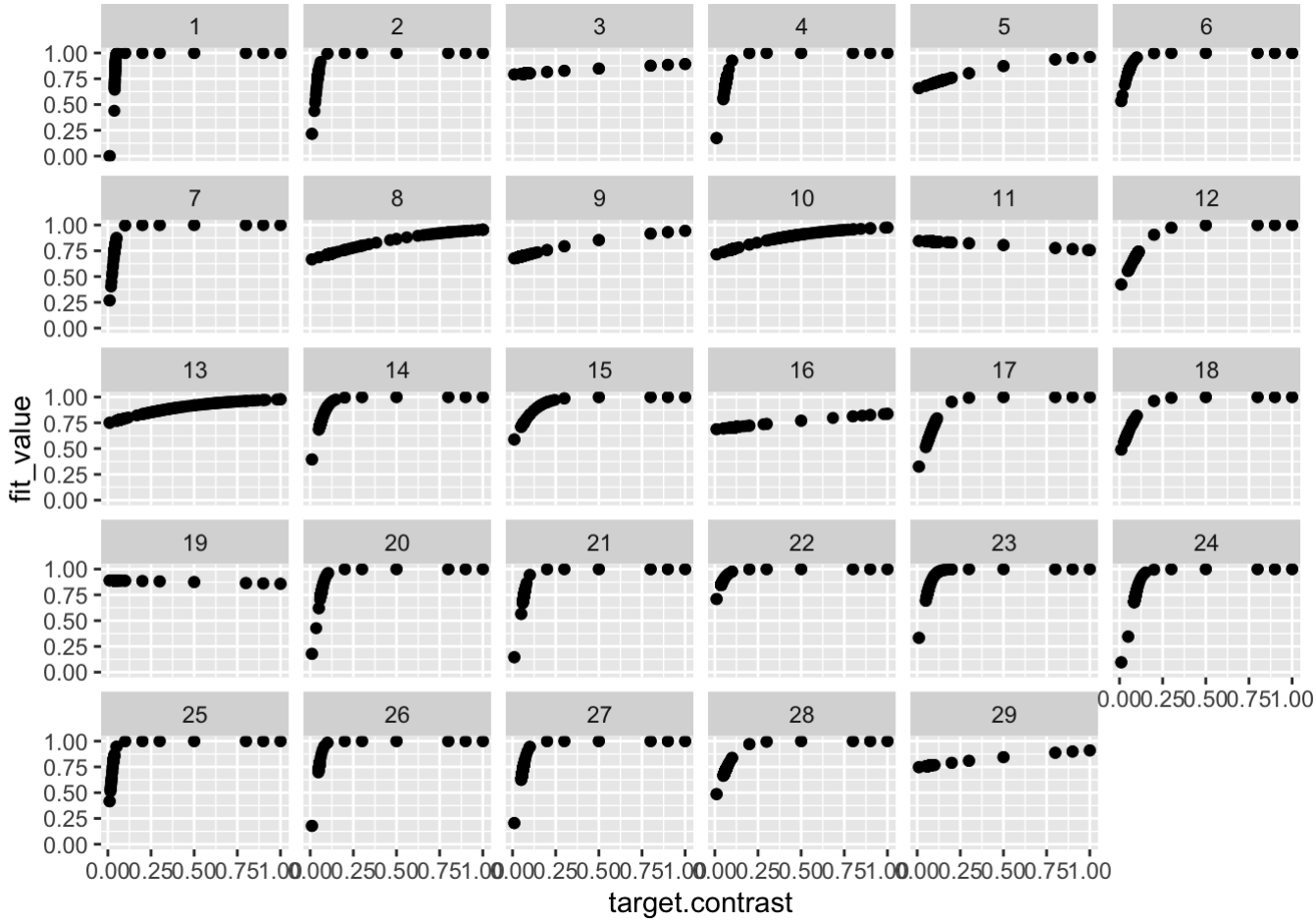
```
m1 <- glm(right_answer ~ target.contrast*subject, data = filter(df_exp,trial.type ==
"staircase" ), family = binomial(link = "logit"))
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
invlogit(coef(m1))
```

```
##               (Intercept)          target.contrast                  subject2
##              6.403916e-07             1.000000e+00             9.999952e-01
##                  subject3                  subject4                  subject5
##              9.999998e-01             9.999952e-01             9.999997e-01
##                  subject6                  subject7                  subject8
##              9.999992e-01             9.999964e-01             9.999997e-01
##                  subject9                 subject10                 subject11
##              9.999997e-01             9.999997e-01             9.999999e-01
##                 subject12                 subject13                 subject14
##              9.999990e-01             9.999998e-01             9.999987e-01
##                 subject15                 subject16                 subject17
##              9.999995e-01             9.999997e-01             9.999984e-01
##                 subject18                 subject19                 subject20
##              9.999992e-01             9.999999e-01             9.999951e-01
##                 subject21                 subject22                 subject23
##              9.999937e-01             9.999996e-01             9.999981e-01
##                 subject24                 subject25                 subject26
##              9.999910e-01             9.999980e-01             9.999944e-01
##                 subject27                 subject28                 subject29
##              9.999961e-01             9.999992e-01             9.999998e-01
##  target.contrast:subject2  target.contrast:subject3  target.contrast:subject4
##             1.145135e-132             6.241067e-164             1.346004e-144
##  target.contrast:subject5  target.contrast:subject6  target.contrast:subject7
##             3.786360e-163             4.602997e-150             2.372017e-132
##  target.contrast:subject8  target.contrast:subject9 target.contrast:subject10
##             3.012820e-163             2.293329e-163             4.537618e-163
## target.contrast:subject11 target.contrast:subject12 target.contrast:subject13
##             1.595643e-164             2.099260e-158             4.266843e-163
## target.contrast:subject14 target.contrast:subject15 target.contrast:subject16
##             3.497137e-151             2.442752e-158             6.674961e-164
## target.contrast:subject17 target.contrast:subject18 target.contrast:subject19
##             1.025591e-155             8.563314e-157             2.105787e-164
## target.contrast:subject20 target.contrast:subject21 target.contrast:subject22
##             2.541380e-142             3.584018e-142             9.595477e-151
## target.contrast:subject23 target.contrast:subject24 target.contrast:subject25
##             6.700431e-148             7.672048e-147             1.685355e-129
## target.contrast:subject26 target.contrast:subject27 target.contrast:subject28
##             7.626064e-137             4.700550e-144             4.378820e-156
## target.contrast:subject29
##             9.797932e-164
```

```
df_exp_stair <- df_exp %>%
  filter(trial.type == "staircase") %>%
  mutate(fit_value = fitted.values(m1))

df_exp_stair %>%
  ggplot(aes(x = target.contrast, y = fit_value)) + geom_point() + facet_wrap(~subjec
t)
```



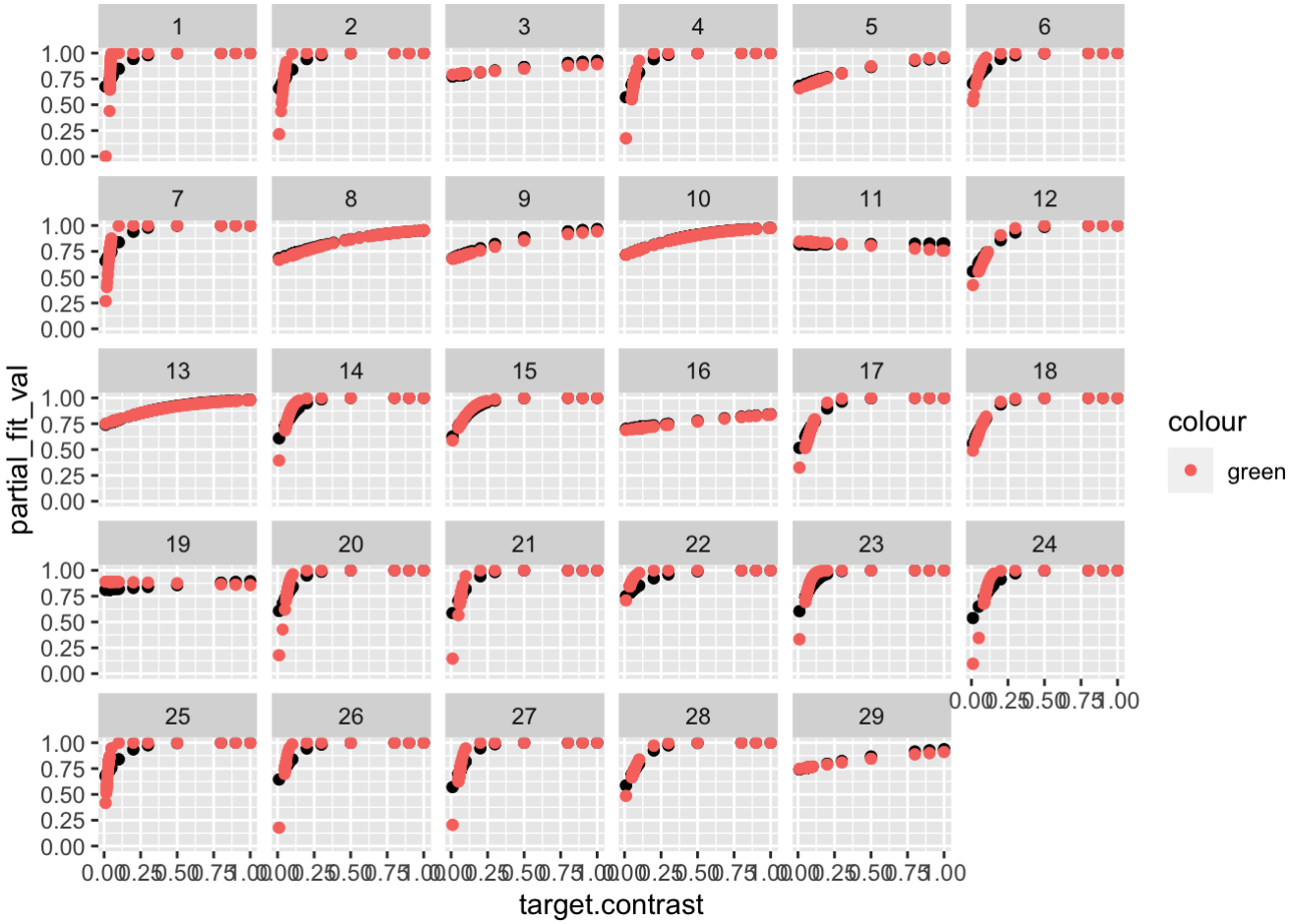
As can be seen by the above plot we do not have something that resembles a sigmoid function.The “shape” is almost there but with a lot of holes. Having a data point for each x with a step of 0.01 should result in a perfect sigmoid fit.

```
iv. on top of those plots, add the estimated functions (on the _target.contrast_ rang
e from 0-1) for each subject based on partial pooling model (use `glmer` from the pac
kage `lme4`) where unique intercepts and slopes for _target.contrast_ are modelled fo
r each _subject_
```

```
m2 <- glmer(right_answer ~ target.contrast + (1+target.contrast|subject), data = df_e
xp_stair, family = binomial(link = "logit"))
summary(m2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: right_answer ~ target.contrast + (1 + target.contrast | subject)
##    Data: df_exp_stair
##
##      AIC      BIC   logLik deviance df.resid
##   5988.5   6021.6  -2989.2   5978.5     5598
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -5.7671  0.0068  0.5532  0.5915  0.9264
##
## Random effects:
##  Groups  Name            Variance Std.Dev. Corr
##  subject (Intercept)      0.2717  0.5213
##          target.contrast 42.7575  6.5389   -0.84
## Number of obs: 5603, groups:  subject, 29
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)       0.5619     0.1595   3.523 0.000427 ***
## target.contrast   8.7132     2.3879   3.649 0.000263 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr)
## trgt.cntrst -0.909
```

```
df_exp_stair <- df_exp_stair %>%
  mutate(partial_fit_val = fitted.values(m2))

df_exp_stair %>%
  ggplot(aes(x = target.contrast, y = partial_fit_val)) + geom_point() + geom_point(a
es(x = target.contrast, y = fit_value, col = "green")) + facet_wrap(~subject)
```




```
v. in your own words, describe how the partial pooling model allows for a better fit
for each subject
```

Exercise 2

Now we **only** look at the *experiment* trials (*trial.type*)

```
df_exp_exp <- df_exp %>%
  filter(trial.type == "experiment")
```

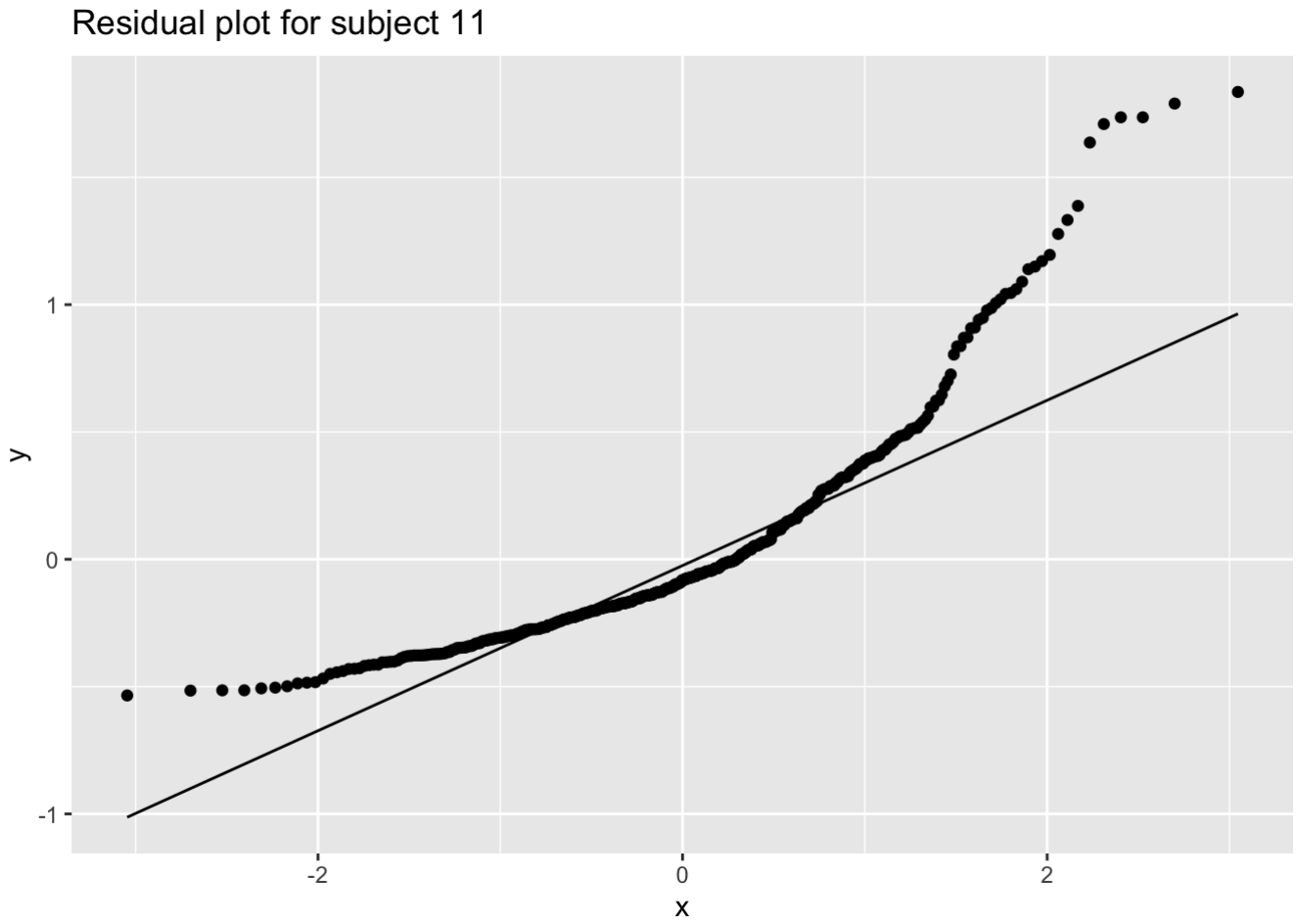
1. Pick four subjects and plot their Quantile-Quantile (Q-Q) plots for the residuals of their objective response times (*rt.obj*) based on a model where only intercept is modelled.

```
m3 <- lmer(rt.obj ~ (1|subject), data = df_exp_exp)

df_exp_exp <- df_exp_exp %>%
  mutate(rt.obj_fit_val = fitted(m3)) %>%
  mutate(rt.obj_resid = resid(m3))
```

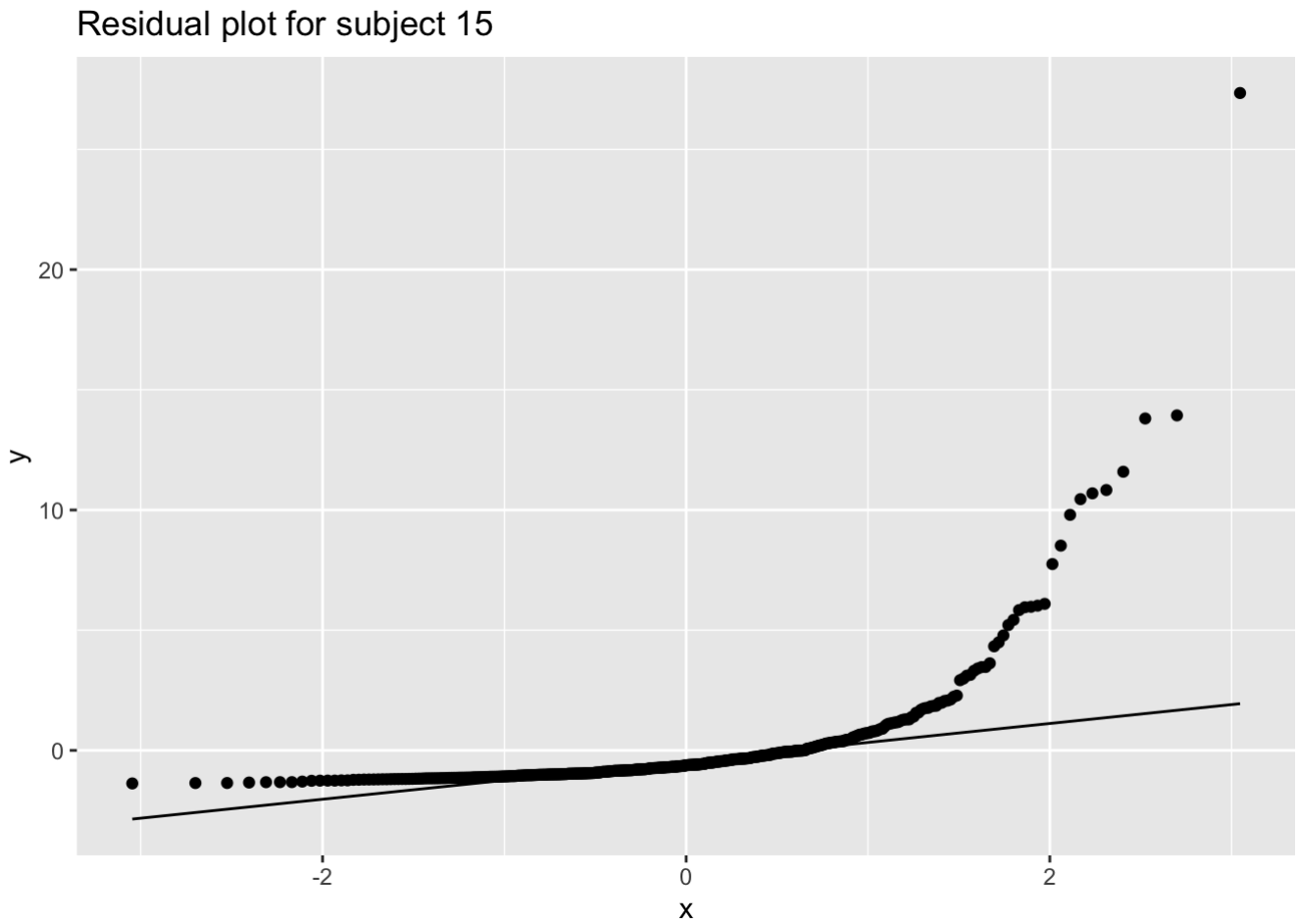
```
#For-loop not printing out plot.. Don't know why as I've done similair loops before.
for (i in sample(1:length(unique(df_exp_exp$subject)),4)){
  df_exp_exp %>%
    filter(subject == as.character(i)) %>%
    ggplot(aes(sample = rt.obj_resid)) + stat_qq()+stat_qq_line()
}
```

```
#Boring tedious way
subjects <- sample(1:length(unique(df_exp_exp$subject)),4)
```

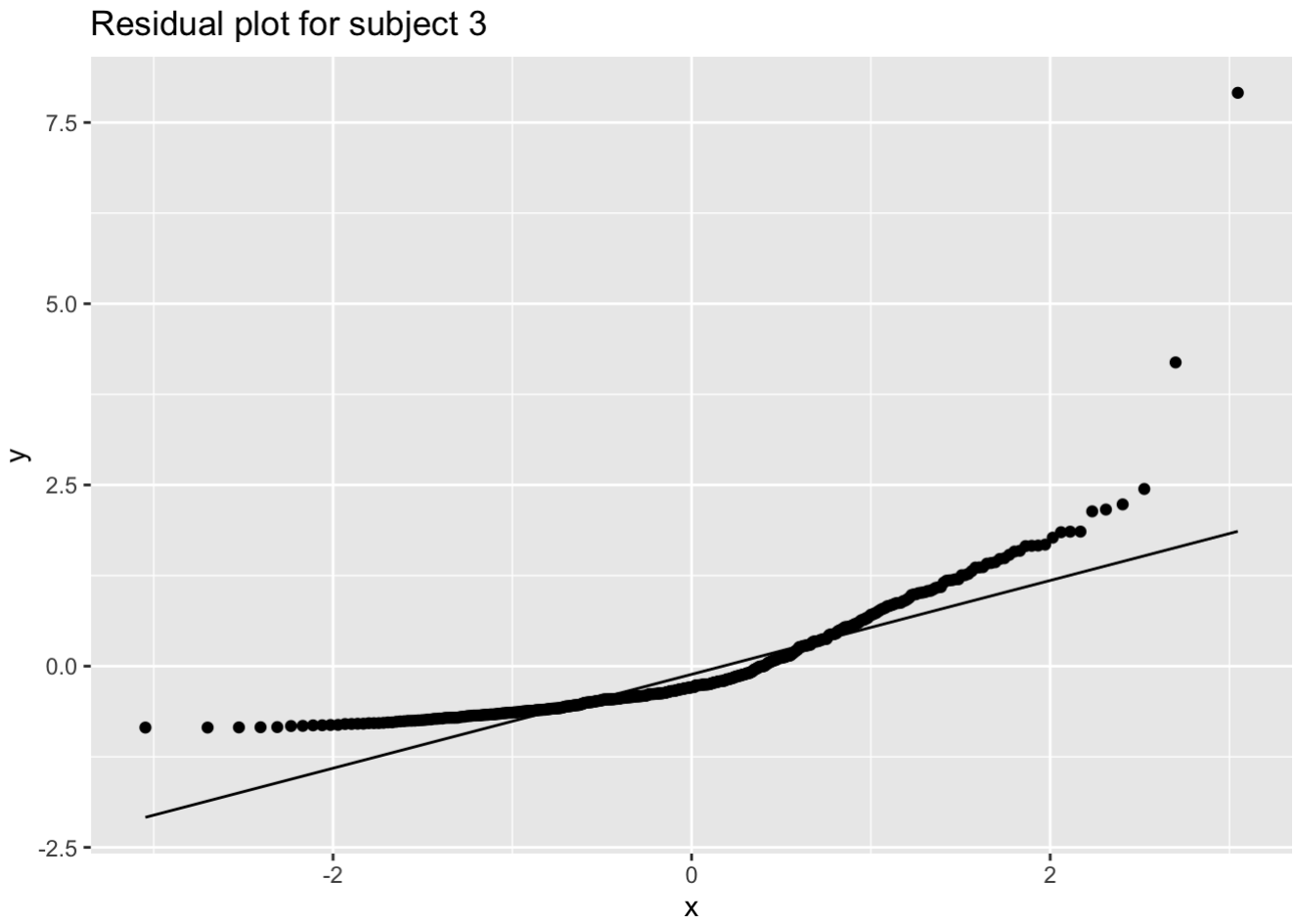
```
df_exp_exp %>%
  filter(subject == as.character(subjects[1])) %>%
  ggplot(aes(sample = rt.obj_resid)) + stat_qq() + stat_qq_line() + labs(title = past
e("Residual plot for subject", subjects[1],sep = " "))
```



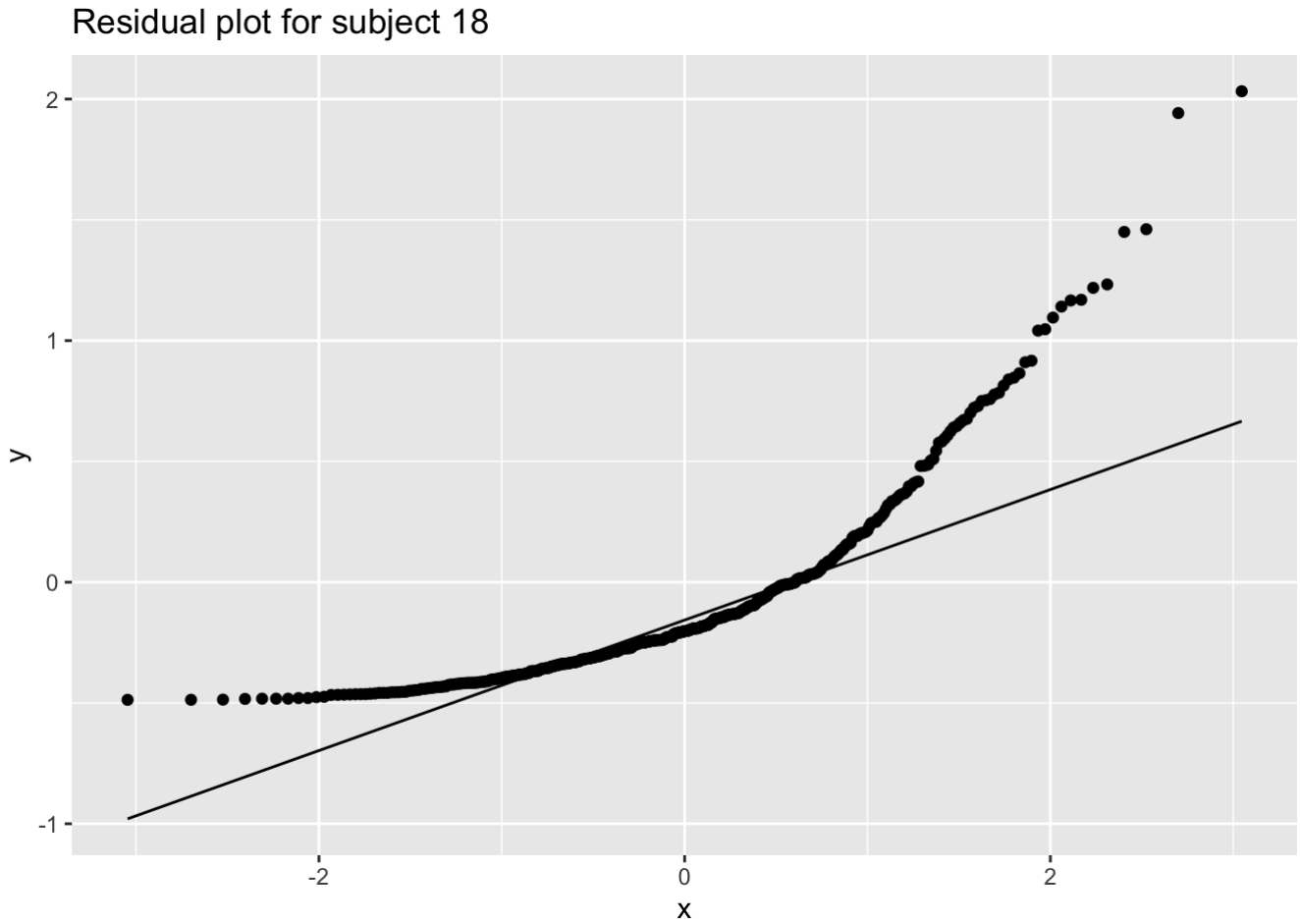

```
df_exp_exp %>%
  filter(subject == as.character(subjects[2])) %>%
  ggplot(aes(sample = rt.obj_resid)) + stat_qq() + stat_qq_line() + labs(title = past
e("Residual plot for subject", subjects[2],sep = " "))
```




```
df_exp_exp %>%
  filter(subject == as.character(subjects[3])) %>%
  ggplot(aes(sample = rt.obj_resid)) + stat_qq() + stat_qq_line() + labs(title = past
e("Residual plot for subject", subjects[3],sep = " "))
```




```
df_exp_exp %>%
  filter(subject == as.character(subjects[4])) %>%
  ggplot(aes(sample = rt.obj_resid)) + stat_qq() + stat_qq_line() + labs(title = past
e("Residual plot for subject", subjects[4],sep = " "))
```



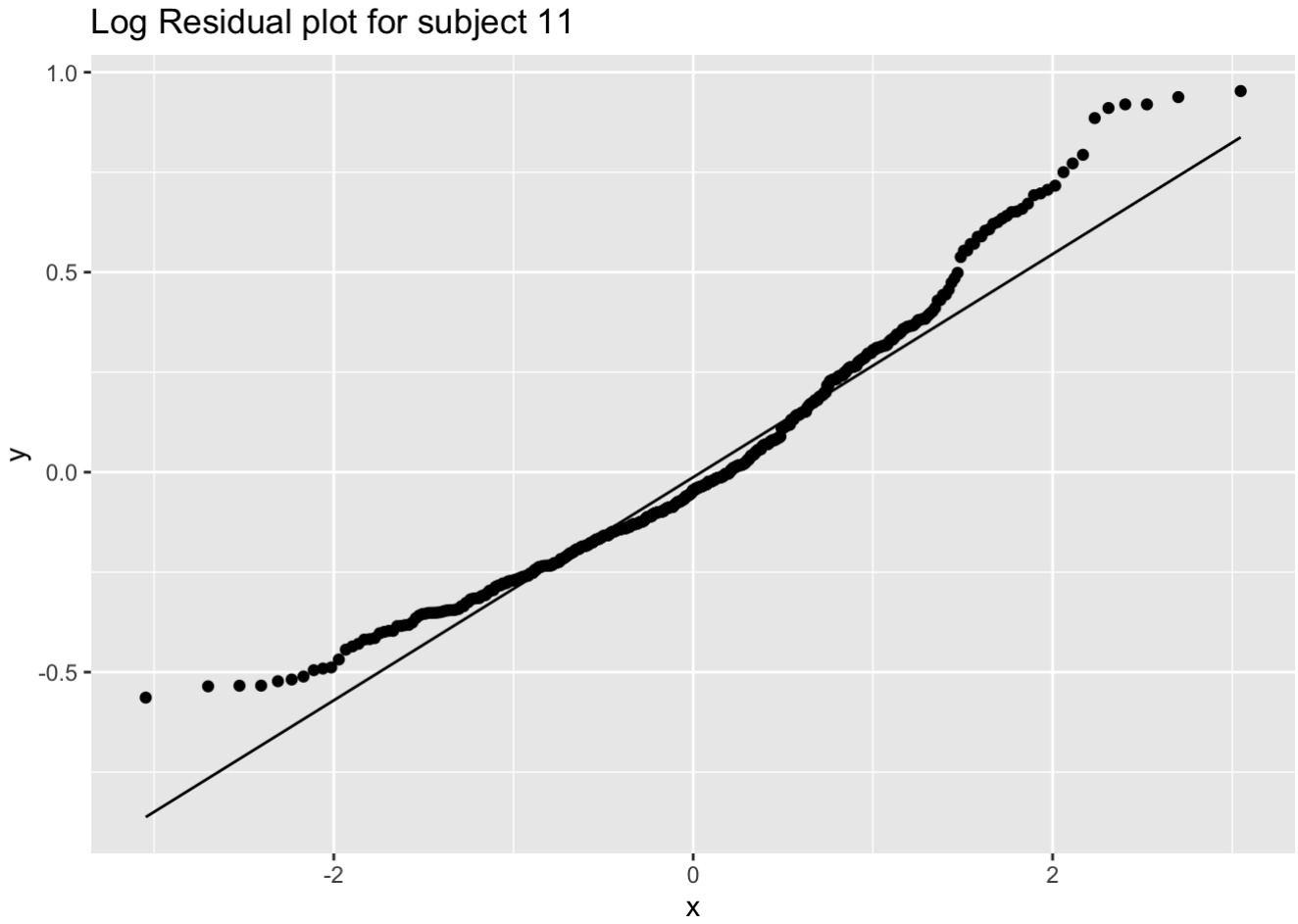

```
i. comment on these
```

The residuals does not look normally distributed some worse than others. As this is an assumption of the linear mixed-effect models measures to counteract is required.

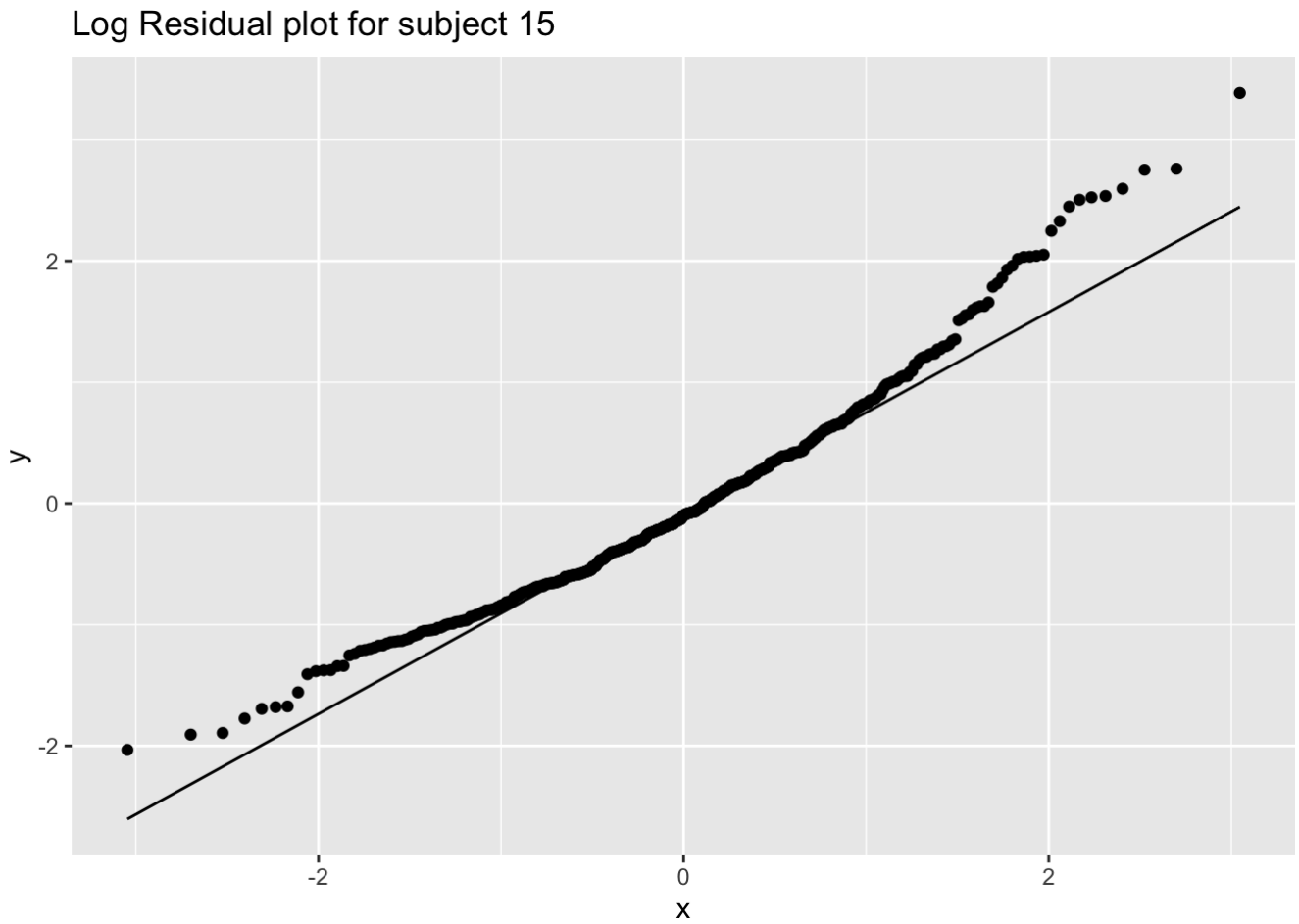
```
ii. does a log-transformation of the response time data improve the Q-Q-plots?
```

```
m4 <- lmer(log(rt.obj) ~ (1|subject), data = df_exp_exp)

df_exp_exp <- df_exp_exp %>%
  mutate(rt.obj_fit_val_log = fitted(m4)) %>%
  mutate(rt.obj_resid_log = resid(m4))
```

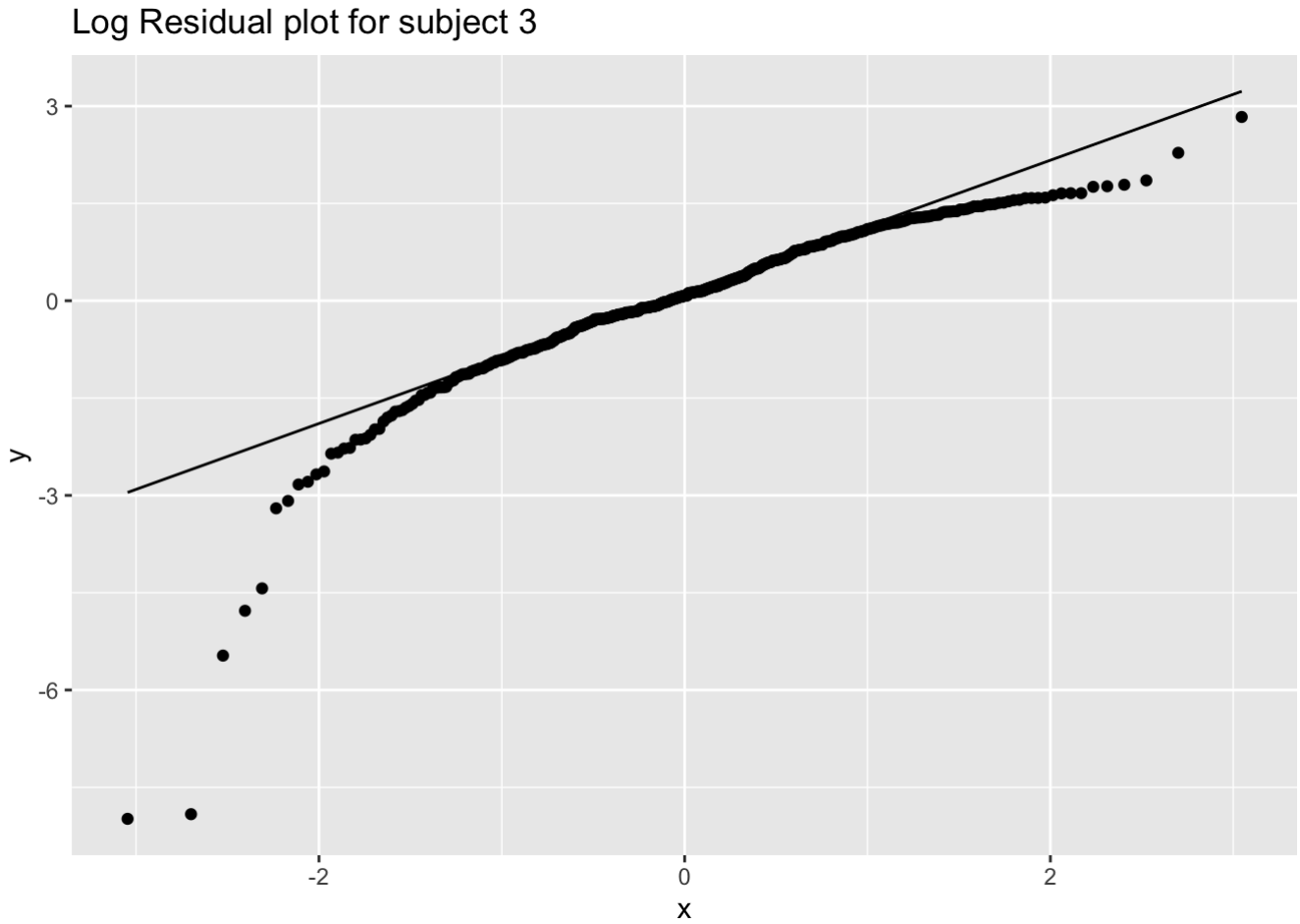
```
#Boring tedious way with log
df_exp_exp %>%
  filter(subject == as.character(subjects[1])) %>%
  ggplot(aes(sample = rt.obj_resid_log)) + stat_qq() + stat_qq_line() + labs(title =
 paste("Log Residual plot for subject", subjects[1],sep = " "))
```



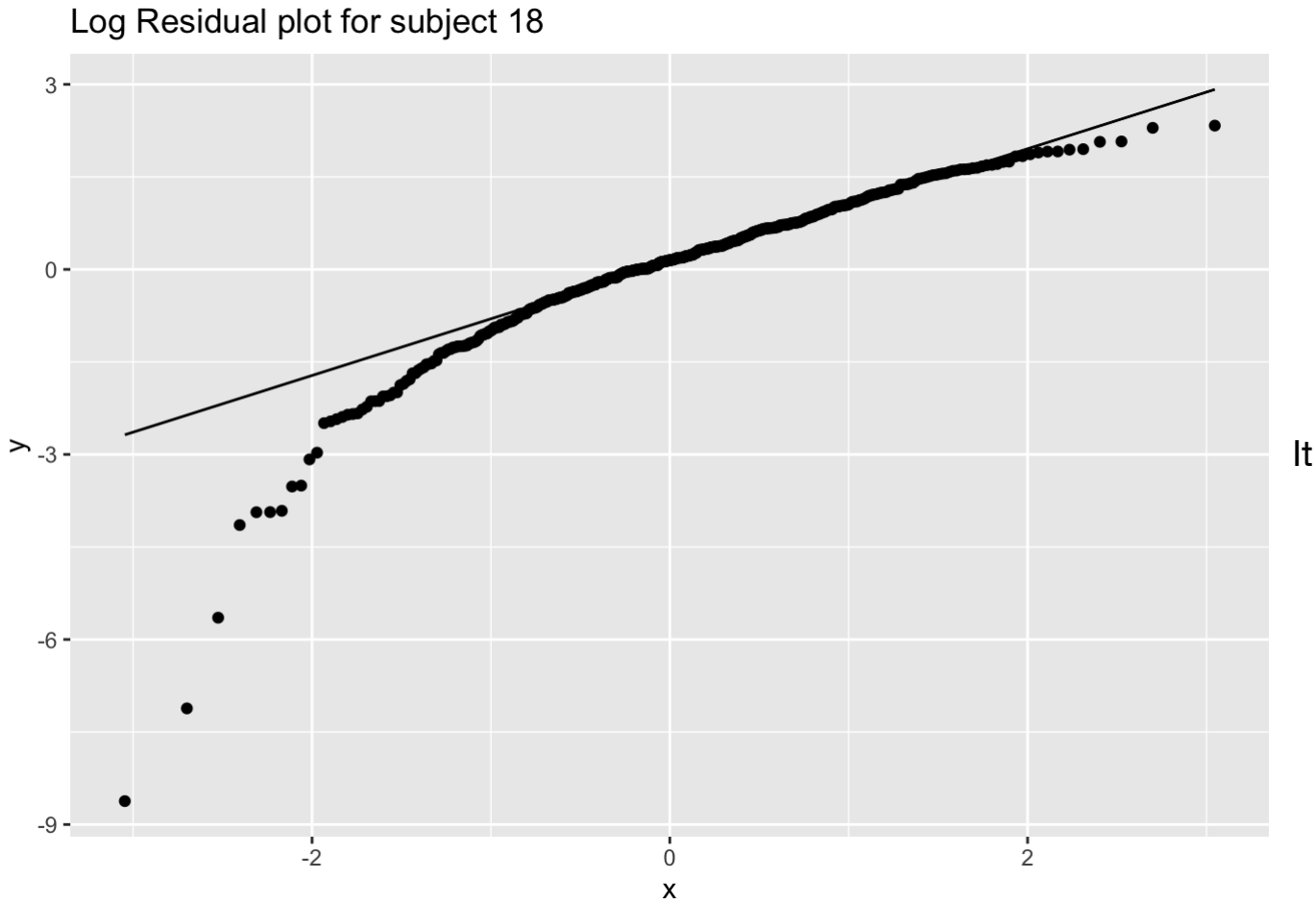

```
df_exp_exp %>%
  filter(subject == as.character(subjects[2])) %>%
  ggplot(aes(sample = rt.obj_resid_log)) + stat_qq() + stat_qq_line() + labs(title =
 paste("Log Residual plot for subject", subjects[2],sep = " "))
```




```
df_exp_exp %>%
  filter(subject == as.character(subjects[3])) %>%
  ggplot(aes(sample = rt.obj_resid_log)) + stat_qq() + stat_qq_line() + labs(title =
 paste("Log Residual plot for subject", subjects[3],sep = " "))
```




```
df_exp_exp %>%
  filter(subject == as.character(subjects[4])) %>%
  ggplot(aes(sample = rt.obj_resid_log)) + stat_qq() + stat_qq_line() + labs(title =
 paste("Log Residual plot for subject", subjects[4],sep = " "))
```



It generally generated some better QQ-plots for some of the subjects. But there are still some inconsistency leaving some of the QQ-plots still showing skewness.

2. Now do a partial pooling model modelling objective response times as dependent on *task*? (set `REML=FALSE` in your `lmer` -specification)

```
m5 <- lmer(rt.obj ~ task + (1|subject) + (1|pas), data = df_exp_exp, REML = FALSE)
summary(m5)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: rt.obj ~ task + (1 | subject) + (1 | pas)
##    Data: df_exp_exp
##
##      AIC      BIC   logLik deviance df.resid
##  61917.5  61962.1 -30952.7  61905.5    12522
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
##  -0.640  -0.153  -0.065   0.046 101.556
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept) 0.1013   0.3183
##  pas      (Intercept) 0.0342   0.1849
##  Residual             8.1542   2.8555
## Number of obs: 12528, groups:  subject, 29; pas, 4
##
## Fixed effects:
##                  Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)     1.085e+00  1.185e-01  8.312e+00   9.152 1.27e-05 ***
## taskquadruplet -1.611e-01  6.251e-02  1.250e+04  -2.576  0.00999 **
## tasksingles    -1.557e-01  6.283e-02  1.248e+04  -2.478  0.01321 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.262
## tasksingles -0.267  0.495
```

```
MuMIn::r.squaredGLMM(m5)
```

```
## Warning: 'r.squaredGLMM' now calculates a revised statistic. See the help page.
```

```
##               R2m        R2c
## [1,] 0.0006726226 0.01701042
```

```
i. which would you include among your random effects and why? (support your choices w
ith relevant measures, taking into account variance explained and number of parameter
s going into the modelling)
```

```
partpoolm1.1 <- lmer(log(rt.obj) ~ task + (1 | subject), data = df_exp_exp, REML=FALS
E)
partpoolm1.2 <- lmer(log(rt.obj) ~ task + (1 | trial), data = df_exp_exp, REML=FALSE)
partpoolm1.3 <- lmer(log(rt.obj) ~ task + (1 | trial)+ (1|subject), data = df_exp_ex
p, REML=FALSE)
partpoolm1.4 <- lmer(log(rt.obj) ~ task + (1 + task | subject), data = df_exp_exp, RE
ML=FALSE)
```

```
## boundary (singular) fit: see ?isSingular
```

```
## Warning: Model failed to converge with 1 negative eigenvalue: -7.6e+02
```

```
partpoolm1.5 <- lmer(log(rt.obj) ~ task + (1 + task | subject) + (1 | trial), data =
 df_exp_exp, REML = F)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00270478 (tol = 0.002, component 1)
```

```
anova(partpoolm1.1,partpoolm1.2,partpoolm1.3,partpoolm1.4,partpoolm1.5)
```

```
## Data: df_exp_exp
## Models:
## partpoolm1.1: log(rt.obj) ~ task + (1 | subject)
## partpoolm1.2: log(rt.obj) ~ task + (1 | trial)
## partpoolm1.3: log(rt.obj) ~ task + (1 | trial) + (1 | subject)
## partpoolm1.4: log(rt.obj) ~ task + (1 + task | subject)
## partpoolm1.5: log(rt.obj) ~ task + (1 + task | subject) + (1 | trial)
##              npar   AIC   BIC logLik deviance   Chisq Df Pr(>Chisq)
## partpoolm1.1    5 29685 29722 -14838    29675
## partpoolm1.2    5 32560 32597 -16275    32550    0.00  0
## partpoolm1.3    6 29460 29505 -14724    29448 3102.21  1     <2e-16 ***
## partpoolm1.4   10 29560 29635 -14770    29540    0.00  4          1
## partpoolm1.5   11 29328 29409 -14653    29306  234.65  1     <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

m1.3 and m1.5 both seem to perform well while this seem to be due to having random intercept for both trial and subject. Adding a random slope of task for subjects improves the model slightly following AIC.

A random intercept for both task and subject is theoretically warranted as well as the effect of task being different between individuals.

partpoolm1.5 is therefore selected.

```
ii. explain in your own words what your chosen models says about response times betwe
en the different tasks
```

```
summary(partpoolm1.5)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: log(rt.obj) ~ task + (1 + task | subject) + (1 | trial)
##    Data: df_exp_exp
##
##      AIC      BIC   logLik deviance df.resid
##  29327.7  29409.5 -14652.8  29305.7    12517
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -10.5363  -0.4958  -0.0279   0.5119   8.0799
##
## Random effects:
##  Groups   Name           Variance Std.Dev. Corr
##  trial    (Intercept)    0.02995  0.17307
##  subject  (Intercept)    0.14916  0.38621
##           taskquadruplet 0.00400  0.06325   0.35
##           tasksingles    0.03227  0.17965   0.38 -0.58
##  Residual                0.57917  0.76103
## Number of obs: 12528, groups:  trial, 432; subject, 29
##
## Fixed effects:
##                Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)    -0.26407    0.07317 29.75926  -3.609  0.00111 **
## taskquadruplet -0.07225    0.02054 28.64575  -3.518  0.00147 **
## tasksingles    -0.17868    0.03740 29.11742  -4.778 4.66e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt  0.099
## tasksingles  0.279 -0.113
## optimizer (nloptwrap) convergence code: 0 (OK)
## Model failed to converge with max|grad| = 0.00270478 (tol = 0.002, component 1)
```

The estimated effect sizes for quadruplet and singles are negative, which indicates that these tasks will result in lower reaction time compared to single_task.

3. Now add *pas* and its interaction with *task* to the fixed effects

    i. how many types of group intercepts (random effects) can you add without ending up with convergence issues or singular fits?

```
partpoom2.1 <- lmer(log(rt.obj) ~ task*pas + (1|subject), data = df_exp_exp, REML = F
)
partpoolm2.2 <- lmer(log(rt.obj) ~ task*pas + (1|trial) + (1|subject), data = df_exp_
exp, REML = F)
partpoolm2.3 <- lmer(log(rt.obj) ~ task*pas + (1|trial) + (1|subject) +  (1|cue), dat
a = df_exp_exp, REML = F)
partpoolm2.4 <- lmer(log(rt.obj) ~ task*pas + (1|trial) + (1|subject) +  (1|cue) + (1
|target.contrast), data = df_exp_exp, REML = F)
```

```
## boundary (singular) fit: see ?isSingular
```

When adding a fourth intercept an error occurs (singular fit)

```
ii. create a model by adding random intercepts (without modelling slopes) that result
s in a singular fit - then use `print(VarCorr(<your.model>), comp='Variance')` to ins
pect the variance vector - explain why the fit is singular (Hint: read the first para
graph under details in the help for `isSingular`)'
```

```
print(VarCorr(partpoolm2.4), comp='Variance')
```

```
##  Groups          Name        Variance
##  trial           (Intercept) 0.0268422
##  cue             (Intercept) 0.0040356
##  target.contrast (Intercept) 0.0000000
##  subject         (Intercept) 0.1628691
##  Residual                    0.5715602
```

Some of the variances approaches zero.

```
iii. in your own words - how could you explain why your model would result in a singu
lar fit?
```

The model will give an error of “is singular” when the random effect’s variance is nearly zero. Could be due to different things. Adding many random parameters will results in a model, where they each do NOT explain a lot of variance in the data. Or a certain selection of random effect is not warranted. It can also occur when the variables are highly correlated (i.e. correlation close to either -1 or 1).

Exercise 3

1. Initialise a new data frame, `data.count` . *count* should indicate the number of times they categorized their experience as *pas* 1-4 for each *task*. I.e. the data frame would have for subject 1: for task:singles, pas1 was used # times, pas2 was used # times, pas3 was used # times and pas4 was used # times. You would then do the same for task:pairs and task:quadruplet

```
## you can start from this if you want to, but you can also make your own from scratc
h
data.count <- data.frame(count = numeric(),
                         pas = numeric(), ## remember to make this into a factor afte
rwards
                         task = numeric(), ## and this too
                         subject = numeric()) ## and this too
```

```
df_count <- df_exp %>%
  group_by(subject, task, pas) %>%
  summarise(count = n()) %>%
  mutate(pas = as.factor(pas))
```

```
## `summarise()` has grouped output by 'subject', 'task'. You can override using the
`.groups` argument.
```

2. Now fit a multilevel model that models a unique “slope” for *pas* for each *subject* with the interaction between *pas* and *task* and their main effects being modelled.

```
m3.1 <- glmer(count ~ pas*task + (1+pas|subject), family = poisson(link = "log"), dat
a = df_count)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00343245 (tol = 0.002, component 1)
```

```
summary(m3.1)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas * task + (1 + pas | subject)
##    Data: df_count
##
##      AIC      BIC   logLik deviance df.resid
##   3148.4   3232.7  -1552.2   3104.4      318
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.3872 -0.7853 -0.0472  0.7552  6.5435
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3322   0.5764
##          pas2        0.3801   0.6165   -0.75
##          pas3        1.1956   1.0934   -0.84  0.63
##          pas4        2.3731   1.5405   -0.86  0.42  0.72
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          4.03550    0.10974  36.773  < 2e-16 ***
## pas2                -0.02364    0.11960  -0.198 0.843332
## pas3                -0.51322    0.20715  -2.478 0.013230 *
## pas4                -0.77251    0.29073  -2.657 0.007880 **
## taskquadruplet       0.11495    0.03127   3.676 0.000237 ***
## tasksingles         -0.23090    0.03418  -6.755 1.43e-11 ***
## pas2:taskquadruplet -0.11377    0.04605  -2.470 0.013500 *
## pas3:taskquadruplet -0.20912    0.05287  -3.956 7.63e-05 ***
## pas4:taskquadruplet -0.21508    0.05230  -4.113 3.91e-05 ***
## pas2:tasksingles     0.19537    0.04830   4.045 5.23e-05 ***
## pas3:tasksingles     0.24291    0.05369   4.524 6.07e-06 ***
## pas4:tasksingles     0.56339    0.05101  11.044  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr tsksng ps2:tskq ps3:tskq
## pas2        -0.742
## pas3        -0.829  0.613
## pas4        -0.847  0.412  0.703
## taskqudrplt -0.151  0.138  0.080  0.057
## tasksingles -0.138  0.126  0.073  0.052  0.484
## ps2:tskqdrp  0.102 -0.198 -0.054 -0.039 -0.679 -0.328
## ps3:tskqdrp  0.089 -0.082 -0.125 -0.034 -0.592 -0.286  0.402
## ps4:tskqdrp  0.090 -0.083 -0.048 -0.093 -0.598 -0.289  0.406    0.354
## ps2:tsksngl  0.098 -0.189 -0.052 -0.037 -0.342 -0.708  0.490    0.203
## ps3:tsksngl  0.088 -0.081 -0.124 -0.033 -0.308 -0.637  0.209    0.486
## ps4:tsksngl  0.092 -0.085 -0.049 -0.091 -0.324 -0.670  0.220    0.192
##             ps4:tskq ps2:tsks ps3:tsks
## pas2
## pas3
## pas4
```

```
## taskqudrplt
## tasksingles
## ps2:tskqdrp
## ps3:tskqdrp
## ps4:tskqdrp
## ps2:tsksngl  0.205
## ps3:tsksngl  0.184    0.451
## ps4:tsksngl  0.507    0.474    0.427
## optimizer (Nelder_Mead) convergence code: 0 (OK)
## Model failed to converge with max|grad| = 0.00343245 (tol = 0.002, component 1)
```

```
i. which family should be used?
```

Poisson which is basically a binomial logisitc regression where the boundaries approaches/goes to 0.

```
ii. why is a slope for _pas_ not really being modelled?
```

Because pas i being treated as a factor we're not really modelling a slope but rather the differences between the levels of pas **compared to the first level.**

```
iii. if you get a convergence error, try another algorithm (the default is the _Nelde
r_Mead_) - try (_bobyqa_) for which the `dfoptim` package is needed. In `glmer`, you
can add the following for the `control` argument: `glmerControl(optimizer="bobyqa")`
(if you are interested, also have a look at the function `allFit`)
```

```
pacman::p_load(dfoptim)
```

Poison modeling

```
m3.2 <- glmer(count ~ pas*task + (1+pas|subject), family = poisson(link = "log"),cont
rol = glmerControl(optimizer="bobyqa"),data = df_count)
summary(m3.2)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas * task + (1 + pas | subject)
##    Data: df_count
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3148.4   3232.7  -1552.2   3104.4      318
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.3871 -0.7853 -0.0469  0.7550  6.5438
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3324   0.5765
##          pas2        0.3803   0.6167   -0.75
##          pas3        1.1960   1.0936   -0.84  0.63
##          pas4        2.3736   1.5407   -0.86  0.42  0.72
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)          4.03570    0.10976  36.769  < 2e-16 ***
## pas2                -0.02378    0.11963  -0.199 0.842458
## pas3                -0.51365    0.20718  -2.479 0.013166 *
## pas4                -0.77292    0.29075  -2.658 0.007853 **
## taskquadruplet       0.11490    0.03127   3.674 0.000239 ***
## tasksingles         -0.23095    0.03418  -6.756 1.42e-11 ***
## pas2:taskquadruplet -0.11375    0.04605  -2.470 0.013508 *
## pas3:taskquadruplet -0.20901    0.05287  -3.954 7.70e-05 ***
## pas4:taskquadruplet -0.21500    0.05230  -4.111 3.94e-05 ***
## pas2:tasksingles     0.19536    0.04830   4.045 5.23e-05 ***
## pas3:tasksingles     0.24299    0.05369   4.526 6.02e-06 ***
## pas4:tasksingles     0.56346    0.05101  11.045  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr tsksng ps2:tskq ps3:tskq
## pas2        -0.742
## pas3        -0.829  0.613
## pas4        -0.847  0.412  0.703
## taskqudrplt -0.151  0.138  0.080  0.057
## tasksingles -0.138  0.126  0.073  0.052  0.484
## ps2:tskqdrp  0.102 -0.198 -0.054 -0.039 -0.679 -0.328
## ps3:tskqdrp  0.089 -0.082 -0.125 -0.034 -0.592 -0.286  0.402
## ps4:tskqdrp  0.090 -0.083 -0.048 -0.093 -0.598 -0.289  0.406    0.354
## ps2:tsksngl  0.098 -0.188 -0.052 -0.037 -0.342 -0.708  0.490    0.203
## ps3:tsksngl  0.088 -0.080 -0.124 -0.033 -0.308 -0.637  0.209    0.486
## ps4:tsksngl  0.092 -0.085 -0.049 -0.091 -0.324 -0.670  0.220    0.192
##             ps4:tskq ps2:tsks ps3:tsks
## pas2
## pas3
```

```
## pas4
## taskqudrplt
## tasksingles
## ps2:tskqdrp
## ps3:tskqdrp
## ps4:tskqdrp
## ps2:tsksngl  0.205
## ps3:tsksngl  0.184    0.451
## ps4:tsksngl  0.507    0.474    0.427
```

iv. when you have a converging fit - fit a model with only the main effects of _pas_ and _task_. Compare this with the model that also includes the interaction

```
m3.3 <- glmer(count ~ pas+task + (1+pas|subject), family = poisson(link = "log"),cont
rol = glmerControl(optimizer="bobyqa"),data = df_count)
summary(m3.3)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas + task + (1 + pas | subject)
##    Data: df_count
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3398.5   3459.8  -1683.3   3366.5      324
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -5.5885 -0.9001 -0.0477  0.8253  6.5100
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3325   0.5766
##          pas2        0.3805   0.6169   -0.75
##          pas3        1.1895   1.0906   -0.84  0.63
##          pas4        2.4222   1.5563   -0.86  0.42  0.73
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)     4.004589   0.108724  36.833   <2e-16 ***
## pas2           -0.006528   0.116616  -0.056   0.9554
## pas3           -0.509918   0.204452  -2.494   0.0126 *
## pas4           -0.663832   0.291962  -2.274   0.0230 *
## taskquadruplet  0.003294   0.018188   0.181   0.8563
## tasksingles     0.004307   0.018177   0.237   0.8127
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr
## pas2        -0.742
## pas3        -0.832  0.623
## pas4        -0.850  0.412  0.723
## taskqudrplt -0.084  0.000  0.001 -0.002
## tasksingles -0.084  0.000  0.000  0.000  0.501
```

```
v. indicate which of the two models, you would choose and why
```

First we will add another column containing the total number of counts in each grouping of participant and task. The log() verison of that column will work as our **offset** variable in future models.

```
df_count <- df_count %>%
  group_by(task,subject) %>%
  mutate(total_count = sum(count))
```

Modeling with offset

Poission regression can either model count or rate data. So far we have modelled count data. But our count data is a fraction of larger grouping or time interval. In our case the frequency will be estimated within the grouping of task and subject. Subject 1 & task pairs has 170 data points where 4 of them is in pas = 4. The frequency is therefore 4/170.

Disclaimer I am not quite sure whether it is best practise to divide by subject in the grouping or if should only be done by task as we account for indivudial variance with the random effect of subject???

```
m3.4 <- glmer(count ~ pas*task + (1+pas|subject), family = poisson(link = "log"),cont
rol = glmerControl(optimizer="bobyqa"),data = df_count, offset = log(total_count))
summary(m3.4)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas * task + (1 + pas | subject)
##    Data: df_count
##  Offset: log(total_count)
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3132.5   3216.7  -1544.2   3088.5      318
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.3832 -0.7914 -0.0361  0.8026  6.4078
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3017   0.5493
##          pas2        0.3786   0.6153   -0.72
##          pas3        1.1754   1.0842   -0.86  0.62
##          pas4        2.2269   1.4923   -0.90  0.43  0.72
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                     Estimate Std. Error z value Pr(>|z|)
## (Intercept)         -1.29022    0.10482 -12.309  < 2e-16 ***
## pas2                -0.02788    0.11938  -0.234 0.815350
## pas3                -0.51782    0.20545  -2.520 0.011722 *
## pas4                -0.75192    0.28146  -2.672 0.007551 **
## taskquadruplet       0.10589    0.03128   3.385 0.000711 ***
## tasksingles         -0.23847    0.03419  -6.975 3.06e-12 ***
## pas2:taskquadruplet -0.11152    0.04606  -2.421 0.015468 *
## pas3:taskquadruplet -0.20335    0.05287  -3.846 0.000120 ***
## pas4:taskquadruplet -0.20713    0.05231  -3.960 7.51e-05 ***
## pas2:tasksingles     0.19916    0.04831   4.123 3.74e-05 ***
## pas3:tasksingles     0.24851    0.05370   4.628 3.70e-06 ***
## pas4:tasksingles     0.56857    0.05102  11.143  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr tsksng ps2:tskq ps3:tskq
## pas2        -0.714
## pas3        -0.847  0.610
## pas4        -0.880  0.419  0.704
## taskqudrplt -0.158  0.138  0.080  0.059
## tasksingles -0.144  0.127  0.074  0.054  0.484
## ps2:tskqdrp  0.107 -0.198 -0.055 -0.040 -0.679 -0.328
## ps3:tskqdrp  0.093 -0.082 -0.126 -0.035 -0.592 -0.286  0.402
## ps4:tskqdrp  0.094 -0.083 -0.048 -0.097 -0.598 -0.289  0.406    0.354
## ps2:tsksngl  0.102 -0.189 -0.052 -0.038 -0.342 -0.708  0.490    0.203
## ps3:tsksngl  0.092 -0.081 -0.125 -0.034 -0.308 -0.637  0.209    0.486
## ps4:tsksngl  0.097 -0.085 -0.049 -0.094 -0.324 -0.670  0.220    0.192
##             ps4:tskq ps2:tsks ps3:tsks
## pas2
```

```
## pas3
## pas4
## taskqudrplt
## tasksingles
## ps2:tskqdrp
## ps3:tskqdrp
## ps4:tskqdrp
## ps2:tsksngl  0.205
## ps3:tsksngl  0.184    0.451
## ps4:tsksngl  0.507    0.474    0.427
```

```
m3.5 <- glmer(count ~ pas+task + (1+pas|subject), family = poisson(link = "log"),cont
rol = glmerControl(optimizer="bobyqa"),data = df_count, offset = log(total_count))
summary(m3.5)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas + task + (1 + pas | subject)
##    Data: df_count
##  Offset: log(total_count)
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##   3381.9   3443.1  -1674.9   3349.9      324
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -5.5004 -0.9089 -0.0148  0.8785  6.2895
##
## Random effects:
##  Groups  Name        Variance Std.Dev. Corr
##  subject (Intercept) 0.3020   0.5495
##          pas2        0.3789   0.6155   -0.72
##          pas3        1.1695   1.0814   -0.86  0.63
##          pas4        2.2744   1.5081   -0.90  0.42  0.73
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                  Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -1.324e+00  1.038e-01 -12.754   <2e-16 ***
## pas2           -8.617e-03  1.164e-01  -0.074   0.9410
## pas3           -5.104e-01  2.027e-01  -2.517   0.0118 *
## pas4           -6.387e-01  2.827e-01  -2.260   0.0238 *
## taskquadruplet -2.290e-03  1.819e-02  -0.126   0.8998
## tasksingles     3.191e-06  1.818e-02   0.000   0.9999
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) pas2   pas3   pas4   tskqdr
## pas2        -0.712
## pas3        -0.850  0.621
## pas4        -0.884  0.419  0.724
## taskqudrplt -0.088  0.000  0.001 -0.002
## tasksingles -0.088  0.000  0.000  0.000  0.501
```

Comparing interaction and offset models

```
#check R^2
MuMIn::r.squaredGLMM(m3.2)
```

```
##                 R2m       R2c
## delta     0.1577062 0.9750530
## lognormal 0.1577375 0.9752462
## trigamma  0.1576745 0.9748567
```

```
MuMIn::r.squaredGLMM(m3.3)
```

```
##                 R2m       R2c
## delta     0.1375957 0.9747420
## lognormal 0.1376233 0.9749376
## trigamma  0.1375677 0.9745434
```

```
MuMIn::r.squaredGLMM(m3.4)
```

```
##                  R2m        R2c
## delta     0.02414973 0.14650923
## lognormal 0.04667814 0.28318238
## trigamma  0.00720645 0.04371939
```

```
MuMIn::r.squaredGLMM(m3.5)
```

```
##                   R2m        R2c
## delta     0.020714560 0.14477964
## lognormal 0.040114255 0.28036933
## trigamma  0.006172595 0.04314193
```

```
#check with anova
anova(m3.2,m3.3,m3.4,m3.5)
```

```
## Data: df_count
## Models:
## m3.3: count ~ pas + task + (1 + pas | subject)
## m3.5: count ~ pas + task + (1 + pas | subject)
## m3.2: count ~ pas * task + (1 + pas | subject)
## m3.4: count ~ pas * task + (1 + pas | subject)
##      npar    AIC    BIC  logLik deviance   Chisq Df Pr(>Chisq)
## m3.3   16 3398.5 3459.8 -1683.3   3366.5
## m3.5   16 3381.9 3443.1 -1674.9   3349.9  16.683  0
## m3.2   22 3148.4 3232.7 -1552.2   3104.4 245.425  6  < 2.2e-16 ***
## m3.4   22 3132.5 3216.7 -1544.2   3088.5  15.945  0
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The interaction effects are significant in both our model with and without offset. As showed by our culminated R2 the interaction effect also adds a tiny bit of explained variance while also performing better following AIC and BIC which punishes unceccesary model complexity.

The inclusion of an offset variable is theoretical warranted when working with frequencies in poission regression as argued by Gelman and Hill in (Data Analysis Using Regression and Multilevel/Hierarchical Models, 2007). The offset variable

also shows a marginal improvement in AIC/BIC but with a enormous reduction in culminated R2.

My model of choice will be m3.4 as it contains the needed interaction effect and offset variable which also performs the best following the AIC and BIC.

```
mean(df_count$count)
```

```
## [1] 53.32647
```

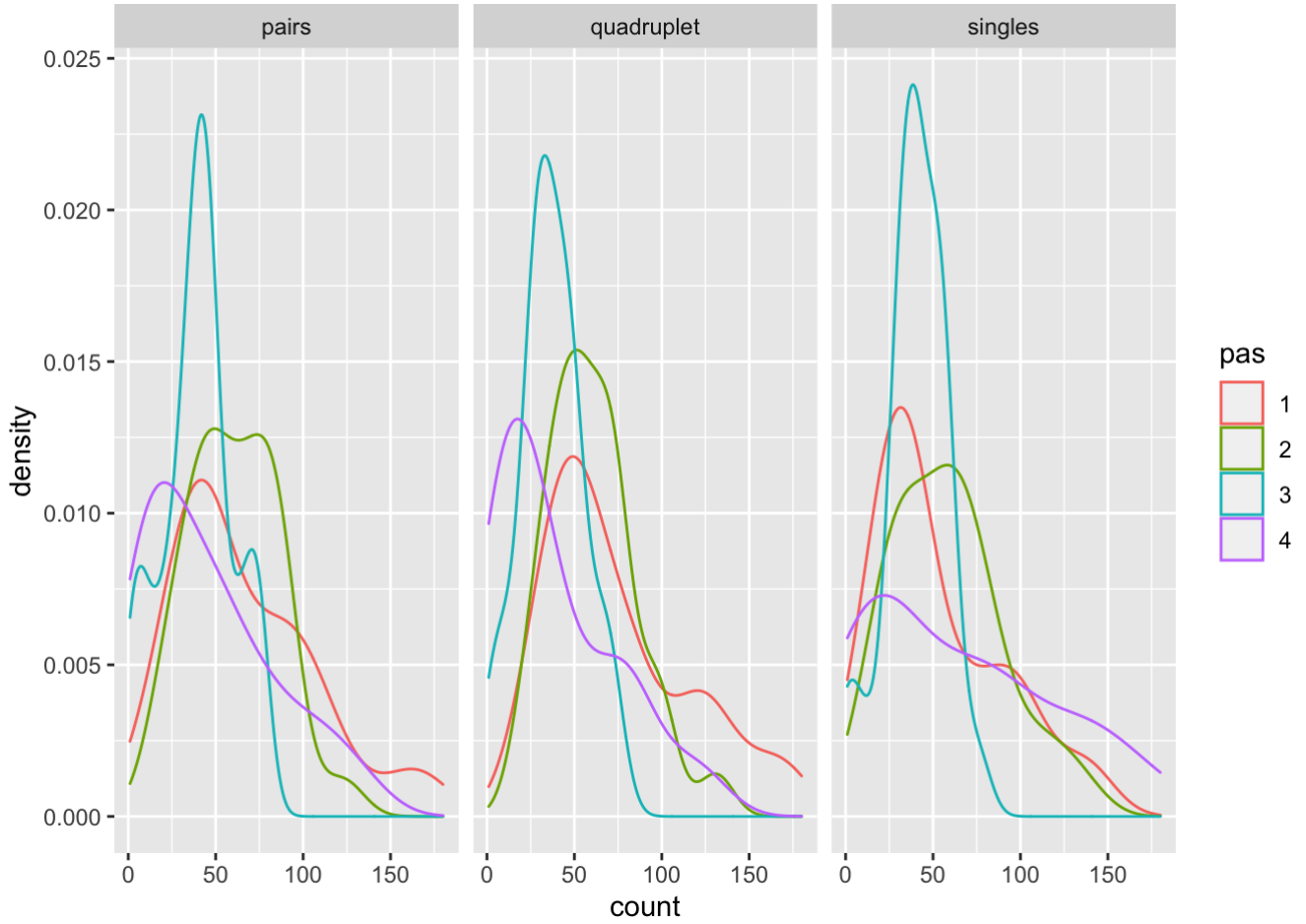
```
var(df_count$count)
```

```
## [1] 1214.002
```

as $mean var $ this is an indication of that there will be overdispersion in all our models. But as non of my known dispersiontest (If the Residual Deviance is greater than the degrees of freedom, then over-dispersion exists) and quasipoission distributions work on mixed effect modelling I don't know what can be done about it. Ideally something would be done like a quasipoission regression.

```
vi. based on your chosen model - write a short report on what this says about the dis
tribution of ratings as dependent on _pas_ and _task_
```

```
ggplot(df_count, aes(col = pas, x = count)) + geom_density() + facet_wrap(~task)
```




```
exp(fixef(m3.4))
```

```
##          (Intercept)                 pas2                 pas3                 pas4
##            0.2752108            0.9725073            0.5958187            0.4714597
##      taskquadruplet          tasksingles pas2:taskquadruplet pas3:taskquadruplet
##            1.1117039            0.7878326            0.8944698            0.8159922
## pas4:taskquadruplet     pas2:tasksingles     pas3:tasksingles     pas4:tasksingles
##            0.8129117            1.2203738            1.2821197            1.7657324
```

- exp(α)= effect on the mean μ, when X = 0
- exp(β) = with every unit increase in X, the predictor variable has multiplicative effect of exp(β) on the mean of Y, that is μ
- If β = 0, then exp(β) = 1, and the expected count is exp(α) and, Y and X are not related.
- If β > 0, then exp(β) > 1, and the expected count is exp(β) times larger than when X = 0
- If β < 0, then exp(β) < 1, and the expected count is exp(β) times smaller than when X = 0

Pas fix

Pas2 will reduce the frequency of count by *(1-0.9725)* percentage compared to pas1. As shown by the exp(fixef(m3.4)) pas3 and pas4 will result in an even more drastic reduction compared to pas1.

Task fix

Quadruplet will increase count frequency with *(1-1.11)* percentage compared to task_pairs. Though task_singles decrease the frequency with *(1-0.78)* percentage compared to task_pairs.

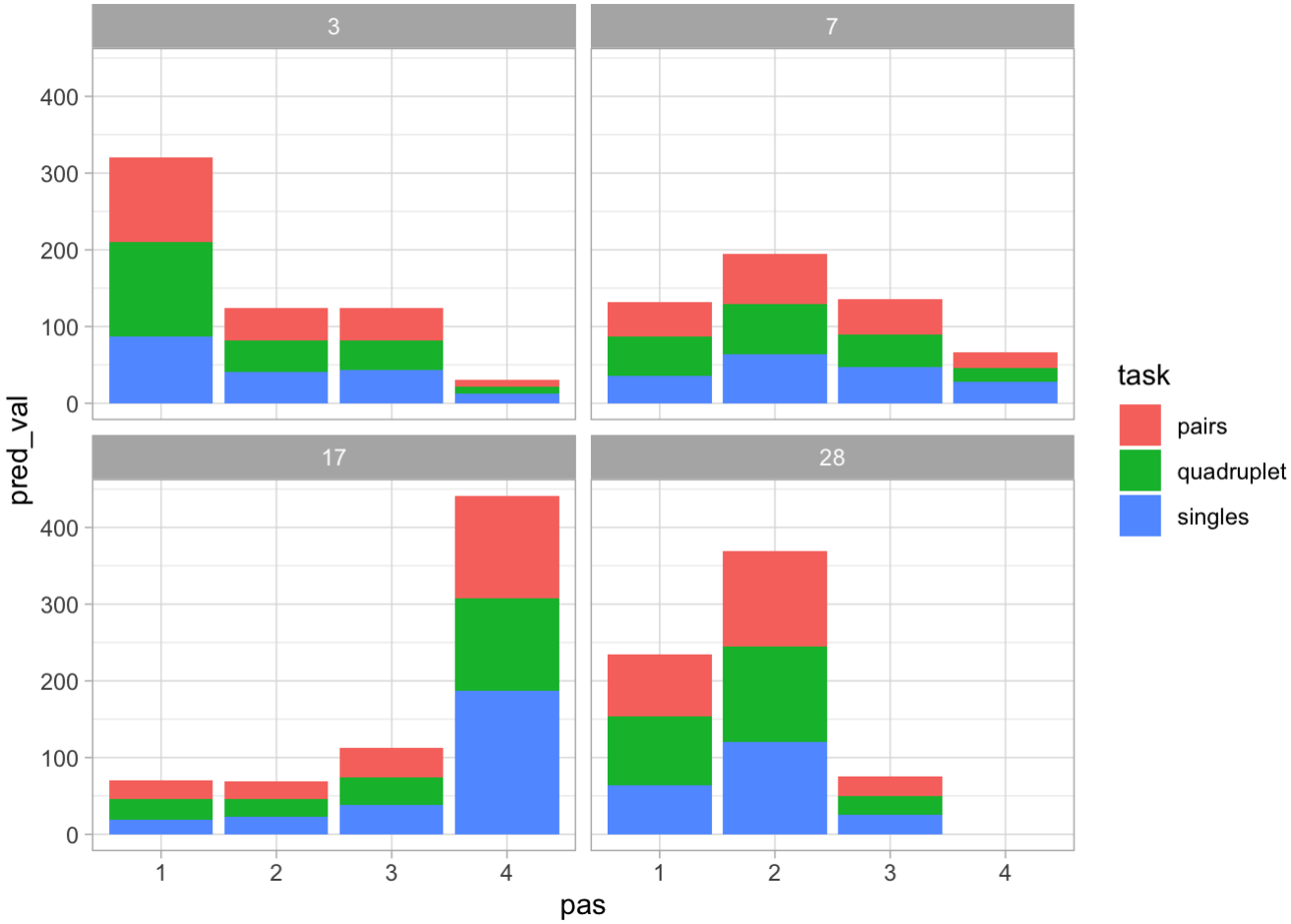
Interaction effect

The interaction effects show that task_singles interaction with pas level 2:4 increase our frequency. While the interaction between task_quadruplet and pas level 2:4 will significantly decrease our frequency. In a percentage wise interpretation the effect of some of the interactions are even larger than the fixed effects alone. This is seems kinda odd as adding the interaction effect didn't improve the models with a big margin.

```
vii. include a plot that shows the estimated amount of ratings for four subjects of y
our choosing
```

As our current selected model is a model of frequency it wouldn't be fitting for plotting "amount". There the similair count model m3.2 without the offset is selected for this assigment.

```
df_count_fil <- df_count %>%
  filter(subject == '3' | subject == '7' | subject == '17' | subject == '28')


df_count_fil_pred <- cbind(df_count_fil, pred_val = exp(predict(m3.2, newdata = df_co
unt_fil)))

ggplot(df_count_fil_pred, aes(x = pas, y = pred_val, fill = task)) +
  geom_bar(stat = 'identity') +
  facet_wrap(~ subject) +
  theme_light()
```



3. Finally, fit a multilevel model that models *correct* as dependent on *task* with a unique intercept for each *subject* Two options:
4. I could calculate the amount of corrects and do a poission regression on the count or frequency of correct responses.

```
#Something like this where I also add some of
df_exp_count_correct <- df_exp %>%
  group_by(subject,task,pas) %>%
  summarise(n_correct = sum(right_answer == 1))
```

```
## `summarise()` has grouped output by 'subject', 'task'. You can override using the
`.groups` argument.
```

```
df_exp_count_correct <- df_exp_count_correct %>%
  group_by(subject) %>%
  mutate(n_answers = sum(n_correct)) %>%
  mutate(pas = as.factor(pas))
```

```
model1_pois <- glmer(n_correct ~ task + (1|subject), data = df_exp_count_correct,
                offset = log(n_answers), family = poisson(link = "log"))
summary(model1_pois)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: n_correct ~ task + (1 | subject)
##    Data: df_exp_count_correct
##  Offset: log(n_answers)
##
##      AIC      BIC   logLik deviance df.resid
##   7720.9   7736.2  -3856.4   7712.9      336
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -6.8366 -2.7035 -0.8973  2.2792 17.5566
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.003766 0.06137
## Number of obs: 340, groups:  subject, 29
##
## Fixed effects:
##                Estimate Std. Error  z value Pr(>|z|)
## (Intercept)    -2.46639    0.01882 -131.033   <2e-16 ***
## taskquadruplet -0.02958    0.02122   -1.394   0.1633
## tasksingles     0.04724    0.02087    2.264   0.0236 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.558
## tasksingles -0.567  0.503
```

```
exp(fixef(model1_pois))
```

```
##    (Intercept) taskquadruplet    tasksingles
##     0.08489079     0.97085114     1.04837246
```

```
MuMIn::r.squaredGLMM(model1_pois)
```

```
## Warning: The null model is correct only if all variables used by the original
## model remain unchanged.
```

```
##                    R2m          R2c
## delta     8.594200e-05 4.081108e-04
## lognormal 3.947110e-04 1.874355e-03
## trigamma  7.280081e-06 3.457075e-05
```

2. I could just do a normal binomial regression trying to predict correct.

```
model1_binom <- glmer(right_answer ~ task + (1|subject), data = df_exp,
                      family = binomial(link = "logit"))
summary(model1_binom)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: right_answer ~ task + (1 | subject)
##    Data: df_exp
##
##      AIC      BIC   logLik deviance df.resid
##  19927.2  19958.4  -9959.6  19919.2    18127
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -2.7426 -1.0976  0.5098  0.6101  0.9111
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.1775   0.4214
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     1.10071    0.08387  13.124  < 2e-16 ***
## taskquadruplet -0.09825    0.04190  -2.345    0.019 *
## tasksingles     0.18542    0.04336   4.276  1.9e-05 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.256
## tasksingles -0.247  0.495
```

```
invlogit(fixef(model1_binom))
```

```
##    (Intercept) taskquadruplet    tasksingles
##      0.7503932      0.4754570      0.5462226
```

```
MuMIn::r.squaredGLMM(model1_binom)
```

```
## Warning: The null model is correct only if all variables used by the original
## model remain unchanged.
```

```
##                    R2m       R2c
## theoretical 0.003979585 0.05497746
## delta       0.002526101 0.03489776
```

```
i. does _task_ explain performance?
```

Both models shows task significantly predicts right_answer. Binomial models shows a rather lower R2c = 0.055 but the poisson model is attrosious.

```
ii. add _pas_ as a main effect on top of _task_ - what are the consequences of that?
```

Due to my time and your reading time I will only continue using the binomial approach. ;)

```
model2_binom <- glmer(right_answer ~ task + pas + (1|subject), data = df_exp,
                      family = binomial(link = "logit"))
summary(model2_binom)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: right_answer ~ task + pas + (1 | subject)
##    Data: df_exp
##
##      AIC      BIC   logLik deviance df.resid
##  17425.0  17464.0  -8707.5  17415.0    18126
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.1096 -0.6101  0.3181  0.5653  1.6476
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.2004   0.4477
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##                 Estimate Std. Error z value Pr(>|z|)
## (Intercept)    -0.950104   0.098399  -9.656   <2e-16 ***
## taskquadruplet -0.029418   0.045016  -0.653    0.513
## tasksingles    -0.008914   0.046889  -0.190    0.849
## pas             1.014031   0.022900  44.281   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng
## taskqudrplt -0.247
## tasksingles -0.189  0.489
## pas         -0.421  0.030 -0.083
```

```
invlogit(fixef(model2_binom))
```

```
##    (Intercept) taskquadruplet    tasksingles            pas
##      0.2788640      0.4926460      0.4977716      0.7338084
```

```
MuMIn::r.squaredGLMM(model2_binom)
```

```
## Warning: The null model is correct only if all variables used by the original
## model remain unchanged.
```

```
##                    R2m       R2c
## theoretical 0.2728668 0.3146160
## delta       0.1925254 0.2219821
```

R2c has gotten a boost up to 0.31 from 0.055. The effect size of task_quadruplet and task_singles has almost remained the same but the is now no longer signifcant.

```
anova(model1_binom, model2_binom)
```

```
## Data: df_exp
## Models:
## model1_binom: right_answer ~ task + (1 | subject)
## model2_binom: right_answer ~ task + pas + (1 | subject)
##              npar   AIC   BIC  logLik deviance  Chisq Df Pr(>Chisq)
## model1_binom    4 19927 19958 -9959.6    19919
## model2_binom    5 17425 17464 -8707.5    17415 2504.2  1  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Model 2 appears to be the best model.

```
iii. now fit a multilevel model that models _correct_ as dependent on _pas_ with a un
ique intercept for each _subject_
```

```
model3_binom <- glmer(right_answer ~ pas + (1|subject), data = df_exp,
                      family = binomial(link = "logit"))
```

```
iv. finally, fit a model that models the interaction between _task_ and _pas_  and th
eir main effects
```

```
model4_binom <- glmer(right_answer ~ pas*task + (1|subject), data = df_exp,
                      family = binomial(link = "logit"))
```

```
v. describe in your words which model is the best in explaining the variance in accur
acy
```

```
anova(model1_binom,model2_binom,model3_binom,model4_binom)
```

```
## Data: df_exp
## Models:
## model3_binom: right_answer ~ pas + (1 | subject)
## model1_binom: right_answer ~ task + (1 | subject)
## model2_binom: right_answer ~ task + pas + (1 | subject)
## model4_binom: right_answer ~ pas * task + (1 | subject)
##              npar   AIC   BIC  logLik deviance     Chisq Df Pr(>Chisq)
## model3_binom    3 17422 17445 -8707.7    17416
## model1_binom    4 19927 19958 -9959.6    19919    0.0000  1    1.00000
## model2_binom    5 17425 17464 -8707.5    17415 2504.2020  1    < 2e-16 ***
## model4_binom    7 17423 17478 -8704.4    17409    6.1866  2    0.04535 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Model 1 has a significantly higher BIC and AIC than the rest. We can therefore conclude including pas as predictor of right_answer is key for explaining the variance in the data.

Currently the choice is between model2, model3, model4. So I suggest we do a cross-validation test to test the generalisability of the models.

Cross-Validation

```
pacman::p_load(caret)
#sample
rand_sample <- createDataPartition(df_exp $ right_answer, p = 0.8, list = FALSE)
#training
train_df <- df_exp[rand_sample,]
#test
test_df <- df_exp[-rand_sample,]
```

```
#Train on train_df
cv_model2 <- glmer(right_answer ~ task + pas + (1|subject), data = train_df,
                     family = binomial(link = "logit"))
cv_model3 <- glmer(right_answer ~ pas + (1|subject), data = train_df,
                     family = binomial(link = "logit"))
cv_model4 <- glmer(right_answer ~ pas*task + (1|subject), data = train_df,
                     family = binomial(link = "logit"))
```

```
#Predict
predictions2 <- predict(cv_model2, test_df, type = "response")
predictions3 <- predict(cv_model3, test_df, type = "response")
predictions4 <- predict(cv_model4, test_df, type = "response")
#Turn into binary response
prediction_bin2 <- if_else(predictions2 > 0.5, 1, 0)
prediction_bin3 <- if_else(predictions3 > 0.5, 1, 0)
prediction_bin4 <- if_else(predictions4 > 0.5, 1, 0)

#append predicted values to test_df
test_df <- test_df %>%
  mutate(pred_val2 = as.factor(prediction_bin2), pred_val3 = as.factor(prediction_bin
3), pred_val4 = as.factor(prediction_bin4))
```

```
#Confusion Matrix for model2
confusionMatrix(data=test_df$pred_val2, reference = test_df$right_answer)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  194  208
##          1  715 2509
##
##                Accuracy : 0.7454
##                  95% CI : (0.7309, 0.7596)
##     No Information Rate : 0.7493
##     P-Value [Acc > NIR] : 0.7115
##
##                   Kappa : 0.1681
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.2134
##             Specificity : 0.9234
##          Pos Pred Value : 0.4826
##          Neg Pred Value : 0.7782
##              Prevalence : 0.2507
##          Detection Rate : 0.0535
##    Detection Prevalence : 0.1109
##       Balanced Accuracy : 0.5684
##
##        'Positive' Class : 0
##
```

```
#Confusion Matrix for model3
confusionMatrix(data=test_df$pred_val3, reference = test_df$right_answer)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  194  209
##          1  715 2508
##
##                Accuracy : 0.7452
##                  95% CI : (0.7307, 0.7593)
##     No Information Rate : 0.7493
##     P-Value [Acc > NIR] : 0.7244
##
##                   Kappa : 0.1675
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.2134
##             Specificity : 0.9231
##          Pos Pred Value : 0.4814
##          Neg Pred Value : 0.7782
##              Prevalence : 0.2507
##          Detection Rate : 0.0535
##    Detection Prevalence : 0.1111
##       Balanced Accuracy : 0.5682
##
##        'Positive' Class : 0
##
```

```
#Confusion Matrix for model4
confusionMatrix(data=test_df$pred_val4, reference = test_df$right_answer)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0  183  200
##          1  726 2517
##
##                Accuracy : 0.7446
##                  95% CI : (0.7301, 0.7588)
##     No Information Rate : 0.7493
##     P-Value [Acc > NIR] : 0.7493
##
##                   Kappa : 0.1582
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.20132
##             Specificity : 0.92639
##          Pos Pred Value : 0.47781
##          Neg Pred Value : 0.77613
##              Prevalence : 0.25069
##          Detection Rate : 0.05047
##    Detection Prevalence : 0.10563
##       Balanced Accuracy : 0.56385
##
##        'Positive' Class : 0
##
```

sum up As all models have exactly the same accuracy we could continue with doing k-fold cross-validation. But for now I'll just argue that we care more about true-negative rate. Correctly classifying a person with a tumor is not fun, but it would be even worse to miss diagnose someone as a false-negative which could then result in death due to lack of treatment. I know the analogy doesn't quite fit in this experiment but I need some way to choose. ;)

model4 shows the best true-negative-rate and will therefore be chosen as the final model.

portfolio 2.2, Methods 3, 2021, autumn semester

Sigurd Fyhn Sørensen

13-10-2021

Exercises and objectives

The objectives of the exercises of this assignment are based on:
https://doi.org/10.1016/j.concog.2019.03.007
(https://doi.org/10.1016/j.concog.2019.03.007)

4. Download and organise the data from experiment 1
5. Use log-likelihood ratio tests to evaluate logistic regression models
6. Test linear hypotheses
7. Estimate psychometric functions for the Perceptual Awareness Scale and evaluate them

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This is part 2 of Assignment 2 and will be part of your final portfolio

EXERCISE 4 - Download and organise the data from experiment 1

```
pacman::p_load(tidyverse, lmerTest, lme4, stargazer, carret, readbulk, TinyTex, rstan
arm, reabulk)
```

Go to https://osf.io/ecxsj/files/ (https://osf.io/ecxsj/files/) and download the files associated with Experiment 1 (there should be 29).
The data is associated with Experiment 1 of the article at the following DOI
https://doi.org/10.1016/j.concog.2019.03.007
(https://doi.org/10.1016/j.concog.2019.03.007)

1. Put the data from all subjects into a single data frame - note that some of the subjects do not have the *seed* variable. For these subjects, add this variable and make in *NA* for all observations. (The *seed* variable will not be part of the analysis and is not an experimental variable)

I identified some na's in the seed variable.

```
sum(is.na(df_exp1$seed))
```

```
## [1] 2646
```

precisely 2646. So no need to change anything as the NA's has aleady been added to the seed variable.

```
i. Factorise the variables that need factorising
```

```
df_exp1 <- df_exp1 %>%
  mutate(right_answer = if_else(target.type == "odd" & obj.resp == "o" | target.type
 == "even" & obj.resp == "e",1,0)) %>%
  mutate(right_answer = as.numeric(right_answer))


df_exp1 <- df_exp1 %>%
  mutate(right_answer = as.numeric(right_answer)) %>%
  mutate(subject = as.factor(subject)) %>%
  mutate(task = as.factor(task)) %>%
  mutate(target.type = as.factor (target.type)) %>%
  mutate(trial.type = as.factor(trial.type)) %>%
  mutate(pas = as.factor(pas))
```

```
ii. Remove the practice trials from the dataset (see the _trial.type_ variable)
```

```
levels(df_exp1$trial.type)
```

```
## [1] "experiment" "practice"
```

```
df_exp1 <- df_exp1 %>%
  filter(trial.type == "experiment")
```

```
iii. Create a _correct_ variable
```

This has been done in the *r preprocess* chunk.

```
iv. Describe how the _target.contrast_ and _target.frames_ variables differ compared
to the data from part 1 of this assignment.
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.1     0.1     0.1     0.1     0.1     0.1
```

In experiment 2 (part 1 of this assignment) **target.frames** = 3 in all trials. In experiment 1 (part 2 of this assignment) **target.frame** = (1:6) across trials.

In experiment 2 (part 1 of this assignment) target.contrast: Min. 1st Qu. Median Mean 3rd Qu. Max. 0.01000 0.05683 0.06329 0.08771 0.09392 1.00000

In experiment 1 (part 2 of this assignment) target.contrast: Min. 1st Qu. Median Mean 3rd Qu. Max. 0.1 0.1 0.1 0.1 0.1 0.1

EXERCISE 5 - Use log-likelihood ratio tests to evaluate logistic regression models

1. Do logistic regression - *correct* as the dependent variable and *target.frames* as the independent variable. (Make sure that you understand what *target.frames* encode). Create two models - a pooled model and a partial-pooling model. The partial-pooling model should include a subject-specific intercept.

```
m5.1.1 <- glmer(right_answer ~ target.frames + (1|subject)+ (1|trial), data = df_exp
1, family = binomial(link = "logit"))

m5.1.2 <- glm(right_answer ~ target.frames, data = df_exp1, family = binomial(link =
"logit"))
```

```
i. the likelihood-function for logistic regression is: $L(p)={\displaystyle\prod_{i=
1}^Np^{y_i}(1-p)^{(1-y_i)}}$ (Remember the probability mass function for the Bernoull
i Distribution). Create a function that calculates the likelihood.
```

```
likelihood_function <- function(model, y){
  p = fitted.values(model)
  temp_vec = p^y*(1-p)^(1-y)
  return(prod(temp_vec))
}
```

```
ii. the log-likelihood-function for logistic regression is: $l(p) = {\displaystyle\su
m_{i=1}^N}[y_i\ln{p}+(1-y_i)\ln{(1-p)}$. Create a function that calculates the log-li
kelihood
```

```
log_likelihood_function <- function(model, y){
  p = fitted.values(model)
  temp_vec = y * log(p) + (1-y) * log(1-p)
  return(sum(temp_vec))
}
```

```
iii. apply both functions to the pooling model you just created. Make sure that the l
og-likelihood matches what is returned from the _logLik_ function for the pooled mode
l. Does the likelihood-function return a value that is surprising? Why is the log-lik
elihood preferable when working with computers with limited precision?
```

```
likelihood_function(m5.1.2, df_exp1$right_answer)
```

```
## [1] 0
```

```
log_likelihood_function(m5.1.2, df_exp1$right_answer)
```

```
## [1] -10865.25
```

I wouldn't say it is surprising that the likelihood function returns 0. If one iteration of i in the product chain $L(p) = \prod_{i=1}^{N} p^{y_i}(1-p)^{(1-y_i)}$ would be 0 the final product would result in 0.

The argument for preferring log-likelihood is much similair to the argument of why likelihood returns 0. Taking the log not only simplifies the subsequent mathematical analysis, but it also helps numerically because the product of a large number of small probabilities can easily be lesser than the numerical precision of the computer, and this is resolved by computing instead the sum of the log probabilities.

Log-likelihood is also preferential in regards of machine learning and optimization. It is less computational expensive and complicated to maximize a sum compared to a product. Furthermore, we can make use of ln(ab)=ln(a)+ln(b). Now we can maximize simply by taking derivatives and setting equal to 0.

```
iv. now show that the log-likelihood is a little off when applied to the partial pool
ing model - (the likelihood function is different for the multilevel function - see s
ection 2.1 of https://www.researchgate.net/profile/Douglas-Bates/publication/2753537_
Computational_Methods_for_Multilevel_Modelling/links/00b4953b4108d73427000000/Computa
tional-Methods-for-Multilevel-Modelling.pdf if you are interested)
```

```
likelihood_function(m5.1.1, df_exp1$right_answer)
```

```
## [1] 0
```

```
log_likelihood_function(m5.1.1, df_exp1$right_answer)
```

```
## [1] -10563.5
```

```
logLik(m5.1.1)
```

```
## 'log Lik.' -10622.03 (df=4)
```

You can see the difference between -10563.5 (our function estimate) and another build in function logLik = -10622.03.

2. Use log-likelihood ratio tests to argue for the addition of predictor variables, start from the null model, `glm(correct ~ 1, 'binomial', data)`, then add subject-level intercepts, then add a group-level effect of *target.frames* and finally add subject-level slopes for *target.frames*. Also assess whether or not a correlation between the subject-level slopes and the subject-level intercepts should be included.

```
m5.2.1 <- glm(right_answer ~ 1, family = binomial(link = "logit"), data = df_exp1)
m5.2.2 <- glm(right_answer ~ target.frames , family = binomial(link= "logit"), data =
df_exp1)
m5.2.3 <- glmer(right_answer ~ target.frames + (1|subject), family = binomial(link =
"logit"), data = df_exp1)

m5.2.4 <- glmer(right_answer ~ target.frames  + (1+ target.frames|subject), family =
 binomial(link = "logit"), data = df_exp1)
summary(m5.2.4)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: right_answer ~ target.frames + (1 + target.frames | subject)
##    Data: df_exp1
##
##      AIC      BIC   logLik deviance df.resid
##  20907.7  20948.3 -10448.8  20897.7    25039
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -16.1429   0.0812   0.2454   0.4868   1.2632
##
## Random effects:
##  Groups  Name          Variance Std.Dev. Corr
##  subject (Intercept)   0.06086  0.2467
##          target.frames 0.05117  0.2262   -0.87
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept)   -1.09010    0.05878  -18.55   <2e-16 ***
## target.frames  0.83317    0.04433   18.80   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr)
## target.frms -0.817
```

Following the guideline of the lme4 creators approach to selecting correlated or uncorrelated random effects in your model. https://cran.r-project.org/web/packages/lme4/vignettes/lmer.pdf (https://cran.r-project.org/web/packages/lme4/vignettes/lmer.pdf) (page. 7) The correlation

between target.frames and intercept is fairly high (-0.817) therefore this needs to be specified in our model by specifying (x|y). This isn't assessed by log-likelihood but rather correlation test between slope estimates and intercept.

```
anova( m5.2.4, m5.2.3, m5.2.1, m5.2.3)
```

	n...	AIC	BIC	logLik	deviance	Chisq	Df
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
MODEL3	1	26685.08	26693.21	-13341.54	26683.08	*NA*	*NA*
MODEL2	3	21250.06	21274.45	-10622.03	21244.06	5439.0149	2
MODEL4	3	21250.06	21274.45	-10622.03	21244.06	0.0000	0
MODEL1	5	20907.65	20948.29	-10448.83	20897.65	346.4127	2

4 rows

A model calculating the log-likelihoood ratio test. Since all our models are nested I've conditioned the order of variables following log-likelihood. If models weren't nested I would need to order by the amount of columns in the design matrix (X+Z).

```
log_like_ratio_test <- function(model1, model2){
  ratio = ifelse( logLik(model2) <= logLik(model1) , -2*(logLik(model2) - logLik(mode
l1)) , (-2*(logLik(model1) - logLik(model2)))) #log-like-ratio

  df_val = abs(df.residual(model1) - df.residual(model2)) #calculate the degree of fr
eedom
  p_val = pchisq(ratio, df = df_val, lower.tail = FALSE) #chi-square test
  name = deparse(substitute(c(model1, model2)))
  return(c(model_comp = name, log_lik_ratio = ratio, p_value = p_val))
}


log_like_ratio_test(m5.2.4, m5.2.1)
```

```
##          model_comp       log_lik_ratio             p_value
## "c(m5.2.4, m5.2.1)"  "5785.42764596847"                 "0"
```

```
log_like_ratio_test(m5.2.4, m5.2.2)
```

```
##              model_comp           log_lik_ratio                 p_value
##     "c(m5.2.4, m5.2.2)"      "832.846524015516" "3.25400522669055e-180"
```

```
log_like_ratio_test(m5.2.4, m5.2.3)
```

```
##             model_comp          log_lik_ratio                 p_value
##    "c(m5.2.4, m5.2.3)"     "346.412703775855" "5.99014236897163e-76"
```

```
#With package
pacman::p_load(lmtest)
lrtest(m5.2.3 , m5.2.4)
```

	#Df	LogLik	Df	Chisq	Pr(>Chisq)
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	3	-10622.03	*NA*	*NA*	*NA*
2	5	-10448.83	2	346.4127	5.990142e-76

2 rows

As shown, my functions output is similar to that of the packages lmtest::lrtest().

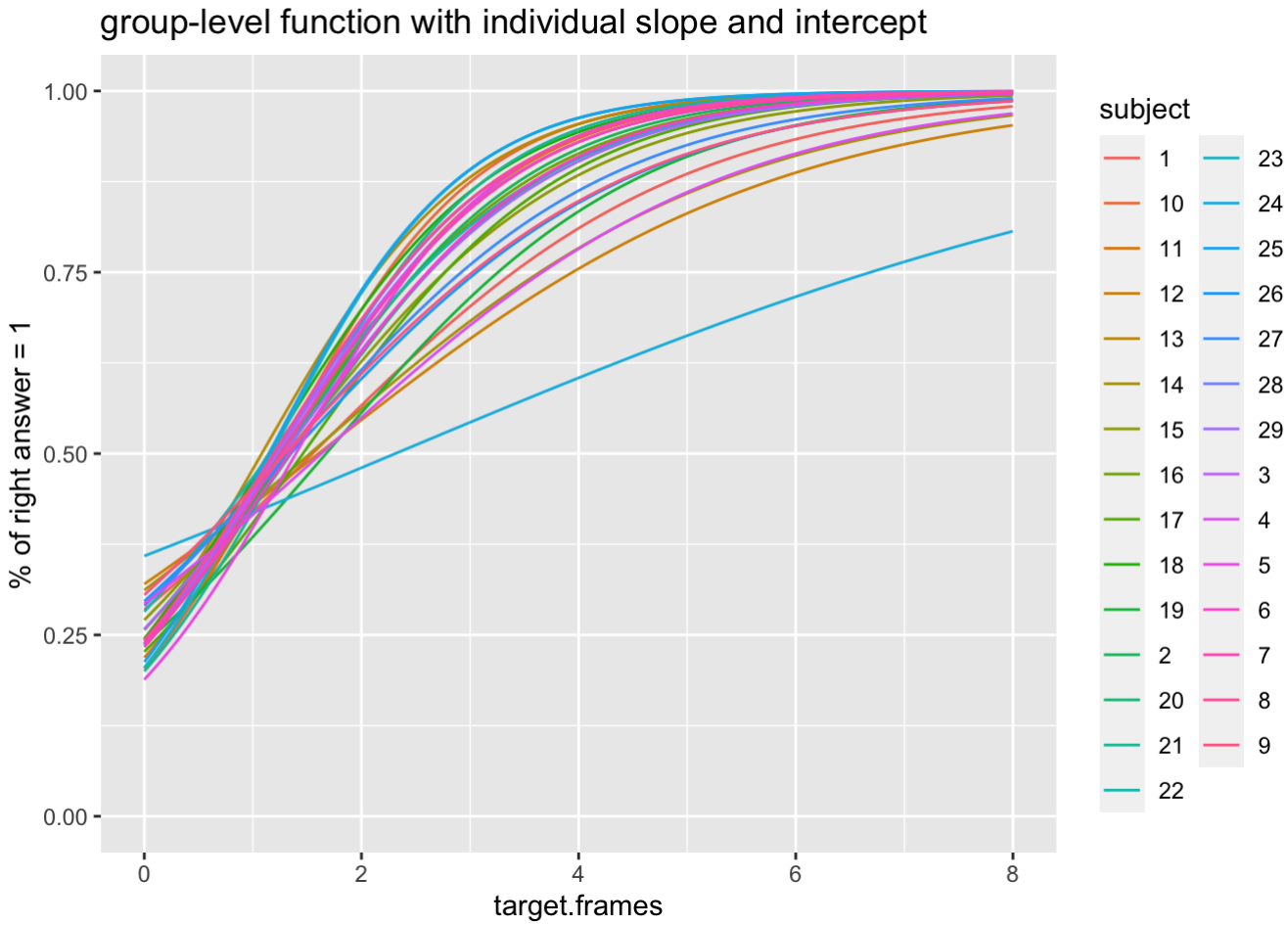
```
i. write a short methods section and a results section where you indicate which model
you chose and the statistics relevant for that choice. Include a plot of the estimate
d group-level function with `xlim=c(0, 8)` that includes the estimated subject-specif
ic functions.
```

logLik: We performed a log-likelihood ratio test on our nested models with the most complex model as reference level. With a significance level of 0.05 we will reject the null-hypothesis on all comparisons. This supports the choice of selecting the most complex ie. model4.

AIC: While log-likelihood ratio does account for model complexity by chi-square distribution varying depending on Df. Aikai-Information-Criteria (AIC) has shown to be a better measure for penalizing model complexity. the single level intercept model has the lowest AIC score and will therefore be the preferred model following AIC.

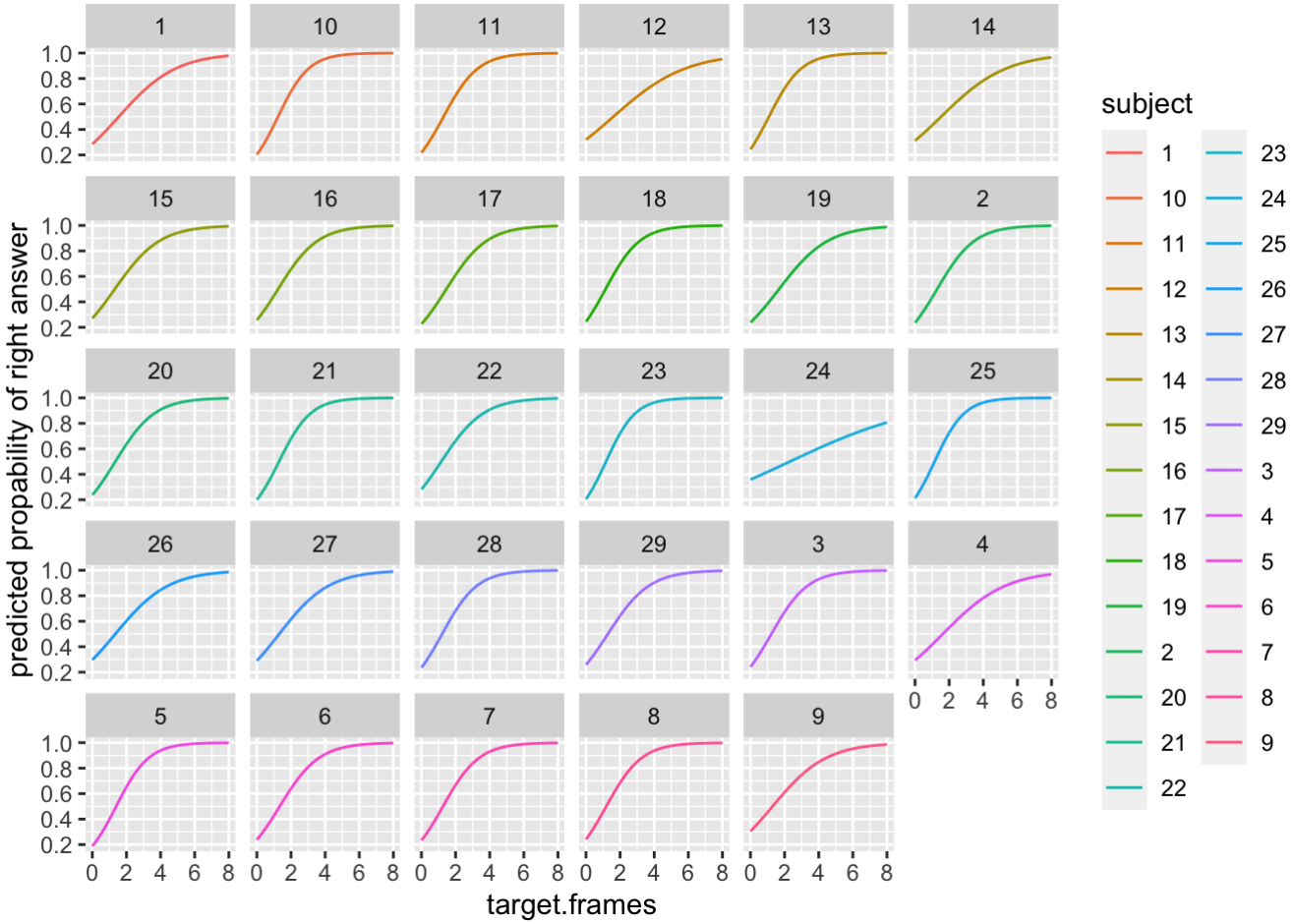
```
newdata1 <- data.frame(target.frames = rep(seq(0,8,0.01),29) , subject = as.character
(rep(seq(1:29),801))) %>%
  mutate(subject = as.factor(subject))
#predict values for each subject target.frames 0:8
newdata1$yhat <- predict(m5.2.4, newdata = newdata1, type = "response")
```

```
ggplot(newdata1, aes(x = target.frames, y = yhat, color = subject)) + geom_line() +
  ylim(0,1) + labs(title = "group-level function with individual slope and intercept"
, y = "% of right answer = 1")
```



While this plot is well suited for looking at the variance between subjects slope and intercept it is difficult to distinguish subjects from one another. If we facet by subject it will allow individual inspection.

```
ggplot(newdata1, aes(x = target.frames, y = yhat)) + geom_line(aes(colour = subject)) + labs(y = "predicted propability of right answer") +  facet_wrap(~subject) + xlim(0, 8)
```



While graphs are nice numbers can also be very informative in illustrating the spread in our beta parameters b0 and b1.

```
#get coef
coef_m5.2.4 <- coef(m5.2.4)
slope <- invlogit(coef_m5.2.4$subject[,2])
intercept <- invlogit(coef_m5.2.4$subject[,1])
cbind(intercept = summary(intercept), slope = summary(slope))
```

```
##         intercept     slope
## Min.    0.1885187 0.5624312
## 1st Qu. 0.2332623 0.6643165
## Median  0.2419339 0.7031253
## Mean    0.2543926 0.6944146
## 3rd Qu. 0.2850686 0.7278806
## Max.    0.3588732 0.7600487
```

Supported by the plots and statistical summary we can can conclude that there isn't that much spread in the individual variance at subject level.

```
ii. also include in the results section whether the fit didn't look good for any of t
he subjects. If so, identify those subjects in the report, and judge (no statistical
test) whether their performance (accuracy) differed from that of the other subjects.
Was their performance better than chance? (Use a statistical test this time) (50 %)
```

```
pacman::p_load(caret)
```

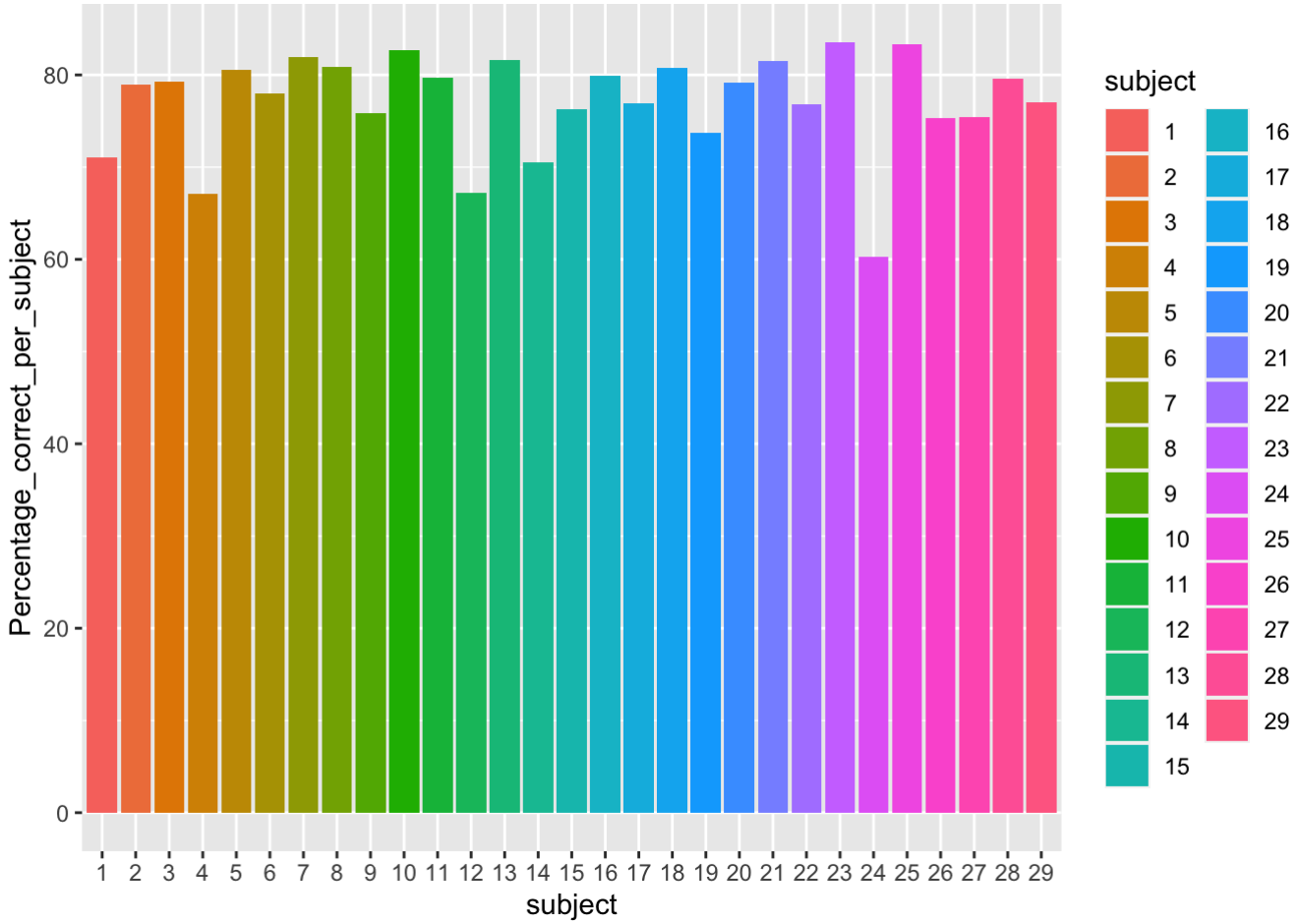
Generating an idea of accuracy in the entire data-set.

```
df_exp1<- df_exp1 %>%
  mutate(fitted_m.5.2.4 = fitted(m5.2.4)) %>%
  mutate(bin_fit_m5.2.4 = if_else(fitted_m.5.2.4 >= 0.5, 1, 0))

confusionMatrix(data = as.factor(df_exp1$bin_fit_m5.2.4), reference = as.factor(df_ex
p1$right_answer))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0  2101  2219
##          1  3525 17199
##
##                Accuracy : 0.7706
##                  95% CI : (0.7654, 0.7758)
##     No Information Rate : 0.7754
##     P-Value [Acc > NIR] : 0.9634
##
##                   Kappa : 0.2825
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.37344
##             Specificity : 0.88572
##          Pos Pred Value : 0.48634
##          Neg Pred Value : 0.82991
##              Prevalence : 0.22464
##          Detection Rate : 0.08389
##    Detection Prevalence : 0.17250
##       Balanced Accuracy : 0.62958
##
##        'Positive' Class : 0
##
```

We illustrate differences in subject accuracy

```
df_exp_temp <- df_exp1 %>%
  group_by(subject) %>%
  summarise(Percentage_correct_per_subject = sum(bin_fit_m5.2.4 == right_answer)/leng
th(right_answer)*100)

ggplot(df_exp_temp, aes(x = subject, y = Percentage_correct_per_subject, fill = subje
ct)) + geom_col()
```



Based on a visual inspection of predicted correct answers we can see that subject 24 has the worst predicted accuracy score. We will therefore perform as one-sample test to see if he performs better than chance.

```
df_exp1_fil24 <- df_exp1 %>%
  filter(subject == 24)

#t-test on collected data
t.test(as.numeric(df_exp1_fil24$right_answer), mu = 0.5, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  as.numeric(df_exp1_fil24$right_answer)
## t = 4.026, df = 873, p-value = 3.083e-05
## alternative hypothesis: true mean is greater than 0.5
## 95 percent confidence interval:
##  0.5398963       Inf
## sample estimates:
## mean of x
## 0.5675057
```

```
#t-test on predicted values.
t.test(as.numeric(df_exp1_fil24$bin_fit_m5.2.4), mu = 0.5, alternative = "greater")
```

```
##
##  One Sample t-test
##
## data:  as.numeric(df_exp1_fil24$bin_fit_m5.2.4)
## t = 10.473, df = 873, p-value < 2.2e-16
## alternative hypothesis: true mean is greater than 0.5
## 95 percent confidence interval:
##  0.6407847       Inf
## sample estimates:
## mean of x
## 0.6670481
```

T.test Summary

collected data The accuracy for subject 23 were above chance (50%), (t(863) = 26.944, p = 2.2e^16)

predicted data The predicted accuracy for subject 23 were above chance (50%), (t(863) = 26.275, p = 2.2e^16)

3. Now add *pas* to the group-level effects - if a log-likelihood ratio test justifies this, also add the interaction between *pas* and *target.frames* and check whether a log-likelihood ratio test justifies this

    i. if your model doesn't converge, try a different optimizer

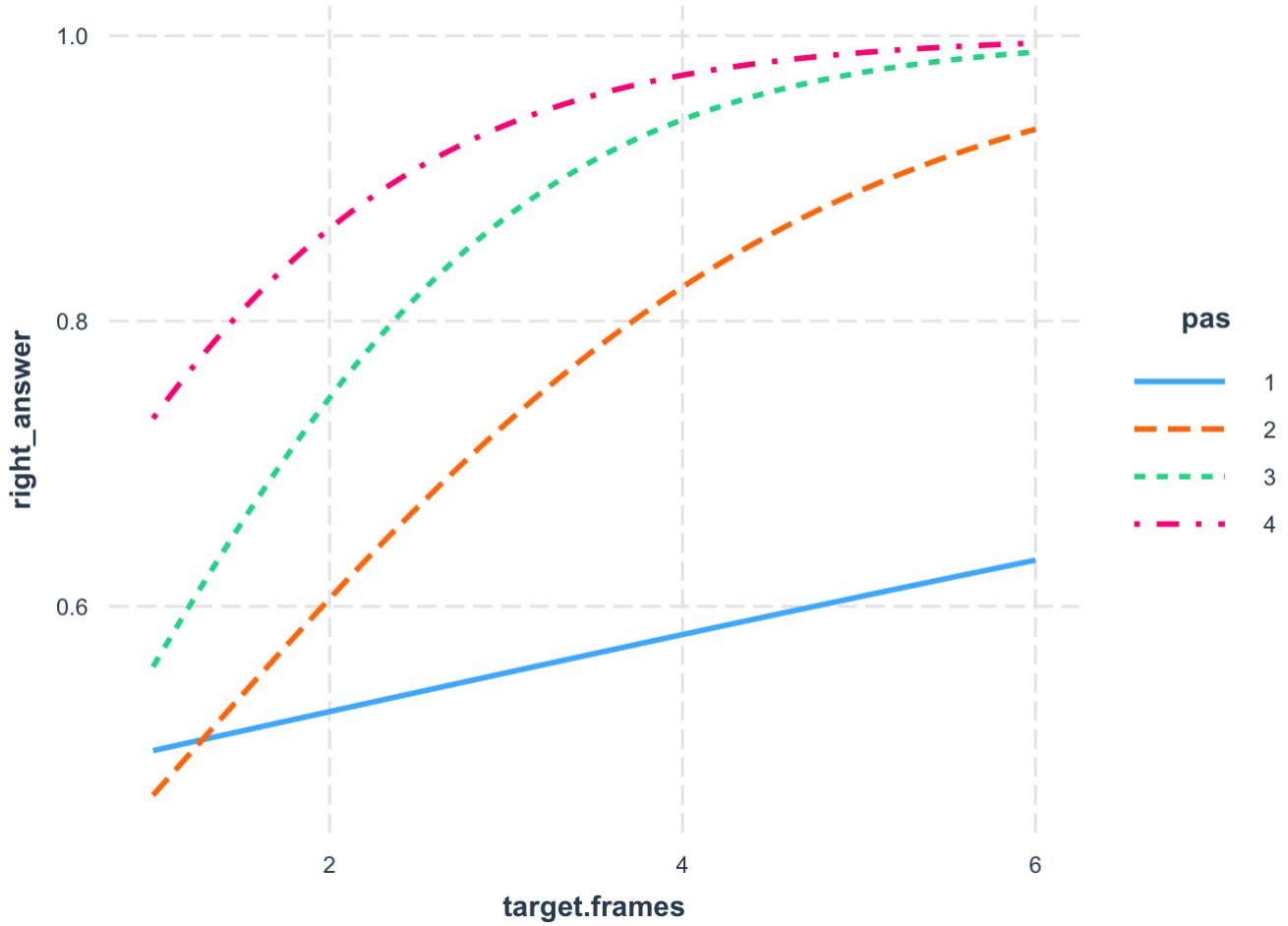
```
m5.3.1 <- glmer(right_answer ~ pas*target.frames + (target.frames||subject), data = d
f_exp1, family = binomial(link = "logit"), control = glmerControl(optimizer="bobyqa"
))



log_like_ratio_test(m5.3.1 , m5.2.4)
```

```
##              model_comp           log_lik_ratio                 p_value
##     "c(m5.3.1, m5.2.4)"      "1405.89148846487" "7.28140374467874e-302"
```

Our log-likelihood-ratio test support the choice of the more complex model. (m5.3.1) including the interaction between pas:target.frames and pas as a fixed effect.

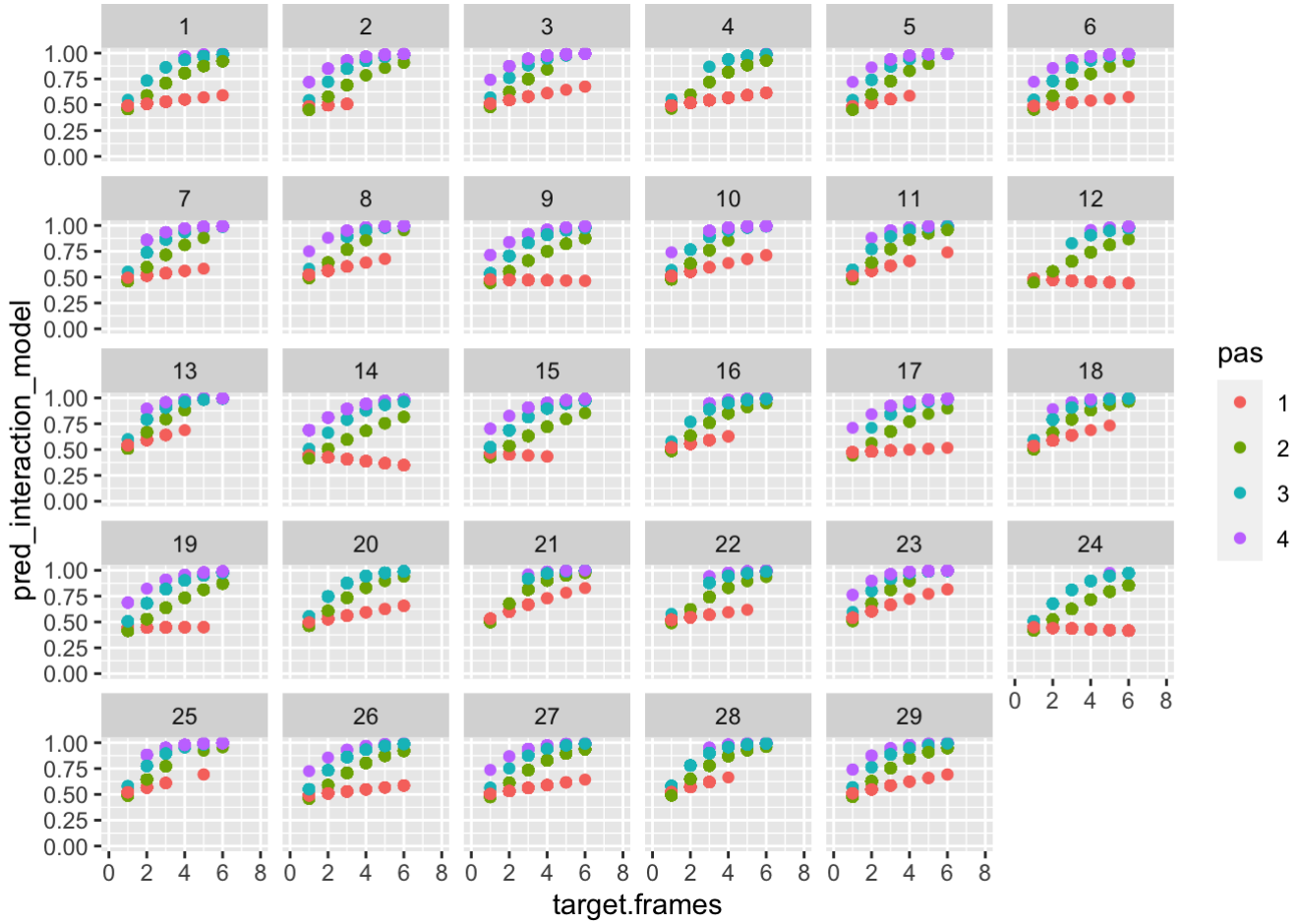
```
ii. plot the estimated group-level functions over `xlim=c(0, 8)` for each of the four
PAS-ratings - add this plot to your report (see: 5.2.i) and add a description of your
chosen model. Describe how _pas_ affects accuracy together with target duration if at
all. Also comment on the estimated functions' behaviour at target.frame=0 - is that b
ehaviour reasonable?
```

```
df_exp1 <- df_exp1 %>%
  mutate(pred_interaction_model = fitted.values(m5.3.1))


interactions::interact_plot(model = m5.3.1 , pred = "target.frames", modx = "pas")
```



While this plot is nice for showing the general interaction between pas:target.frames it does not account for our random intercept and random slop of target.frames. We will therefore continue with another plot that can illustrate these individual differences.

```
ggplot(df_exp1, aes(x = target.frames, y = pred_interaction_model, col = pas )) + geo
m_point() + facet_wrap(~subject) + xlim(0,8) + ylim(0,1)
```



As shown target.frames has individual # EXERCISE 6 - Test linear hypotheses

In this section we are going to test different hypotheses. We assume that we have already proved that more objective evidence (longer duration of stimuli) is sufficient to increase accuracy in and of itself and that more subjective evidence (higher PAS ratings) is also sufficient to increase accuracy in and of itself.
We want to test a hypothesis for each of the three neighbouring differences in PAS, i.e. the difference between 2 and 1, the difference between 3 and 2 and the difference between 4 and 3. More specifically, we want to test the hypothesis that accuracy increases faster with objective evidence if subjective evidence is higher at the same time, i.e. we want to test for an interaction.

1. Fit a model based on the following formula:

   `correct ~ pas * target.frames + (target.frames | subject))`

   i. First, use `summary` (yes, you are allowed to!) to argue that accuracy increases faster with objective evidence for PAS 2 than for PAS 1.

```
coef_OG_scale <-  data.frame(intercept = rep(NA,29), pas2 = rep(NA,29), pas3 = rep(NA
,29), pas4 = rep(NA,29), target.frames = rep(NA,29), pas2_target.frames = rep(NA,29),
pas3_target.frames = rep(NA,29), pas4_target.frames = rep(NA,29))

for (i in 1:length(coef(m5.3.1)$subject)){
  coef_OG_scale[,i] = invlogit(coef(m5.3.1)$subject[,i])
}
```

All beta values were transformed from log-odds to probabilities through an inverse logit transformation. All beta values will be reported on the probability scale for easier interpretation. ’

Intercepts Pas is coded as a factor and we will therefore gain an intercept for all 4 levels of pas. *general baseline intercept* $_0 (intercept/pas1) = 0.4714, z = -2.064, p = 0.0390 $ *Individual baseline intercept per subject* $\beta_0(intercept/pas1) : Median = 0.47, Mean = 0.4716, sd = 0.011$ Every subject has their own baseline/intercept probabillity of giving the right answer when target.frames = 0 and pas = pas1.

$\beta_1(intercept/pas2) = 0.3604, z = -6.471, p = 9.77e-11$ added on top of of their individual baseline for pas1
$\beta_2(intercept/pas3) = 0.3771, z = -3.653, p = 0.000259 * **$ added on top of of their individual baseline for pas1
$\beta_3(intercept/pas4) = 0.5658, z = 1.065, p = 0.286944$ added on top of of their individual baseline for pas1

Slopes We’ve modeled an interaction between a categorical variable(pas1-4) and a numeric (target.frames) with a second level effect of random slope We will therefore have a individual slope of target.frame for each subject. **main effect** $\beta_4(target.frames) = Median = 0.5282, Mean = 0.5269, sd = 0.0234$ **interaction effect** $_5 (target.frames:pas2) = 0.6102, z = 13.006, p < 2e-16 $ $_6 (target.frames:pas3) = 0.6764, z = 16.336, p < 2e-16 $ $_7 (target.frames:pas4) = 0.6776, z = 10.990, p < 2e-16 $

Currently both genereal and individual intercept are below chance (50%). An explanation can be found in the interpritation. Currently our intercept represent target.frames = 0 and pas = pas1. Having been presented with 0 frames does not make much sense in terms of the experiment. But we would expect a 50/50 chance to guess right if you had no information to guess by. The best solution would be to standardize the data around target.type. Giving us a more intuitive interpritation.

Our model supports our assumption of an increase in subjective (pas-score) and objective (target.frames) evidence significantly increases the probability of the right answer.

Furthermore, as shown by our interaction effects increasing subjective evidence (PAS1 -> PAS2/3/4) will significantly modulate objective evidence to increase the accuracy score even faster compared to PAS1.

2. `summary` won't allow you to test whether accuracy increases faster with objective evidence for PAS 3 than for PAS 2 (unless you use `relevel`, which you are not allowed to in this exercise). Instead, we'll be using the function `glht` from the `multcomp` package
    i. To redo the test in 6.1.i, you can create a *contrast* vector. This vector will have the length of the number of estimated group-level effects and any specific contrast you can think of can be specified using this. For redoing the test from 6.1.i, the code snippet below will do

```
design <- model.matrix(m5.3.1)
head(design)
```

```
##   (Intercept) pas2 pas3 pas4 target.frames pas2:target.frames
## 1           1    0    0    0             3                   0
## 2           1    0    0    0             2                   0
## 3           1    0    0    0             1                   0
## 4           1    0    1    0             5                   0
## 5           1    0    1    0             6                   0
## 6           1    0    0    1             4                   0
##   pas3:target.frames pas4:target.frames
## 1                  0                  0
## 2                  0                  0
## 3                  0                  0
## 4                  5                  0
## 5                  6                  0
## 6                  0                  4
```

```
pacman::p_load(multcomp)
contrast.vectors <- matrix(c(0, 0, 0, 0, -1, 1, 0, 0), nrow=1)
ghs <- glht(m5.3.1, contrast.vectors)
print(summary(ghs))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = right_answer ~ pas * target.frames + (target.frames ||
##     subject), data = df_exp1, family = binomial(link = "logit"),
##     control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.33877    0.05723   5.919 3.24e-09 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
ii. Now test the hypothesis that accuracy increases faster with objective evidence fo
r PAS 3 than for PAS 2.
```

```
#Contrast matrix


contrast_matrix_compar <- glht(m5.3.1, linfct = c( "pas2:target.frames = -1", "pas3:t
arget.frames = 1"))
summary(contrast_matrix_compar) #not a 100% how this works.
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = right_answer ~ pas * target.frames + (target.frames ||
##     subject), data = df_exp1, family = binomial(link = "logit"),
##     control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##                           Estimate Std. Error z value Pr(>|z|)
## pas2:target.frames == -1  0.44820    0.03442   42.07  < 1e-10 ***
## pas3:target.frames == 1   0.73767    0.04507   -5.82 1.18e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
#Difference between pas2:target.frames pas3:target.frames
contrast.vector <- matrix(c(0, 0, 0, 0, 0, -1, 1, 0), nrow=1)
gh <- glht(m5.3.1, contrast.vector)
print(summary(gh))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = right_answer ~ pas * target.frames + (target.frames ||
##     subject), data = df_exp1, family = binomial(link = "logit"),
##     control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.28947    0.04578   6.323 2.56e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

objective evidence is also modulated more by PAS3 than PAS2 resulting in a faster growing accuracy by target.frames in PAS3 and than PAS2.

```
iii. Also test the hypothesis that accuracy increases faster with objective evidence
for PAS 4 than for PAS 3
```

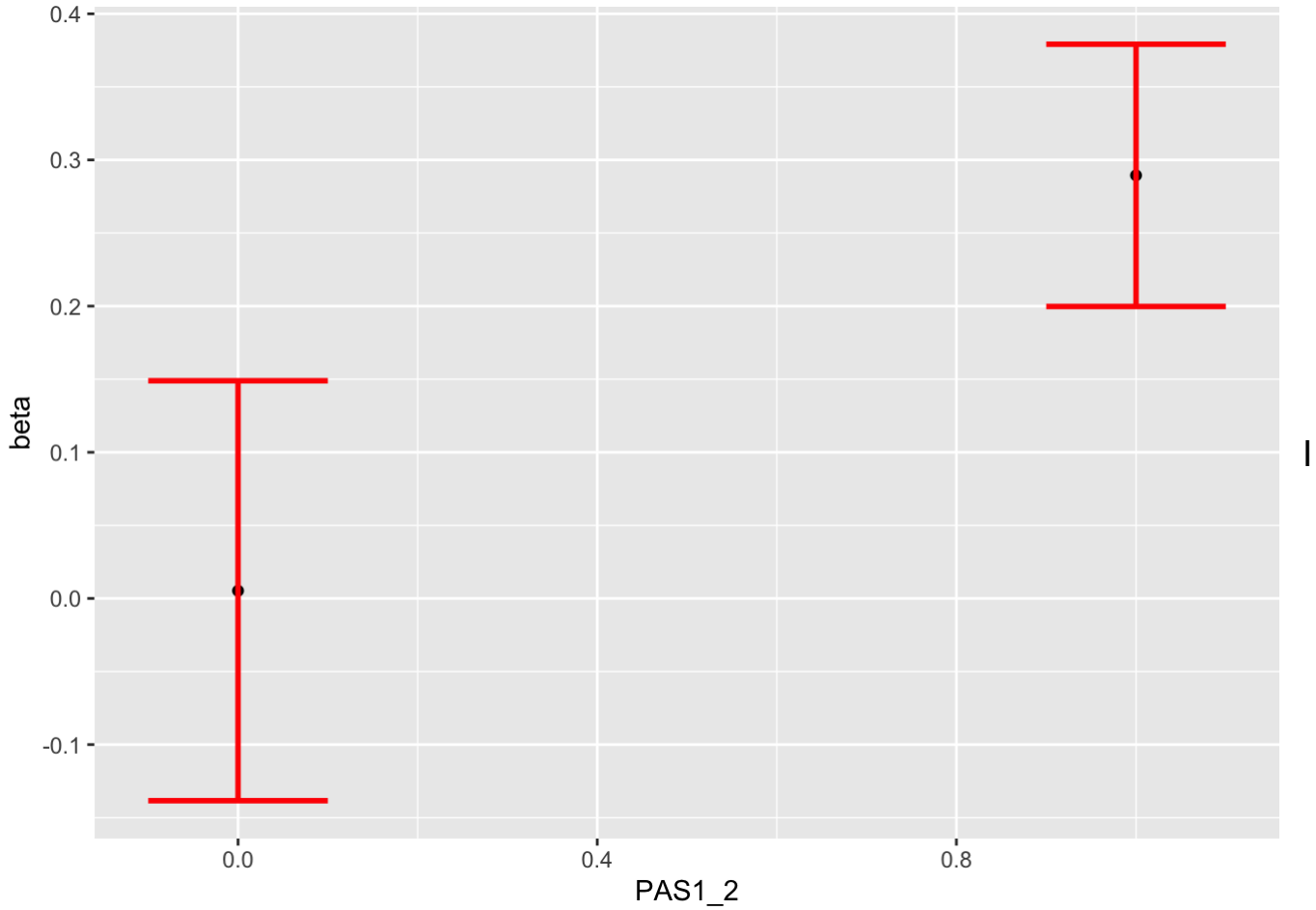
```
#Difference between pas3:target.frames pas4:target.frames
contrast.vector2 <- matrix(c(0, 0, 0, 0, 0, 0, -1, 1), nrow=1)
gh2 <- glht(m5.3.1, contrast.vector2)
print(summary(gh2))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = right_answer ~ pas * target.frames + (target.frames ||
##     subject), data = df_exp1, family = binomial(link = "logit"),
##     control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0 0.005259   0.073293   0.072    0.943
## (Adjusted p values reported -- single-step method)
```

Moving from PAS3-PAS4 does not increase the effect of target.frames significantly.

3. Finally, test that whether the difference between PAS 2 and 1 (tested in 6.1.i) is greater than the difference between PAS 4 and 3 (tested in 6.2.iii)

```
# beta = 0.44821   std.error =  0.03446 pas1-pas2
# beta = 0.005258   std.error = 0.073618 pas3-pas4
confint_gh<- confint(gh)
confint_gh2 <- confint(gh2)

df_confint <- data.frame(beta = c(confint_gh$confint[1], confint_gh2$confint[1]),
           upper = c(confint_gh$confint[2], confint_gh2$confint[2]),
           lower = c(confint_gh$confint[3], confint_gh2$confint[3]),
           PAS1_2 = c(1,0),
           PAS3_4 = c(0,1))

ggplot(df_confint, aes(x = PAS1_2, y = beta)) + geom_point() + geom_errorbar(aes(x =
 PAS1_2, ymin = lower, ymax = upper), width = 0.2, color = "red", size = 1)
```



can not do a t-test as calculating SE_pooled requires the SD.
$SE_p = SD_p * sqrt(1/n_1 + 1/n_2)$. We can calculate the 95% confidence intervals based on SE x 1.96 for lower and high boundaries. Our Confidence intervals does not overlap which would indicate that they are significantly different. THe difference of **PAS1-2** is larger than **PAS3-4**.

EXERCISE 7 - Estimate psychometric functions for the Perceptual Awareness Scale and evaluate them

We saw in 5.3 that the estimated functions went below chance at a target duration of 0 frames (0 ms). This does not seem reasonable, so we will be trying a different approach for fitting here.

We will fit the following function that results in a sigmoid, $f(x) = a + \frac{b-a}{1+e^{\frac{c-x}{d}}}$

It has four parameters: *a*, which can be interpreted as the minimum accuracy level, *b*, which can be interpreted as the maximum accuracy level, *c*, which can be interpreted as the so-called inflexion point, i.e. where the derivative of the sigmoid reaches its maximum and *d*, which can be interpreted as the steepness at the inflexion point. (When *d* goes towards infinity, the slope goes towards a straight line, and when it goes towards 0, the slope goes towards a step function).

We can define a function of a residual sum of squares as below

```
RSS <- function(data, par)
{
    ## "dataset" should be a data.frame containing the variables x (target.frames)
    ## and y (correct)

    ## "par" are our four parameters (a numeric vector)
    ## par[1]=a, par[2]=b, par[3]=c, par[4]=d
    a = par[1]
    b = par[2]
    c = par[3]
    d = par[4]
    x <- data$x
    y <- data$y
    y.hat <- a + ((b-a)/(1+(exp((c-x)/d))))
    RSS <- sum((y - y.hat)^2)
    return(RSS)
}
```

1. Now, we will fit the sigmoid for the four PAS ratings for Subject 7
    i. use the function `optim`. It returns a list that among other things contains the four estimated parameters. You should set the following arguments:
    `par`: you can set *c* and *d* as 1. Find good choices for *a* and *b* yourself (and argue why they are appropriate)
    `fn`: which function to minimise?
    `data`: the data frame with *x*, *target.frames*, and *y*, *correct* in it
    `method`: 'L-BFGS-B'
    `lower`: lower bounds for the four parameters, (the lowest value they can take), you can set *c* and *d* as `-Inf`. Find good choices for *a* and *b* yourself (and argue why they are appropriate)
    `upper`: upper bounds for the four parameters, (the highest value they can take) can set *c* and *d* as `Inf`. Find good choices for *a* and *b* yourself (and argue why they are appropriate)

```
#Optim function with target.frames
data <- df_exp1 %>%
  dplyr::select(target.frames, right_answer) %>%
  rename(x = target.frames, y = right_answer)


optim(par = c(0.5,1,1,1), fn = RSS, data = data, method = "L-BFGS-B", lower = c(0.5,
0.5,-Inf,-Inf),
      upper = c(1,1,Inf,Inf))
```

```
## $par
## [1] 0.5077512 0.9586143 2.9380622 0.4303382
##
## $value
## [1] 3500.344
##
## $counts
## function gradient
##       24       24
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
#Optim function with pas rating.
data1 <- df_exp1 %>%
  filter(subject == 7) %>% #only use subject 7
  dplyr::select(pas, right_answer) %>%
  mutate(pas = as.numeric(pas)) %>% #optim function requires it to be numeric.
  rename(x = pas, y = right_answer)


optim_pas <- optim(par = c(0.5,1,1,1), fn = RSS, data = data1, method = "L-BFGS-B", l
ower = c(0.5,0.5,-Inf,-Inf),
      upper = c(1,1,Inf,Inf))
optim_pas$par
```

```
## [1] 0.5000000 0.9945230 2.6256293 0.3288213
```

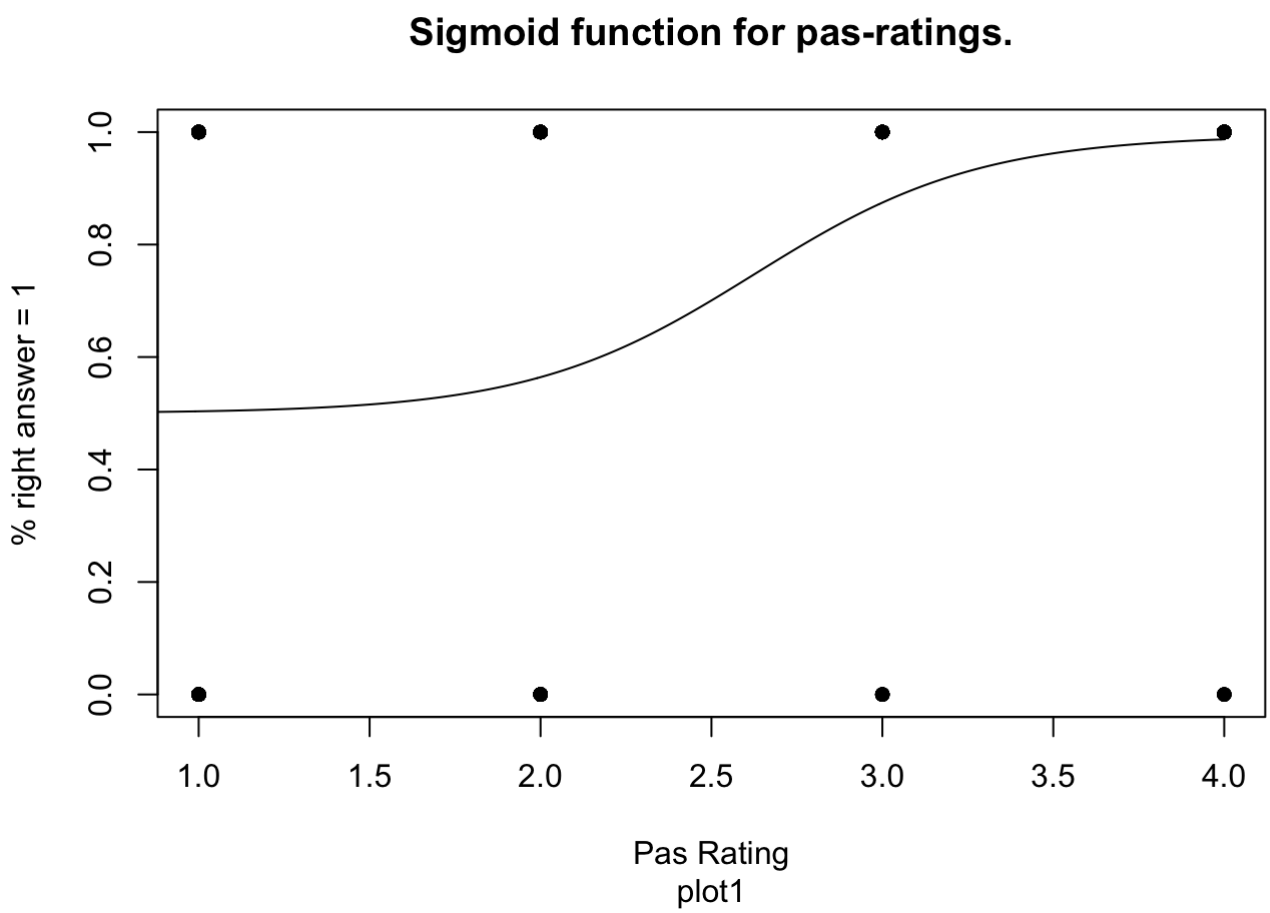
par = c(0.5, 1,1,1,) *a* will start at 0.5 with a min = 0.5 and max = 1. *b* will start at 1 with min = 0.5 and max = 1. The choice of min and max is based on the assignments wish to avoid values below chance. So max and min values of y can span from 0.5:1. It is imaginable that such values could be possible so therefore they have been chosen as the lower and upper boundraies.

Starting point is based on guesses of what the best scenario could be and worst scenario. ie. a = 0.5 and b = 1.

Use the estimated parameters to compute y_hat and plot the function.

```
sigmoid <- function(a,b, c,d,x){
  y = a + ((b-a)/(1+(exp((c-x)/d))))
  return(y)
}

x_pas_optim <- seq(0, 4, 0.01)
y_pas_optim <- sigmoid(optim_pas$par[1],optim_pas$par[2],
                       optim_pas$par[3],optim_pas$par[4],x_pas_optim)
```

```
plot(data1$x, data1$y, pch = 16, xlab = "Pas Rating", ylab = "% right answer = 1")
lines(x_pas_optim, y_pas_optim) + title(main = "Sigmoid function for pas-ratings.", s
ub = "plot1")
```




```
## integer(0)
```

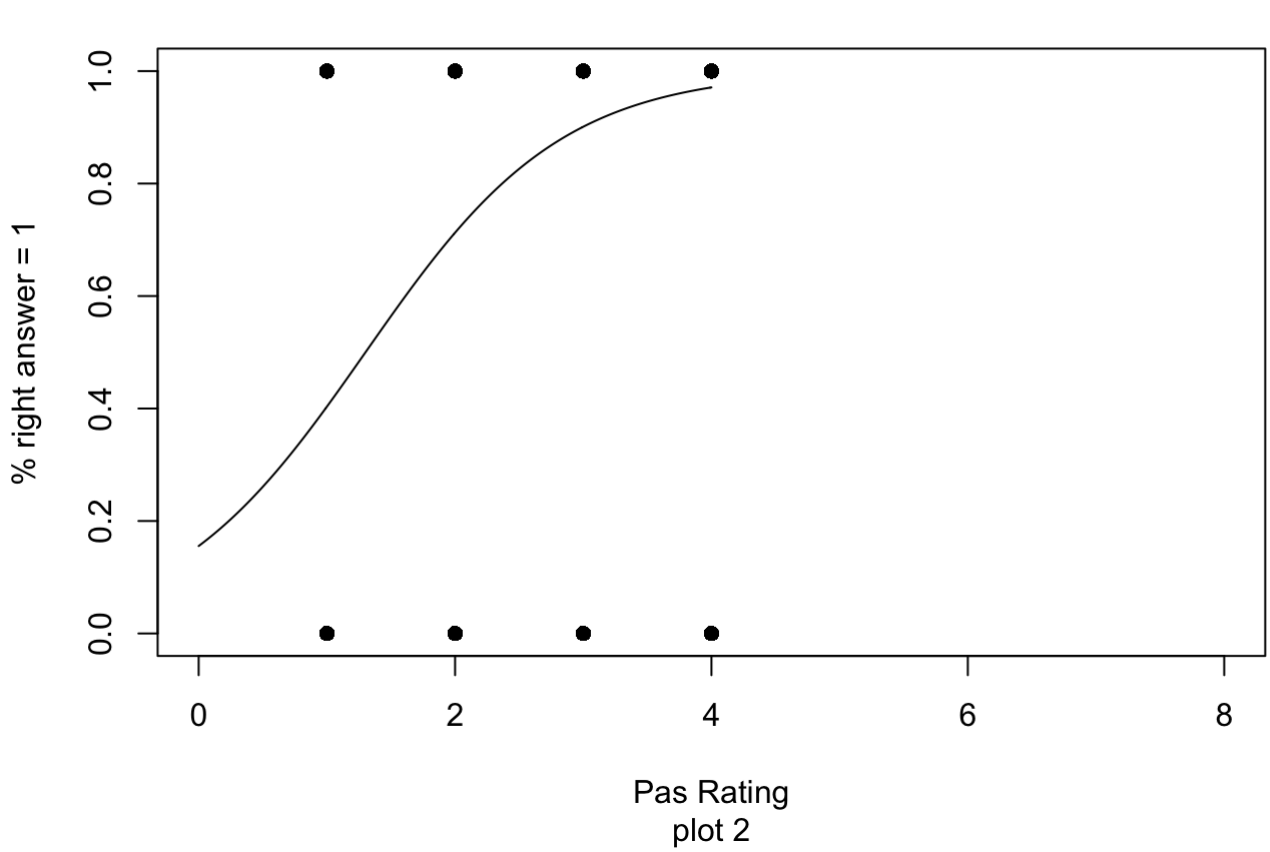
```
ii. Plot the fits for the PAS ratings on a single plot (for subject 7) `xlim=c(0, 8)`
```

```
#Simple model to do illustrate pas ratings for subject 7
m7.1.1 <- glm(right_answer ~ as.numeric(pas), data = filter(df_exp1, subject == 7), f
amily = binomial(link = "logit"))
x_pas <- seq(0, 4, 0.01)
y_pas <- predict(m7.1.1, list(pas = x_pas),type="response")
```

```
#plots for subject 7
plot(as.numeric(df_exp1$pas), df_exp1$right_answer, pch = 16, xlab = "Pas Rating", yl
ab = "% right answer = 1",
     xlim = c(0,8)) + title(main = "subject 7 glm function with no interaction", sub
 = "plot 2")
```

```
## integer(0)
```

```
lines(x_pas, y_pas)
```



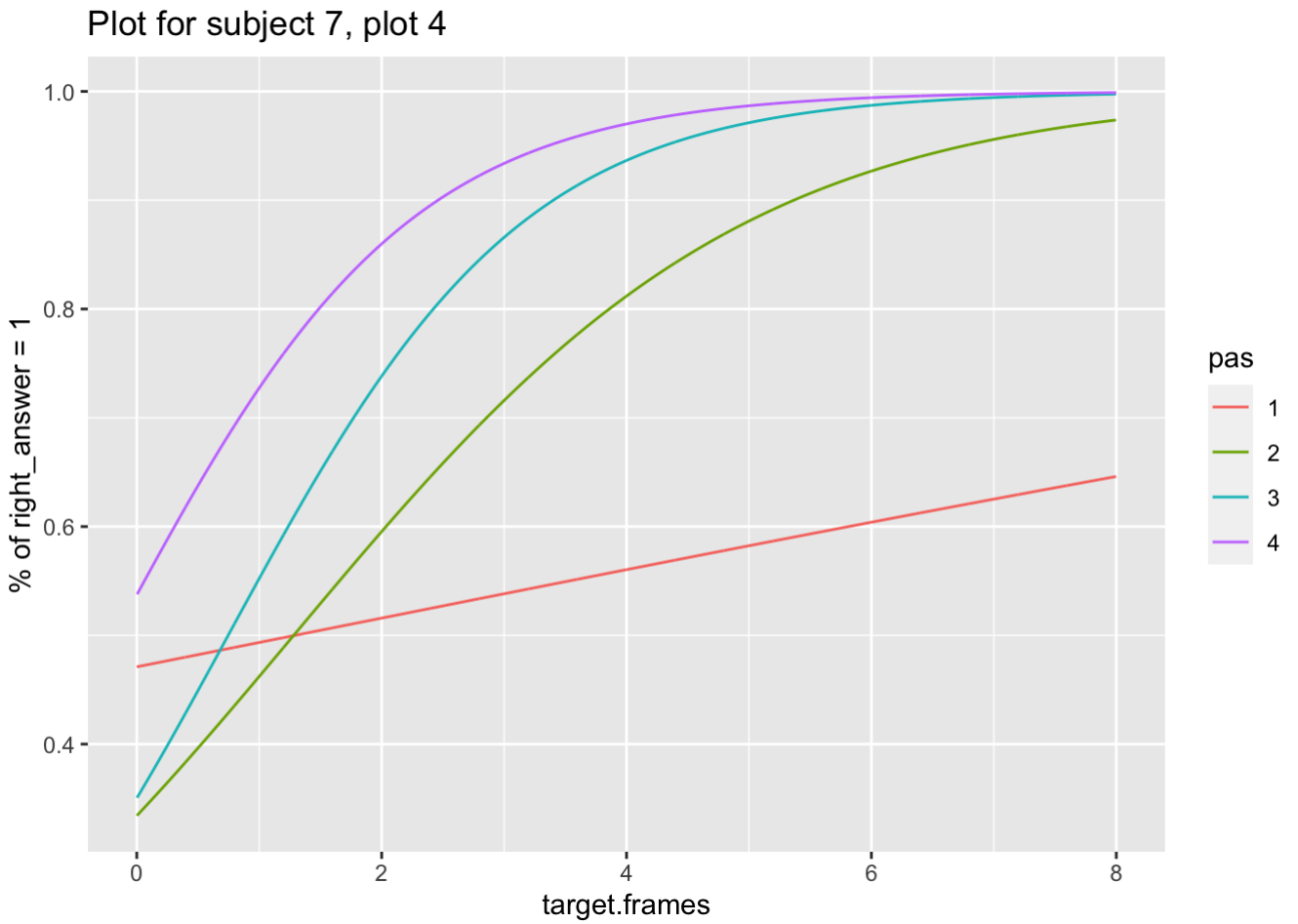
iii. Create a similar plot for the PAS ratings on a single plot (for subject 7), but this time based on the model from 6.1 `xlim=c(0, 8)`

```
newdat <- data.frame(cbind('target.frames' = seq(0, 8, by = 0.001), 'pas' = rep(1:4), 'subject' = rep('7')))
```

```
## Warning in cbind(target.frames = seq(0, 8, by = 0.001), pas = rep(1:4), : number
## of rows of result is not a multiple of vector length (arg 2)
```

```
newdat$subject <- as.factor(newdat$subject)
newdat$pas <- as.factor(newdat$pas)
newdat$target.frames <- as.numeric(newdat$target.frames)

newdat$yhat <- predict(m5.3.1, newdata = newdat, type = "response")
```

```
ggplot(newdat, aes(x = target.frames, y = yhat, color = pas)) + geom_line() + labs(title = "Plot for subject 7, plot 4", y = "% of right_answer = 1")
```




```
iv. Comment on the differences between the fits - mention some advantages and disadva
ntages of each way
```

Comparing plot1 (sigmoid) and plot2 (subject 7): Plot 1 shows that as PAS approaches 4 the probability of guessing right approaches 100%. This is also illustrated in **plot 3** but also adds the detail of how PAS4 interacts with target.frames. While higher level of target.frames along with PAS4 rating will result in the best probability of guessing right.

While **plot 1** gives us the best idea of how PAS rating effects probability of guessing correct. **plot 3** is more informative when it comes to illustrating the combination of target.frames and PAS. But as we will show in the next exercise there is a way to combine the information from both plots into 1 single plot.

2. Finally, estimate the parameters for all subjects and each of their four PAS ratings. Then plot the estimated function at the group-level by taking the mean for each of the four parameters, *a*, *b*, *c* and *d* across subjects. A function should be estimated for each PAS-rating (it should look somewhat similar to Fig. 3 from the article: https://doi.org/10.1016/j.concog.2019.03.007 (https://doi.org/10.1016/j.concog.2019.03.007))

    i. compare with the figure you made in 5.3.ii and comment on the differences between the fits - mention some advantages and

disadvantages of both.

```
#Create a function with a loop for calculating a,b,c and d for every subject within e
very level of PAS.

optim_for_indi <- function(){
  data_frame_parameters <- data.frame(subject = NA, pas = NA , a = NA, b = NA, c = NA
, d = NA)

  for (i in 1:4){
    df_temp1 <- df_exp1 %>%
      filter(pas == i)

    for (ii in 1:length(unique(df_exp1$subject))){
      data_temp <- df_temp1 %>%
        filter(subject == ii) %>%
        dplyr::select(target.frames, right_answer, pas) %>%
        rename(x = target.frames, y = right_answer)

      op_temp = optim(par = c(0.5,0.5,1,1), fn = RSS, data = data_temp, method = "L-B
FGS-B", lower = c(0.5,0.5,-Inf,-Inf),
      upper = c(1,1,Inf,Inf))
      data_frame_parameters <- rbind(data_frame_parameters , c(ii,i,op_temp$par))

    }
  }
  return(data_frame_parameters)
}
```

```
#Get all parameters for each subject in each level of pas
df_param <- optim_for_indi() %>%
  na.omit()
df_param
```

	subject	...	a	b	c	d
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
2	1	1	0.5000000	0.5000000	1.0000000	1.0000000000
3	2	1	0.6072665	0.5000000	0.9206736	0.7154031594
4	3	1	0.6248181	0.5374096	0.8602455	0.4300550152
5	4	1	0.7593121	0.5000000	0.9151872	0.0932777361
6	5	1	0.5000000	0.5000000	1.0000000	1.0000000000
7	6	1	0.6143188	0.5000000	1.0058600	0.9558662745
8	7	1	0.5000000	0.5000000	1.0000000	1.0000000000
9	8	1	0.6165075	0.5000000	0.9436198	0.8557934710

	subject	...	a	b	c	d
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
10	9	1	0.5000000	0.5391376	1.7117410	0.0203760864
11	10	1	0.5000000	0.5000000	1.0000000	1.0000000000

1-10 of 116 rows Previous **1** 2 3 4 5 6 ... 12 Next

```
#Compute the average value of a,b,c and d.
df_param_avg <- df_param %>%
  group_by(pas) %>%
  summarise(a = mean(a), b = mean(b), c = mean(c), d = mean(d))

df_param_avg
```

pas	a	b	c	d
<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	0.5796434	0.5514403	1.638820	0.4571516
2	0.5274091	0.8775394	2.829774	0.1757673
3	0.5768916	0.9556266	2.281409	0.2259202
4	0.7199105	0.9658502	2.189910	0.5010169

4 rows

We currently have a problem of $a > b$ in PAS1. This will result in a declining sigmoid function. However, I can not specify logical statement in lower_boundaries ensuring $b > a$.

```
#x-values
x_target.frames <- seq(0,8,0.01)


#setup data frame for x and y of different PAS score.
df_x_y <- data.frame(x_target.frames)

#Function for estimating y_hat given our a,b,c and d from our optim function.
y_hat_func <- function(){
for (i in 1:4){
  y_hat_temp <- sigmoid(df_param_avg$a[i], df_param_avg$b[i], df_param_avg
                        $c[i], df_param_avg$d[i], x_target.frames)
  df_x_y[,i+1] <- y_hat_temp
}

  df_x_y <- df_x_y %>% #Rename column names into something meaningful.
    rename(x = x_target.frames, y_hat_pas1 = V2 , y_hat_pas2 = V3 , y_hat_pas3 = V4 ,
y_hat_pas4 = V5)
  return(df_x_y)
}

df_final_param <- y_hat_func()
```

```
head(df_final_param)
```

	x <dbl>	y_hat_pas1 <dbl>	y_hat_pas2 <dbl>	y_hat_pas3 <dbl>	y_hat_pas4 <dbl>
1	0.00	0.5788822	0.5274092	0.5769072	0.7229803
2	0.01	0.5788658	0.5274092	0.5769079	0.7230414
3	0.02	0.5788491	0.5274092	0.5769086	0.7231037
4	0.03	0.5788320	0.5274092	0.5769094	0.7231672
5	0.04	0.5788146	0.5274092	0.5769102	0.7232319
6	0.05	0.5787968	0.5274092	0.5769110	0.7232980

6 rows

```
tail(df_final_param)
```

	x <dbl>	y_hat_pas1 <dbl>	y_hat_pas2 <dbl>	y_hat_pas3 <dbl>	y_hat_pas4 <dbl>
796	7.95	0.5514403	0.8775394	0.9556266	0.9658477
797	7.96	0.5514403	0.8775394	0.9556266	0.9658478

	x	y_hat_pas1	y_hat_pas2	y_hat_pas3	y_hat_pas4
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
798	7.97	0.5514403	0.8775394	0.9556266	0.9658478
799	7.98	0.5514403	0.8775394	0.9556266	0.9658479
800	7.99	0.5514403	0.8775394	0.9556266	0.9658479
801	8.00	0.5514403	0.8775394	0.9556266	0.9658480

6 rows

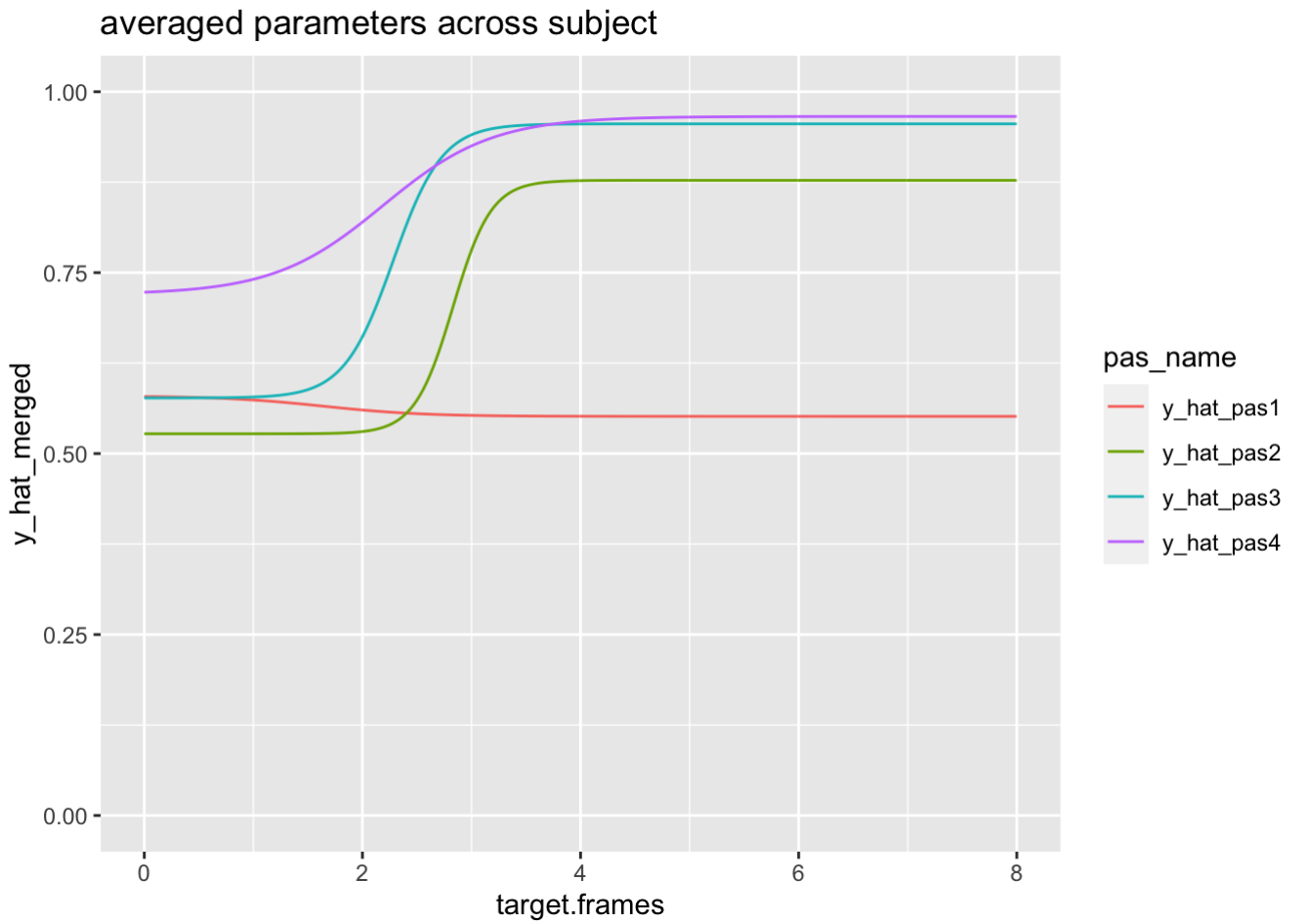
As X -> 8 our y_hat_pas1 estimates based on the sigmoid function defined in earlier chunks will decrease. This does not make sense what so ever but is due to *a* and *b* from our optim function.

```
df_final_param_long <- df_final_param %>%
  pivot_longer(cols = c(y_hat_pas1,y_hat_pas2,y_hat_pas3,y_hat_pas4) , names_to = "pa
s_name", values_to = "y_hat_merged")

df_final_param_long
```

	x	pas_name	y_hat_merged
	<dbl>	<chr>	<dbl>
	0.00	y_hat_pas1	0.5788822
	0.00	y_hat_pas2	0.5274092
	0.00	y_hat_pas3	0.5769072
	0.00	y_hat_pas4	0.7229803
	0.01	y_hat_pas1	0.5788658
	0.01	y_hat_pas2	0.5274092
	0.01	y_hat_pas3	0.5769079
	0.01	y_hat_pas4	0.7230414
	0.02	y_hat_pas1	0.5788491
	0.02	y_hat_pas2	0.5274092

1-10 of 3,204 rows Previous **1** 2 3 4 5 6 … 321 Next

```
ggplot(df_final_param_long, aes(x, y_hat_merged, colour = pas_name)) +
  geom_line() + ylim(0,1) + labs(x = "target.frames", title = "averaged parameters ac
ross subject")
```



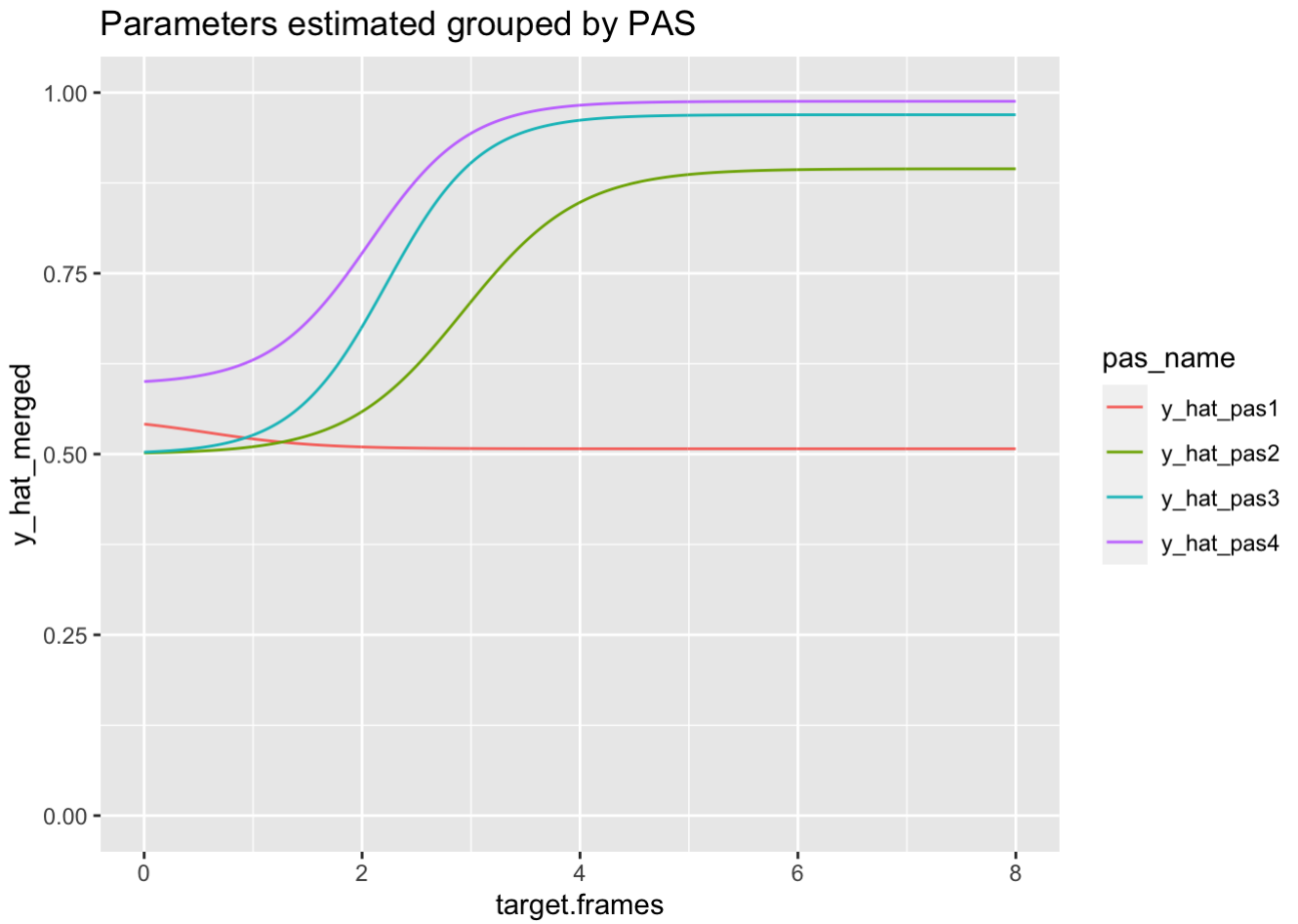
Extra stuff

Checking the difference between taking the mean() of every parameter c(a,b,c,d) and running the optim on data only divided by PAS ratings.

```
#Make function

optim_pas_overall <- function(i){
  #prepare correct subset of  data frame.
  pas_temp <- df_exp1 %>%
    filter(pas == i) %>%
    dplyr::select(target.frames, right_answer) %>%
    rename(x = target.frames, y = right_answer)
  op_temp_pas = optim(par = c(0.5,1,1,1), fn = RSS, data = pas_temp, method = "L-BFGS
-B", lower =  c(0.5,0.5,-Inf,-Inf), upper = c(1,1,Inf,Inf))

  # x and y values
  x_pas_optim <- seq(0, 8, 0.01)
  y_pas_optim <- sigmoid(op_temp_pas$par[1],op_temp_pas$par[2],
                         op_temp_pas$par[3],op_temp_pas$par[4],x_pas_optim)
  return(y_pas_optim)
}
```

```
#Generate y_hats with function for all pas ratings.
data_frame_y_hats <- tibble(x = x_target.frames, y_hat_pas1 = optim_pas_overall(1)
                            , y_hat_pas2 = optim_pas_overall(2)
                            , y_hat_pas3 = optim_pas_overall(3)
                            , y_hat_pas4 = optim_pas_overall(4))
```

```
#Pivot longer to merge into 1 columns with an identity column.
data_frame_y_hats_long <- data_frame_y_hats %>%
  pivot_longer(cols = c(y_hat_pas1,y_hat_pas2,y_hat_pas3,y_hat_pas4) , names_to = "pa
s_name", values_to = "y_hat_merged")

#plot
ggplot(data_frame_y_hats_long, aes(x, y_hat_merged, colour = pas_name)) +
  geom_line() + ylim(0,1) + labs(x = "target.frames", title = "Parameters estimated g
rouped by PAS")
```



Advantage If the relation between *% of guessing correct* and *target.frames* is in nature a sigmoid function. ie. When target.frames goes from 0 -> *inf* then would derivative first be fairly low (difference between 0-1 isn't that great) while at some point we would hit the inflection point (point with highest derivative and effect of increasing target.frame by any unit.) And afterwards the derivative would lessen because increasing target.frames above a specific number would not result in any new information and therefore better accuracy.

Disadvantage/problem This plot/method alows us to set a minimum of our y-values. I've specified lower boundary of a = 0.5 (not allowing below chance). As this was our goal with using the optim function. Though there are several arguments against setting a minimum y-value. (It's cheating. :P)

The Optim function tries to minimise our RSS given our other parameters (a,b,c,d). So if we really were to compare the model from optim with min = 0.5 and our interaction model from exercise 5 we should look at diff in RSS. Specifying absurd high/low as *a* & *b* = 1 would give incredible accuracy by may have inflated RSS and poor prediction accuracy of unseen data.

Portfolio 3 , Methods 3, 2021, autumn semester

Author: Sigurd Fyhn Sørensen

Date: 17-11-21

Exercises and objectives

1) Load the magnetoencephalographic recordings and do some initial plots to understand the data
2) Do logistic regression to classify pairs of PAS-ratings
3) Do a Support Vector Machine Classification on all four PAS-ratings

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This is Assignment 3 and will be part of your final portfolio

EXERCISE 1 - Load the magnetoencephalographic recordings and do some initial plots to understand the data

The files `megmag_data.npy` and `pas_vector.npy` can be downloaded here (http://laumollerandersen.org/data_methods_3/megmag_data.npy) and here (http://laumollerandersen.org/data_methods_3/pas_vector.npy)

In [ ]:
```
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
```

In [ ]:
```
#LOAD DATA ONLINE
""" import requests
import io

response = requests.get('http://laumollerandersen.org/data_methods_3/megmag_d
response.raise_for_status()
data = np.load(io.BytesIO(response.content))


response = requests.get('http://laumollerandersen.org/data_methods_3/pas_vect
response.raise_for_status()
y = np.load(io.BytesIO(response.content)) """
```

Out[ ]:
```
" import requests\nimport io\n\nresponse = requests.get('http://laumollerander
sen.org/data_methods_3/megmag_data.npy')\nresponse.raise_for_status()\ndata =
np.load(io.BytesIO(response.content))\n\n\nresponse = requests.get('http://lau
```

```
mollerandersen.org/data_methods_3/pas_vector.npy')\nresponse.raise_for_status
()\ny = np.load(io.BytesIO(response.content)) "
```

1) Load `megmag_data.npy` and call it `data` using `np.load` . You can use `join` , which can be imported from `os.path` , to create paths from different string segments
i. The data is a 3-dimensional array. The first dimension is number of repetitions of a visual stimulus , the second dimension is the number of sensors that record magnetic fields (in Tesla) that stem from neurons activating in the brain, and the third dimension is the number of time samples. How many repetitions, sensors and time samples are there?

In [ ]:
```
#load data local
data = np.load("/Users/sigurd/Downloads/megmag_data.npy")
y = np.load("/Users/sigurd/Downloads/pas_vector.npy")
```

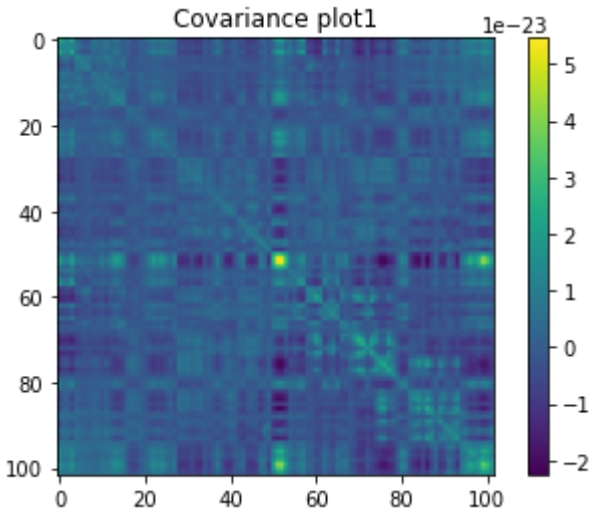
ii. The time range is from (and including) -200 ms to (and including) 800 ms with a sample recorded every 4 ms. At time 0, the visual stimulus was briefly presented.
    Create a 1-dimensional array called `times` that represents this.

In [ ]:
```
time = np.arange(-200,804, 4)
```

iii. Create the sensor covariance matrix Σ_{XX}:
$$\Sigma_{XX} = \frac 1 N \sum_{i=1}^N XX^T$$ N is the number of repetitions and X has s rows and t columns (sensors and time), thus the shape is $X_{s\times t}$. Do the sensors pick up independent signals? (Use `plt.imshow` to plot the sensor covariance matrix)

In [ ]:
```
#1/682 * np.sum(data[0,:,:]@data[0,:,:].T)
cov_matrix = 1/682 * sum([data[x,:,:]@data[x,:,:].T for x in range(682)])
plt.imshow(cov_matrix)
plt.title("Covariance plot1")
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x7fe1c63aa130>
```



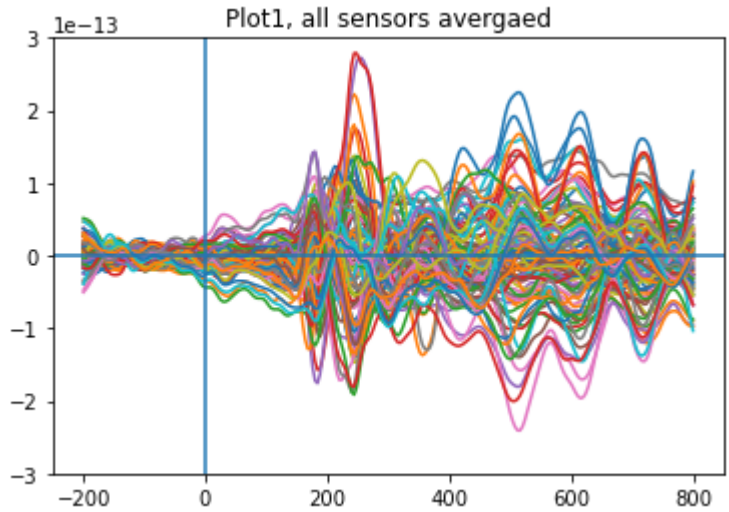
Sensors seem to pick up independent signals, but we cannot be sure there is some covariance between a few of the sensors.

iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resulting array should have two dimensions with time as the first and magnetic field as the second)

In [ ]:
```
rep_mean = np.mean(data, axis = 0) #axis specifies which axis the mean should
```

v. Plot the magnetic field (based on the average) as it evolves over time for each of the sensors (a line for each) (time on the x-axis and magnetic field on the y-axis). Add a horizontal line at $y = 0$ and a vertical line at $x = 0$ using `plt.axvline` and `plt.axhline`

In [ ]:
```
for i in range(102):
    plt.plot(time , rep_mean[i,:])
plt.axvline(0)
plt.axhline(0)
plt.title("Plot1, all sensors avergaed")
plt.ylim(-3e-13, 3e-13)
plt.show()
```



vi. Find the maximal magnetic field in the average. Then use `np.argmax` and `np.unravel_index` to find the sensor that has the maximal magnetic field.

In [ ]:
```
indx_max = np.unravel_index(np.argmax(rep_mean), rep_mean.shape)

print("mean:", rep_mean[indx_max], "\nindx:", indx_max)
```

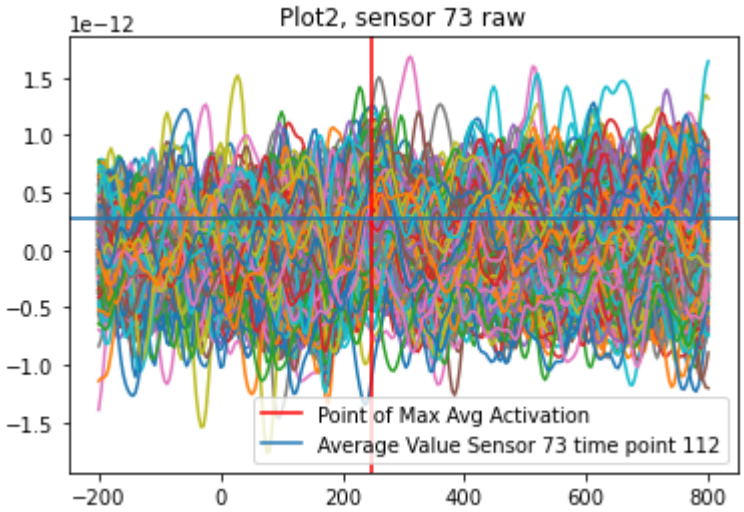
```
mean: 2.7886216843591933e-13
indx: (73, 112)
```

Sensor = 73 at time point = 112 has the highest value. This would be the red peak around 220ms on the above plot.

vii. Plot the magnetic field for each of the repetitions (a line for each) for the sensor that has the maximal magnetic field.

Highlight the time point with the maximal magnetic field in the average (as found in 1.1.v) using `plt.axvline`

In [ ]:
```
data[:,73,:].shape

for i in range(682):
    plt.plot(time, data[i,73,:])
plt.axvline(112*4-200, color = "red", label = "Point of Max Avg Activation") ;
plt.axhline(rep_mean[73,112], label = "Average Value Sensor 73 time point 112
plt.title("Plot2, sensor 73 raw")
plt.legend()
plt.show()
```



Notice the decimal change on the y-axis. Plot1's Y scale is 1e-13 and plot2 1e-12. On plot2 0.2788 which look to be the mean around the hline would be equal to 2.788 in plot1 on the scale of 1e-e13.

viii. Describe in your own words how the response found in the average is represented in the single repetitions. But do make sure to use the concepts _signal_ and _noise_ and comment on any differences on the range of values on the y-axis

Mean value at the time stamp 220ms across all repetitions = 2.7886216843591933e-13. This is roughly the same value we see in the plot illustrating the averages across repetitions. However, there is a lot of noise surrounding the mean signal. The Std for timestamp = 220ms is even higher than the mean. We must therefore conclude that there is a lot of noise.

In [ ]:
```
print('std:', np.std(data[:,73,112]))
print('mean:', np.mean(data[:,73,112]))
```

```
std: 3.189439776492671e-13
mean: 2.7886216843591933e-13
```

2) Now load `pas_vector.npy` (call it `y`). PAS is the same as in Assignment 2, describing the clarity of the subjective experience the subject reported after seeing the briefly presented stimulus

i. Which dimension in the `data` array does it have the same length as?

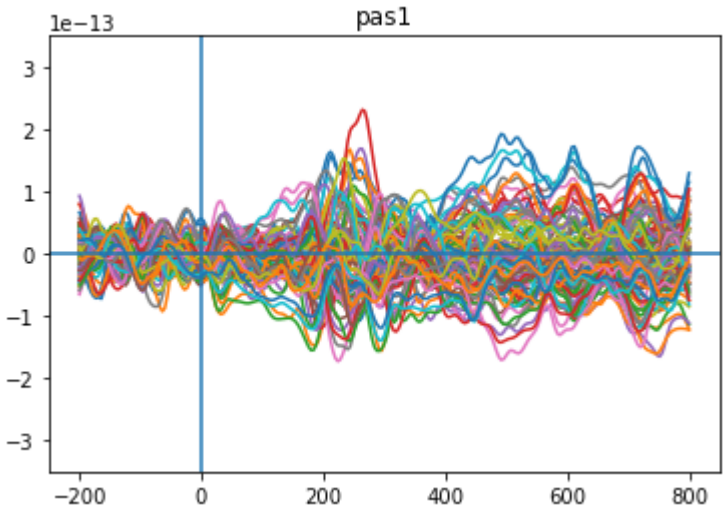
It has the same length as the first dimension of our ndarray with MEG data. This must be

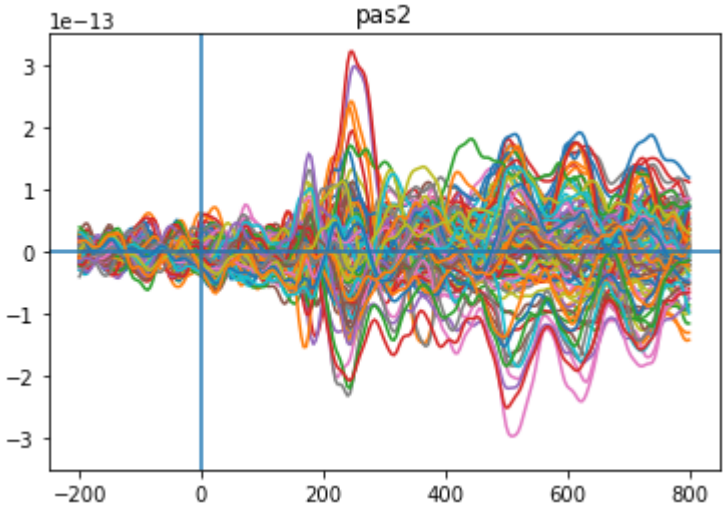
because that there is an individual score for each repetition.

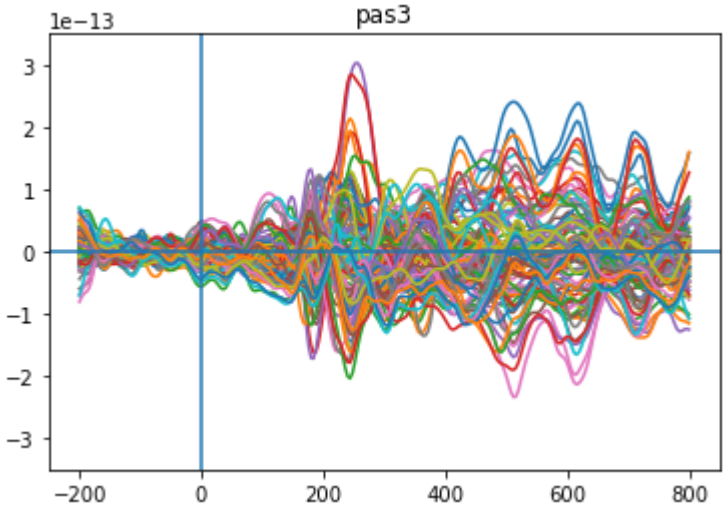
ii. Now make four averages (As in Exercise 1.1.iii), one for each PAS rating, and plot the four time courses (one for each PAS rating) for the sensor found in Exercise 1.1.v

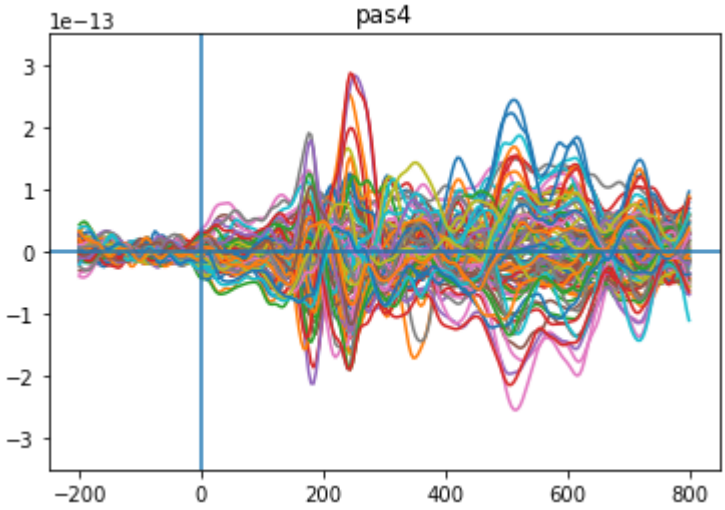
In [ ]:
```
d = {}
#Create a dictonary with 4 levels on for each pas rating containing the data
for i in range(1,5):
    idx_ = np.argwhere(y == i)
    d["pas" + str(i)] = np.squeeze(data[idx_, : , :])
```

In [ ]:
```
rep_mean_dic = {}
#Compute the mean across itterations for each PAS rating.  Save in a dict wit
for key in d:
    rep_mean_dic[str(key)] = np.mean(d[key], axis = 0)

#Plot the avg for each PAS.
def plot_pas_rat():
    for key in rep_mean_dic:
        temp = rep_mean_dic[key]
        for i in range(102):
            plt.plot(time , temp[i,:])
        plt.axvline(0)
        plt.ylim(-3.5e-13, 3.5e-13)
        plt.axhline(0)
        plt.title(str(key))
        plt.show()
plot_pas_rat()
```



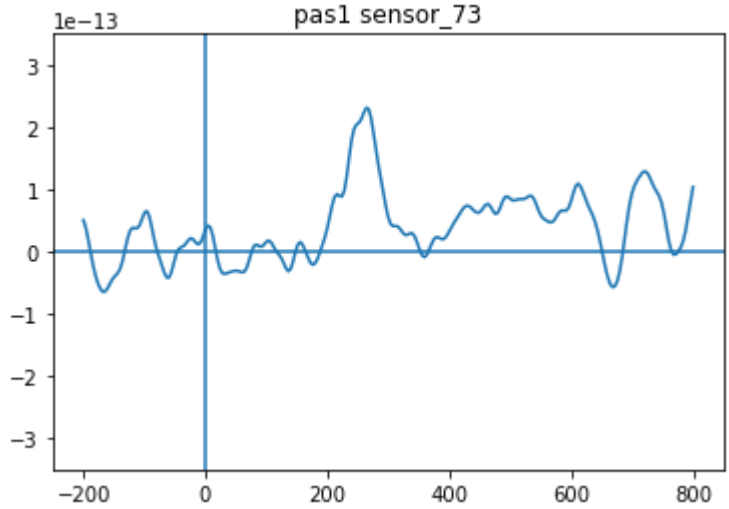


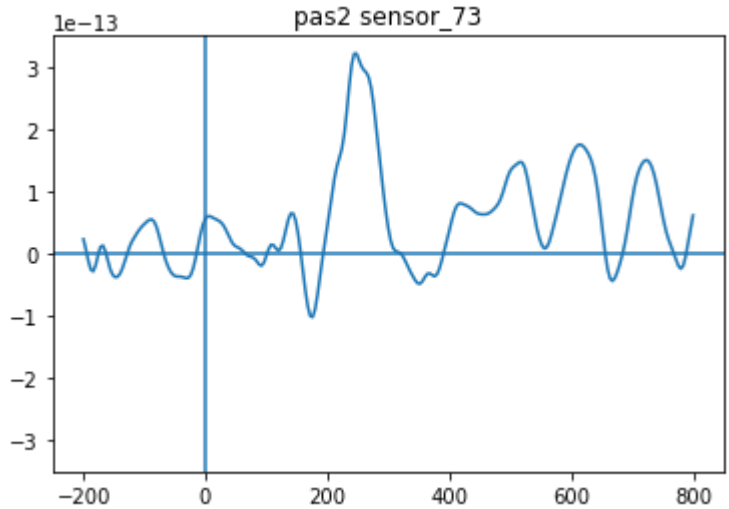


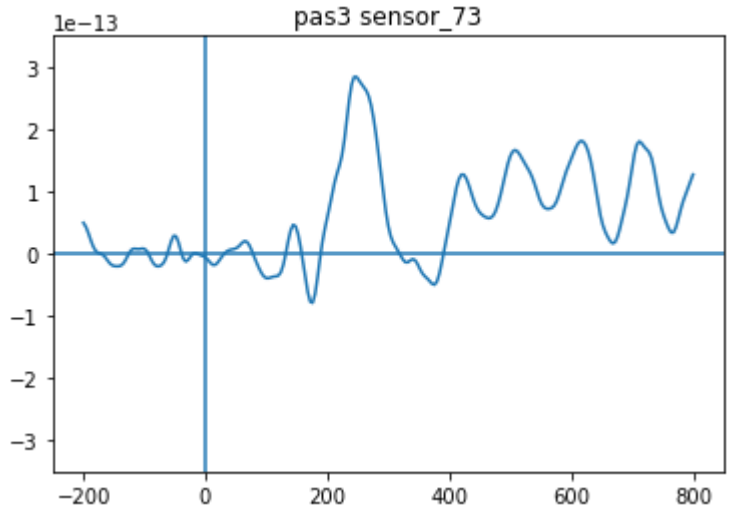


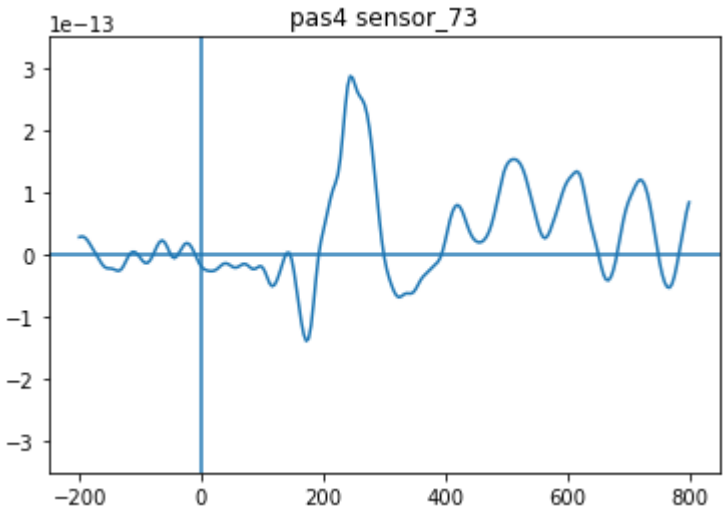
In [ ]:

```
for key in rep_mean_dic:
    temp = rep_mean_dic[key]

    plt.plot(time , temp[73,:])
    plt.axvline(0)
    plt.ylim(-3.5e-13, 3.5e-13)
    plt.axhline(0)
    plt.title(str(key) + " sensor_73")
    plt.show()
```


pas1 sensor_73
1e−13
3
2
1
0
−1
−2
−3
−200
0
200
400
600
800

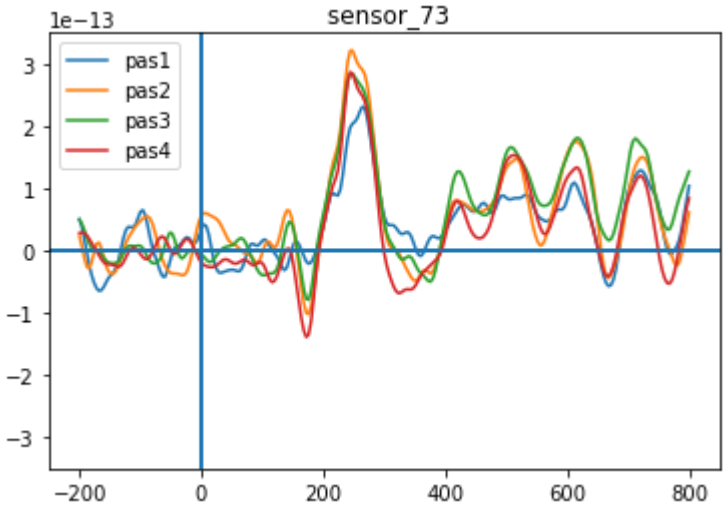

pas2 sensor_73
1e−13
3
2
1
0
−1
−2
−3
−200
0
200
400
600
800


pas3 sensor_73
1e−13
3
2
1
0
−1
−2
−3
−200
0
200
400
600
800



In [ ]:

```
for key in rep_mean_dic:
    temp = rep_mean_dic[key]

    plt.plot(time , temp[73,:], label = str(key))
    plt.axvline(0)
    plt.ylim(-3.5e-13, 3.5e-13)
    plt.axhline(0)
    plt.title(" sensor_73")
    plt.legend()
plt.show()
```



iii. Notice that there are two early peaks (measuring visual activity from the brain), one before 200 ms and one around 250 ms. Describe how the amplitudes of responses are related to the four PAS-scores. Does PAS 2 behave differently than expected?

Higher amplitudes indicate a shift in magnetic fields, which is correlated with brain activity/ hemoglobin increase at target areas. I don't specifically where sensor 73 were placed. But it seems a higher PAS-score results in a higher brain response in the given area.

PAS2 The amplitude before 200ms seem to become more negative the higher the PAS score which is followed by the maximum around 250ms that increases with PAS-rating. However, comparing PAS-rating = 2 at sensor 73 to PAS-rating = 2|3, PAS2 seem to have a higher amplitude at 250ms and lower around 180ms.

EXERCISE 2 - Do logistic regression to classify pairs of PAS-ratings

1) Now, we are going to do Logistic Regression with the aim of classifying the PAS-rating given by the subject

i. We'll start with a binary problem - create a new array called `data_1_2` that only contains PAS responses 1 and 2. Similarly, create a `y_1_2` for the target vector

ii. Scikit-learn expects our observations (`data_1_2`) to be in a 2d-array, which has samples (repetitions) on dimension 1 and features (predictor variables) on dimension 2. Our `data_1_2` is a three-dimensional array. Our strategy will be to collapse our two last dimensions (sensors and time) into one dimension, while keeping the first dimension as it is (repetitions). Use `np.reshape` to create a variable `X_1_2` that fulfils these criteria.

In [ ]:
```
idx = np.argwhere((y ==1) | (y ==2))
data_1_2 = np.squeeze(data[idx,:,:])
y_1_2 = y[idx]
y_1_2 = np.reshape(y_1_2, (len(y_1_2)))

X_1_2 = np.reshape(data_1_2, newshape = (data_1_2.shape[0], data_1_2.shape[1]
```

iii. Import the `StandardScaler` and scale `X_1_2`

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_1_2_std = sc.fit_transform(X_1_2)
```

iv. Do a standard `LogisticRegression` - can be imported from `sklearn.linear_model` - make sure there is no `penalty` applied

In [ ]:
```
from sklearn.linear_model import LogisticRegression
lgR_1 = LogisticRegression(max_iter= 1000, penalty= 'none')
#fit
lgR_1.fit(X_1_2_std, y_1_2)
```

Out[ ]:
```
LogisticRegression(max_iter=1000, penalty='none')
```

v. Use the `score` method of `LogisticRegression` to find out how many labels were classified correctly. Are we overfitting? Besides the score, what would make you suspect that we are overfitting?

In [ ]:
```
lgR_1.score(X_1_2_std, y_1_2)
```

Out[ ]:
```
1.0
```

We get an accuracy score of 1 which indicates overfitting (or a perfect model if it could

generalize to out-of-sample prediction). Another indication that we could be overfitting would be our lack of penalty in our model.

vi. Now apply the _L1_ penalty instead - how many of the coefficients (`.coef_`) are non-zero after this?

In [ ]:
```
lgR_2 = LogisticRegression(penalty = 'l1', solver = "liblinear", C = 1 , max_
lgR_2.fit(X_1_2_std, y_1_2)
```

Out[ ]:
```
LogisticRegression(C=1, max_iter=400, penalty='l1', solver='liblinear')
```

In [ ]:
```
print("accuracy score %f" % (lgR_2.score(X_1_2_std, y_1_2)*100), "%")
print("non-zero coef = %f" % np.sum(lgR_2.coef_ != 0))
print("number of  zero coef = %f"  % np.sum(lgR_2.coef_ == 0))
```

```
accuracy score 100.000000 %
non-zero coef = 194.000000
number of  zero coef = 25408.000000
```

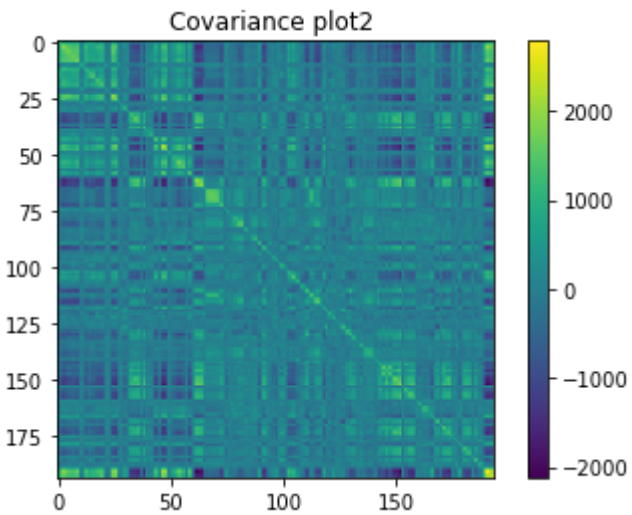
10997 coefficients are non-zero when using l1 regularization/penalizing.

vii. Create a new reduced X that only includes the non-zero coefficients - show the covariance of the non-zero features (two covariance matrices can be made; $X_{reduced}X_{reduced}^T$ or $X_{reduced}^TX_{reduced}$ (you choose the right one)) . Plot the covariance of the features using `plt.imshow`. Compared to the plot from 1.1.iii, do we see less covariance?

In [ ]:
```
idx_non0 = np.where(lgR_2.coef_ != 0)[1]
data_reduced = X_1_2_std[:,idx_non0]
```

In [ ]:
```
cov = data_reduced.T @ data_reduced
plt.imshow(np.cov(cov))
plt.title("Covariance plot2 ")
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x7fe1a2f5f5e0>
```



There is less covariance in plot 2 compared to covariance plot 1. It is difficult to quantify

these differences just by visual inspection, but it gives a good intuition.

2) Now, we are going to build better (more predictive) models by using cross-validation as an outcome measure

i. Import `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

In [ ]:
```
def equalize_targets_binary(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) > 2:
        raise NameError("can't have more than two targets")
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target)) ## find the number of each target
        indices.append(np.where(y == target)[0]) ## find their indices
    min_count = np.min(counts)
    # randomly choose trials
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count,replace=False

    # create the new data sets
    new_indices = np.concatenate((first_choice, second_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y
```

ii. To make sure that our training data sets are not biased to one target (PAS) or the other, create `y_1_2_equal`, which should have an equal number of each target. Create a similar `X_1_2_equal`. The function `equalize_targets_binary` in the code chunk associated with Exercise 2.2.ii can be used. Remember to scale `X_1_2_equal`!

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold

X_1_2_equa , y_1_2_equa  = equalize_targets_binary(data_1_2, y_1_2)
X_1_2_equa_2d = X_1_2_equa.reshape(198,-1)


sc = StandardScaler()
X_1_2_equa_std = sc.fit_transform(X_1_2_equa_2d)

#X_1_2_equa_std = sc.fit_transform(X_1_2_equa)
```

iii. Do cross-validation with 5 stratified folds doing standard `LogisticRegression` (See Exercise 2.1.iv)

In [ ]:
```
kfold = StratifiedKFold(n_splits= 5, random_state = 2, shuffle= True).split(X_

scores = []
lr = LogisticRegression(penalty= "l1", solver = "liblinear")

for k, (train, test) in enumerate(kfold):
    lr.fit(X_1_2_equa_std[train], y_1_2[train])
    score = lr.score(X_1_2_equa_2d[test], y_1_2_equa[test])
```

```
    scores.append(score)
    print('Fold: %2d, Class dist.: %s, Acc: %.3f' % (k+1,
        np.bincount(y_1_2[train]), score))

cv_score = cross_val_score(lr, X_1_2_equa_std, y_1_2_equa, cv= 5)
print("accuracy scores for k-folds:" , cv_score)
```

```
Fold:  1, Class dist.: [ 0 69 89], Acc: 0.500
Fold:  2, Class dist.: [ 0 69 89], Acc: 0.500
Fold:  3, Class dist.: [ 0 71 87], Acc: 0.500
Fold:  4, Class dist.: [ 0 74 85], Acc: 0.487
Fold:  5, Class dist.: [ 0 73 86], Acc: 0.513
accuracy scores for k-folds: [0.625      0.575      0.425      0.64102564 0.46
153846]
```

iv. Do L2-regularisation with the following `Cs= [1e5, 1e1, 1e-5]`. Use the same kind of cross-validation as in Exercise 2.2.iii. In the best-scoring of these models, how many more/fewer predictions are correct (on average)?

In [ ]:

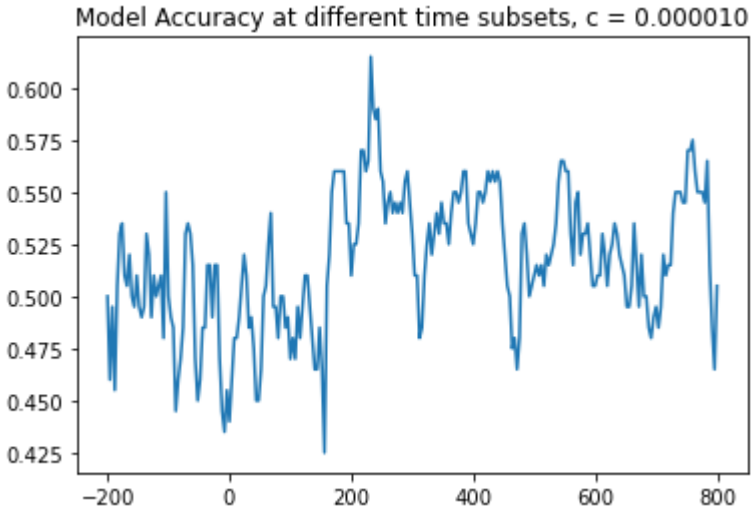
```
def cv_different_C(a):
    for i in a:
        cv_score = []
        lr2 = LogisticRegression(penalty= "l2", C = i)
        cv_score = cross_val_score(lr2, X_1_2_equa_std, y_1_2_equa, cv = 5)
        print("mean accuracy for c =",str(i)+ ":", np.mean(cv_score))

cv_different_C([1e5, 1e1, 1e-5])
```

```
mean accuracy for c = 100000.0: 0.5353846153846155
mean accuracy for c = 10.0: 0.5252564102564102
mean accuracy for c = 1e-05: 0.5956410256410256
```

v. Instead of fitting a model on all `n_sensors * n_samples` features, fit a logistic regression (same kind as in Exercise 2.2.iv (use the `C` that resulted in the best prediction)) for __each__ time sample and use the same cross-validation as in Exercise 2.2.iii. What are the time points where classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

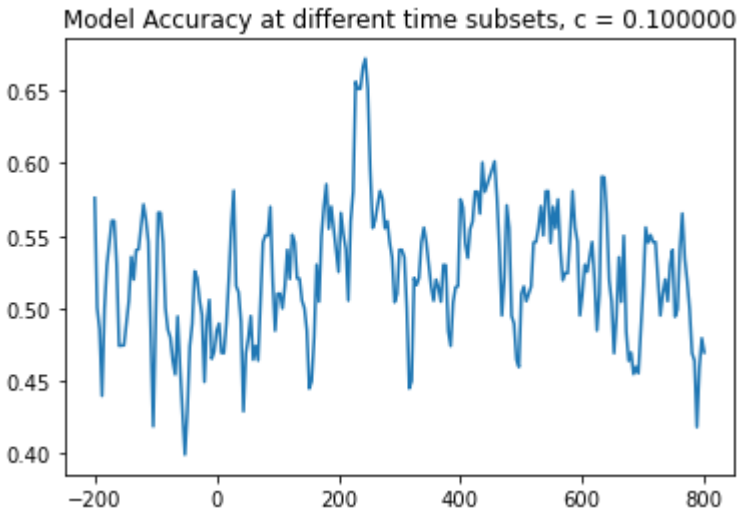
In [ ]:

```
def cv_different_C_i(a, penal, solver_lul, X, y):
    accuracy = []
    lr3 = LogisticRegression(penalty = penal, C = a, solver = solver_lul)
    for i in range(251):
        cv_score = []
        sc = StandardScaler()
        X_std = sc.fit_transform(X[:,:,i])
        cv_score = cross_val_score(lr3, X_std, y)
        accuracy.append(np.mean(cv_score))
    return(accuracy)

def plot_cv_time(a, penal, solver_lul, X , y ):
    cv_acc_means = cv_different_C_i(a, penal, solver_lul, X , y)
    plt.plot(time, cv_acc_means)
    plt.title("Model Accuracy at different time subsets, c = %f" % a )
    plt.show()

plot_cv_time(a = 1e-5, penal = "l2", solver_lul= "lbfgs", X = X_1_2_equa, y =
```



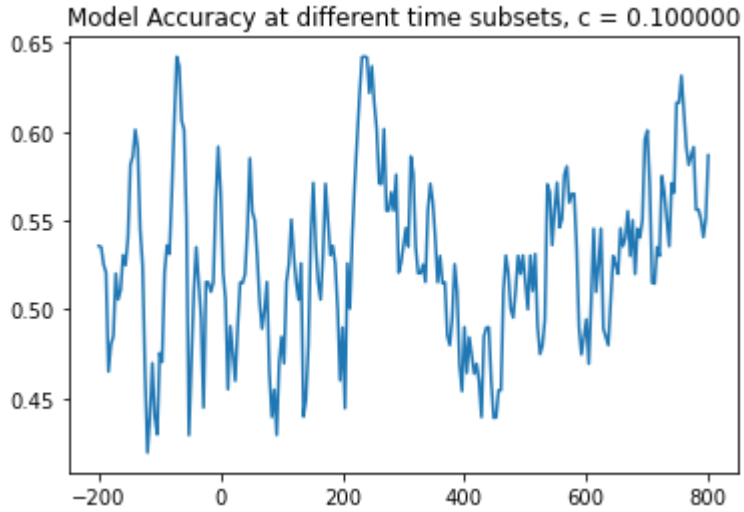
vi. Now do the same, but with L1 regression - set `C=1e-1` - what are the time points when classification is best? (make a plot)?

In [ ]:
```
plot_cv_time(a = 1e-1, penal = "l1", solver_lul= "liblinear", X = X_1_2_equa,
```



vii. Finally, fit the same models as in Exercise 2.2.vi but now for `data_1_4` and `y_1_4` (create a data set and a target vector that only contains PAS responses 1 and 4). What are the time points when classification is best? Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

In [ ]:
```
idx = np.argwhere((y ==1) | (y ==4)) # Get index for pas 1 & 4
data_1_4 = np.squeeze(data[idx,:,:]) #Squeeze the data and filter repittion b
y_1_4 = y[idx] #filter y by idx
y_1_4 = np.reshape(y_1_4, (len(y_1_4))) #get the right y_shape
X_1_4 = np.reshape(data_1_4, newshape = (data_1_4.shape[0], data_1_4.shape[1]
data_1_4_equa , y_1_4_equa = equalize_targets_binary(data_1_4, y_1_4) #equali



plot_cv_time(a = 1e-1, penal = "l1", solver_lul= "liblinear", X = data_1_4_eq
```



3) Is pairwise classification of subjective experience possible? Any surprises in the classification accuracies, i.e. how does the classification score fore PAS 1 vs 4 compare to the classification score for PAS 1 vs 2?

Pairwise classification of PAS scores does not seem to be possible. Surprisingly the accuracy for PAS1-2 classification is higher compared to PAS1-2. This indicates that the difference between PAS1-2 is larger or more linearly separable than PAS1-4. But in general the accuracy is not much better than chance.

EXERCISE 3 - Do a Support Vector Machine Classification on all four PAS-ratings

1) Do a Support Vector Machine Classification
i. First equalize the number of targets using the function associated with each PAS-rating using the function associated with Exercise 3.1.i

In [ ]:
```
def equalize_targets(data, y):
    np.random.seed(7)
    targets = np.unique(y) ## find the number of targets
    if len(targets) > 4:
        raise NameError("can't have more than two targets")
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target)) ## find the number of each target
        indices.append(np.where(y == target)[0]) ## find their indices
    min_count = np.min(counts)
    # randomly choose trials
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count,replace=False
    third_choice = np.random.choice(indices[2], size = min_count, replace = F
    fourth_choice = np.random.choice(indices[3], size = min_count, replace = 
    # create the new data sets
    new_indices = np.concatenate((first_choice, second_choice, third_choice, 
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y
```

In [ ]:

```
data_equa, y_equa = equalize_targets(data, y)
```

In [ ]:
```
sum(y_equa == 1), sum(y_equa == 2), sum(y_equa == 3), sum(y_equa == 4)
```

Out[ ]:
```
(99, 99, 99, 99)
```

ii. Run two classifiers, one with a linear kernel and one with a radial basis (other options should be left at their defaults) - the number of features is the number of sensors multiplied the number of samples. Which one is better predicting the category?

In [ ]:
```
data_equa_2d = np.reshape(data_equa, newshape= (data_equa.shape[0], -1))
data_equa_2d_std = sc.fit_transform(data_equa_2d)
```

In [ ]:
```
from sklearn.svm import SVC
from sklearn import svm
#Linear function
sv = SVC(kernel='linear')
sv.fit(data_equa_2d_std, y_equa)
```

Out[ ]:
```
SVC(kernel='linear')
```

In [ ]:
```
#radial basis function
sv2 = SVC(kernel = "rbf")
sv2.fit(data_equa_2d_std, y_equa)
```

Out[ ]:
```
SVC()
```

In [ ]:
```
print("score for linear function:",sv.score(data_equa_2d_std, y_equa), "\n sc
```

```
score for linear function: 1.0
 score for rbf: 0.9873737373737373
```

iii. Run the sample-by-sample analysis (similar to Exercise 2.2.v) with the best kernel (from Exercise 3.1.ii). Make a plot with time on the x-axis and classification score on the y-axis with a horizontal line at the chance level (what is the chance level for this analysis?)

In [ ]:
```
def cv_different_svm(X, y):
    accuracy = []
    lr3 = SVC(kernel = "rbf", random_state= 10)
    for i in range(251):
        cv_score = []
        sc = StandardScaler()
        X_std = sc.fit_transform(X[:,:,i])
        cv_score = cross_val_score(lr3, X_std, y)
        accuracy.append(np.mean(cv_score))
        print("append:", i)
    return(accuracy)


svm_time_score = cv_different_svm(data_equa, y_equa)
```

```
append: 0
append: 1
append: 2
append: 3
append: 4
append: 5
append: 6
append: 7
append: 8
append: 9
append: 10
append: 11
append: 12
append: 13
append: 14
append: 15
append: 16
append: 17
append: 18
append: 19
append: 20
append: 21
append: 22
append: 23
append: 24
append: 25
append: 26
append: 27
append: 28
append: 29
append: 30
append: 31
append: 32
append: 33
append: 34
append: 35
append: 36
append: 37
append: 38
append: 39
append: 40
append: 41
append: 42
append: 43
append: 44
append: 45
append: 46
append: 47
append: 48
append: 49
append: 50
append: 51
append: 52
append: 53
append: 54
append: 55
append: 56
append: 57
append: 58
append: 59
append: 60
append: 61
append: 62
append: 63
```

```
append: 64
append: 65
append: 66
append: 67
append: 68
append: 69
append: 70
append: 71
append: 72
append: 73
append: 74
append: 75
append: 76
append: 77
append: 78
append: 79
append: 80
append: 81
append: 82
append: 83
append: 84
append: 85
append: 86
append: 87
append: 88
append: 89
append: 90
append: 91
append: 92
append: 93
append: 94
append: 95
append: 96
append: 97
append: 98
append: 99
append: 100
append: 101
append: 102
append: 103
append: 104
append: 105
append: 106
append: 107
append: 108
append: 109
append: 110
append: 111
append: 112
append: 113
append: 114
append: 115
append: 116
append: 117
append: 118
append: 119
append: 120
append: 121
append: 122
append: 123
append: 124
append: 125
append: 126
append: 127
```

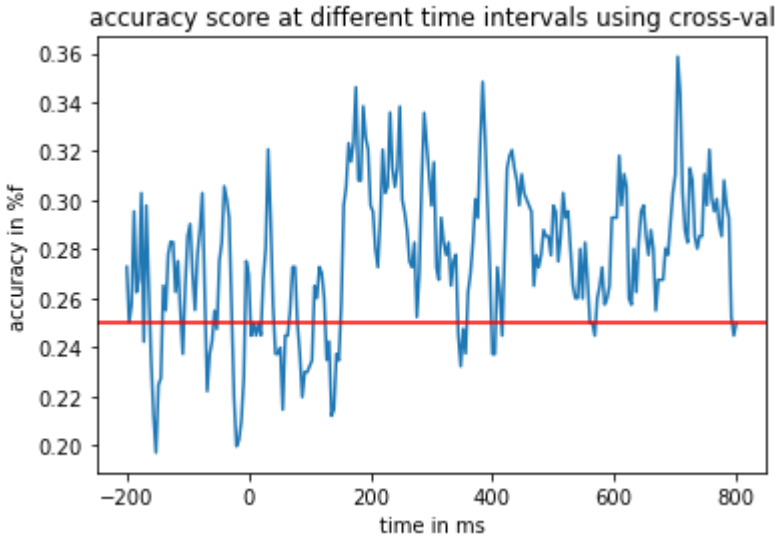
```
append: 128
append: 129
append: 130
append: 131
append: 132
append: 133
append: 134
append: 135
append: 136
append: 137
append: 138
append: 139
append: 140
append: 141
append: 142
append: 143
append: 144
append: 145
append: 146
append: 147
append: 148
append: 149
append: 150
append: 151
append: 152
append: 153
append: 154
append: 155
append: 156
append: 157
append: 158
append: 159
append: 160
append: 161
append: 162
append: 163
append: 164
append: 165
append: 166
append: 167
append: 168
append: 169
append: 170
append: 171
append: 172
append: 173
append: 174
append: 175
append: 176
append: 177
append: 178
append: 179
append: 180
append: 181
append: 182
append: 183
append: 184
append: 185
append: 186
append: 187
append: 188
append: 189
append: 190
append: 191
```

```
append: 192
append: 193
append: 194
append: 195
append: 196
append: 197
append: 198
append: 199
append: 200
append: 201
append: 202
append: 203
append: 204
append: 205
append: 206
append: 207
append: 208
append: 209
append: 210
append: 211
append: 212
append: 213
append: 214
append: 215
append: 216
append: 217
append: 218
append: 219
append: 220
append: 221
append: 222
append: 223
append: 224
append: 225
append: 226
append: 227
append: 228
append: 229
append: 230
append: 231
append: 232
append: 233
append: 234
append: 235
append: 236
append: 237
append: 238
append: 239
append: 240
append: 241
append: 242
append: 243
append: 244
append: 245
append: 246
append: 247
append: 248
append: 249
append: 250
```

In [ ]:

```
plt.plot(time, svm_time_score)
plt.axhline(0.25, color = "red")
plt.xlabel("time in ms")
plt.ylabel("accuracy in %f")
```

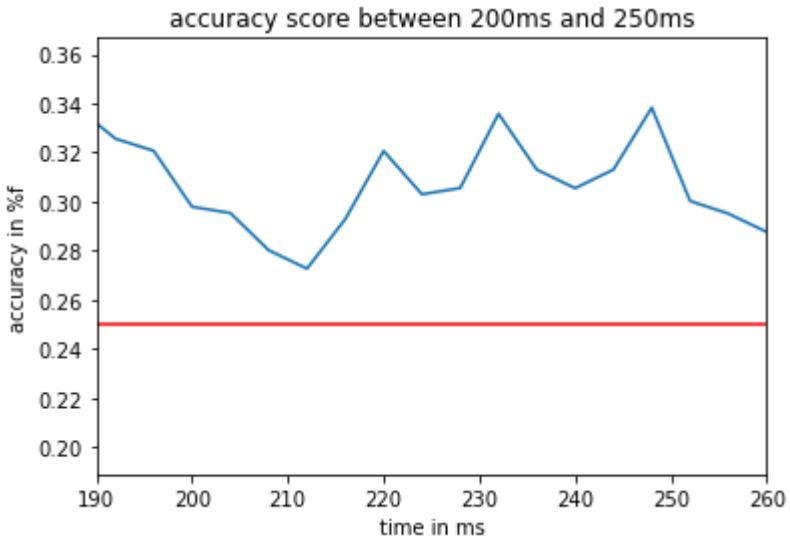
```
plt.title("accuracy score at different time intervals using cross-val")
plt.show()
```



iv. Is classification of subjective experience possible at around 200-250 ms?

In [ ]:

```
plt.plot(time, svm_time_score)
plt.axhline(0.25, color = "red")
plt.xlim(190,260)
plt.xlabel("time in ms")
plt.ylabel("accuracy in %f")
plt.title("accuracy score between 200ms and 250ms")
plt.show()
```



Our support vector machine classification performs better than chance (25%) when applied sequentially to the frames between 200 ms and 250ms. The max accuracy of our sequential modelling of PAS rating does not excede 33% at any time point. I would therefore argue that with our current model a classification of subjective experience in the time span between 200-250 ms is not possible.

2) Finally, split the equalized data set (with all four ratings) into a training part and test part, where the test part if 30 % of the trials. Use `train_test_split` from `sklearn.model_selection`

i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit`the training set and `predict` on the test set. This time your features are the number of sensors multiplied by the number of samples.

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_equa, y_equa, test_s
#Reshape
X_train_2d = np.reshape(X_train, newshape= (X_train.shape[0], -1))
X_test_2d = np.reshape(X_test, newshape= (X_test.shape[0], -1))

X_train_2d = sc.fit_transform(X_train_2d)
X_test_2d = sc.transform(X_test_2d)


sv3 = SVC(kernel='linear', random_state= 10)
sv4 = SVC(kernel = "rbf", random_state= 10)
sv3.fit(X_train_2d, y_train)
sv4.fit(X_train_2d, y_train)

y_pred_svm_split = sv3.predict(X_test_2d)
y_pred_svm2_split = sv4.predict(X_test_2d)
```

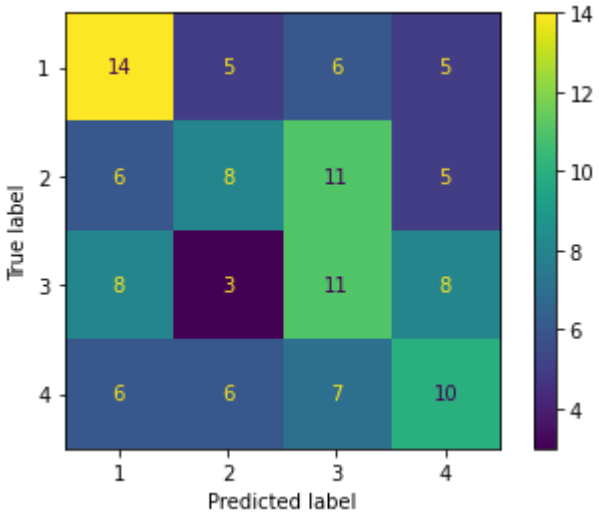
In [ ]:
```
print("linear score: " , sv3.score(X_test_2d, y_test), "\n rbf score:" ,sv4.s
```

```
linear score:  0.36134453781512604
 rbf score: 0.35294117647058826
```

ii. Create a _confusion matrix_. It is a 4x4 matrix. The row names and the column names are the PAS-scores. There will thus be 16 entries. The PAS1xPAS1 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS1, $\hat y_{pas1}$. The PAS1xPAS2 entry will be the number of actual PAS1, y_{pas1} that were predicted as PAS2, $\hat y_{pas2}$ and so on for the remaining 14 entries. Plot the matrix

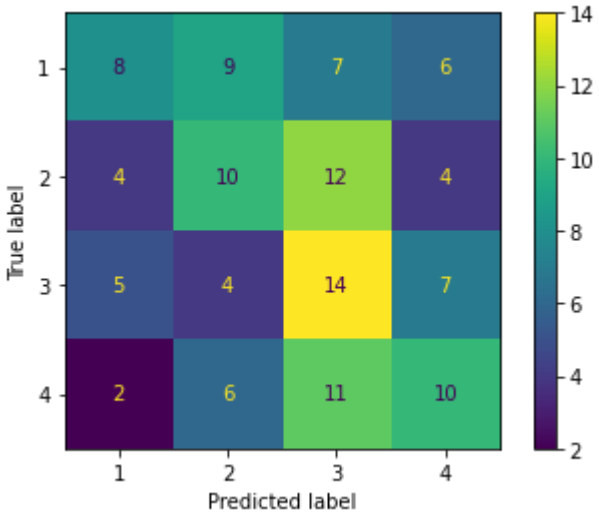
In [ ]:
```
from sklearn.metrics import confusion_matrix , ConfusionMatrixDisplay
#Confusion matrix for linaer kernel
confmat = confusion_matrix(y_test, y_pred_svm_split, labels= sv3.classes_)
confmat_disp = ConfusionMatrixDisplay(confmat, display_labels= sv3.classes_)
confmat_disp.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe19ccab7
90>
```



In [ ]:
```
confmat = confusion_matrix(y_test, y_pred_svm2_split, labels= sv4.classes_)
confmat_disp = ConfusionMatrixDisplay(confmat, display_labels= sv4.classes_)
confmat_disp.plot()
```

Out[ ]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x7fe1c78ae5
20>
```



iii. Based on the confusion matrix, describe how ratings are misclassified and if that makes sense given that ratings should measure the strength/quality of the subjective experience. Is the classifier biased towards specific ratings?

Linear kernel

36.13% accuracy. It chooses a PAS rating as being the dominant. With the current random_state it is PAS2. But with other states it chooses randomly. We can therefore conclude that the guessing is more up to chance than to observed regularities. Since the features of PAS 2,3,4 is so similar our linear SVM really struggles with distinguishing between those.

RBF kernel

35.29% accuracy. You would hope that RBF SVM were able to distinguish between the non-linear separable data. Changing the random_state again results in a change of PAS bias. So

it seems that even introducing non-linear dimensionality expansion through the RBF kernel does not supply anything to the model that would be much better than just chance. This is also supported by our relatively low accuracy scores.

Getting a better accuracy with deep learning NN

While SVM and LogisticRegression has its usefulness it is also restricted to only having one layer of tensor operations. Deep learning has proven useful when dealing with large amount of structured data when classifying perceptual tasks. Our data is stored in a 3D tensor (sample, sensor/feature, time) for dealing with such data the default architecture is a convolutional network or the long-short-term-memory (LSTM) network. But for now let us just work with stackable Dense layers and the same tensor shape as fed to the SVM and Logistic classifiers.

In [ ]:
```
import keras
from keras.models import Sequential
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score , KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
from keras.regularizers import l2
import tensorflow as tf
from keras.layers import LSTM, Dropout , Dense
from sklearn.decomposition import PCA
```

In [ ]:
```
#load data local
data = np.load("/Users/sigurd/Downloads/megmag_data.npy")
y = np.load("/Users/sigurd/Downloads/pas_vector.npy")
```

Dense NN

In [ ]:
```
# encode class values as integers
encoder = LabelEncoder()
encoder.fit(y_equa)
encoded_Y = encoder.transform(y_equa)
# convert integers to dummy variables (i.e. one hot encoded)
dummy_y = np_utils.to_categorical(encoded_Y)
```

In [ ]:
```
pc = PCA(n_components= 30)
data_equa_pca_2d_std = pc.fit_transform(data_equa_2d_std)
data_equa_pca_2d_std.shape
```

Out[ ]:
```
(396, 30)
```

In [ ]:
```
#Model fit without function
model = Sequential()
model.add(Dense(16, input_dim = 30, activation='relu'))
#model.add(layer= layers.Conv2D( 16 , 8 , padding='same', activation='relu'))
model.add(Dense(16, activation = 'relu', kernel_regularizer= l2(0.01), bias_r
model.add(Dense(4, activation='softmax'))
```

```
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['ac
```

Train, Train_val and Test split

Note to self: The test split should actually only be standardized and transformed based on the parameters of the train data split. Right now we've some data leakage between the test and train splits, but we will ignore it for now.

In [ ]:
```
from sklearn.model_selection import train_test_split
train_data, test_data, train_y, test_y = train_test_split(data_equa_pca_2d_st

data_val = train_data[:99]
data_train = train_data[99:]

y_val = train_y[:99]
y_train = train_y[99:]

"""from sklearn.preprocessing import Normalizer
scaler = Normalizer()

data_equa_2d_norm = scaler.fit_transform(data_equa_2d)
train_data2, test_data2, train_y2, test_y2 = train_test_split(data_equa_2d_no
```

Out[ ]:
```
'from sklearn.preprocessing import Normalizer\nscaler = Normalizer()\n\ndata_e
qua_2d_norm = scaler.fit_transform(data_equa_2d)\ntrain_data2, test_data2, tra
in_y2, test_y2 = train_test_split(data_equa_2d_norm, dummy_y, stratify= dummy_
y)'
```

In [ ]:
```
#train
model.fit(train_data, train_y, epochs = 10, batch_size= 8, verbose = 1, valid

#evalutate on test data
model.evaluate(test_data, test_y)
```

```
Epoch 1/10
38/38 [==============================] - 0s 4ms/step - loss: 0.6346 - accurac
y: 0.8249 - val_loss: 0.5574 - val_accuracy: 0.8384
Epoch 2/10
38/38 [==============================] - 0s 2ms/step - loss: 0.5722 - accurac
y: 0.8586 - val_loss: 0.5960 - val_accuracy: 0.8384
Epoch 3/10
38/38 [==============================] - 0s 2ms/step - loss: 0.5291 - accurac
y: 0.8923 - val_loss: 0.6141 - val_accuracy: 0.7980
Epoch 4/10
38/38 [==============================] - 0s 2ms/step - loss: 0.5015 - accurac
y: 0.8855 - val_loss: 0.6355 - val_accuracy: 0.7980
Epoch 5/10
38/38 [==============================] - 0s 2ms/step - loss: 0.4801 - accurac
y: 0.8990 - val_loss: 0.6648 - val_accuracy: 0.7677
Epoch 6/10
38/38 [==============================] - 0s 2ms/step - loss: 0.4652 - accurac
y: 0.9091 - val_loss: 0.6922 - val_accuracy: 0.7677
Epoch 7/10
38/38 [==============================] - 0s 2ms/step - loss: 0.4553 - accurac
y: 0.9091 - val_loss: 0.7270 - val_accuracy: 0.7475
Epoch 8/10
38/38 [==============================] - 0s 2ms/step - loss: 0.4402 - accurac
y: 0.9091 - val_loss: 0.7502 - val_accuracy: 0.7374
Epoch 9/10
```

```
38/38 [==============================] - 0s 2ms/step - loss: 0.4307 - accurac
y: 0.9226 - val_loss: 0.7961 - val_accuracy: 0.6869
Epoch 10/10
38/38 [==============================] - 0s 2ms/step - loss: 0.4215 - accurac
y: 0.9091 - val_loss: 0.7932 - val_accuracy: 0.6970
4/4 [==============================] - 0s 1ms/step - loss: 0.7932 - accuracy:
0.6970
```

Out[ ]: `[0.7932385802268982, 0.6969696879386902]`

report first model

Our validation score: accuracy = 69.7%, loss = 0.79.

Test score: accuracy = 69.6%, loss = 0.79.

Our test and validations scores are fairly similair this may be due to the data leakage between train and test.

Cross-validation of a new model

In [ ]:
```
# define baseline model
def baseline_model():
        # create model
        model = Sequential()
        model.add(Dense(32, input_dim= 30, activation='relu'))
        #model.add(layer= layers.Conv2D( 16 , 8 , padding='same', activation=
        model.add(Dense(64, activation = 'relu'))
        model.add(Dense(32, activation = 'relu'))
        model.add(Dense(4, activation='softmax'))
        # Compile model
        model.compile(loss='categorical_crossentropy', optimizer='adam', metr
        return model
```

In [ ]:
```
estimator = KerasClassifier(build_fn= baseline_model , epochs=20, batch_size=
#kFold
kfold = KFold(n_splits=5, shuffle=True)
```

In [ ]:
```
#Cross val based on NN
results = cross_val_score(estimator, data_equa_pca_2d_std , dummy_y, cv=kfold
```

```
Epoch 1/20
22/22 [==============================] - 0s 1ms/step - loss: 9.7627 - accurac
y: 0.2216
Epoch 2/20
22/22 [==============================] - 0s 1ms/step - loss: 3.1832 - accurac
y: 0.2909
Epoch 3/20
22/22 [==============================] - 0s 1ms/step - loss: 2.2075 - accurac
y: 0.3404
Epoch 4/20
22/22 [==============================] - 0s 2ms/step - loss: 1.7746 - accurac
y: 0.4084
Epoch 5/20
22/22 [==============================] - 0s 2ms/step - loss: 1.3294 - accurac
y: 0.4993
Epoch 6/20
22/22 [==============================] - 0s 1ms/step - loss: 1.1060 - accurac
y: 0.5104
```

```
Epoch 7/20
22/22 [==============================] - 0s 1ms/step - loss: 1.0671 - accurac
y: 0.6102
Epoch 8/20
22/22 [==============================] - 0s 1ms/step - loss: 0.9191 - accurac
y: 0.6074
Epoch 9/20
22/22 [==============================] - 0s 1ms/step - loss: 0.8164 - accurac
y: 0.6619
Epoch 10/20
22/22 [==============================] - 0s 1ms/step - loss: 0.7483 - accurac
y: 0.7305
Epoch 11/20
22/22 [==============================] - 0s 1ms/step - loss: 0.6619 - accurac
y: 0.7649
Epoch 12/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5972 - accurac
y: 0.7858
Epoch 13/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5930 - accurac
y: 0.8123
Epoch 14/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5725 - accurac
y: 0.8115
Epoch 15/20
22/22 [==============================] - 0s 983us/step - loss: 0.4772 - accura
cy: 0.8559
Epoch 16/20
22/22 [==============================] - 0s 992us/step - loss: 0.4313 - accura
cy: 0.8792
Epoch 17/20
22/22 [==============================] - 0s 1ms/step - loss: 0.4235 - accurac
y: 0.8825
Epoch 18/20
22/22 [==============================] - 0s 994us/step - loss: 0.3864 - accura
cy: 0.8951
Epoch 19/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3203 - accurac
y: 0.9407
Epoch 20/20
22/22 [==============================] - 0s 984us/step - loss: 0.2998 - accura
cy: 0.9505
6/6 [==============================] - 0s 985us/step - loss: 3.0446 - accurac
y: 0.3000
Epoch 1/20
22/22 [==============================] - 0s 956us/step - loss: 8.3519 - accura
cy: 0.2242
Epoch 2/20
22/22 [==============================] - 0s 1ms/step - loss: 3.4763 - accurac
y: 0.3389
Epoch 3/20
22/22 [==============================] - 0s 1ms/step - loss: 2.2756 - accurac
y: 0.3927
Epoch 4/20
22/22 [==============================] - 0s 1ms/step - loss: 1.5508 - accurac
y: 0.4805
Epoch 5/20
22/22 [==============================] - 0s 1ms/step - loss: 1.3454 - accurac
y: 0.5501
Epoch 6/20
22/22 [==============================] - 0s 1ms/step - loss: 1.2797 - accurac
y: 0.5651
Epoch 7/20
22/22 [==============================] - 0s 1ms/step - loss: 1.0112 - accurac
```

```
y: 0.6400
Epoch 8/20
22/22 [==============================] - 0s 1ms/step - loss: 0.8030 - accurac
y: 0.7189
Epoch 9/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5746 - accurac
y: 0.7966
Epoch 10/20
22/22 [==============================] - 0s 883us/step - loss: 0.6111 - accura
cy: 0.7833
Epoch 11/20
22/22 [==============================] - 0s 907us/step - loss: 0.5339 - accura
cy: 0.8386
Epoch 12/20
22/22 [==============================] - 0s 886us/step - loss: 0.4401 - accura
cy: 0.8901
Epoch 13/20
22/22 [==============================] - 0s 894us/step - loss: 0.5636 - accura
cy: 0.8018
Epoch 14/20
22/22 [==============================] - 0s 857us/step - loss: 0.4469 - accura
cy: 0.8665
Epoch 15/20
22/22 [==============================] - 0s 991us/step - loss: 0.3599 - accura
cy: 0.9119
Epoch 16/20
22/22 [==============================] - 0s 983us/step - loss: 0.3339 - accura
cy: 0.9297
Epoch 17/20
22/22 [==============================] - 0s 996us/step - loss: 0.2711 - accura
cy: 0.9568
Epoch 18/20
22/22 [==============================] - 0s 1ms/step - loss: 0.2519 - accurac
y: 0.9562
Epoch 19/20
22/22 [==============================] - 0s 990us/step - loss: 0.2121 - accura
cy: 0.9855
Epoch 20/20
22/22 [==============================] - 0s 936us/step - loss: 0.1976 - accura
cy: 0.9791
6/6 [==============================] - 0s 994us/step - loss: 3.0082 - accurac
y: 0.3165
Epoch 1/20
22/22 [==============================] - 0s 973us/step - loss: 6.5288 - accura
cy: 0.2171
Epoch 2/20
22/22 [==============================] - 0s 1ms/step - loss: 2.7782 - accurac
y: 0.2577
Epoch 3/20
22/22 [==============================] - 0s 1ms/step - loss: 1.7732 - accurac
y: 0.3914
Epoch 4/20
22/22 [==============================] - 0s 1ms/step - loss: 1.4443 - accurac
y: 0.4608
Epoch 5/20
22/22 [==============================] - 0s 1ms/step - loss: 1.3159 - accurac
y: 0.5156
Epoch 6/20
22/22 [==============================] - 0s 984us/step - loss: 1.0569 - accura
cy: 0.5583
Epoch 7/20
22/22 [==============================] - 0s 1ms/step - loss: 1.0550 - accurac
y: 0.6079
Epoch 8/20
```

```
22/22 [==============================] - 0s 1ms/step - loss: 0.9140 - accurac
y: 0.6560
Epoch 9/20
22/22 [==============================] - 0s 1ms/step - loss: 0.7555 - accurac
y: 0.7307
Epoch 10/20
22/22 [==============================] - 0s 1ms/step - loss: 0.7269 - accurac
y: 0.7297
Epoch 11/20
22/22 [==============================] - 0s 1ms/step - loss: 0.6280 - accurac
y: 0.7840
Epoch 12/20
22/22 [==============================] - 0s 1ms/step - loss: 0.6283 - accurac
y: 0.8036
Epoch 13/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5324 - accurac
y: 0.8265
Epoch 14/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5077 - accurac
y: 0.8560
Epoch 15/20
22/22 [==============================] - 0s 985us/step - loss: 0.4486 - accura
cy: 0.8902
Epoch 16/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3926 - accurac
y: 0.9358
Epoch 17/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3816 - accurac
y: 0.9312
Epoch 18/20
22/22 [==============================] - 0s 954us/step - loss: 0.3242 - accura
cy: 0.9564
Epoch 19/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3215 - accurac
y: 0.9362
Epoch 20/20
22/22 [==============================] - 0s 1ms/step - loss: 0.2933 - accurac
y: 0.9354
6/6 [==============================] - 0s 1ms/step - loss: 2.8747 - accuracy:
0.3544
Epoch 1/20
22/22 [==============================] - 0s 992us/step - loss: 8.7003 - accura
cy: 0.2243
Epoch 2/20
22/22 [==============================] - 0s 1ms/step - loss: 3.1328 - accurac
y: 0.2783
Epoch 3/20
22/22 [==============================] - 0s 1ms/step - loss: 2.1028 - accurac
y: 0.3245
Epoch 4/20
22/22 [==============================] - 0s 1ms/step - loss: 1.5756 - accurac
y: 0.4152
Epoch 5/20
22/22 [==============================] - 0s 1ms/step - loss: 1.3347 - accurac
y: 0.4942
Epoch 6/20
22/22 [==============================] - 0s 1ms/step - loss: 1.0977 - accurac
y: 0.5758
Epoch 7/20
22/22 [==============================] - 0s 1000us/step - loss: 1.0335 - accur
acy: 0.5943
Epoch 8/20
22/22 [==============================] - 0s 1ms/step - loss: 0.8614 - accurac
y: 0.6763
```

```
Epoch 9/20
22/22 [==============================] - 0s 976us/step - loss: 0.7685 - accura
cy: 0.7148
Epoch 10/20
22/22 [==============================] - 0s 1ms/step - loss: 0.6692 - accurac
y: 0.7864
Epoch 11/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5924 - accurac
y: 0.8063
Epoch 12/20
22/22 [==============================] - 0s 966us/step - loss: 0.5502 - accura
cy: 0.8378
Epoch 13/20
22/22 [==============================] - 0s 1ms/step - loss: 0.5737 - accurac
y: 0.8201
Epoch 14/20
22/22 [==============================] - 0s 926us/step - loss: 0.4892 - accura
cy: 0.8710
Epoch 15/20
22/22 [==============================] - 0s 1ms/step - loss: 0.4142 - accurac
y: 0.9045
Epoch 16/20
22/22 [==============================] - 0s 922us/step - loss: 0.4273 - accura
cy: 0.8914
Epoch 17/20
22/22 [==============================] - 0s 944us/step - loss: 0.3666 - accura
cy: 0.9264
Epoch 18/20
22/22 [==============================] - 0s 898us/step - loss: 0.3616 - accura
cy: 0.9220
Epoch 19/20
22/22 [==============================] - 0s 920us/step - loss: 0.3205 - accura
cy: 0.9518
Epoch 20/20
22/22 [==============================] - 0s 926us/step - loss: 0.2651 - accura
cy: 0.9730
6/6 [==============================] - 0s 1ms/step - loss: 2.0605 - accuracy:
0.3797
Epoch 1/20
22/22 [==============================] - 0s 1ms/step - loss: 6.6385 - accurac
y: 0.2529
Epoch 2/20
22/22 [==============================] - 0s 1ms/step - loss: 3.3002 - accurac
y: 0.2869
Epoch 3/20
22/22 [==============================] - 0s 1ms/step - loss: 2.2598 - accurac
y: 0.3827
Epoch 4/20
22/22 [==============================] - 0s 1ms/step - loss: 1.5680 - accurac
y: 0.4756
Epoch 5/20
22/22 [==============================] - 0s 1ms/step - loss: 1.3596 - accurac
y: 0.5144
Epoch 6/20
22/22 [==============================] - 0s 1ms/step - loss: 1.0795 - accurac
y: 0.5901
Epoch 7/20
22/22 [==============================] - 0s 967us/step - loss: 0.8951 - accura
cy: 0.6590
Epoch 8/20
22/22 [==============================] - 0s 1ms/step - loss: 0.7758 - accurac
y: 0.6753
Epoch 9/20
22/22 [==============================] - 0s 968us/step - loss: 0.6115 - accura
```

```
cy: 0.7791
Epoch 10/20
22/22 [==============================] - 0s 944us/step - loss: 0.5326 - accura
cy: 0.8578
Epoch 11/20
22/22 [==============================] - 0s 982us/step - loss: 0.4658 - accura
cy: 0.8584
Epoch 12/20
22/22 [==============================] - 0s 899us/step - loss: 0.4256 - accura
cy: 0.8871
Epoch 13/20
22/22 [==============================] - 0s 922us/step - loss: 0.3642 - accura
cy: 0.9415
Epoch 14/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3000 - accurac
y: 0.9698
Epoch 15/20
22/22 [==============================] - 0s 1ms/step - loss: 0.2693 - accurac
y: 0.9663
Epoch 16/20
22/22 [==============================] - 0s 1ms/step - loss: 0.3086 - accurac
y: 0.9497
Epoch 17/20
22/22 [==============================] - 0s 960us/step - loss: 0.2598 - accura
cy: 0.9646
Epoch 18/20
22/22 [==============================] - 0s 1ms/step - loss: 0.2018 - accurac
y: 0.9961
Epoch 19/20
22/22 [==============================] - 0s 1ms/step - loss: 0.2066 - accurac
y: 0.9794
Epoch 20/20
22/22 [==============================] - 0s 981us/step - loss: 0.1567 - accura
cy: 0.9931
6/6 [==============================] - 0s 993us/step - loss: 2.9845 - accurac
y: 0.3544
```

In [ ]:

```
print("Baseline: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
Baseline: 34.10% (2.88%)
```

Report Cross-validated model

We created a new model with 1 additional hidden layer and larger tensor outputs which was test and validated through a 10-fold cross-validation. Features were standardized based on the Z distribution and extracted by PCA. For model hyperparameters a batch_size = 15 and epochs = 20 showed the best results. Our main metric of interest was accuracy with categorical cross entropy as our loss function.

The mean accuracy score across all 10-fold were 34.10% with a std = 2.88%. This is lower than our first model which was not cross-validated but only validated on a single split and tested on a final unseen split. This was expected due to CV being a stronger tool against overfitting. It is still better than chance level which is 25% 100*1/4 or see below chunk for exemplification of chance level. Right now we only have 396 samples which is quite small for a deep neural network. So by collecting additional data we could expect our classifier to improve significantly.

In [ ]:

```
import copy
```

```
chance_level = []
for i in range (100):
    test_labels_copy = copy.copy(y_equa)
    np.random.shuffle(test_labels_copy)
    hits_array = np.array(y_equa) == np.array(test_labels_copy)
    chance_level.append(float(np.sum(hits_array)) / len(y_equa))
np.mean(chance_level)
```

Out[ ]: 0.25

Portfolio 4, Methods 3 2021

Author: Sigurd Fyhn Sørensen

30-11-21

In [ ]:
```
import numpy as np
import pandas as pd
import scipy as sp
import matplotlib.pyplot as plt
```

Exercises and objectives

1) Use principal component analysis to improve the classification of subjective experience
2) Use logistic regression with cross-validation to find the optimal number of principal components

REMEMBER: In your report, make sure to include code that can reproduce the answers requested in the exercises below (**MAKE A KNITTED VERSION**)
REMEMBER: This is Assignment 4 and will be part of your final portfolio

EXERCISE 1 - Use principal component analysis to improve the classification of subjective experience

We will use the same files as we did in Assignment 3 The files `megmag_data.npy` and `pas_vector.npy` can be downloaded here (http://laumollerandersen.org/data_methods_3/megmag_data.npy) and here (http://laumollerandersen.org/data_methods_3/pas_vector.npy)
The function `equalize_targets` is supplied - this time, we will only work with an equalized data set. One motivation for this is that we have a well-defined chance level that we can compare against. Furthermore, we will look at a single time point to decrease the dimensionality of the problem

1) Create a covariance matrix, find the eigenvectors and the eigenvalues

```
i. Load `megmag_data.npy` and call it `data` using `np.load`.
You can use `join`, which can be imported from `os.path`, to
create paths from different string segments
```

In [ ]:
```
"""import requests
import io

response = requests.get('http://laumollerandersen.org/data_methods_3/megmag_d
response.raise_for_status()
data = np.load(io.BytesIO(response.content))
```

```
response = requests.get('http://laumollerandersen.org/data_methods_3/pas_vect
response.raise_for_status()
y = np.load(io.BytesIO(response.content))"""
```

Out[ ]:
```
"import requests\nimport io\n\nresponse = requests.get('http://laumolleranders
en.org/data_methods_3/megmag_data.npy')\nresponse.raise_for_status()\ndata = n
p.load(io.BytesIO(response.content))\n\n\nresponse = requests.get('http://laum
ollerandersen.org/data_methods_3/pas_vector.npy')\nresponse.raise_for_status()
\ny = np.load(io.BytesIO(response.content))"
```

In [ ]:
```
data = np.load("/Users/sigurd/Downloads/megmag_data.npy")
y = np.load("/Users/sigurd/Downloads/pas_vector.npy")
```

ii. Equalize the number of targets in `y` and `data` using `equalize_targets`

In [ ]:
```
def equalize_targets(data, y):
    np.random.seed(7)
    targets = np.unique(y)
    counts = list()
    indices = list()
    for target in targets:
        counts.append(np.sum(y == target))
        indices.append(np.where(y == target)[0])
    min_count = np.min(counts)
    first_choice = np.random.choice(indices[0], size=min_count, replace=False
    second_choice = np.random.choice(indices[1], size=min_count, replace=Fals
    third_choice = np.random.choice(indices[2], size=min_count, replace=False
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=Fals

    new_indices = np.concatenate((first_choice, second_choice,
                                  third_choice, fourth_choice))
    new_y = y[new_indices]
    new_data = data[new_indices, :, :]

    return new_data, new_y

data_equa, y_equa = equalize_targets(data, y)
```

iii. Construct `times=np.arange(-200, 804, 4)` and find the index corresponding to 248 ms - then reduce the dimensionality of `data` from three to two dimensions by only choosing the time index corresponding to 248 ms (248 ms was where we found the maximal average response in Assignment 3)

In [ ]:
```
times = np.arange(-200, 804, 4)

data_equa_reduc = data_equa[:,:,np.argwhere(times == 248).item()]
data_equa_reduc.shape
```

Out[ ]:
```
(396, 102)
```

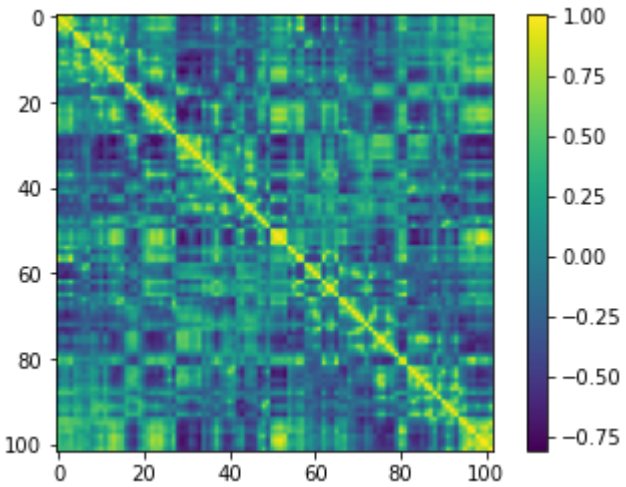
iv. Scale the data using `StandardScaler`

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
data_equa_reduc_std = sc.fit_transform(data_equa_reduc)
```

v. Calculate the sample covariance matrix for the sensors (you can use `np.cov`) and plot it (either using `plt.imshow` or `sns.heatmap` (`import seaborn as sns`))

In [ ]:
```
cov = np.cov(data_equa_reduc_std.T)
plt.imshow(cov)
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x7ff130d501c0>
```



vi. What does the off-diagonal activation imply about the independence of the signals measured by the 102 sensors?

The covariance values around the diagonal (sensors with close index numbers) seem to have a high covariance (light green). If sensors were placed in a numerical order those with close index number would be close to one another. Following the assumption of sensor location the covariance matrix makes a lot of sense. Sensors located close to each other record similar activity and therefore share covariance.

vii. Run `np.linalg.matrix_rank` on the covariance matrix - what integer value do you get? (we'll use this later)

In [ ]:
```
e = np.linalg.matrix_rank(cov)
e
```

Out[ ]:
```
97
```

viii. Find the eigenvalues and eigenvectors of the covariance matrix using `np.linalg.eig` - note that some of the numbers returned are complex numbers, consisting of a real and an imaginary part (they have a *j* next to them). We are going to ignore this by only looking at the real parts of the eigenvectors and -values. Use `np.real` to retrieve only the real parts

In [ ]:
```
eigen_value, eigen_vec  = np.linalg.eig(cov)
#remove complex numbers
real_eigen_val = np.real(eigen_value)
real_eigen_vec = np.real(eigen_vec)
```

2) Create the weighting matrix W and the projected data, Z

i. We need to sort the eigenvectors and eigenvalues according to the absolute values of the eigenvalues (use `np.abs` on the eigenvalues).

In [ ]:

```
real_eigen_val = np.abs(real_eigen_val)
```

ii. Then, we will find the correct ordering of the indices and create an array, e.g. `sorted_indices` that contains these indices. We want to sort the values from highest to lowest. For that, use `np.argsort`, which will find the indices that correspond to sorting the values from lowest to highest. Subsequently, use `np.flip`, which will reverse the order of the indices.

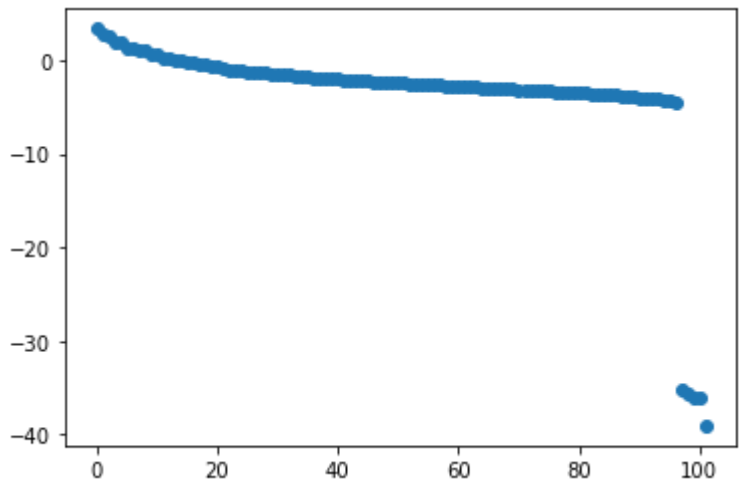
In [ ]:
```
indx = np.argsort(real_eigen_val)
indx = np.flip(indx)
```

iii. Finally, create arrays of sorted eigenvalues and eigenvectors using the `sorted_indices` array just created. For the eigenvalues, it should like this `eigenvalues = eigenvalues[sorted_indices]` and for the eigenvectors: `eigenvectors = eigenvectors[:, sorted_indices]`

In [ ]:
```
real_eigen_val = real_eigen_val[indx]
```

iv. Plot the log, `np.log`, of the eigenvalues, `plt.plot(np.log(eigenvalues), 'o')` - are there some values that stand out from the rest? In fact, 5 (noise) dimensions have already been projected out of the data - how does that relate to the matrix rank (Exercise 1.1.vii)

In [ ]:
```
plt.plot(np.log(real_eigen_val), 'o')
plt.show()


np.argwhere(np.log(real_eigen_val) == np.log(real_eigen_val).min())
```



Out[ ]:
```
array([[101]])
```

As shown with both e = np.linalg.matrix_rank(cov) and np.argwhere(np.log(real_eigen_val) == np.log(real_eigen_val).min()) == 101. So this is the index number for the lowest value on the log scale and original scale. The last five eigenvalues are now so small they barely explain any variance. The amount of variance they acutaly explain is already explained by the other eigenvectors. We have a matrix rank of 97, ie. we can have 97 unique eigenvectors with their own explaining any more than that will only explain that which is already explained.

We've 102 features and 102-97 = 5, ie. we must have 5 vectors doing no new explaining and therefore their low eigenvalue.

np.log(1e-15)

v. Create the weighting matrix, `W` (it is the sorted eigenvectors)

In [ ]:
```
sort_real_eigen_vec = real_eigen_vec[:,indx]
```

vi. Create the projected data, `Z` , $Z = XW$ - (you can check you did everything right by checking whether the X you get from $X = ZW^T$ is equal to your original X, `np.isclose` may be of help)

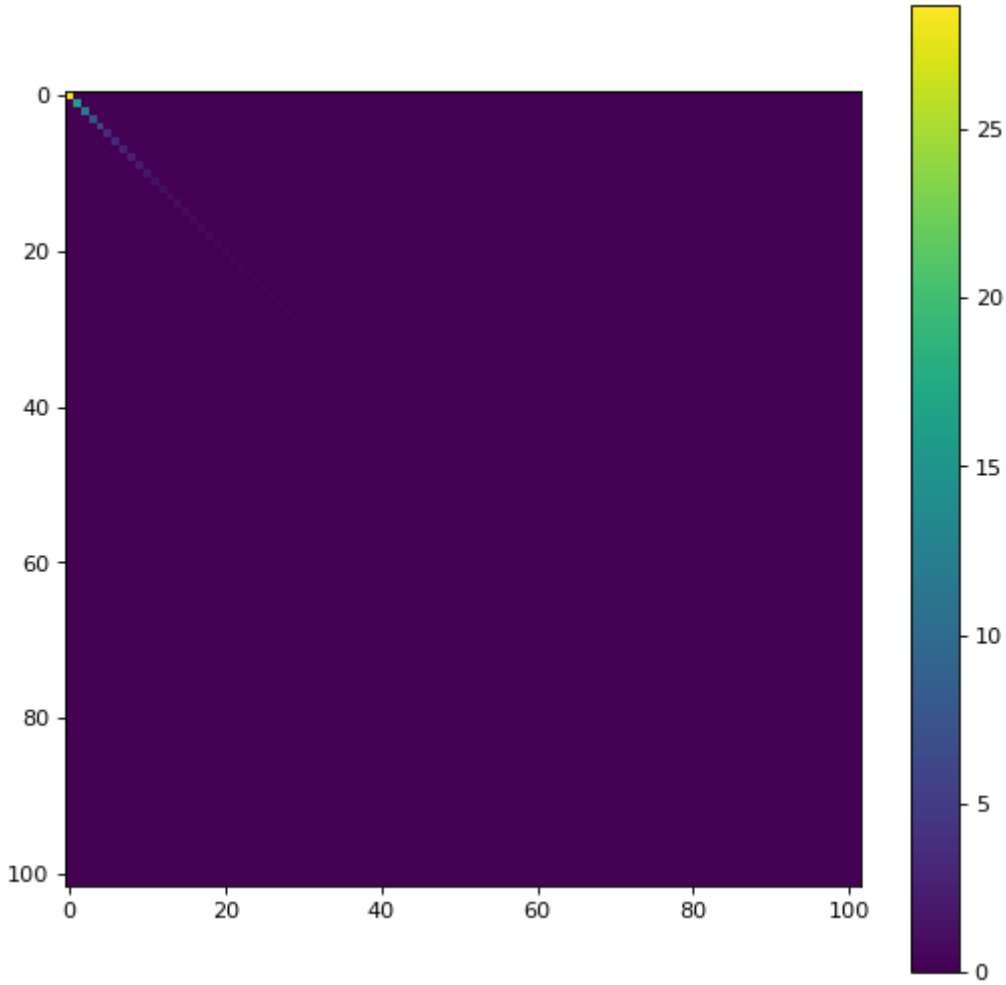
In [ ]:
```
X_new = data_equa_reduc_std @ sort_real_eigen_vec
X_old = X_new @ sort_real_eigen_vec.T

np.isclose( data_equa_reduc_std, X_old )
```

Out[ ]:
```
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       ...,
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True]])
```

vii. Create a new covariance matrix of the principal components (n=102) - plot it! What has happened off-diagonal and why?

In [ ]:
```
from matplotlib.pyplot import figure

figure(figsize=(8, 8), dpi=80)

cov2 = np.cov(X_new.T)
plt.imshow(cov2)
plt.colorbar()

print(X_new)
```

```
[[-1.36356497e+00  7.73472913e-01 -3.77969254e+00 ...  1.02613193e-09
   1.02613207e-09  1.32908514e-09]
 [ 6.48447356e+00  2.80100060e+00  3.13556004e+00 ...  3.13545317e-09
   3.13545303e-09  3.78206968e-09]
 [ 5.14241649e+00  4.77568372e+00  3.02071757e+00 ... -5.73524582e-10
  -5.73525248e-10 -6.60052108e-10]
 ...
 [ 4.80074660e+00  4.18979694e+00  6.59773262e-01 ... -6.00444911e-10
  -6.00444911e-10 -7.59573807e-10]
 [ 2.00350207e+00 -1.41860866e-02 -3.46460258e+00 ... -8.26901780e-10
  -8.26901780e-10 -1.13380059e-09]
 [ 2.20807406e+00 -6.17523595e+00  7.68468211e+00 ...  3.57630243e-11
   3.57634684e-11  1.11151721e-10]]
```



Now we got no covariance except on the diagonal. Which makes sense since the point of the PCA is to remove covariance between features by making them orthogonal. Because our eigenvectors are sorted from high -> low eigenvalues our eigenvectors with most variance will be indexed first. Around index 20 the rest of the covariance/variance of the feature is so low that it doesn't show on the plot.

EXERCISE 2 - Use logistic regression with cross-validation to find the optimal number of principal components

1) We are going to run logistic regression with in-sample validation

In [ ]:
```
from sklearn.linear_model import LogisticRegressionCV, LogisticRegression
```

i. First, run standard logistic regression (no regularization) based on $Z_{d\times k}$ and `y` (the target vector). Fit (`.fit`) 102 models based on: $k = [1, 2, \ldots, 101, 102]$ and $d = 102$. For each fit get the classification accuracy, (`.score`), when applied to $Z_{d\times k}$ and y. This is an in-sample validation. Use the solver `newton-cg` if the default solver doesn't converge

In [ ]:
```
def n_pca():
    accuracy = []
    lr = LogisticRegression(solver = "newton-cg", random_state= 10)
    for i in range(102):
        i = i + 1
        lr.fit(X_new[:,range(i)], y_equa)
```

```
        score_lul = lr.score(X_new[:,range(i)], y_equa)
        print(score_lul)
        accuracy.append(score_lul)
    return(accuracy)
accuracy_n_feat = n_pca()
```

```
0.27525252525252525
0.3207070707070707
0.3333333333333333
0.351010101010101
0.3409090909090909
0.36363636363636365
0.3712121212121212
0.3661616161616162
0.3787878787878788
0.3787878787878788
0.3813131313131313
0.39141414141414144
0.3838383838383838
0.4116161616161616
0.42424242424242425
0.4393939393939394
0.4292929292929293
0.4393939393939394
0.43686868686868685
0.4444444444444444
0.45707070707070707
0.46464646464646464
0.4494949494949495
0.4621212121212121
0.46464646464646464
0.4772727272727273
0.48484848484848486
0.4797979797979798
0.4823232323232323
0.4898989898989899
0.494949494949495
0.5151515151515151
0.5151515151515151
0.5227272727272727
0.51010101010101
0.5176767676767676
0.5176767676767676
0.5227272727272727
0.5303030303030303
0.5378787878787878
0.5328282828282829
0.5303030303030303
0.5378787878787878
0.5176767676767676
0.5227272727272727
0.5277777777777778
0.5176767676767676
0.5151515151515151
0.5202020202020202
0.5252525252525253
0.5353535353535354
0.5378787878787878
0.5378787878787878
0.547979797979798
0.547979797979798
0.547979797979798
0.5378787878787878
0.5429292929292929
```

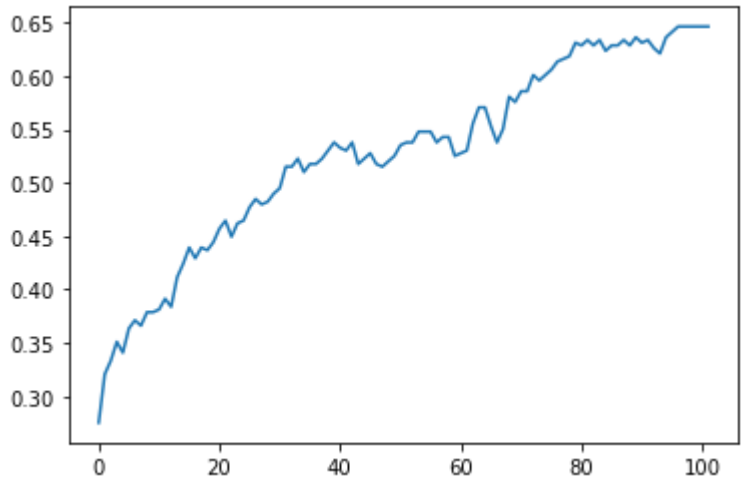
```
0.5429292929292929
0.5252525252525253
0.5277777777777778
0.5303030303030303
0.5555555555555556
0.5707070707070707
0.5707070707070707
0.553030303030303
0.5378787878787878
0.5505050505050505
0.5808080808080808
0.5757575757575758
0.5858585858585859
0.5858585858585859
0.601010101010101
0.5959595959595959
0.601010101010101
0.6060606060606061
0.6136363636363636
0.6161616161616161
0.6186868686868687
0.6313131313131313
0.6287878787878788
0.6338383838383839
0.6287878787878788
0.6338383838383839
0.6237373737373737
0.6287878787878788
0.6287878787878788
0.6338383838383839
0.6287878787878788
0.6363636363636364
0.6313131313131313
0.6338383838383839
0.6262626262626263
0.6212121212121212
0.6363636363636364
0.6414141414141414
0.6464646464646465
0.6464646464646465
0.6464646464646465
0.6464646464646465
0.6464646464646465
0.6464646464646465
```

In [ ]:
```
print("number of pca_values:" , len(accuracy_n_feat))
```

```
number of pca_values: 102
```

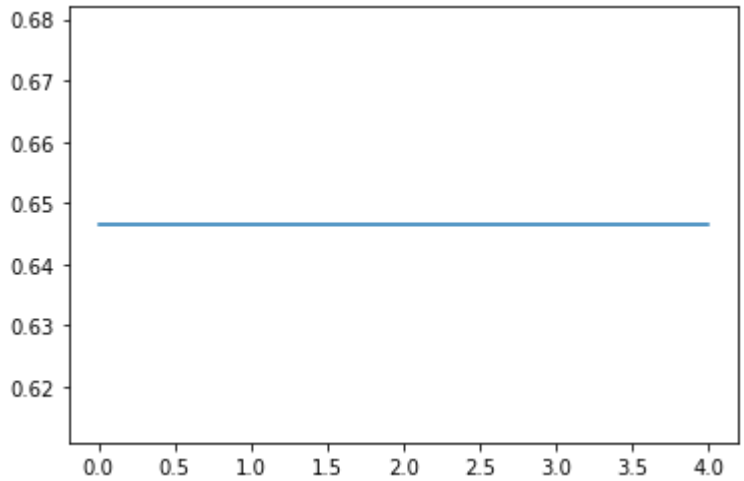
ii. Make a plot with the number of principal components on the *x*-axis and classification accuracy on the *y*-axis - what is the general trend and why is this so?

In [ ]:
```
plt.plot(accuracy_n_feat)
plt.show()
```



iii. In terms of classification accuracy, what is the effect of adding the five last components? Why do you think this is so?

In [ ]:
```
plt.plot(accuracy_n_feat[-6:-1])
plt.show()
```



Our new PCA feature matrix is sorted based by descending eigenvalues.
As we can see adding the last features with the comparatively lowest eigenvalues does not add anything to the model. This is due to our matrix rank = 97 as argued earlier any additional vectors beyond will do no new explaining.

2) Now, we are going to use cross-validation - we are using `cross_val_score` and `StratifiedKFold` from `sklearn.model_selection`

In [ ]:
```
from sklearn.model_selection import cross_val_score, StratifiedKFold
```

i. Define the variable: `cv = StratifiedKFold()` and run `cross_val_score` (remember to set the `cv` argument to your created `cv` variable). Use the same `estimator` in `cross_val_score` as in Exercise 2.1.i. Find the mean score over the 5 folds (the default of `StratifiedKFold`) for each k, $k = [1, 2, \ldots, 101, 102]$

In [ ]:
```
def n_pca_cv(X, y):
    cv = StratifiedKFold()
    accuracy = []
    lr = LogisticRegression(solver = "newton-cg", random_state= 10)
    for i in range(102):
```

```
        i = i + 1
        lr.fit(X[:,range(i)], y)
        score_lul = cross_val_score(lr ,X[:,range(i)], y, cv = cv)
        print(np.mean(score_lul))
        accuracy.append(np.mean(score_lul))
    return(accuracy)


accuracy_n_feat_cv = n_pca_cv(X = X_new, y = y_equa)
```

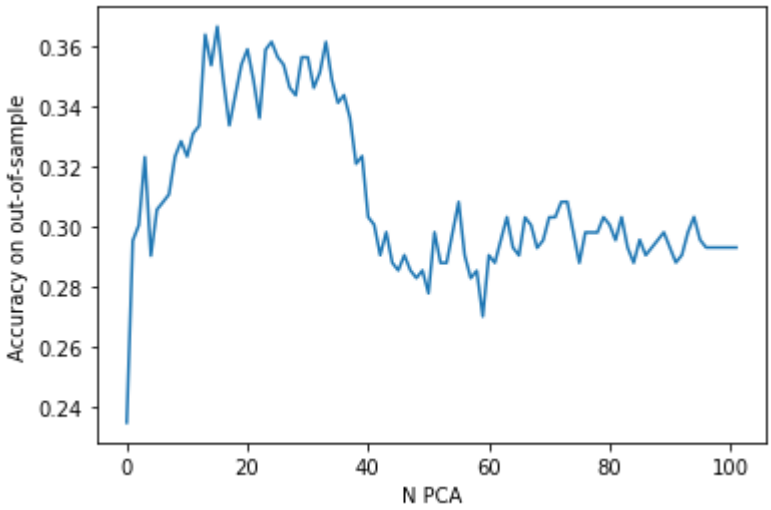
```
0.23474683544303798
0.2952215189873418
0.3003481012658228
0.3229746835443038
0.2901898734177215
0.30550632911392406
0.30797468354430374
0.3105696202531646
0.323132911392405
0.32819620253164555
0.3231645569620253
0.3307911392405064
0.3333544303797469
0.36360759493670886
0.3534493670886076
0.36629746835443033
0.3486392405063291
0.3334810126582279
0.3435759493670886
0.3536708860759494
0.3587341772151899
0.34854430379746837
0.33591772151898736
0.35860759493670885
0.36123417721518986
0.3561708860759494
0.3536075949367089
0.3459810126582279
0.34348101265822784
0.35610759493670885
0.35607594936708864
0.3459493670886076
0.3510126582278481
0.3611708860759494
0.34854430379746837
0.34098101265822783
0.34351265822784816
0.33591772151898736
0.3207278481012658
0.3232594936708861
0.3030379746835443
0.30047468354430384
0.29037974683544304
0.298006329113924
0.28784810126582283
0.28531645569620256
0.29037974683544304
0.28531645569620256
0.2827848101265823
0.28528481012658224
0.27772151898734176
0.2979746835443038
0.2878164556962025
0.28781645569620257
```

```
0.2979430379746836
0.3080379746835443
0.2903481012658228
0.28275316455696203
0.2851898734177215
0.2700316455696202
0.2903481012658228
0.28784810126582283
0.2954430379746836
0.3030063291139241
0.29291139240506325
0.2902848101265823
0.3029113924050633
0.3003481012658228
0.29278481012658225
0.2953164556962026
0.3029430379746835
0.3029746835443038
0.3080379746835443
0.30810126582278474
0.2979430379746836
0.28781645569620257
0.2979430379746836
0.29791139240506326
0.29791139240506326
0.30300632911392406
0.30050632911392405
0.2953481012658228
0.3029746835443038
0.29287974683544304
0.28784810126582283
0.29547468354430373
0.2903481012658228
0.29287974683544304
0.29537974683544305
0.2979746835443038
0.2929113924050633
0.28791139240506325
0.2904430379746835
0.298006329113924
0.30306962025316453
0.2954746835443038
0.29291139240506325
0.29291139240506325
0.29291139240506325
0.29291139240506325
0.29291139240506325
0.29291139240506325
```

ii. Make a plot with the number of principal components on the *x*-axis and classification accuracy on the *y*-axis - how is this plot different from the one in Exercise 2.1.ii?

In [ ]:

```
plt.plot(accuracy_n_feat_cv)
plt.xlabel("N PCA")
plt.ylabel("Accuracy on out-of-sample")
plt.show()
```



In exercise 2.1.ii we saw that for each additional feature added we got a better and better accuracy rating. Whether this fit can generalize beyond in-sample prediction is tested now. As illustrated by our plot adding additional PCA features only improves our models generalizability up until 40 added features with the best accuracy around 15 features. Beyond 40 features we're overfitting the training data too much.

iii. What is the number of principal components, $k_{max\_accuracy}$, that results in the greatest classification accuracy when cross-validated?

In [ ]:
```
print("indx for max value/ number of PCA features:",np.argmax(accuracy_n_feat
```

```
indx for max value/ number of PCA features: 15
 value: 36.629746835443036 %
```

iv. How many percentage points is the classification accuracy increased with relative to the to the full-dimensional, d, dataset

In [ ]:
```
data_equa_2d = np.reshape(data_equa, newshape = (data_equa.shape[0], -1))
cv = StratifiedKFold()
lr2 = LogisticRegression(random_state = 10, solver= "newton-cg")
data_equa_2d_std = sc.fit_transform(data_equa_2d)
lr2.fit(data_equa_2d_std, y_equa)
acc_all_data = np.mean(cross_val_score(lr2, data_equa_2d_std, y_equa, cv = cv
print("accuracy score all data:", acc_all_data , "\n diff from PCA:",  np.max
```

```
accuracy score all data: 0.27528481012658224
 diff from PCA: 0.0910126582278481
```

Cross validated Logistic regression conducted on the entire data results in an accuracy score of 24.24%. And CV logistic regression on the PCA data is 9% points better compared to the model on the entire data.

v. How do the analyses in Exercises 2.1 and 2.2 differ from one another? Make sure to comment on the differences in optimization criteria.

Exercise 2.1 is validated on the training data, this is prone to overfitting, and you can therefore not be sure that your result can generalize. Exercise 2.2 is validated on out-of-sample data. This makes sure that our model can give good accuracies on unseen data. (Which is what we want in ML). Our "real" optimization criteria has not changed. Logistic regression still only tries to find the best line based on the training data. If you talk of

"optimization" in terms of model reliability. The differences found in the explanation of Exercise 2.1 and Exercise 2.2 is used to underline this.

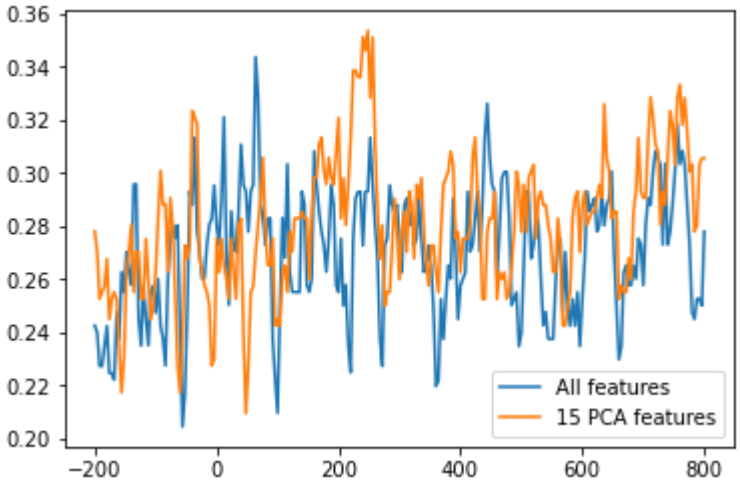
3) We now make the assumption that $k_{max\_accuracy}$ is representative for each time sample (we only tested for 248 ms). We will use the PCA implementation from *scikit-learn*, i.e. import `PCA` from `sklearn.decomposition` .

i. For **each** of the 251 time samples, use the same estimator and cross-validation as in Exercises 2.1.i and 2.2.i. Run two analyses - one where you reduce the dimensionality to $k_{max\_accuracy}$ dimensions using `PCA` and one where you use the full data. Remember to scale the data (for now, ignore if you get some convergence warnings - you can try to increase the number of iterations, but this is not obligatory)

In [ ]:
```
from sklearn.decomposition import PCA

def n_feat_lr():
    score_pca = []
    score_big = []
    cv = StratifiedKFold()
    lr = LogisticRegression(solver = "newton-cg", random_state= 10)
    lr2 = LogisticRegression(solver = "newton-cg", random_state= 10)
    for i in range(251):
        #transform X
        sc = StandardScaler()
        X = sc.fit_transform(data_equa[:,:,i])
        pc = PCA(n_components = np.argmax(accuracy_n_feat_cv))
        X_pca = pc.fit_transform(X)
        #fit data
        lr.fit(X, y_equa)
        lr2.fit(X_pca, y_equa)

        score_big.append(np.mean(cross_val_score(lr, X, y_equa, cv = cv)))
        score_pca.append(np.mean(cross_val_score(lr2, X_pca, y_equa, cv = cv)


    return score_big, score_pca


score_all, score_pca = n_feat_lr()
```

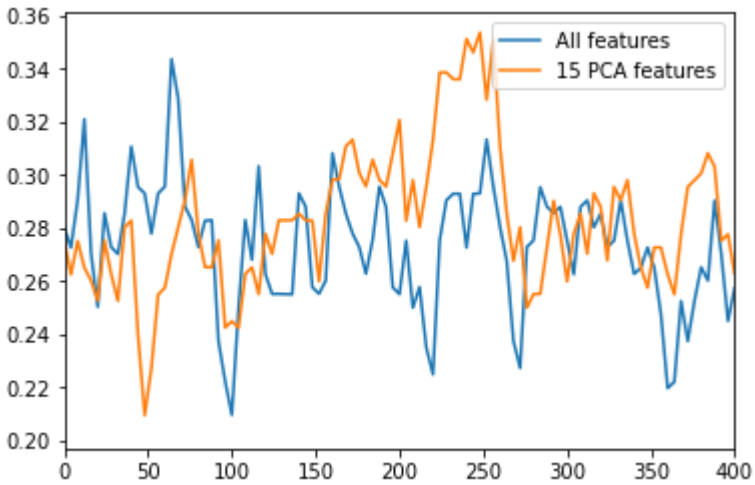
ii. Plot the classification accuracies for each time sample for the analysis with PCA and for the one without in the same plot. Have time (ms) on the *x*-axis and classification accuracy on the *y*-axis

In [ ]:
```
plt.plot(times ,score_all, label = "All features")
plt.plot(times ,score_pca, label = "15 PCA features")
plt.legend()
plt.show()
```



iii. Describe the differences between the two analyses - focus on the time interval between 0 ms and 400 ms - describe in your own words why the logistic regression performs better on the PCA-reduced dataset around the peak magnetic activity

In [ ]:
```
plt.plot(times ,score_all, label = "All features")
plt.plot(times ,score_pca, label = "15 PCA features")
plt.xlim(0,400)
plt.legend()
plt.show()
```



The PCA model performs better around peak magnetic activity because of the amount of data. Because we define peak magnetic activity as the max value of averaged activity across all samples we're also liable to see the most activity when all samples show high activation. When all samples are activated at the same time we're bound to have a lot more covariance between our features compared to when only few showed activity. So around 200-250ms we see an activation across all samples which result in large covariance which PCA manages to remove through eigenvector and eigenvalue calculation. This is why PCA performs better around the 200-250 ms mark compared to the non-pca all feature model with alot of covariance at those time points.