Assignment 1, Methods 3, 2021, autumn semester

Aleksander M. Wael

29/09/2021

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('politeness.csv') ## read in data

# Omit na
politeness <- na.omit(politeness)
```

Exercise 1 - describing the dataset and making some initial plots

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

Data comes from an experiment in which researches are interested in differences of voice pitch in formal and informal settings. The dataset contains 224 observations with 7 variables to describe the data. These variables are labeled: - Subject: The anonymized participant ID. - Gender: The gender of the participant, either F (female) or M (male). - Scenario: Categorized as 7 different scenarios of dialogue, labeled 1-7. - Attitude: The attitude of the scenario, either formal or informal. - Total duration: Duration of the scenario. - f0mn: The mean pitch of the participants voice in a scenario. - hiss_count: How many hisses the subject utters during the scenario.

i. Also consider whether any of the variables in *politeness* should be encoded as factors or have the factor encoding removed. Hint: `?factor`

```
# First 4 variables are
politeness$subject <- as.factor(politeness$subject)
politeness$gender <- as.factor(politeness$gender)
politeness$scenario <- as.factor(politeness$scenario)
politeness$attitude <- as.factor(politeness$attitude)
politeness$total_duration <- as.numeric(politeness$total_duration)
politeness$f0mn <- as.numeric(politeness$f0mn)
politeness$hiss_count <- as.numeric(politeness$hiss_count)

summary(lm(total_duration ~ attitude, data = politeness))
```

```
##
## Call:
## lm(formula = total_duration ~ attitude, data = politeness)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -22.67 -17.99 -11.75  16.47  76.96
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  25.1444     2.4072  10.446   <2e-16 ***
## attitudepol  -0.7275     3.3883  -0.215     0.83
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 24.67 on 210 degrees of freedom
## Multiple R-squared:  0.0002195,  Adjusted R-squared:  -0.004541
## F-statistic: 0.0461 on 1 and 210 DF,  p-value: 0.8302
```

I encode the first 4 variables as factors because i want to treat them as categorical variables.

2. 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

```
# Make subset of data to only include F1
scenario_integer_df <- politeness %>%
  filter(subject == "F1")

scenario_factor_df <- politeness %>%
  filter(subject == "F1")

# Treat subject as integer
# I already changed scenario to factor previously, so will change it in the integer_df
scenario_integer_df$scenario <- as.integer(scenario_integer_df$scenario)

# Make model with the different encodings
model_as_integer <- lm(f0mn ~ scenario, data = scenario_integer_df)
model_as_factor <- lm(f0mn ~ scenario, data = scenario_factor_df)
```

i. Include the model matrices, X from the General Linear Model, for these two models in your report and describe the different interpretations of *scenario* that these entail

```
# Include model matrix
model.matrix(model_as_integer)
```

```
##    (Intercept) scenario
## 1            1        1
## 2            1        1
## 3            1        2
## 4            1        2
## 5            1        3
## 6            1        3
## 7            1        4
## 8            1        4
## 9            1        5
## 10           1        5
## 11           1        6
## 12           1        6
## 13           1        7
## 14           1        7
## attr(,"assign")
## [1] 0 1
```

```
model.matrix(model_as_factor)
```

```
##    (Intercept) scenario2 scenario3 scenario4 scenario5 scenario6 scenario7
## 1            1         0         0         0         0         0         0
## 2            1         0         0         0         0         0         0
## 3            1         1         0         0         0         0         0
## 4            1         1         0         0         0         0         0
## 5            1         0         1         0         0         0         0
## 6            1         0         1         0         0         0         0
## 7            1         0         0         1         0         0         0
## 8            1         0         0         1         0         0         0
## 9            1         0         0         0         1         0         0
## 10           1         0         0         0         1         0         0
## 11           1         0         0         0         0         1         0
## 12           1         0         0         0         0         1         0
## 13           1         0         0         0         0         0         1
## 14           1         0         0         0         0         0         1
## attr(,"assign")
## [1] 0 1 1 1 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$scenario
## [1] "contr.treatment"
```

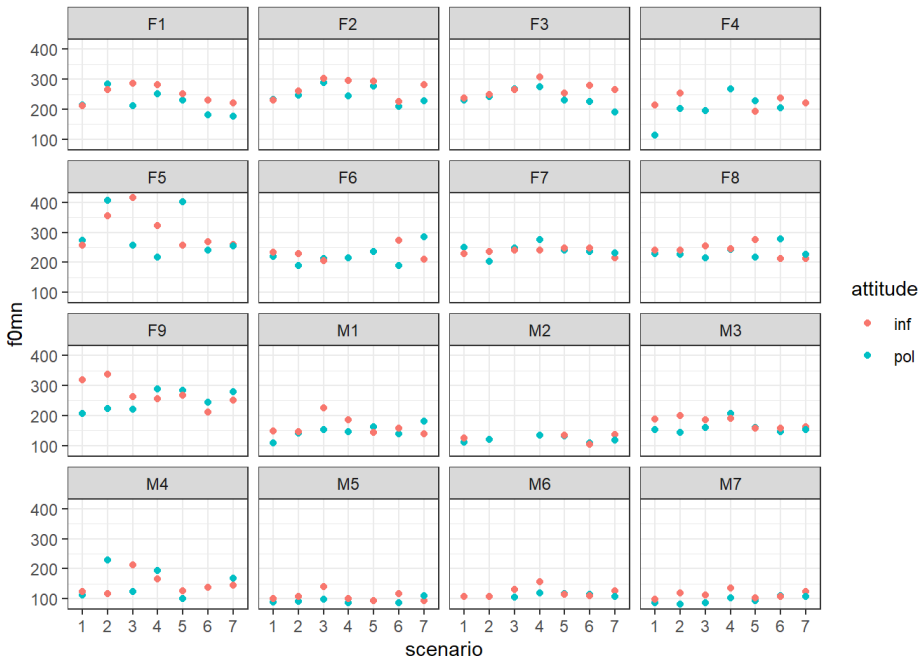
When scenario is an integer, it means the scenario number is treated as a continuous variable. The interpretation would be that scenario 7 is 7 x scenario 1, and instead of talking about “what” scenario has an effect you are modelling “how much” scenario, which doesn’t make sense. This can be seen in the model matrix where the scenario column increases by scenario level. When treating scenario as a factor, we acknowledge that scenario is an independent category and should be modeled as such. In the model matrix, there is a column for each scenario, and the rows which correspond to the specific scenario have an entry of 1. Therefore, a row can’t be both e.g. scenario 2 and 3, as is true in the experimental design.

ii. Which coding of *scenario*, as a factor or not, is more fitting?

As mentioned, we are interested in treating each scenario as separate entities that aren’t ranked based on their numeric values, i.e. scenario 7 is not larger than scenario 5; it is simply a different condition to compare to.

3. 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*

```
# Plotting
ggplot(data = politeness, aes(x = scenario, y = f0mn, color = attitude)) +
  geom_point() +
  facet_wrap(~subject)+
  theme_bw()
```



i. Describe the differences between subjects

Some subjects have greater differences between attitude conditions, and the baseline f0mn also appears to be varying. There is between-subject variance, which we should like to account for in our model.

Exercise 2 - comparison of models

For this part, make sure to have `lmerTest` installed.
You can install it using `install.packages("lmerTest")` and load it using `library(lmerTest)`
`lmer` is used for multilevel modelling

```
mixed.model <- lmer(formula=..., data=...)
example.formula <- formula(dep.variable ~ first.level.variable + (1 | second.level.variable))
```

1) Build four models and do some comparisons

i. a single level model that models *f0mn* as dependent on *gender*

```
m1 <- lm(f0mn ~ gender, data = politeness)
summary(m1)
```

```
##
## Call:
## lm(formula = f0mn ~ gender, data = politeness)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -134.283  -24.928   -6.783   20.517  168.217
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  247.583      3.588   69.01   <2e-16 ***
## genderM     -115.821      5.476  -21.15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.46 on 210 degrees of freedom
## Multiple R-squared:  0.6806, Adjusted R-squared:  0.679
## F-statistic: 447.4 on 1 and 210 DF,  p-value: < 2.2e-16
```

ii. a two-level model that adds a second level on top of i. where unique intercepts are modelled for each *scenario*

```
m2 <- lmerTest::lmer(f0mn ~ gender + (1|scenario), data = politeness, REML = FALSE)
summary(m2)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender + (1 | scenario)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2162.3   2175.7  -1077.1   2154.3      208
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.2617 -0.6192 -0.1537  0.4899  4.2318
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  scenario (Intercept)   71.82   8.475
##  Residual             1471.08  38.355
## Number of obs: 212, groups:  scenario, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  247.768      4.735   11.793   52.32  2.5e-15 ***
## genderM     -115.870      5.324  205.219  -21.76  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##         (Intr)
## genderM -0.483
```

```
summary(lmerTest::lmer(f0mn ~ gender + (1|scenario), data = politeness, REML = FALSE))
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender + (1 | scenario)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2162.3   2175.7  -1077.1   2154.3      208
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.2617 -0.6192 -0.1537  0.4899  4.2318
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  scenario (Intercept)   71.82   8.475
##  Residual             1471.08  38.355
## Number of obs: 212, groups:  scenario, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  247.768      4.735   11.793   52.32  2.5e-15 ***
## genderM     -115.870      5.324  205.219  -21.76  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##         (Intr)
## genderM -0.483
```

iii. a two-level model that only has *subject* as an intercept

```
m3 <- lmerTest::lmer(f0mn ~ gender + (1|subject), data = politeness, REML = FALSE)
summary(m3)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender + (1 | subject)
##    Data: politeness
##
##      AIC      BIC   logLik deviance df.resid
##   2112.0   2125.5  -1052.0   2104.0      208
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.2405 -0.5471 -0.1431  0.4360  3.8443
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept)  511.2   22.61
##  Residual             1026.7   32.04
## Number of obs: 212, groups:  subject, 16
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  246.547      8.083   15.984  30.501 1.36e-15 ***
## genderM     -115.193     12.239   16.076  -9.412 6.08e-08 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##         (Intr)
## genderM -0.660
```

iv. a two-level model that models intercepts for both *scenario* and *subject*

```
m4 <- lmerTest::lmer(f0mn ~ gender + (1|scenario) + (1|subject), data = politeness, REML = FALSE)
summary(m4)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender + (1 | scenario) + (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 (Intercept)  89.36    9.453
##  Residual             940.25   30.664
## Number of obs: 212, groups:  subject, 16; scenario, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  246.778      8.829   19.248  27.952  < 2e-16 ***
## genderM     -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)
## genderM -0.604
```

v. which of the models has the lowest residual standard deviation, also compare the Akaike Information Criterion `AIC` ?

```
# Finding residual standard deviation
tibble(sigma(m1), sigma(m2), sigma(m3), sigma(m4))
```

sigma(m1) <dbl>	sigma(m2) <dbl>	sigma(m3) <dbl>	sigma(m4) <dbl>
39.46268	38.3546	32.04227	30.66355

1 row

```
# AIC values
tibble(AIC(m1), AIC(m2), AIC(m3), AIC(m4))
```

AIC(m1) <dbl>	AIC(m2) <dbl>	AIC(m3) <dbl>	AIC(m4) <dbl>
2163.971	2162.257	2112.048	2105.176

1 row

```
anova(m2, m1, m3, m4)
```

	npar <dbl>	AIC <dbl>	BIC <dbl>	logLik <dbl>	deviance <dbl>	Chisq <dbl>	Df <dbl>	Pr(>Chisq) <dbl>
m1	3	2163.971	2174.041	-1078.986	2157.971	*NA*	*NA*	*NA*
m2	4	2162.257	2175.684	-1077.129	2154.257	3.713650	1	0.053969266
m3	4	2112.048	2125.474	-1052.024	2104.048	50.209453	0	*NA*
m4	5	2105.176	2121.958	-1047.588	2095.176	8.872474	1	0.002895025

4 rows

The model with gender as a predictor with random intercepts for both scenario and subject has the lowest resid. SD (30.66) and AIC score (2105.18) ($p < 0.05$).

vi. which of the second-level effects explains the most variance?

We see by comparing residual standard deviation and AIC scores of m2 (intercepts for scenario) and m3 (intercepts for subjects) to the “base” model (only predicted by gender) that random intercepts per subject explain more variance than intercepts per scenario. Using only random intercepts by subject as a second-level effect has a lower sigma value (32.04) and AIC score (2112.1) than by scenario.

2) Why is our single-level model bad?

The single level model ignores some important hierarchies in the data which we are well aware of exist. It disregards the fact that subjects might differ on baseline pitch, and that there may be differences across scenarios. Whilst it might explain some of the variance in the data, yes, it is to simple to be a complete answer.

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_

```
# Creating new df
politeness2 <- politeness %>%
  group_by(subject, gender) %>%
  summarize(mean_of_f0mn = mean(f0mn))
```

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

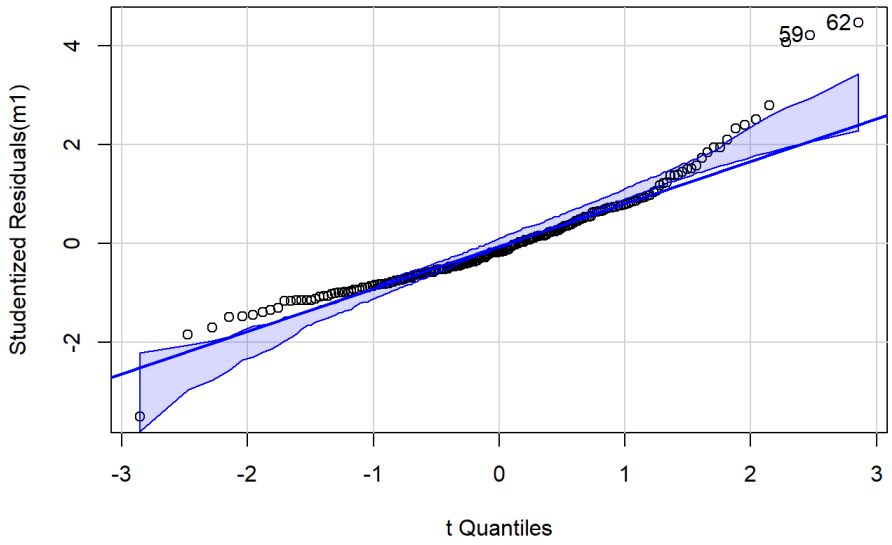
ii. build a single-level model that models *f0mn* as dependent on *gender* using this new dataset

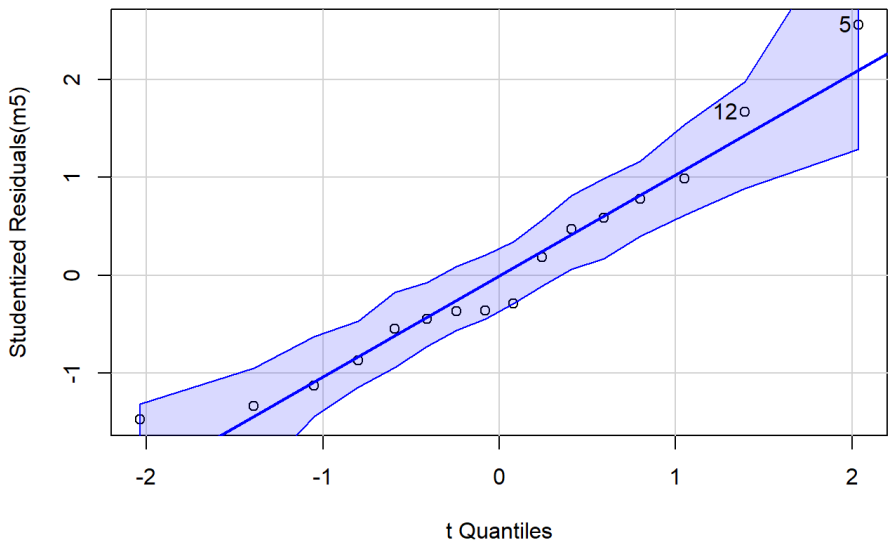
```
# Building new model
m5 <- lm(mean_of_f0mn ~ gender, data = politeness2)
summary(m5)
```

```
##
## Call:
## lm(formula = mean_of_f0mn ~ gender, data = politeness2)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -34.606 -15.493  -8.212  15.702  52.859
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  246.370      8.635  28.530 8.34e-14 ***
## genderM     -115.092     13.055  -8.816 4.35e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 25.91 on 14 degrees of freedom
## Multiple R-squared:  0.8474, Adjusted R-squared:  0.8365
## F-statistic: 77.72 on 1 and 14 DF,  p-value: 4.346e-07
```

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) fulfill the assumptions of the General Linear Model better?)

```
par(car::qqPlot(m1), car::qqPlot(m5))
```






```
## [[1]]
## NULL
##
## [[2]]
## NULL
```

```
tibble(sigma(m1), sigma(m5))
```

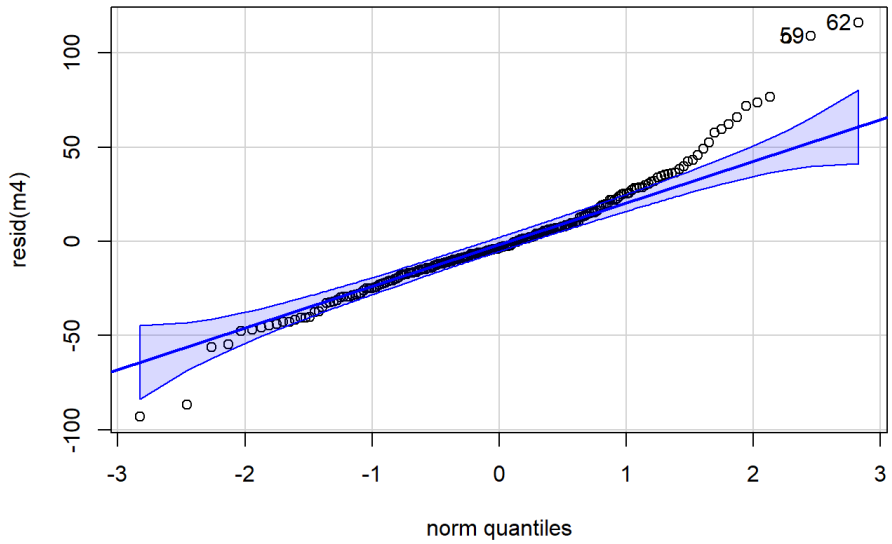
sigma(m1) <dbl>	sigma(m5) <dbl>
39.46268	25.906

1 row

The QQ-plot of the pooled f0 scores seems better than that of the individual scores. Pooling "pulls" observations to the mean, so it can fix some problems with outliers. This can also be seen by the lower sigma value for the model with the pooled scores. Although it fulfills the assumptions of the model better, it reduces the resolution of the data by removing the difference in f0 scores between subjects.

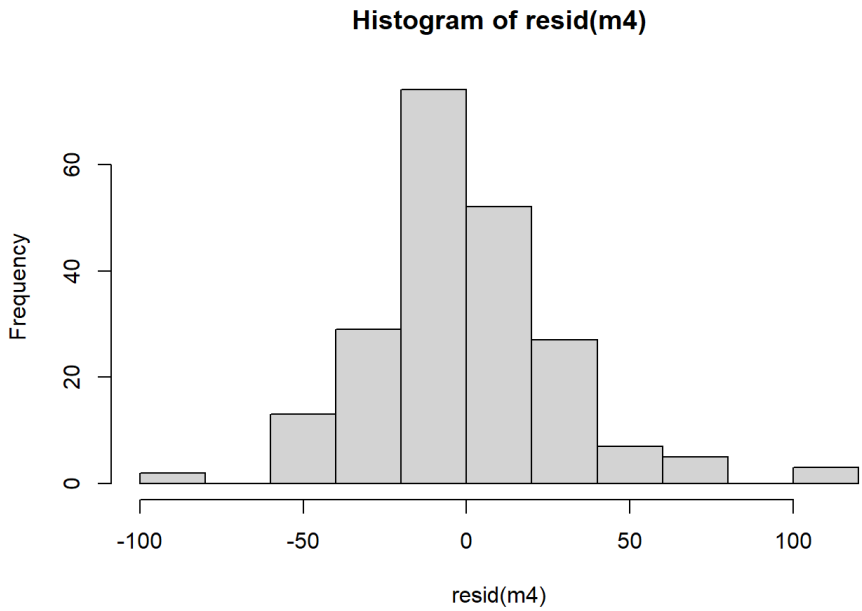
iv. Also make a quantile-quantile plot for the residuals of the multilevel model with two intercepts. Does it look alright?

```
# Plotting both qq plot and histogram of residuals
car::qqPlot(resid(m4))
```




```
## 62 59
## 59 56
```

```
hist(resid(m4))
```

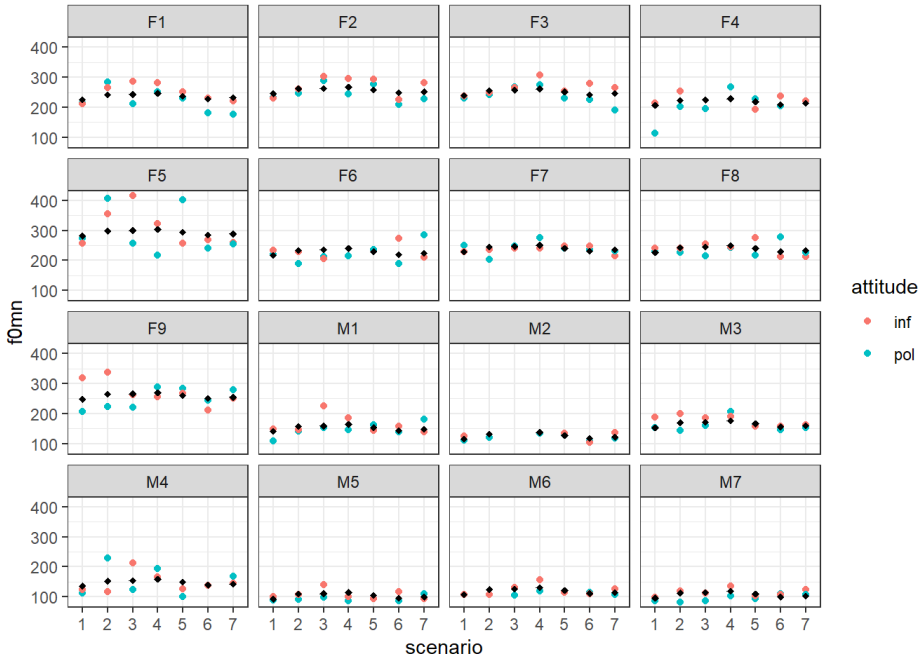


The residuals are a bit right-skewed, but it does look alright, perhaps a slight violation of assumption.

3) 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)

```
# Plotting
ggplot(data = politeness, aes(x = scenario, y = f0mn, color = attitude)) +
  geom_point() +
  geom_point(aes(x = scenario, y = fitted(m4)), color = "black", shape = 18)+
  facet_wrap(~subject)+
  theme_bw()
```



Exercise 3 - now with attitude

1) 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*

```
m6 <- lmerTest::lmer(f0mn ~ gender + attitude + (1|scenario) + (1|subject), data = politeness, REML = FALSE)
summary(m6)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender + attitude + (1 | scenario) + (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 (Intercept)  99.22    9.961
##  Residual             878.39   29.638
## Number of obs: 212, groups:  subject, 16; scenario, 7
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  254.408      9.117   21.800  27.904  < 2e-16 ***
## genderM     -115.447     12.161   16.000  -9.494 5.63e-08 ***
## attitudepol  -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) gendrM
## genderM     -0.583
## attitudepol -0.231  0.006
```

ii. make a separate model that besides the main effects of *attitude* and *gender* also include their interaction

```
m7 <- lmerTest::lmer(f0mn ~ gender * attitude + (1|scenario) + (1|subject), data = politeness, REML = FALSE)
summary(m7)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: f0mn ~ gender * attitude + (1 | scenario) + (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 (Intercept)  99.08    9.954
##  Residual             876.46   29.605
## Number of obs: 212, groups:  subject, 16; scenario, 7
##
## Fixed effects:
##                     Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)          255.632      9.289   23.556  27.521  < 2e-16 ***
## genderM             -118.251     12.841   19.922  -9.209 1.28e-08 ***
## attitudepol          -17.198      5.395  190.331  -3.188  0.00168 **
## genderM:attitudepol    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) gendrM atttdp
## genderM     -0.605
## attitudepol -0.299  0.216
## gndrM:tttdp  0.195 -0.323 -0.654
```

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

The model describes that, although pitch decreases in the polite attitude compared to the informal attitude, Korean men's pitch seem (in our sample) to decrease less than women's pitch, though the interaction is not significant.

2) 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.

```
# Running anova to quickly compare AIC values, making a df for sigmal values and one for SSR
anova(m4, m6, m7)
```

	npar	AIC	BIC	logLik	deviance	Chisq	Df	Pr(>Chisq)
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
m4	5	2105.176	2121.958	-1047.588	2095.176	*NA*	*NA*	*NA*
m6	6	2094.489	2114.628	-1041.244	2082.489	12.6867631	1	0.0003682532
m7	7	2096.034	2119.530	-1041.017	2082.034	0.4551491	1	0.4998998177

3 rows

```
tibble(sigma(m4), sigma(m6), sigma(m7))
```

sigma(m4)	sigma(m6)	sigma(m7)
<dbl>	<dbl>	<dbl>
30.66355	29.63771	29.60505

1 row

```
tibble(sum(resid(m4)^2), sum(resid(m6)^2), sum(resid(m7)^2))
```

sum(resid(m4)^2)	sum(resid(m6)^2)	sum(resid(m7)^2)
<dbl>	<dbl>	<dbl>

sum(resid(m4)^2) <dbl>	sum(resid(m6)^2) <dbl>	sum(resid(m7)^2) <dbl>
181913	169681.1	169305.6

1 row

3) 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
ii. what can you conclude about the effect of gender and attitude on pitch (if anything)?
iii. motivate why you would include separate intercepts for subjects and scenarios (if you think they should be included)
iv. describe the variance components of the second level (if any)
v. include a Quantile-Quantile plot of your chosen model

Findings based on politeness dataset by Winter & Grawunder

Note on dataset

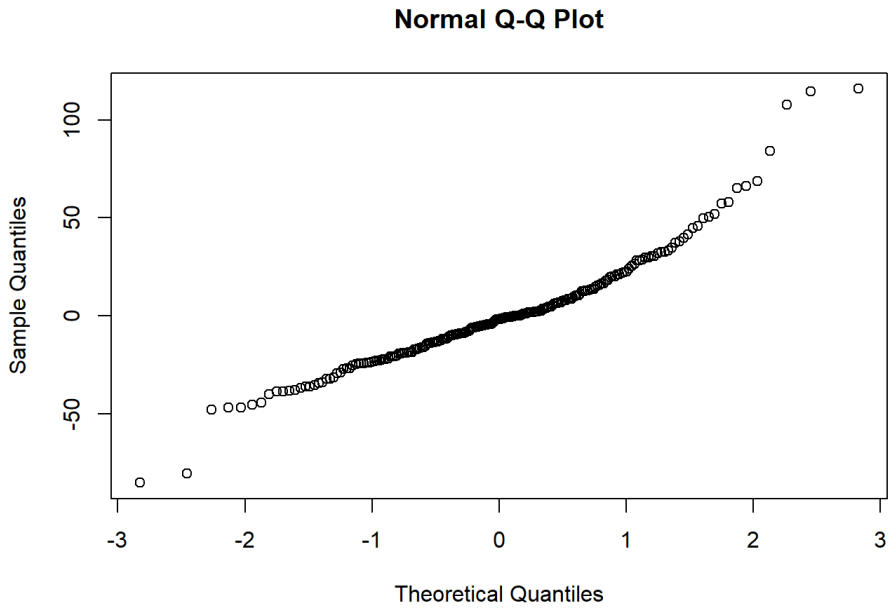
This dataset was collected to study the phonetic profile of Korean formal and informal speech registers (Winter & Grawunder, 2012). The dataset includes variables of: “Subject” - the anonymized participant ID; “Gender” - the gender of the participant, either F (female) or M (male); “Scenario” - categorized as 7 different scenarios of dialogue, labeled 1-7; “Attitude” - the attitude of the scenario, either formal or informal; “Total duration” - duration of the scenario; “f0mn” - the mean pitch of the participants voice in a scenario and “hiss_count” - how many hisses the subject utters during the scenario.

Building the model

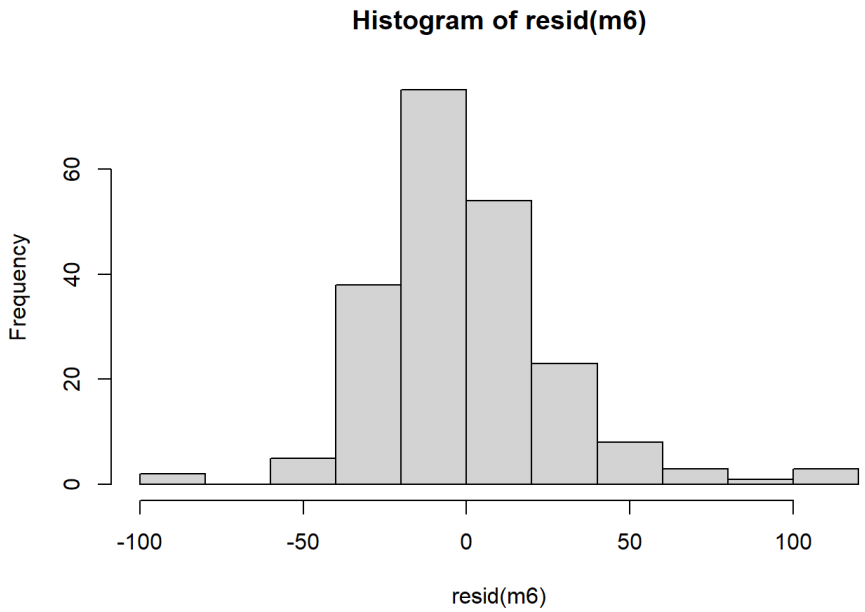
The variable of interest in the paper is mean pitch (f0mn), and I’d like to know what variables affect this variable. Therefore, it is the outcome variable of my model. Since it is established that pitch is highly correlated to biological sex, I have included gender as a main (fixed) effect. As the original paper hypothesized about pitch differences in formal and informal settings, attitude should of course also be included as a main effect. Furthermore, it is likely that subject have varying baseline pitch frequencies, which is the case for adding a random intercepts per subject. The same goes for the variable scenario, which indicates the situation in which the dialogue exchange takes place. The input in R then becomes f0mn ~ gender + attitude + (1 | scenario) + (1 | subject). The main effects are both significant at $p < 0.05$, and the model also has the lowest AIC-value of all tested models. There could be a case for adding the interaction between gender and attitude, as one might hypothesize whether there’s a difference in degree of pitch change between men and women across attitude, though this was tested and yielded a weaker model with no significant effect of the interaction and a higher AIC value, and as also not particularly relevant to the study.

Checking assumptions of normality of residuals

```
qqnorm(resid(m6))
```




```
hist(resid(m6))
```



The residuals follow a somewhat normal distribution, with a slight right skew (positive skew), which propose a challenge to my assumptions.

Results from model

The predictor “gender” has a significant effect ($p < 0.05$) on mean pitch, showing a decrease from 254.41 Hz for women to 138.96 Hz for men. In comparing polite and informal speech, there is a 14.82 Hz reduction in mean pitch going from the informal to the polite condition ($p < 0.05$). We also see the random intercepts explaining a good amount of the variance in the data, especially random intercepts by subject (score of 514.9).

Assignment 2 (Part 1), Methods 3, 2021, autumn semester

Aleksander Moeslund Wael

01/10/2021

Loading packages

```
pacman::p_load(tidyverse, lmerTest, lme4, gridExtra, dfoptim)
```

Exercise 1

1.1) Put the data from all subjects into a single data frame

```
files <- list.files(path = "experiment_2",
                    pattern = ".csv",
                    full.names = T)

df <- read_csv(files)
```

```
## Rows: 18131 Columns: 17
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (5): trial.type, task, target.type, obj.resp, subject
## dbl (12): pas, trial, jitter.x, jitter.y, odd.digit, target.contrast, target...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

1.2) Describe the data and construct extra variables from the existing variables

The dataset contains 18131 observations measured on 17 variables. Data from 29 subjects is included. The data originates from a study on measuring subjective experience in a visual task, where participants have to rate a visual stimulus based on the Perceptual Awareness Scale (PAS). Thorough description of the variables are in next task.

1.2.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.

Adding variable “correct” to display if subject was correct

```
# Adding empty variable
df <- df %>%
  mutate(obj.resp.2 = obj.resp)

# Renaming rows in obj.resp.2 to get same units as target.type
df$obj.resp.2 <- replace(df$obj.resp.2, df$obj.resp.2 == "e", "even")
df$obj.resp.2 <- replace(df$obj.resp.2, df$obj.resp.2 == "o", "odd")

# Adding value for correct and incorrect answers
df_correct <- df %>%
  filter(obj.resp.2 == target.type) %>%
  mutate(correct = "1")

# Joining with my df
df <- left_join(df, df_correct)
```

```
## Joining, by = c("trial.type", "pas", "trial", "jitter.x", "jitter.y", "odd.digit", "target.contrast", "target.frames", "c
ue", "task", "target.type", "rt.subj", "rt.obj", "even.digit", "seed", "obj.resp", "subject", "obj.resp.2")
```

```
# Remaining are NAs, so replace with 0
df$correct <- replace(df$correct, is.na(df$correct), "0")
```

1.2.ii. Describe what the following variables in the data frame contain, trial.type, pas, 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 variables 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: Indicates whether subject is doing the staircase task (first experiment) or the follow-up experiment. Should be class character, as it is a category. pas: Indicates subjects response to trial on the Perceptual Awareness Scale (PAS). It is treated as factor. trial: A numbered list for every trial the subject completes, i.e. presses e or o in either of the trial types., per subject. I should think character class for now (might change). target.contrast: The contrast between the background and the digit (target). Between 0-1, treated as numeric. cue: The specific cue pattern, will treat as character. task: Whether cue pattern is 2 (singles), 4 (pairs) or 8 (quadruplets) digits. Will treat as character. target.type: Whether target type is an odd or even number - will treat as character. rt.subj: Reaction time for response to PAS pr. trail - will treat as numeric. rt.obj: Reaction time for responding if target is even or odd - will treat as numeric. obj.resp: Subjects response to target is either even or odd - will treat as character. subject: Participant ID, ordered from 001. Treated as character/factor. correct: Whether subject answered correctly in the trail, 1 for correct and 0 for incorrect. Is logical (binary), NOTE: treated as a factor due to an error when conducting analysis.

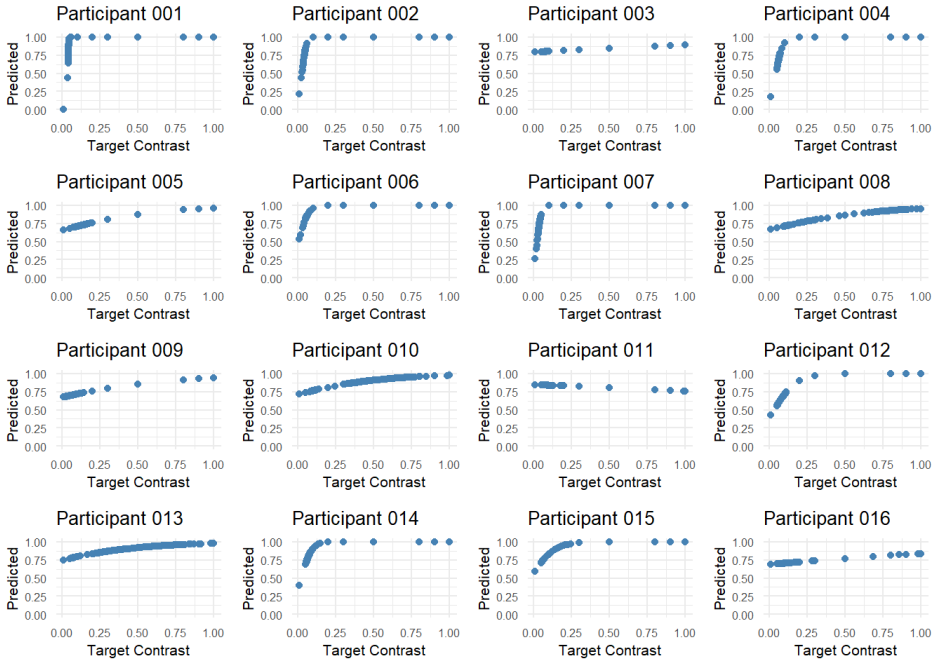
```
# Assigning variables to proper class
df$pas <- as.factor(df$pas)
df$trial <- as.character(df$trial)
df$target.contrast <- as.numeric(df$target.contrast)
df$cue <- as.character(df$cue)
df$rt.subj <- as.numeric(df$rt.subj)
df$rt.obj <- as.numeric(df$rt.obj)
df$target.contrast <- as.numeric(df$target.contrast)
df$correct <- as.factor(df$correct)
df$subject <- as.factor(df$subject)
```

1. 2. 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 enough data to plot the logistic functions?

```
# Making a df
staircase <- df %>%
  filter(df$trial.type == 'staircase')

# No pooling function that returns plot pr. participant
np_function <- function(i){
  dat <- staircase[which(staircase$subject == i),] # subsetting
  model <- glm(correct ~ target.contrast, family = 'binomial', data = dat) # running per participant
  fitted <- model$fitted.values # fitted values
  plot_dat <- data.frame(cbind(fitted, 'target.contrast' = dat$target.contrast)) # append to df with my variables

  plot <- ggplot(plot_dat, aes(x = target.contrast, y = fitted))+
    geom_point(color = 'steelblue') +
    xlab('Target Contrast') +
    ylab('Predicted') +
    ylim(c(0,1))+
    ggtitle(paste0('Participant ', as.character(i))) +
    theme_minimal() +
    theme(plot.title = element_text(size = 10), axis.title=element_text(size = 8), axis.text=element_text(size=6))

  return(plot)
}

# Applying function to subjects and plotting with grid.arrange
subjects <- c("001", "002", "003", "004", "005", "006", "007", "008", "009", "010", "011", "012", "013", "014", "015", "016"
)
plots <- lapply(subjects, FUN=np_function)
```

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

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

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

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

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

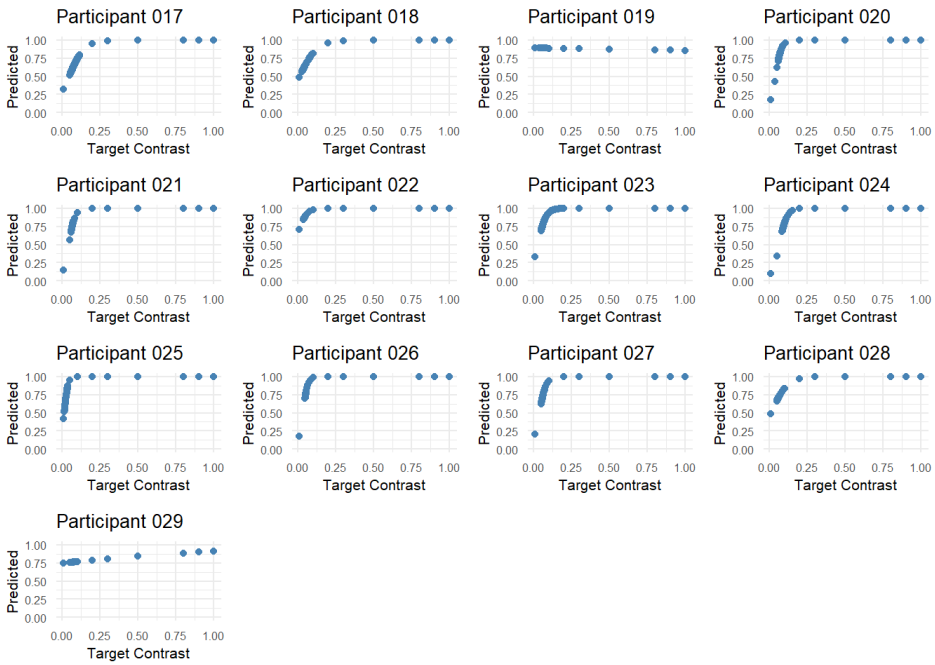
```
do.call(grid.arrange,  plots)
```




```
# With remaining participants
subjects <- c("017", "018", "019", "020", "021", "022", "023", "024", "025", "026", "027", "028", "029")
plots <- lapply(subjects, FUN=np_function)
```

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

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

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

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

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

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

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

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

```
do.call(grid.arrange,  plots)
```



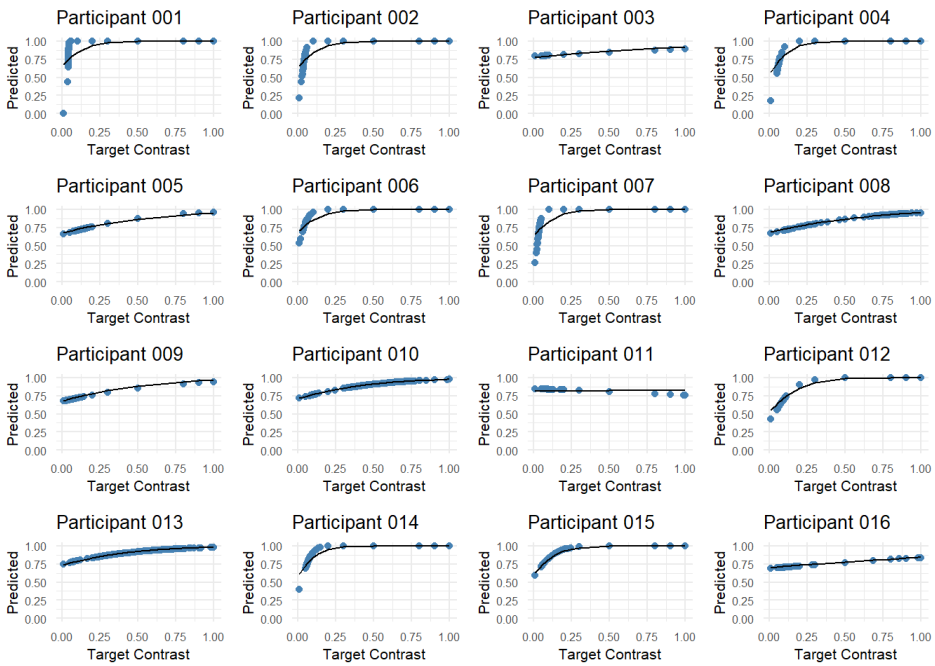
We have ~180 rows pr subject, which is pretty good, but it isn't a meaningful analysis. The no pooling analysis does not inform us about the population, but shows that there is between-subject variance, seen by the difference in sigmoid-shapes for each plot.

1. 2. iv. on top of those plots, add the estimated functions (on the target.contrast range from 0-1) for each subject based on partial pooling model (use glmer from the package lme4) where unique intercepts and slopes for target.contrast are modelled for each subject

```
# Making a df
staircase <- df %>%
  filter(df$trial.type == 'staircase')

# Model with random intercepts and slopes (partial pooling)
m2 <- glmer(correct ~ target.contrast + (1 + target.contrast | subject), family = 'binomial', data = staircase)

# Function from before, but altered to add estimated functions
pp_function <- function(i){
  dat <- staircase[which(staircase$subject == i),]
  model <- glm(correct ~ target.contrast, family = 'binomial', data = dat)
  fitted <- model$fitted.values
  plot_dat <- data.frame(cbind(fitted, 'target.contrast' = dat$target.contrast))
  fitted2 <- fitted.values(m2) # Adding fitted values for partial pooling model
  plot_dat_2 <- staircase %>% # Subsetting this df also pr subject
    mutate("fitted.values" = fitted2) %>%
    filter(subject == i)

  plot <- ggplot(plot_dat, aes(x = target.contrast, y = fitted))+
    geom_point(color = 'steelblue') +
    geom_line(data = plot_dat_2, aes(x = target.contrast, y = fitted.values)) + # THIS IS THE PARTIAL POOLING MODEL
    xlab('Target Contrast') +
    ylab('Predicted') +
    ylim(c(0,1))+
    ggtitle(paste0('Participant ', as.character(i))) +
    theme_minimal() +
    theme(plot.title = element_text(size = 10), axis.title=element_text(size = 8), axis.text=element_text(size=6))

  return(plot)
}

# Applying function to subjects and plotting with grid.arrange
subjects <- c("001", "002", "003", "004", "005", "006", "007", "008", "009", "010", "011", "012", "013", "014", "015", "016"
)
plots <- lapply(subjects, FUN=pp_function)
```

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

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

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

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

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

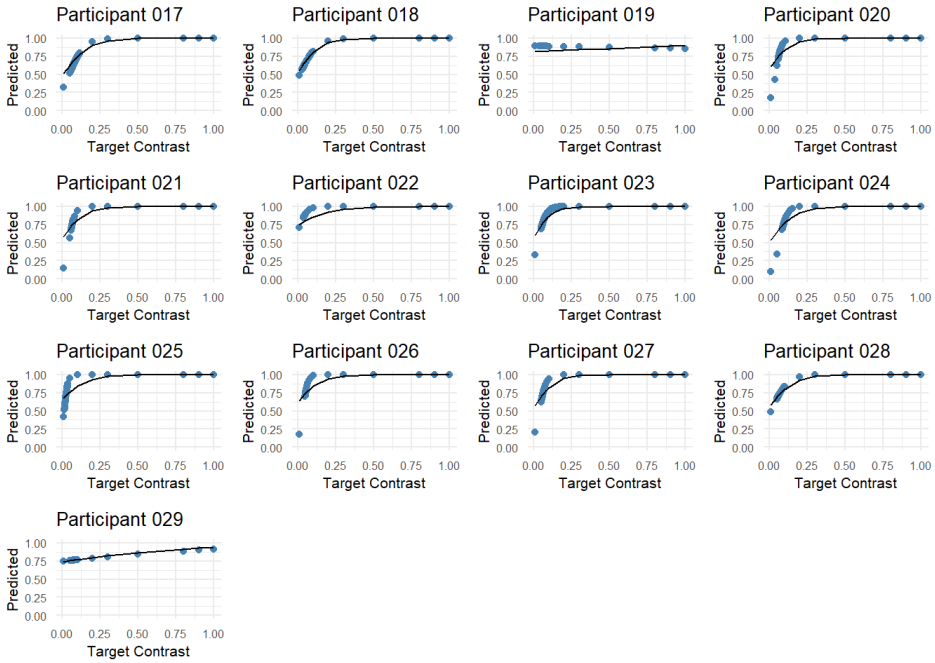
```
# With remaining participants
do.call(grid.arrange,  plots)
```



```
subjects <- c("017", "018", "019", "020", "021", "022", "023", "024", "025", "026", "027", "028", "029")
plots <- lapply(subjects, FUN=pp_function)
```

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

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

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

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

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

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

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

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

```
do.call(grid.arrange,  plots)
```



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

Partial pooling allows for the model to be generalizable (i.e. “less accurate” fit compared to no pooling), but still accounts for subject differences in baseline (intercept) and performance (slopes). Although it makes little sense to have slopes per subject, as we’re only modelling single subject data, but 29 times.

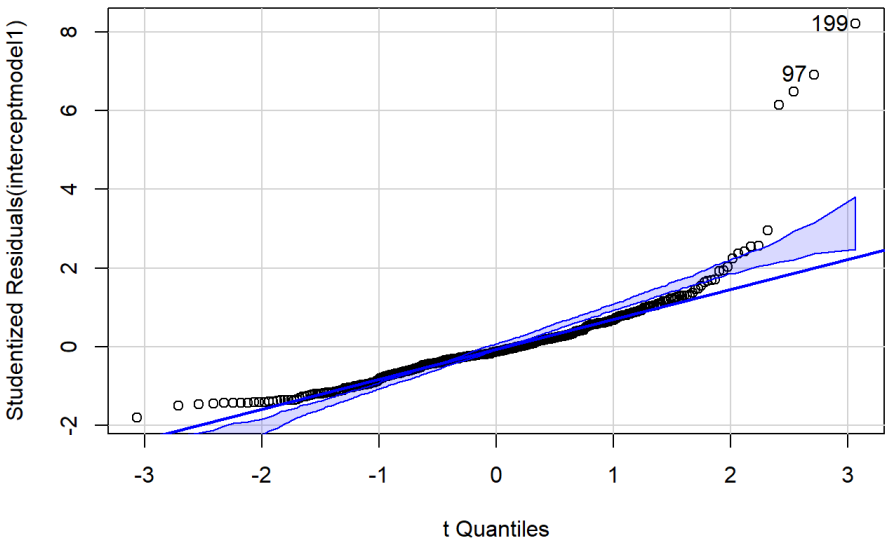
Exercise 2

```
# Making df
df_experiment <- df %>%
  filter(trial.type == "experiment")
```

2. 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

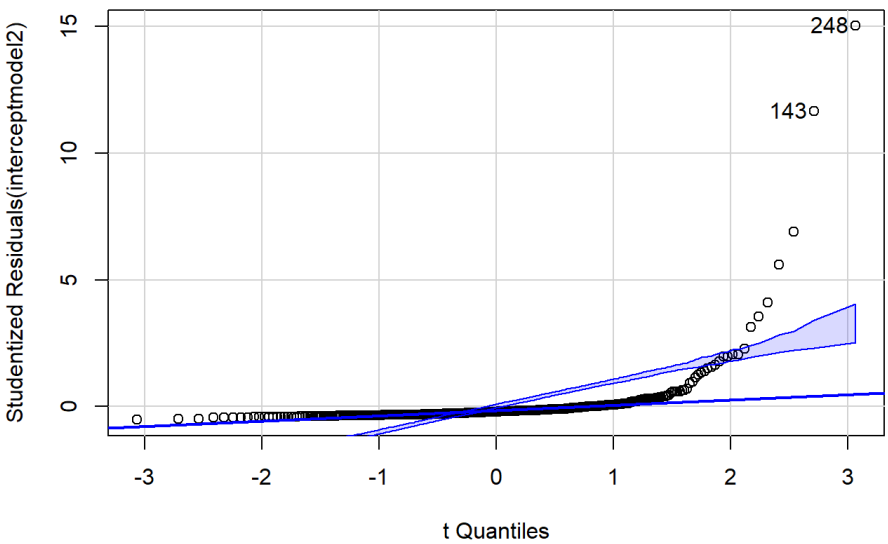
```
# Modelling simple response time
response_time <- lm(rt.obj ~ 1, data = df_experiment)
df_experiment$fitted_rt <- fitted(response_time)
```

```
# Subsetting for each subject
subject1 <- df_experiment %>%
  filter(subject == "001")
subject2 <- df_experiment %>%
  filter(subject == "002")
subject3 <- df_experiment %>%
  filter(subject == "003")
subject4 <- df_experiment %>%
  filter(subject == "004")

# Modelling each subject
interceptmodel1 <- lm(rt.obj ~ 1, data = subject1)
interceptmodel2 <- lm(rt.obj ~ 1, data = subject2)
interceptmodel3 <- lm(rt.obj ~ 1, data = subject3)
interceptmodel4 <- lm(rt.obj ~ 1, data = subject4)

# Plotting
car::qqPlot(interceptmodel1)
```



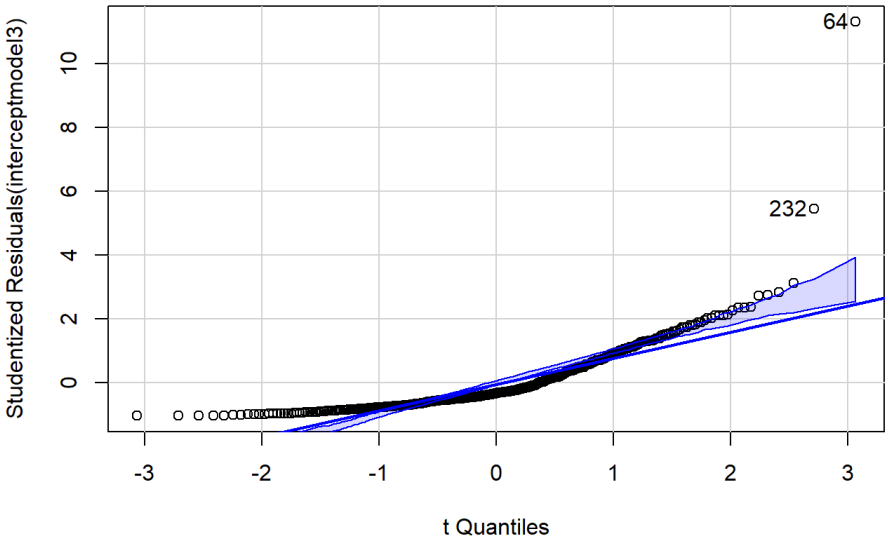

```
## [1]  97 199
```

```
car::qqPlot(interceptmodel2)
```



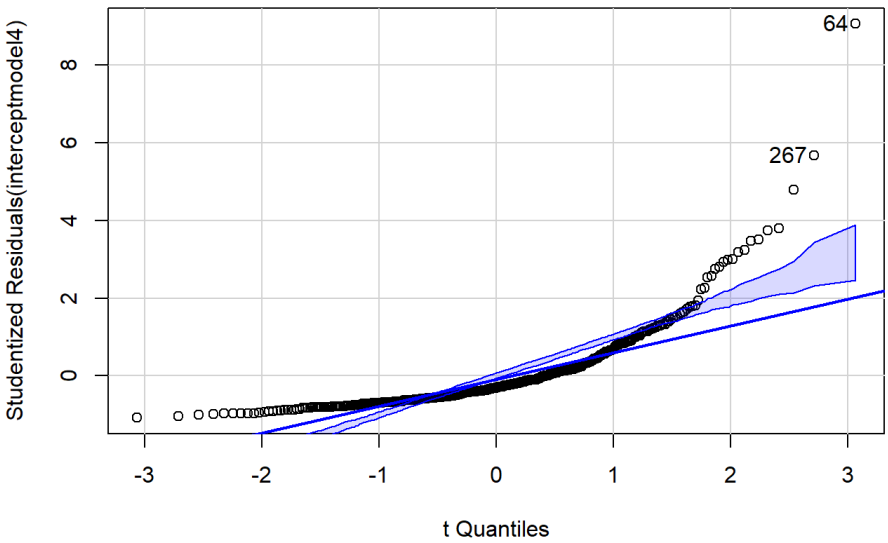

```
## [1] 143 248
```

```
car::qqPlot(interceptmodel3)
```




```
## [1]  64 232
```

```
car::qqPlot(interceptmodel4)
```



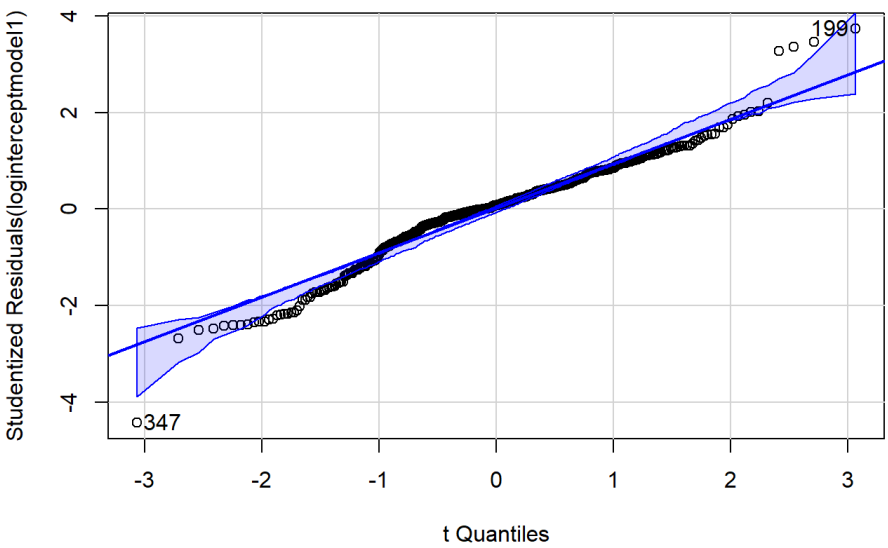

```
## [1]  64 267
```

2. 1. i. comment on these

They are all slightly normal-distributed, but not satisfactory as there is heavy skewness in some of the plots and some heavy outliers in the top end mainly.

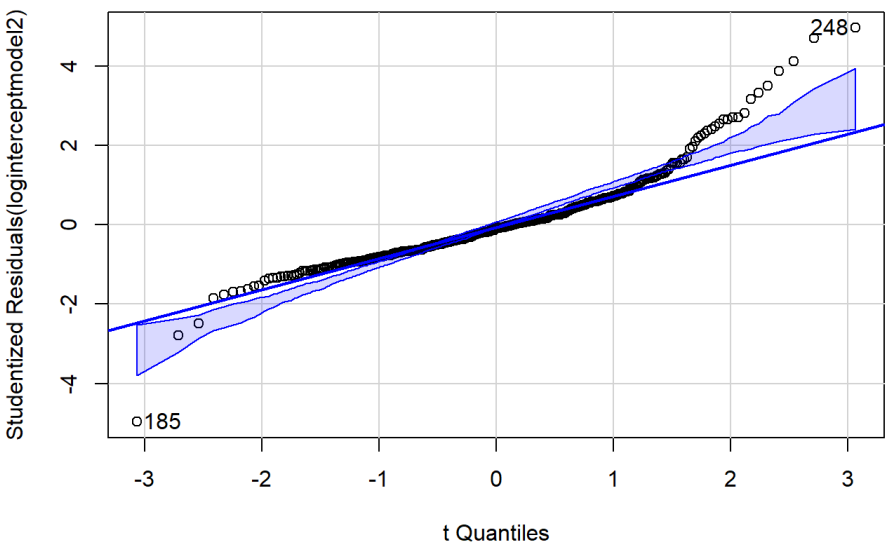
2. 1. ii. does a log-transformation of the response time data improve the Q-Q-plots?

```
# Log transforming for each subject
logsubject1 <- subject1 %>%
  mutate(log_rt = log(rt.obj))
logsubject2 <- subject2 %>%
  mutate(log_rt = log(rt.obj))
logsubject3 <- subject3 %>%
  mutate(log_rt = log(rt.obj))
logsubject4 <- subject4 %>%
  mutate(log_rt = log(rt.obj))

# Modelling
loginterceptmodel1 <- lm(log_rt ~ 1, data = logsubject1)
loginterceptmodel2 <- lm(log_rt ~ 1, data = logsubject2)
loginterceptmodel3 <- lm(log_rt ~ 1, data = logsubject3)
loginterceptmodel4 <- lm(log_rt ~ 1, data = logsubject4)

# Plotting
car::qqPlot(loginterceptmodel1)
```



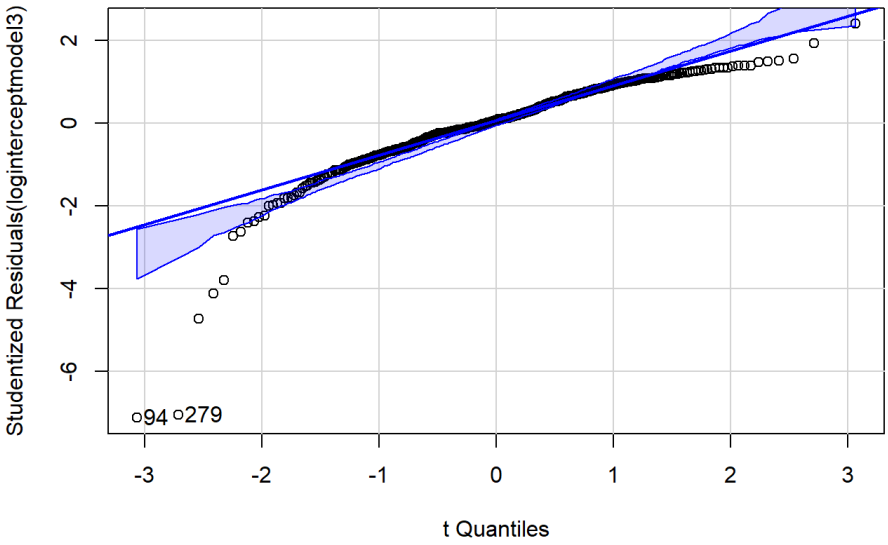

```
## [1] 199 347
```

```
car::qqPlot(loginterceptmodel2)
```



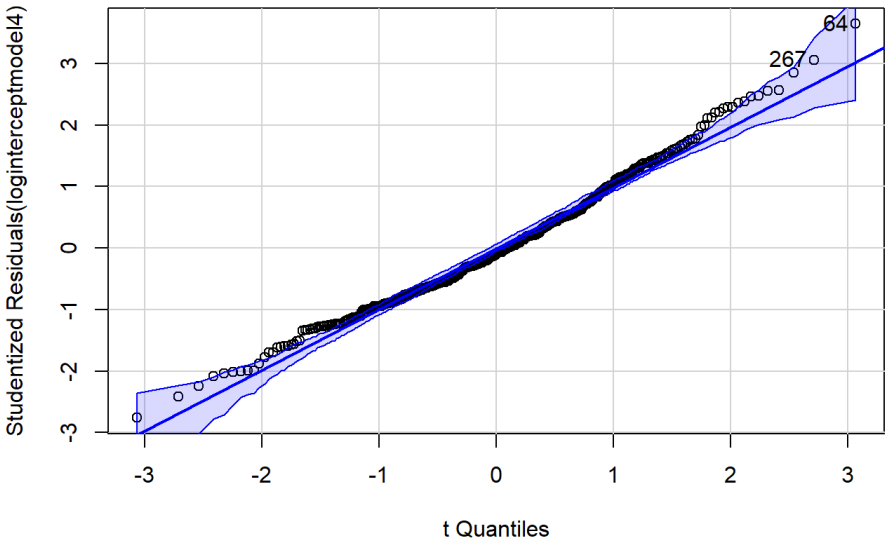

```
## [1] 185 248
```

```
car::qqPlot(loginterceptmodel3)
```




```
## [1]  94 279
```

```
car::qqPlot(loginterceptmodel4)
```




```
## [1]  64 267
```

I would say the log transformation reduced the tail sizes (outliers), which betters the distribution for subject 1 and 2. Though distribution for subject 3 is now less normal than before transformation.

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

```
rt_partialpooling1 <- lmer(rt.obj ~ task + (1|subject), REML = FALSE, data = df_experiment)
rt_partialpooling2 <- lmer(rt.obj ~ task + (1|subject) + (1|trial), REML = FALSE, data = df_experiment)
```

2. 2. i. which would you include among your random effects and why? (support your choices with relevant measures, taking into account variance explained and number of parameters going into the modelling)

```
rt_no_ranef <- lm(rt.obj ~ task, data = df_experiment)
anova(rt_partialpooling1, rt_partialpooling2, rt_no_ranef)
```

	npar	AIC	BIC	logLik	deviance	Chisq
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>

	npar <dbl>	AIC <dbl>	BIC <dbl>	logLik <dbl>	deviance <dbl>	Chisq <dbl>
rt_no_ranef	4	62056.06	62085.81	-31024.03	62048.06	*NA*
rt_partialpooling1	5	61940.21	61977.39	-30965.11	61930.21	117.84801407
rt_partialpooling2	6	61942.17	61986.78	-30965.08	61930.17	0.04584006

3 rows | 1-7 of 9 columns

```
summary(rt_partialpooling1)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: rt.obj ~ task + (1 | subject)
##    Data: df_experiment
##
##      AIC      BIC   logLik deviance df.resid
##  61940.2  61977.4 -30965.1  61930.2    12523
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
##  -0.630  -0.155  -0.072   0.051 101.443
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept) 0.1147   0.3386
##  Residual             8.1739   2.8590
## Number of obs: 12528, groups:  subject, 29
##
## Fixed effects:
##                   Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)     1.120e+00  7.689e-02  4.775e+01  14.568  < 2e-16 ***
## taskquadruplet -1.532e-01  6.257e-02  1.250e+04  -2.449  0.01433 *
## tasksingles    -1.915e-01  6.257e-02  1.250e+04  -3.061  0.00221 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr
## taskqudrplt -0.407
## tasksingles -0.407  0.500
```

Optimally, i should like to model the hierarchies i'd expect exist in the data. That would definitively include random intercepts per subject, as I assume reaction time is in some sense dependent on subject. Although this doesn't explain much of the variance, it does reduce the AIC slightly. I would probably also consider doing random intercepts for trial, but seeing as this increases the AIC value, i don't regard it as the right choice, as i only want to add terms that increase the models explanatory power.

2. 2. ii. explain in your own words what your chosen models says about response times between the different tasks

My model shows that task significantly predicts reaction time for all three tasks ($p < 0.05$). Subjects have highest reaction time with doubles, then quadruplets, then singles.

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

```
rt_partialpooling3 <- lmer(rt.obj ~ task*pas + (1|subject), REML = FALSE, data = df_experiment)
summary(rt_partialpooling3)
```

```
## Linear mixed model fit by maximum likelihood . t-tests use Satterthwaite's
##   method [lmerModLmerTest]
## Formula: rt.obj ~ task * pas + (1 | subject)
##    Data: df_experiment
##
##      AIC      BIC   logLik deviance df.resid
##  61917.4  62021.5 -30944.7  61889.4    12514
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
##  -0.667  -0.152  -0.064   0.047 101.556
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  subject  (Intercept) 0.09749  0.3122
##  Residual             8.14982  2.8548
## Number of obs: 12528, groups:  subject, 29
##
## Fixed effects:
##                       Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)          1.335e+00  9.786e-02  1.569e+02  13.644  < 2e-16 ***
## taskquadruplet      -3.229e-01  1.068e-01  1.251e+04  -3.023  0.00251 **
## tasksingles         -2.156e-01  1.155e-01  1.252e+04  -1.866  0.06205 .
## pas2                -1.394e-01  1.150e-01  1.228e+04  -1.213  0.22521
## pas3                -3.359e-01  1.314e-01  1.184e+04  -2.557  0.01058 *
## pas4                -5.795e-01  1.341e-01  9.034e+03  -4.321 1.57e-05 ***
## taskquadruplet:pas2  2.273e-01  1.582e-01  1.252e+04   1.437  0.15067
## tasksingles:pas2     1.251e-01  1.661e-01  1.253e+04   0.753  0.45131
## taskquadruplet:pas3  2.624e-01  1.822e-01  1.252e+04   1.440  0.14979
## tasksingles:pas3     9.453e-02  1.852e-01  1.252e+04   0.510  0.60972
## taskquadruplet:pas4  2.583e-01  1.797e-01  1.251e+04   1.437  0.15068
## tasksingles:pas4     7.097e-02  1.746e-01  1.252e+04   0.406  0.68444
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng pas2   pas3   pas4   tskq:2 tsks:2 tskq:3
## taskqudrplt -0.562
## tasksingles -0.517  0.474
## pas2        -0.553  0.478  0.441
## pas3        -0.499  0.420  0.386  0.425
## pas4        -0.504  0.410  0.373  0.414  0.395
## tskqdrplt:2  0.378 -0.677 -0.319 -0.697 -0.282 -0.276
## tsksngls:p2  0.368 -0.330 -0.697 -0.672 -0.279 -0.270  0.481
## tskqdrplt:3  0.328 -0.587 -0.279 -0.280 -0.681 -0.238  0.397  0.194
## tsksngls:p3  0.327 -0.296 -0.625 -0.276 -0.677 -0.242  0.199  0.437  0.484
## tskqdrplt:4  0.333 -0.595 -0.282 -0.284 -0.249 -0.658  0.402  0.195  0.348
## tsksngls:p4  0.343 -0.314 -0.662 -0.290 -0.255 -0.681  0.211  0.460  0.185
##             tsks:3 tskq:4
## taskqudrplt
## tasksingles
## pas2
## pas3
## pas4
## tskqdrplt:2
## tsksngls:p2
## tskqdrplt:3
## tsksngls:p3
## tskqdrplt:4  0.176
## tsksngls:p4  0.413  0.507
```

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

```
rt_partialpooling4 <- lmer(rt.obj ~ task*pas + (1|subject) + (1|trial), REML = FALSE, data = df_experiment)
rt_partialpooling5 <- lmer(rt.obj ~ task*pas + (1|subject) + (1|trial) + (1|odd.digit), REML = FALSE, data = df_experiment)
rt_partialpooling6 <- lmer(rt.obj ~ task*pas + (1|subject) + (1|trial) + (1|odd.digit) + (1|cue), REML = FALSE, data = df_ex
periment)
rt_partialpooling7 <- lmer(rt.obj ~ task*pas + (1|subject) + (1|trial) + (1|odd.digit) + (1|cue) + (1|pas), REML = FALSE, da
ta = df_experiment)
```

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

Only the last one failed to converge. It is hard to see in the code chunk, but the model which didn't converge was the following: rt_partialpooling7 <- lmer(rt.obj ~ task*pas + (1|subject) + (1|trial) + (1|odd.digit) + (1|cue) + (1|pas), REML = FALSE, data = df_experiment)

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

```
print(VarCorr(rt_partialpooling7), comp='Variance')
```

```
##  Groups    Name        Variance
##  trial     (Intercept) 0.00153000
##  cue       (Intercept) 0.08946868
##  subject   (Intercept) 0.09769533
##  pas       (Intercept) 0.00000000
##  odd.digit (Intercept) 0.00016599
##  Residual              8.11287188
```

The fit is singular because random intercept for pas explains no variance, i.e. is estimated at 0, and can therefore not be calculated. This makes sense since i already added pas in the model previously.

2. 3. iii. in your own words - how could you explain why your model would result in a singular fit?

If you add terms which explain close to zero variance, i.e. too low eigenvalue (below some tolerance level in the function, i assume), the model cannot be fit and fails to converge.

Exercise 3

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

```
# Making df
data.count <- df %>%
  group_by(subject, task, pas) %>%
  dplyr::summarise("count" = n())
```

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

3. 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

```
pasmodel <- glmer(count ~ pas*task + (pas|subject), data = data.count, family = poisson, control = glmerControl(optimizer="b
obyqa"))
summary(pasmodel)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: poisson  ( log )
## Formula: count ~ pas * task + (pas | subject)
##    Data: data.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  36.770  < 2e-16 ***
## pas2                -0.02378    0.11963  -0.199 0.842456
## pas3                -0.51365    0.20717  -2.479 0.013164 *
## pas4                -0.77292    0.29075  -2.658 0.007851 **
## taskquadruplet       0.11490    0.03127   3.674 0.000239 ***
## tasksingles         -0.23095    0.03418  -6.756 1.42e-11 ***
## pas2:taskquadruplet -0.11376    0.04605  -2.470 0.013508 *
## pas3:taskquadruplet -0.20902    0.05287  -3.954 7.69e-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
```

3. 2. i. which family should be used?

Poisson, because it's good for modelling frequency, which we are doing with counting the PAS-scores.

3. 2. ii. why is a slope for *pas* not really being modelled?

Because pas isn't continuous, we can't model a proper slope, so this is a "pseudo"-slope.

3. 2. iii. if you get a convergence error, try another algorithm (the default is the *Nelder_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`)

I did get the error, added the optimizer, now it converges.

3. 2. 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

```
pasmodel2 <- glmer(count ~ pas + task + (pas|subject), data = data.count, family = poisson, control = glmerControl(optimizer
="bobyqa"))
anova(pasmodel, pasmodel2)
```

	npar <dbl>	AIC <dbl>	BIC <dbl>	logLik <dbl>	deviance <dbl>	Chisq <dbl>	Df <dbl>
pasmodel2	16	3398.549	3459.812	-1683.274	3366.549	*NA*	*NA*
pasmodel	22	3148.441	3232.678	-1552.220	3104.441	262.1076	6

2 rows | 1-8 of 9 columns

The interaction model performs better with all interaction effects being significant ($p < 0.05$), and with a lower AIC value ($p < 0.05$).

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

```
tibble("SSR interaction model" = sum(residuals(pasmodel)^2), "SSR no interaction" = sum(residuals(pasmodel2)^2))
```

SSR interaction model <dbl>	SSR no interaction <dbl>
699.4866	962.4628

1 row

I would choose the model with the interaction effect included. The model performs better as shown, and the interaction effects are significant, which is an important feature of the data. The SSR score is also lower for this model.

3. 2. vi. based on your chosen model - write a short report on what this says about the distribution of ratings as dependent on *pas* and *task*

Building the model

The model i've chosen is count ~ pas * task + (pas | subject). Results from this model indicate that the frequency of observations for a certain combination of PAS-score, task and subject (the count variable) is significantly predicted by PAS-score and task and the interaction between the two ($p < 0 .05$). The model includes random slopes for PAS-score per subject (intercept).

Results

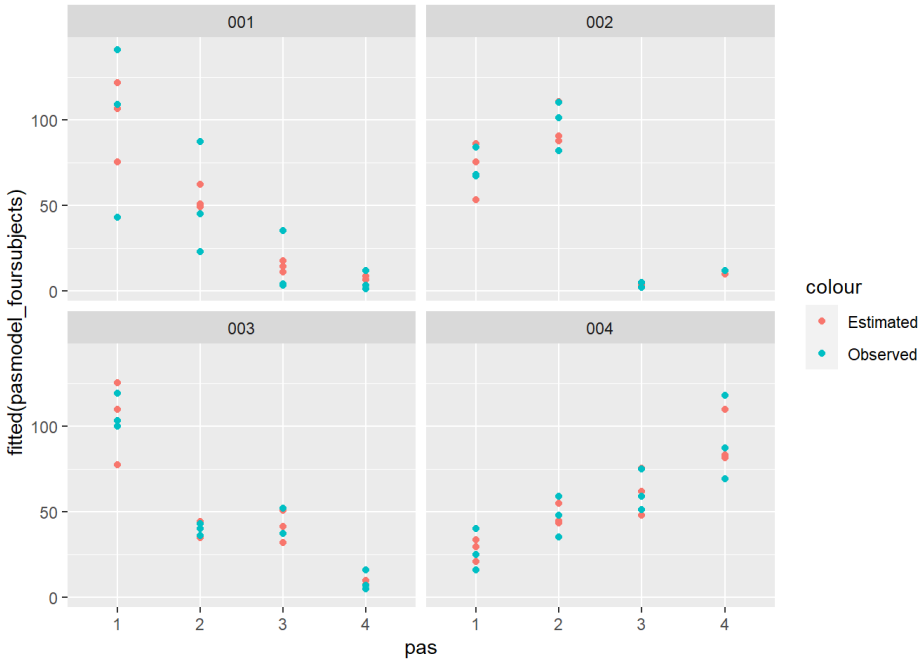
The model suggests that count decreases as PAS-score increases, and count is largest in task "quadruplets", then "doubles" then "singles". Though it is more useful to look at the interaction effects here - when PAS-score increases, the task "quadruplets" decreases compared to "doubles", and "singles" increases compared to "doubles". This effectively means that subjects are actually more "perceptually aware" doing the "singles" task, even though count of PAS-score generally decreases (as per pas main effect) and count of "singles" task generally decreases (as per task main effect).

3. 2. vii. include a plot that shows the estimated amount of ratings for four subjects of your choosing

```
# FITTING THE INTERACTION MODEL
# Subsetting
pas_foursubjects <- data.count %>%
  filter(subject == "001"|subject == "002"|subject == "003"|subject == "004")

# Modelling
pasmodel_foursubjects <- glmer(count ~ pas*task + (pas|subject), data = pas_foursubjects, family = poisson)
```

```
## Warning in checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
## Model failed to converge with max|grad| = 0.00304025 (tol = 0.002, component 1)
```

```
# Plotting
pas_foursubjects %>%
  ggplot() +
  geom_point(aes(x = pas, y = fitted(pasmodel_foursubjects), color = "Estimated")) +
  geom_point(aes(x = pas, y = count, color = "Observed")) +
  facet_wrap(~subject)
```



3. 3. Finally, fit a multilevel model that models correct as dependent on task with a unique intercept for each subject

```
df_end_me <- glmer(correct ~ task + (1 | subject), data = df, family = "binomial")
```

3. 3. i. does task explain performance?

```
summary(df_end_me)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + (1 | subject)
##    Data: df
##
##      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    0.08386  13.125  < 2e-16 ***
## taskquadruplet -0.09825    0.04190  -2.345    0.019 *
## tasksingles     0.18542    0.04337   4.276 1.91e-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
```

Task significantly predicts correctness for all task levels (all p < 0.05).

3. 3. ii. add pas as a main effect on top of task - what are the consequences of that?

```
df_end_me_more <- glmer(correct ~ task + pas + (1 | subject), data = df, family = "binomial")
summary(df_end_me_more)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ task + pas + (1 | subject)
##    Data: df
##
##      AIC      BIC   logLik deviance df.resid
##  17424.9  17479.5  -8705.5  17410.9    18124
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.4872 -0.6225  0.3240  0.5767  1.6144
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.1979   0.4449
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     0.08530    0.09143   0.933    0.351
## taskquadruplet -0.03055    0.04497  -0.679    0.497
## tasksingles    -0.01059    0.04687  -0.226    0.821
## pas2            0.95477    0.04419  21.604   <2e-16 ***
## pas3            1.97709    0.06219  31.793   <2e-16 ***
## pas4            3.12732    0.08626  36.255   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) tskqdr tsksng pas2   pas3
## taskqudrplt -0.259
## tasksingles -0.225  0.489
## pas2        -0.212  0.021 -0.040
## pas3        -0.165  0.030 -0.045  0.355
## pas4        -0.123  0.016 -0.080  0.257  0.236
```

Since task is no longer significant, it seems that pas explains more of the variance, i.e. a better predictor.

3. 3. iii. now fit a multilevel model that models correct as dependent on pas with a unique intercept for each subject

```
df_end_me_more_now <- glmer(correct ~ pas + (1 | subject), data = df, family = "binomial")
summary(df_end_me_more_now)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ pas + (1 | subject)
##    Data: df
##
##      AIC      BIC   logLik deviance df.resid
##  17421.4  17460.4  -8705.7  17411.4    18126
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -8.5665 -0.6243  0.3244  0.5754  1.6017
##
## Random effects:
##  Groups  Name        Variance Std.Dev.
##  subject (Intercept) 0.1981   0.4451
## Number of obs: 18131, groups:  subject, 29
##
## Fixed effects:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.07044    0.08773   0.803    0.422
## pas2         0.95575    0.04410  21.671   <2e-16 ***
## pas3         1.97892    0.06201  31.914   <2e-16 ***
## pas4         3.12940    0.08579  36.476   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr) pas2   pas3
## pas2 -0.223
## pas3 -0.173  0.352
## pas4 -0.136  0.253  0.231
```

3. 3. iv. finally, fit a model that models the interaction between task and pas and their main effects

```
df_end_me_more_now_pls <- glm(correct ~ task * pas, data = df, family = "binomial")
summary(df_end_me_more_now_pls)
```

```
##
## Call:
## glm(formula = correct ~ task * pas, family = "binomial", data = df)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.5349  -1.2539   0.5089   0.7922   1.1028
##
## Coefficients:
##                      Estimate Std. Error z value Pr(>|z|)
## (Intercept)          0.210096   0.045777   4.590 4.44e-06 ***
## taskquadruplet       0.002246   0.062960   0.036    0.972
## tasksingles         -0.032082   0.068755  -0.467    0.641
## pas2                 0.787873   0.070763  11.134  < 2e-16 ***
## pas3                 1.723013   0.099679  17.286  < 2e-16 ***
## pas4                 2.764730   0.142179  19.445  < 2e-16 ***
## taskquadruplet:pas2 -0.049612   0.098605  -0.503    0.615
## tasksingles:pas2     0.073425   0.103517   0.709    0.478
## taskquadruplet:pas3 -0.108583   0.140751  -0.771    0.440
## tasksingles:pas3     0.077543   0.143470   0.540    0.589
## taskquadruplet:pas4 -0.125402   0.199278  -0.629    0.529
## tasksingles:pas4     0.229071   0.195933   1.169    0.242
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 20418  on 18130  degrees of freedom
## Residual deviance: 17853  on 18119  degrees of freedom
## AIC: 17877
##
## Number of Fisher Scoring iterations: 5
```

3. 3. v. describe in your words which model is the best in explaining the variance in accuracy.

```
anova(df_end_me, df_end_me_more, df_end_me_more_now, df_end_me_more_now_pls)
```

	npar <dbl>	AIC <dbl>	BIC <dbl>	logLik <dbl>	deviance <dbl>
df_end_me	4	19927.21	19958.43	-9959.605	19919.21
df_end_me_more_now	5	17421.39	17460.41	-8705.694	17411.39
df_end_me_more	7	17424.91	17479.55	-8705.456	17410.91
df_end_me_more_now_pls	12	17877.34	17971.00	-8926.668	17853.34

4 rows | 1-6 of 9 columns

The model which predicts correct by pas with intercepts per subject has the lowest AIC value and a significant chi-square value. This model strikes the balance between complexity and explanatory power, that means it explains the accuracy whilst being simple compared to the other models. Adding predictors without increasing the amount of variance explained doesn't increase the models performance, but does increase complexity, which isn't an inherently good thing.

Assignment 2 (Part 2), Methods 3, 2021, autumn semester

Aleksander Wael

10/13/2021

Loading packages

```
pacman::p_load(tidyverse, lmerTest, lme4, gridExtra, dfoptim, readbulk, boot, multcomp)
```

EXERCISE 4 - Download and organise the data from experiment 1

4.1 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)

```
df <- read_bulk("experiment_1")
```

```
## Reading 001.csv
```

```
## Reading 002.csv
```

```
## Reading 003.csv
```

```
## Reading 004.csv
```

```
## Reading 005.csv
```

```
## Reading 006.csv
```

```
## Reading 007.csv
```

```
## Reading 008.csv
```

```
## Reading 009.csv
```

```
## Reading 010.csv
```

```
## Reading 011.csv
```

```
## Reading 012.csv
```

```
## Reading 013.csv
```

```
## Reading 014.csv
```

```
## Reading 015.csv
```

```
## Reading 016.csv
```

```
## Reading 017.csv
```

```
## Reading 018.csv
```

```
## Reading 019.csv
```

```
## Reading 020.csv
```

```
## Reading 021.csv
```

```
## Reading 022.csv
```

```
## Reading 023.csv
```

```
## Reading 024.csv
```

```
## Reading 025.csv
```

```
## Reading 026.csv
```

```
## Reading 027.csv
```

```
## Reading 028.csv
```

```
## Reading 029.csv
```

read_bulk() automatically fills empty rows with NA, so this step isn't needed.

4.1.i. Factorise the variables that need factorising

```
# Assigning variables to proper class
df$pas <- as.factor(df$pas)
df$trial <- as.character(df$trial)
df$target.contrast <- as.numeric(df$target.contrast)
df$cue <- as.character(df$cue)
df$rt.subj <- as.numeric(df$rt.subj)
df$rt.obj <- as.numeric(df$rt.obj)
df$target.contrast <- as.numeric(df$target.contrast)
df$target.frames <- as.integer(df$target.frames)
df$subject <- as.factor(df$subject)
```

4.1.ii. Remove the practice trials from the dataset (see the trial.type variable)

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

4.1.iii. Create a correct variable

Adding variable "correct" to display if subject was correct

```
# Adding empty variable
df <- df %>%
  mutate(obj.resp.2 = obj.resp)

# Renaming rows in obj.resp.2 to get same units as target.type
df$obj.resp.2 <- replace(df$obj.resp.2, df$obj.resp.2 == "e", "even")
df$obj.resp.2 <- replace(df$obj.resp.2, df$obj.resp.2 == "o", "odd")

# Adding value for correct and incorrect answers
df_correct <- df %>%
  filter(obj.resp.2 == target.type) %>%
  mutate(correct = "1")

# Joining with my df
df <- left_join(df, df_correct)
```

```
## Joining, by = c("trial.type", "pas", "trial", "jitter.x", "jitter.y", "odd.digit", "target.contrast", "target.frames", "c
ue", "task", "target.type", "rt.subj", "rt.obj", "even.digit", "seed", "obj.resp", "subject", "File", "obj.resp.2")
```

```
# Remaining are NAs, so replace with 0
df$correct <- replace(df$correct, is.na(df$correct), "0")

# Treating as numeric, otherwise i get an error later on
df$correct <- as.numeric(df$correct)
```

4.1.iv. Describe how the target.contrast and target.frames variables differ compared to the data from part 1 of this assignment

target.contrast is not manipulated in this experiment and is set at 0.1. target.frames is now manipulated and ranges from 1-6 frames. It was opposite in the previous assignment.

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

5.1 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.

```
# Pooled model
m1_pooled <- glm(correct ~ target.frames, data = df, family = "binomial")
```

```
# Partially-pooled model
m1_partial <- glmer(correct ~ target.frames + (1|subject), data = df, family = "binomial")
```

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

```
likelihood_fun <- function(i) {
  p <- fitted(i) # Vector of fitted values
  y <- as.vector(model.response(model.frame(i), type = "numeric")) # Observed y-values
  likelihood <- prod(p^y*(1-p)^(1-y)) # The likelihood function for logistic regression
  return(likelihood)
}
```

5.1.ii. The log-likelihood-function for logistic regression is: $l(p) = \sum_{i=1}^{N} [y_i \ln p + (1-y_i) \ln (1-p)$. Create a function that calculates the log-likelihood

```
log_likelihood_fun <- function(i) {
  p <- fitted(i) # Vector of fitted values
  y <- as.vector(model.response(model.frame(i), type = "numeric")) # Observed y-values
  log_likelihood <- sum(y*log(p)+(1-y)*log(1-p)) # The log-likelihood function for logistic regression
  return(log_likelihood)
}
```

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

```
likelihood_fun(m1_pooled)
```

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

```
log_likelihood_fun(m1_pooled)
```

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

```
logLik(m1_pooled)
```

```
## 'log Lik.' -10865.25 (df=2)
```

Output is the same from my function and lokLik() function. The log-likelihood is better, because the likelihood function has to calculate more decimals (or the decimals are more meaningful, because the likelihood will often approximate 0)

5.1.iv. now show that the log-likelihood is a little off when applied to the partial pooling model

```
log_likelihood_fun(m1_partial)
```

```
## [1] -10565.53
```

```
logLik(m1_partial)
```

```
## 'log Lik.' -10622.03 (df=3)
```

There is a difference in output from the two functions.

5.2. 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

```
m0 <- glm(correct ~ 1, data = df, family = 'binomial') # Null-model
m2 <- glmer(correct ~ 1 + (1|subject), data = df, family = 'binomial') # Null-model with subject intercepts
m1_partial # Model from before, predicted by target.frames and subject intercepts
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ target.frames + (1 | subject)
##    Data: df
##       AIC       BIC    logLik  deviance  df.resid
##  21250.06  21274.45 -10622.03  21244.06     25041
## Random effects:
##  Groups  Name        Std.Dev.
##  subject (Intercept) 0.3936
## Number of obs: 25044, groups:  subject, 29
## Fixed Effects:
##   (Intercept)  target.frames
##       -0.9597         0.7549
```

```
m3 <- glmer(correct ~ target.frames + (target.frames|subject), data = df, family = "binomial")

anova(m2, m0, m1_partial, m3) # This function also give logLik values
```

```
## Data: df
## Models:
## m0: correct ~ 1
## m2: correct ~ 1 + (1 | subject)
## m1_partial: correct ~ target.frames + (1 | subject)
## m3: correct ~ target.frames + (target.frames | subject)
##            npar   AIC   BIC logLik deviance   Chisq Df Pr(>Chisq)
## m0            1 26685 26693 -13342    26683
## m2            2 26319 26335 -13158    26315  367.98  1  < 2.2e-16 ***
## m1_partial    3 21250 21274 -10622    21244 5071.04  1  < 2.2e-16 ***
## m3            5 20908 20948 -10449    20898  346.41  2  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(m1_partial, m3)
```

```
## Data: df
## Models:
## m1_partial: correct ~ target.frames + (1 | subject)
## m3: correct ~ target.frames + (target.frames | subject)
##            npar   AIC   BIC logLik deviance  Chisq Df Pr(>Chisq)
## m1_partial    3 21250 21274 -10622    21244
## m3            5 20908 20948 -10449    20898 346.41  2  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

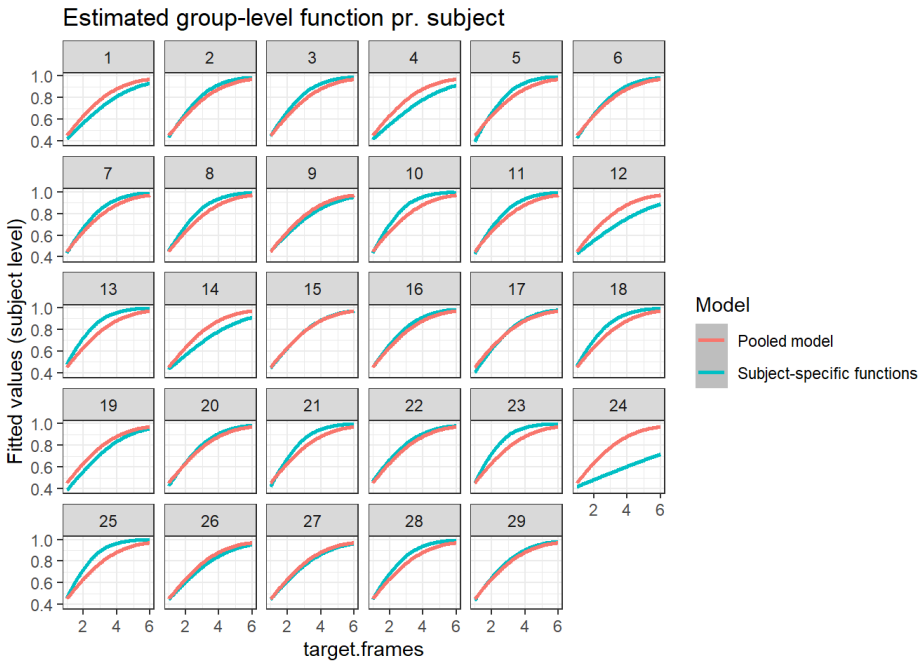
The next exercise has explanations.

5.2.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 estimated group-level function with xlim=c(0, 8) that includes the estimated subject-specific functions.

I use the anova function to display the logLik values of several models with varying complexity. Since m3 and m1_partial have the highest logLik values (most likely to be true), I compare these to see if the difference between them is significant. The difference between the two models is significant ($p < 0.05$), which is why i would choose the more complex model with target.frames as a fixed effect and with intercepts per subject and random slopes for target.frames. The group-level function is plotted below.

```
df %>% # Plot of group-level function per subject
  ggplot() +
  geom_smooth(aes(x = target.frames, y = fitted(m3), color = "Subject-specific functions")) +
  geom_smooth(aes(x = target.frames, y = fitted(m1_pooled), color = "Pooled model")) +
  facet_wrap( ~ subject) +
  labs(title = "Estimated group-level function pr. subject", color = "Model") +
  labs(x = "target.frames", y = "Fitted values (subject level)")+
  theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



5.2.ii. also include in the results section whether the fit didn't look good for any of the 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 %)

Subject 24 has a quite poor fit, and it seems this participant performs worse than other participants. I will compare it to the chance-level of 50% below.

```
df_24 <- df %>%
  filter(subject == "24")

t.test(df_24$correct, mu = 0.5)
```

```
##
##  One Sample t-test
##
## data:  df_24$correct
## t = 4.026, df = 873, p-value = 6.167e-05
## alternative hypothesis: true mean is not equal to 0.5
## 95 percent confidence interval:
##  0.5345964 0.6004150
## sample estimates:
## mean of x
## 0.5675057
```

When running a one sample t-test which the null hypothesis of "true mean is equal to 0.5", we obtain a p-value of less then 0.05, meaning that we reject the null hypothesis. According to this, subject 24 performs better than chance at ~ 56.7% accuracy.

5.3. 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

```
m4 <- glmer(correct ~ target.frames + pas + (target.frames|subject), family = binomial, data = df)

m5 <- glmer(correct ~ target.frames * pas + (target.frames|subject), family = binomial, data = df, control = glmerControl(op
timizer = "bobyqa"))

anova(m4, m5)
```

```
## Data: df
## Models:
## m4: correct ~ target.frames + pas + (target.frames | subject)
## m5: correct ~ target.frames * pas + (target.frames | subject)
##    npar   AIC   BIC  logLik deviance  Chisq Df Pr(>Chisq)
## m4    8 19880 19945 -9931.8    19864
## m5   11 19506 19596 -9742.0    19484 379.58  3  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is justified to add the interaction effect based on the logLik score.

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

It converges now with the bobyqa optimizer.

5.3.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 behaviour reasonable?

```
group_level <- glm(correct ~ target.frames * pas, family = 'binomial', data = df)

df %>%
  ggplot()+
  geom_smooth(aes(x = target.frames, y = fitted(group_level), color = pas), method = "loess")+
  xlab("target.frames")+
  ylab("Accuracy")+
  labs(title = "Group-level accuracy for PAS-ratings", color = "PAS-score")+
  xlim(c(1,6))+
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 0.975
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 1.025
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 7.4808e-028
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 1
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at
## 0.975
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
## 1.025
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
## number 7.4808e-028
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : There are other near
## singularities as well. 1
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : pseudoinverse used at 6.025
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : neighborhood radius 2.025
```

```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : reciprocal condition number 2.1006e-014
```

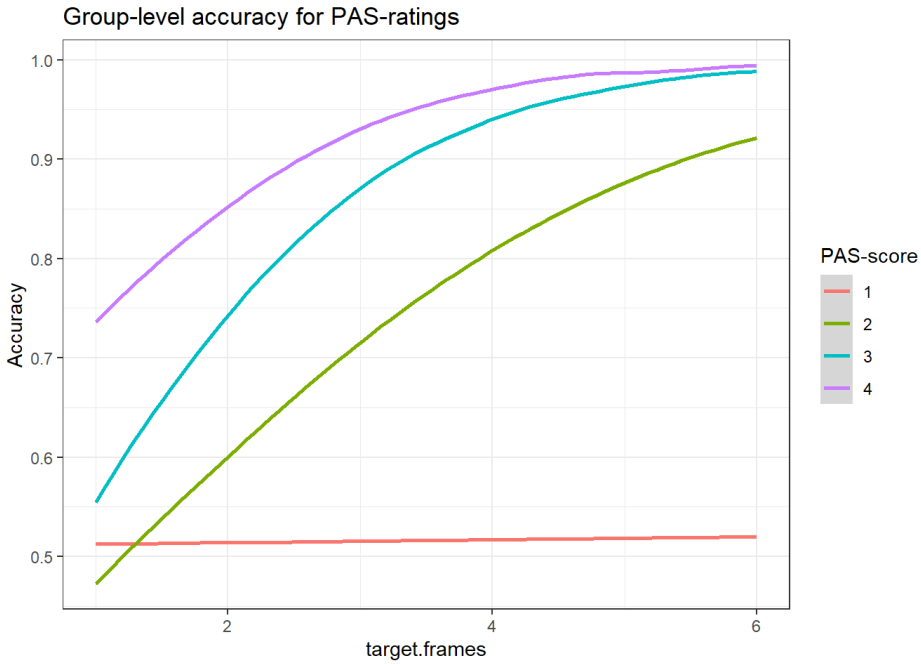
```
## Warning in simpleLoess(y, x, w, span, degree = degree, parametric =
## parametric, : There are other near singularities as well. 1
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : pseudoinverse used at
## 6.025
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : neighborhood radius
## 2.025
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : reciprocal condition
## number 2.1006e-014
```

```
## Warning in predLoess(object$y, object$x, newx = if
## (is.null(newdata)) object$x else if (is.data.frame(newdata))
## as.matrix(model.frame(delete.response(terms(object)), : There are other near
## singularities as well. 1
```



The plot shows that for PAS 1, increasing target frames doesn't really increase accuracy. In other words: if you report PAS 1, you weren't "perceptually aware", regardless of target.frames. Therefore, the accuracy hovers around chance-level (a bit above). The plot also shows the interaction between target.frames and PAS nicely, because the graphs aren't aligned, showing a difference in accuracy-development for each PAS-rating based on target.frames. At PAS rating 3 and 4, large values of target.frames approximates almost 100% probability of having answered correctly. This makes sense, because if you report higher levels of awareness (PAS 3 or 4), you will probably have higher accuracy.

One of the fitted values are below 0.5 when target.frames = 1, which doesn't make much sense since chance-level accuracy is 50%. This is a limitation of the model, as we of course know that 0.5 is the lowest expected accuracy.

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.

6.1. Fit a model based on the following formula: correct ~ pas * target.frames + (target.frames | subject))

```
# Already done, here is a summary of it.
summary(m5)
```

```
## Generalized linear mixed model fit by maximum likelihood (Laplace
##   Approximation) [glmerMod]
##  Family: binomial  ( logit )
## Formula: correct ~ target.frames * pas + (target.frames | subject)
##    Data: df
## Control: glmerControl(optimizer = "bobyqa")
##
##      AIC      BIC   logLik deviance df.resid
##  19506.1  19595.5  -9742.0  19484.1    25033
##
## Scaled residuals:
##      Min       1Q   Median       3Q      Max
## -19.0108   0.0537   0.1606   0.4849   1.4465
##
## Random effects:
##  Groups  Name          Variance Std.Dev. Corr
##  subject (Intercept)   0.03698  0.1923
##          target.frames 0.02058  0.1434   -0.76
## Number of obs: 25044, groups:  subject, 29
##
## Fixed effects:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)        -0.12163    0.06411  -1.897 0.057801 .
## target.frames       0.11480    0.03707   3.097 0.001955 **
## pas2               -0.57139    0.08937  -6.394 1.62e-10 ***
## pas3               -0.53850    0.13935  -3.864 0.000111 ***
## pas4                0.20159    0.24978   0.807 0.419619
## target.frames:pas2  0.44718    0.03473  12.878  < 2e-16 ***
## target.frames:pas3  0.74869    0.04589  16.314  < 2e-16 ***
## target.frames:pas4  0.75927    0.06830  11.116  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) trgt.f pas2   pas3   pas4   trg.:2 trg.:3
## target.frms -0.811
## pas2        -0.461  0.305
## pas3        -0.307  0.207  0.248
## pas4        -0.173  0.123  0.121  0.091
## trgt.frms:2  0.481 -0.428 -0.874 -0.244 -0.124
## trgt.frms:3  0.392 -0.358 -0.278 -0.891 -0.111  0.370
## trgt.frms:4  0.275 -0.260 -0.163 -0.120 -0.918  0.225  0.200
```

6.1.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.

Looking at the fixed effects, we see that increasing target.frames (with PAS at 1) increases the likelihood of answering correct (0.11480 on the log-odds scale). Looking at the interaction effects, we see that, when increasing target.frames with PAS at 2, the likelihood of answering correct is higher than that of PAS 1 (0.44718 on the log-odds scale). Thus it appears accuracy increases faster with target.frames for PAS 2 than for PAS 1. This can also be seen in the previous plot in exercise 5.3.ii.

6.2. 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.

6.2.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.

```
## testing whether PAS 2 is different from PAS 1
contrast.vector <- matrix(c(0, 0, 0, 0, 0, 1, 0, 0), nrow=1)
gh_1 <- glht(m5, contrast.vector)
print(summary(gh_1))
## as another example, we could also test whether there is a difference in
## intercepts between PAS 2 and PAS 3
contrast.vector <- matrix(c(0, -1, 1, 0, 0, 0, 0, 0), nrow=1)
gh_2 <- glht(m5, contrast.vector)
print(summary(gh_2))
```

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

```
# testing if accuracy performance increases faster for pas3 than pas2
contrast.vector <- matrix(c(0, 0, 0, 0, 0, -1, 1, 0), nrow=1)
gh_3 <- glht(m5, contrast.vector)
print(summary(gh_3))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ target.frames * pas + (target.frames |
##     subject), data = df, family = binomial, control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.30151    0.04618   6.529 6.63e-11 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

The positive estimate suggests that accuracy increases faster with target.frames for PAS 3 than PAS 2 ($p < 0.05$).

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

```
# testing if accuracy performance increases faster for pas4 than pas3
contrast.vector <- matrix(c(0, 0, 0, 0, 0, 0, -1, 1), nrow=1)
gh_4 <- glht(m5, contrast.vector)
print(summary(gh_4))
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: glmer(formula = correct ~ target.frames * pas + (target.frames |
##     subject), data = df, family = binomial, control = glmerControl(optimizer = "bobyqa"))
##
## Linear Hypotheses:
##        Estimate Std. Error z value Pr(>|z|)
## 1 == 0  0.01058    0.07430   0.142    0.887
## (Adjusted p values reported -- single-step method)
```

The positive estimate suggests that accuracy increases faster with target.frames for PAS 4 than PAS 3, but the effect is not significant. The difference is smaller compared to the difference of PAS 3 and PAS 2, though.

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

    ## "par" are our four parameters (a numeric vector)
    a = par[1]
    b = par[2]
    c = par[3]
    d = par[4]

    x <- dataset$x
    y <- dataset$y

    y.hat <- a + ((b-a)/(1+exp(1)^((c-x)/d)))

    RSS <- sum((y - y.hat)^2)
    return(RSS)
}
```

7.1. Now, we will fit the sigmoid for the four PAS ratings for Subject 7

7.1.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)

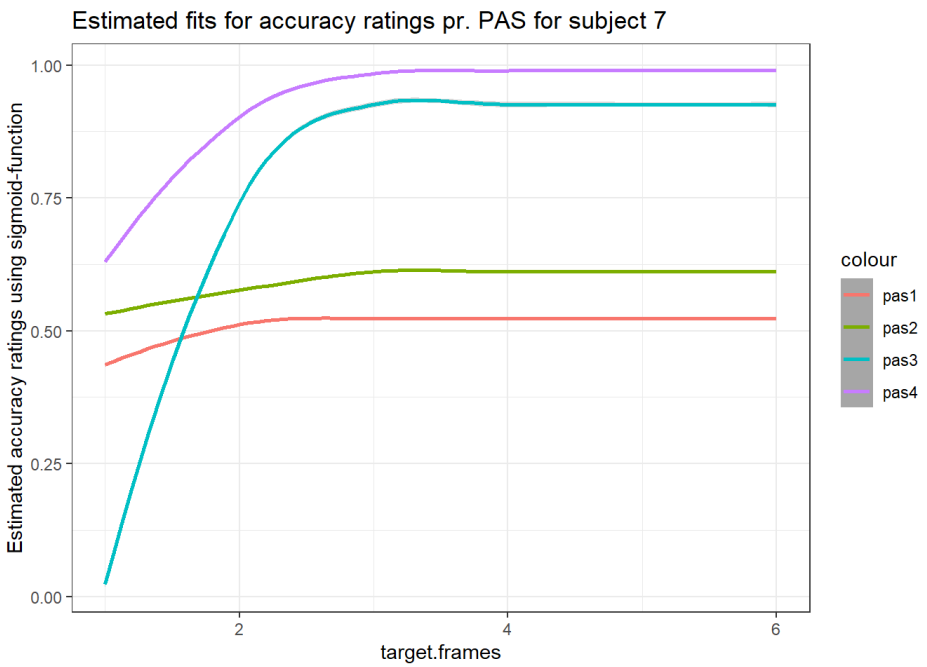
```
# Subsetting df
df_7 <- df %>%
  filter(subject == "7") %>%
  dplyr::select(target.frames, correct, pas) %>%
  rename(x = target.frames, y = correct)

# Specifying par-values
par <- c(0.5, 1, 1, 1)

# Running the optim function for each pas score
optim_7_pas1 <- optim(data = filter(df_7, pas == "1"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf, -Inf
), upper = c(1, 1, Inf, Inf))
optim_7_pas2 <- optim(data = filter(df_7, pas == "2"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf, -Inf
), upper = c(1, 1, Inf, Inf))
optim_7_pas3 <- optim(data = filter(df_7, pas == "3"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf, -Inf
), upper = c(1, 1, Inf, Inf))
optim_7_pas4 <- optim(data = filter(df_7, pas == "4"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf, -Inf
), upper = c(1, 1, Inf, Inf))
```

Argument for a and b values: The lowest accuracy we would expect is 50%, as this is the accuracy we would acquire from randomly drawing. The maximum accuracy would be 100%, if correctness was perfectly predicted from every input value.

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

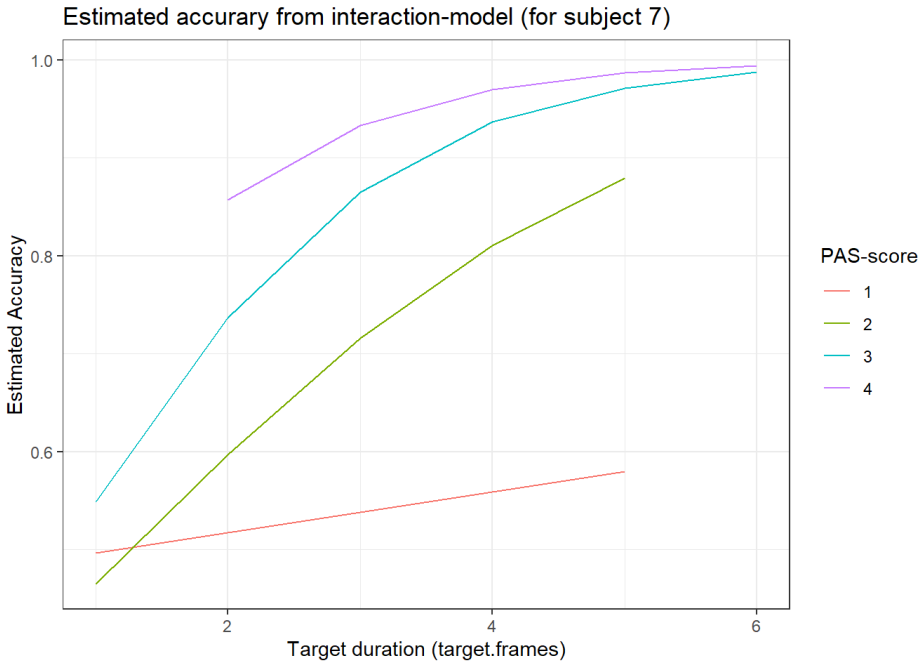
```
# Defining a sigmoid-function that takes parameters from optim-function and x-values from a df.
sigmoid_function <- function(optimfun, x) {
    optimfun$par[1] + ((optimfun$par[2]-optimfun$par[1])/(1+exp(1)^((optimfun$par[3]-x)/optimfun$par[4])))}

# Adding y_hats to dataframe
df_7$y_hat_pas1 <- sigmoid_function(optim_7_pas1,df_7$x)
df_7$y_hat_pas2 <- sigmoid_function(optim_7_pas2,df_7$x)
df_7$y_hat_pas3 <- sigmoid_function(optim_7_pas3,df_7$x)
df_7$y_hat_pas4 <- sigmoid_function(optim_7_pas4,df_7$x)

# plotting
df_7 %>%
  ggplot() +
  geom_smooth(aes(x = x, y = y_hat_pas1, color = "pas1")) +
  geom_smooth(aes(x = x, y = y_hat_pas2, color = "pas2")) +
  geom_smooth(aes(x = x, y = y_hat_pas3, color = "pas3")) +
  geom_smooth(aes(x = x, y = y_hat_pas4, color = "pas4")) +
  labs(title = "Estimated fits for accuracy ratings pr. PAS for subject 7",
       x = "target.frames",
       y = "Estimated accuracy ratings using sigmoid-function") +
  theme_bw()
```

```
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
## `geom_smooth()` using method = 'loess' and formula 'y ~ x'
```



7.1.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)

```
# Subsetting to only use values from subject 7
df_7_estimates <- df
df_7_estimates$fitted <- fitted(m5)
df_7_estimates <- df_7_estimates %>%
  filter(subject == "7")

# Plotting (NO OBSERVATIONS OF PAS = 4 FOR TARGET.FRAMES = 1)
df_7_estimates %>%
  ggplot() +
  geom_line(aes(x = target.frames, y = fitted, color = pas)) +
  labs(title = "Estimated accurary from interaction-model (for subject 7)") +
  labs(x = "Target duration (target.frames)", y = "Estimated Accuracy") +
  labs(color = "PAS-score") +
  theme_bw()
```



7.1.iv. Comment on the differences between the fits - mention some advantages and disadvantages of each way

The model created with the glm-function uses the data from all subjects to create a more appropriate model, whereas simply fitting the “best” sigmoid using the optim function for a single subject’s data will result in overfitting. That being said, using the optim-function forces a sigmoid shape on the data, which i suspect is easier when less data is available, as in the case for subject 7 above. The optim-function also has the advantage of providing the actual parameters for creating the sigmoid function.

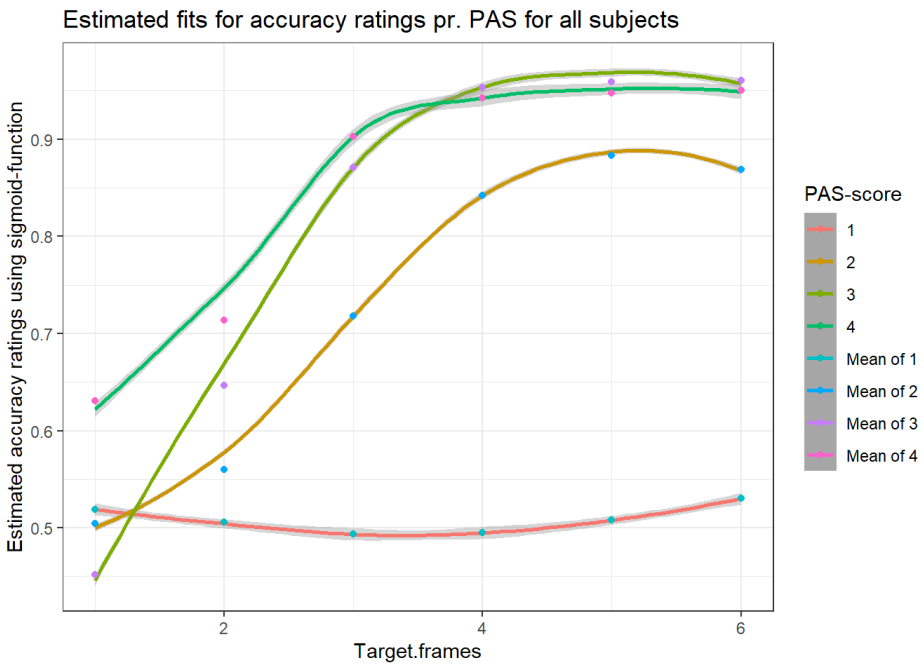
7.2. Finally, estimate the parameters for all subjects and each of their four PAS ratings. Then plot the estimated function at the group-level by taking the mean for each of the four parameters, *a*, *b*, *c* and *d* across subjects. A function should be estimated for each PAS-rating (it should look 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))

```
pas_rating_function <- function(dataframe, participant){

  # Subsetting the df
  dataframe <- dataframe %>%
    dplyr::filter(subject == participant) %>%
    dplyr::select(subject, target.frames, correct, pas) %>%
    dplyr::rename(x = target.frames, y = correct)

  # Specifying par
  par <- c(0.5, 1, 1, 1)

  optim_pas1 <- optim(data = filter(dataframe, pas == "1"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf,
 -Inf), upper = c(1, 1, Inf, Inf))
  optim_pas2 <- optim(data = filter(dataframe, pas == "2"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf,
 -Inf), upper = c(1, 1, Inf, Inf))
  optim_pas3 <- optim(data = filter(dataframe, pas == "3"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf,
 -Inf), upper = c(1, 1, Inf, Inf))
  optim_pas4 <- optim(data = filter(dataframe, pas == "4"), fn = RSS, par = par, method = 'L-BFGS-B', lower = c(0, 0, -Inf,
 -Inf), upper = c(1, 1, Inf, Inf))

  # Now i have 4 variables for each pas score based on the dataframe and participant
  dataframe$a_value_pas_1 <- optim_pas1$par[1]
  dataframe$b_value_pas_1 <- optim_pas1$par[2]
  dataframe$c_value_pas_1 <- optim_pas1$par[3]
  dataframe$d_value_pas_1 <- optim_pas1$par[4]

  dataframe$a_value_pas_2 <- optim_pas2$par[1]
  dataframe$b_value_pas_2 <- optim_pas2$par[2]
  dataframe$c_value_pas_2 <- optim_pas2$par[3]
  dataframe$d_value_pas_2 <- optim_pas2$par[4]

  dataframe$a_value_pas_3 <- optim_pas3$par[1]
  dataframe$b_value_pas_3 <- optim_pas3$par[2]
  dataframe$c_value_pas_3 <- optim_pas3$par[3]
  dataframe$d_value_pas_3 <- optim_pas3$par[4]

  dataframe$a_value_pas_4 <- optim_pas4$par[1]
  dataframe$b_value_pas_4 <- optim_pas4$par[2]
  dataframe$c_value_pas_4 <- optim_pas4$par[3]
  dataframe$d_value_pas_4 <- optim_pas4$par[4]

  # Running the sigmoid-function to get parameter estimates
  dataframe$y_hat_1 <- sigmoid_function(optim_pas1, dataframe$x)
  dataframe$y_hat_2 <- sigmoid_function(optim_pas2, dataframe$x)
  dataframe$y_hat_3 <- sigmoid_function(optim_pas3, dataframe$x)
  dataframe$y_hat_4 <- sigmoid_function(optim_pas4, dataframe$x)

  # Getting mean values per x (target.frames)
  dataframe <- dataframe %>%
    group_by(x) %>%
    mutate(y_hat_1_mean = mean(y_hat_1),
         y_hat_2_mean = mean(y_hat_2),
         y_hat_3_mean = mean(y_hat_3),
         y_hat_4_mean = mean(y_hat_4)) %>%
    ungroup()

  return(dataframe)
}

# Estimated values loop
new_df <- data.frame()

for (i in 1:29){
  newer_df <- pas_rating_function(df, i)
  new_df <- rbind(new_df, newer_df)
}

# Extracting mean parameters from parameters df (very clunky way to do it)
a_mean_pas_1 <- mean(new_df$a_value_pas_1)
b_mean_pas_1 <- mean(new_df$b_value_pas_1)
c_mean_pas_1 <- mean(new_df$c_value_pas_1)
d_mean_pas_1 <- mean(new_df$d_value_pas_1)

a_mean_pas_2 <- mean(new_df$a_value_pas_2)
b_mean_pas_2 <- mean(new_df$b_value_pas_2)
c_mean_pas_2 <- mean(new_df$c_value_pas_2)
d_mean_pas_2 <- mean(new_df$d_value_pas_2)

a_mean_pas_3 <- mean(new_df$a_value_pas_3)
b_mean_pas_3 <- mean(new_df$b_value_pas_3)
c_mean_pas_3 <- mean(new_df$c_value_pas_3)
d_mean_pas_3 <- mean(new_df$d_value_pas_3)
```

```
a_mean_pas_4 <- mean(new_df$a_value_pas_4)
b_mean_pas_4 <- mean(new_df$b_value_pas_4)
c_mean_pas_4 <- mean(new_df$c_value_pas_4)
d_mean_pas_4 <- mean(new_df$d_value_pas_4)

# Calculating mean y_hats pr. pas score
new_df <- new_df %>%
  group_by(x) %>%
  mutate(y_hat_1_mean_grand = mean(y_hat_1),
       y_hat_2_mean_grand = mean(y_hat_2),
       y_hat_3_mean_grand = mean(y_hat_3),
       y_hat_4_mean_grand = mean(y_hat_4)) %>%
  ungroup()
```

```
# Plotting
new_df %>%
  ggplot() +
  geom_smooth(aes(x = x, y = y_hat_1, color = "1"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_2, color = "2"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_3, color = "3"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_4, color = "4"), method = "loess") +
  geom_point(aes(x = x, y = y_hat_1_mean_grand, color = "Mean of 1"))+
  geom_point(aes(x = x, y = y_hat_2_mean_grand, color = "Mean of 2"))+
  geom_point(aes(x = x, y = y_hat_3_mean_grand, color = "Mean of 3"))+
  geom_point(aes(x = x, y = y_hat_4_mean_grand, color = "Mean of 4"))+
  labs(title = "Estimated fits for accuracy ratings pr. PAS for all subjects",
       x = "Target.frames",
       y = "Estimated accuracy ratings using sigmoid-function",
       color = "PAS-score") +
  theme_bw()
```

```
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
## `geom_smooth()` using formula 'y ~ x'
```



7.2.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.

Simply fitting the sigmoid with lowest RSS doesn't take into account the goal of statistics as we see it - it tells us nothing about population, explanatory power, significance etc., but it might be useful in classification tasks. Also, by taking the mean of the estimated values, you reduce the resolution of the data, which a glm-model would take into account. Perhaps the sigmoid-by-hand is faster/easier in some cases for plotting or otherwise.

Assignment 3, Methods 3, 2021, autumn semester

Aleksander Moeslund Wael, 09/12/2021

Importing packages

In [ ]:
```
from matplotlib.colors import Colormap
import matplotlib.pyplot as plt
plt.rcParams['figure.dpi'] = 200 # HIGH DPI PLOTS PLEASE
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

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

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

In [ ]:
```
data = np.load("megmag_data.npy")
```

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 [ ]:
```
data.shape
```

Out[ ]:
```
(682, 102, 251)
```

There are 682 repetitions, 102 sensors and 251 time samples.

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.

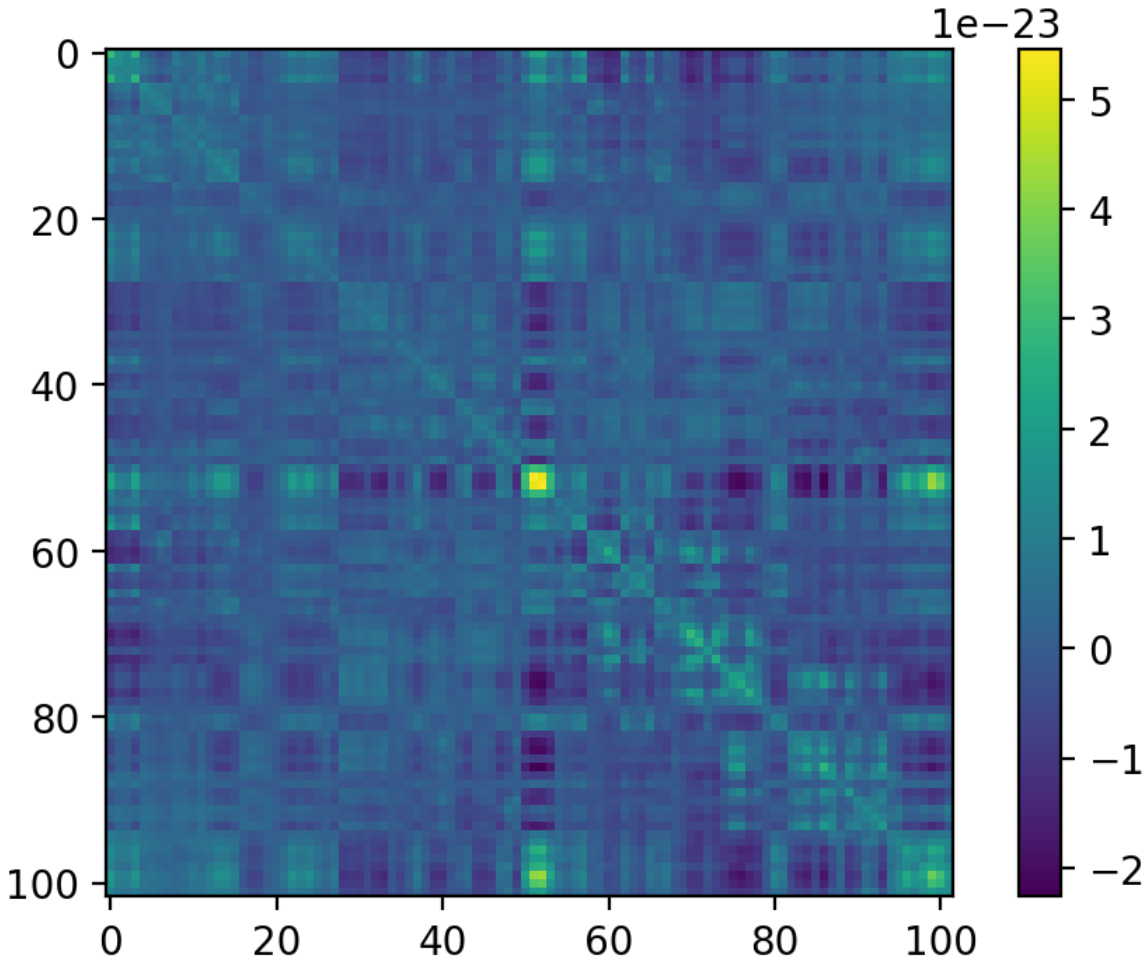
In [ ]:
```
times = np.arange(-200, 804, 4)
```

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 [ ]:
```
covar = np.zeros(shape = (102,102))

for i in range(682):
    X = data[i]
    covar = covar + X @ np.transpose(X)

covar = (1/682)*covar

plt.close("all")
plt.figure()
plt.imshow(covar)
plt.colorbar()
plt.show()
```



There is indeed covariance between the sensors, but it is unclear why this is. It might be that sensors pick up the same signals, OR they pick up different signals but with similar values.

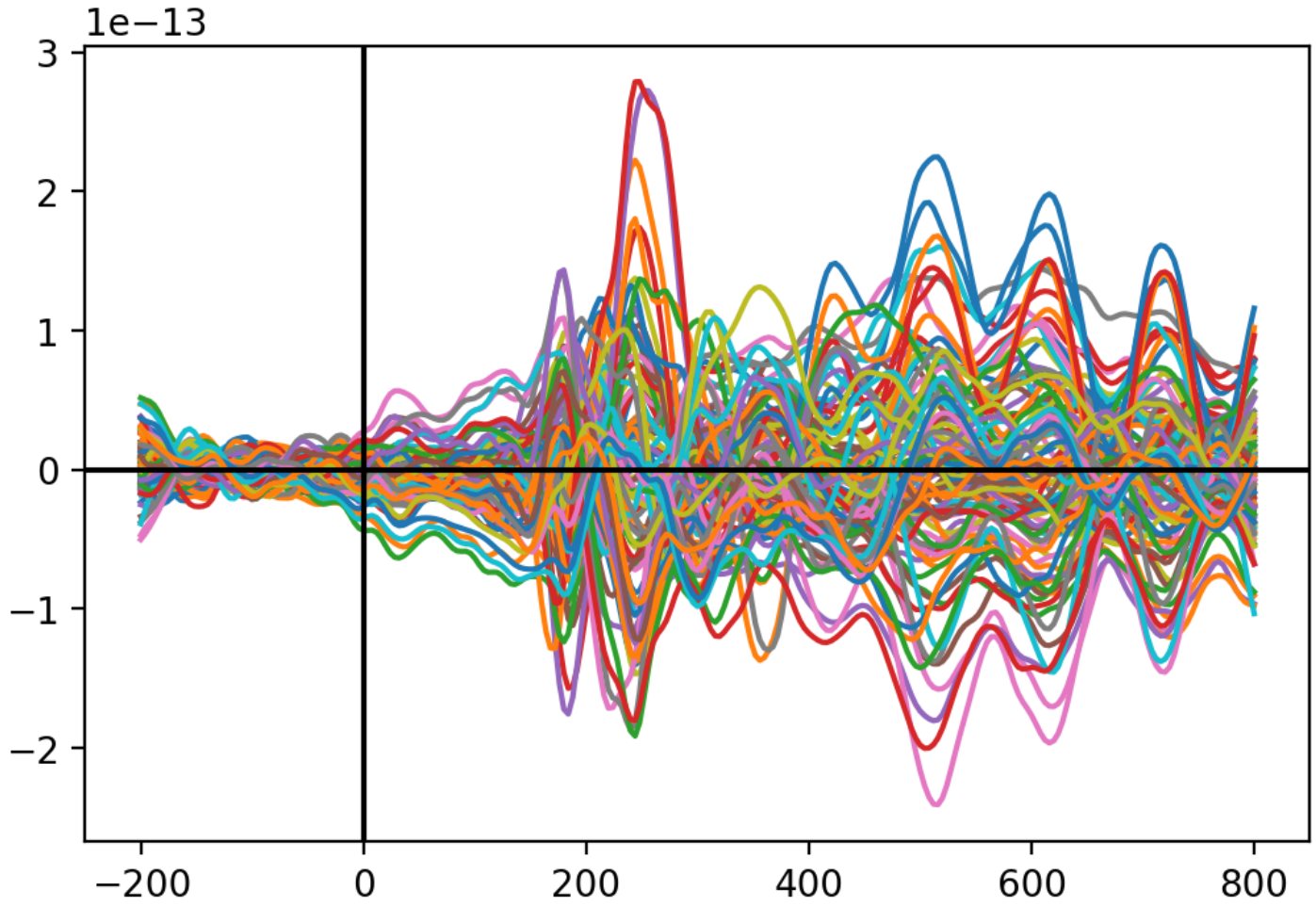
iv. Make an average over the repetition dimension using `np.mean` - use the `axis` argument. (The resulting array should have two dimensions ~~with time as the first and magnetic field as the second~~ with sensor as the first and time as the second)

In [ ]:
```
datamean = data.mean(axis=0)
datamean.shape
```

Out[ ]:
```
(102, 251)
```

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 [ ]:
```
plt.close("all")
plt.figure()
plt.plot(times, datamean.T)
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0, color = "black")
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.

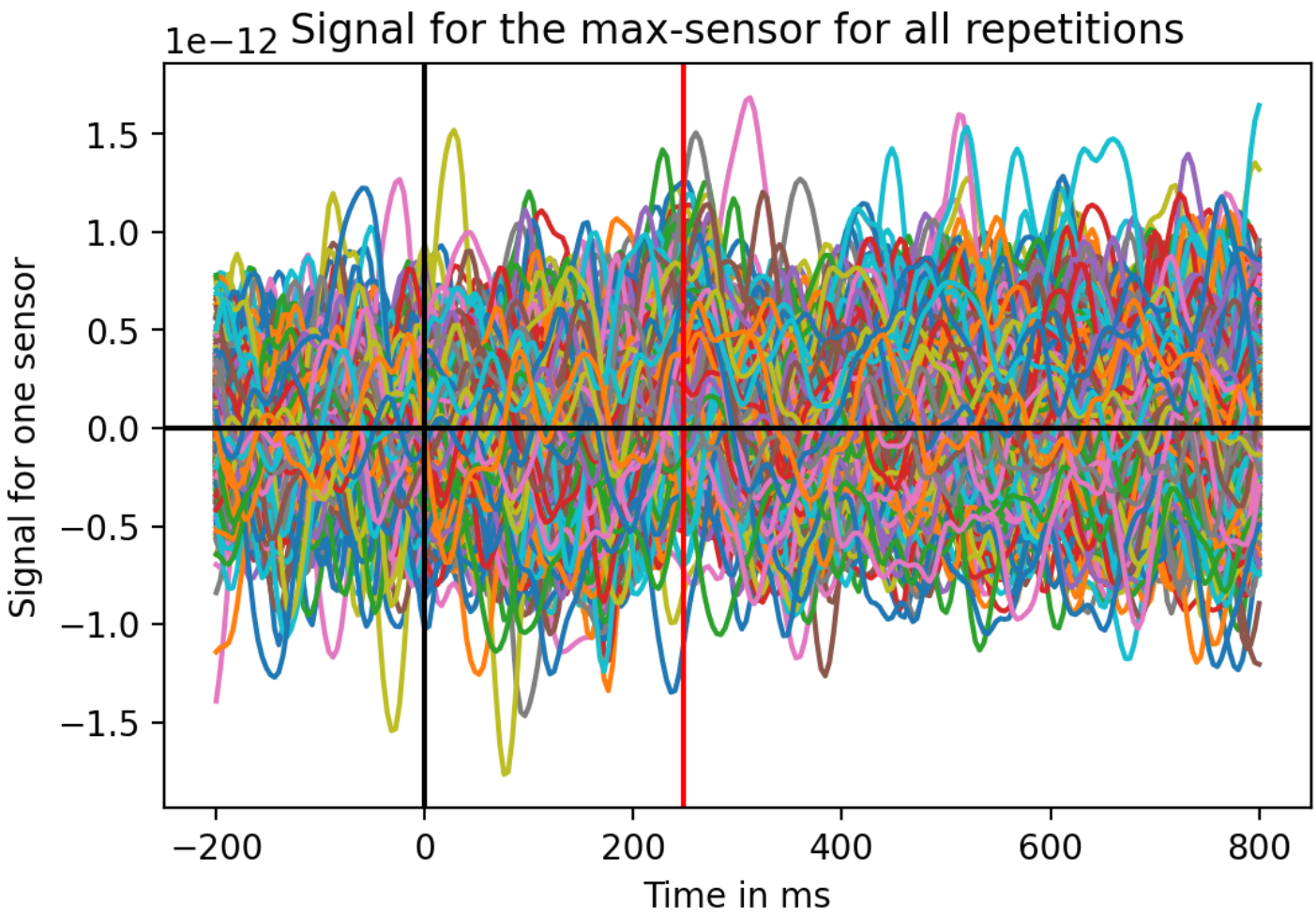
In [ ]:
```
# finding the biggest value
print(np.amax(datamean))

# finding the flat index for the biggest value
print(np.argmax(datamean, axis = 1))

# converting flat index into tuple so i know which sensor it is
print(np.unravel_index(np.argmax(datamean), shape = (102, 251))) # shape = the space where it searches for the index.
```

```
2.7886216843591933e-13
[127  95   5 133  94  94  94 111  96  95  95  95 214 208 207 201  96  95
 246  96 238 112 207 222   0 145 139   0 113 113 112 111 111 112 113 111
 111 143 111 167 142 112 111 203 111 110 172 202 212 151 179 180 179 180
 142 112 143 136 111 200 177 111 144 112 113 143 201 203 157 179 179 111
 112 112  98  95  96  99 128 129 141 141  94  94  94  95  95  95 139 139
  94  95  95  94  95  95 181 206 180 180 180   0]
(73, 112)
```

The sensor with maximal magnetic field (on average) is no. 73

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 [ ]:
```
plt.figure()
plt.plot(times, data[:,73,:].T)
plt.axvline(times[112], color = "red")
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0, color = "black")
plt.xlabel("Time in ms")
plt.ylabel("Signal for one sensor")
plt.title("Signal for the max-sensor for all repetitions")
plt.show()
```



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

- As can be seen by the plot, there is some diffeerence between repetitions. These can be due to hidden variables or noise in the data, which i'm not interested in modelling. Taking the mean of repetitions is a way of isolating the signal which is what matters for analysis.

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

In [ ]:
```
y = np.load("pas_vector.npy")
```

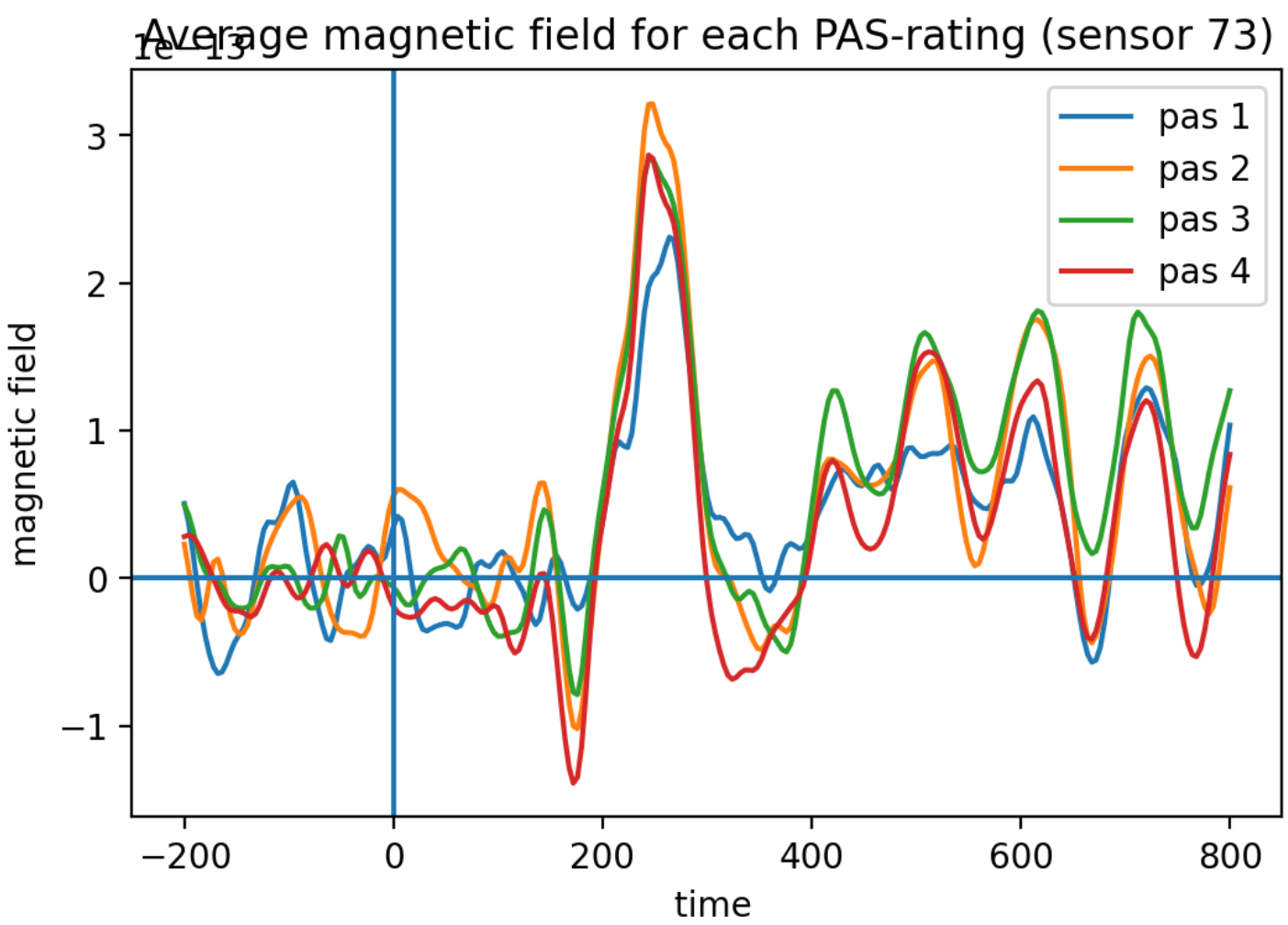
i. Which dimension in the `data` array does it have the same length as?

- It has the same length as repetitions, because there is a PAS rating per repitition.

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~~ 1.1.vi

In [ ]:
```
# subsetting the data, so I'm only looking at sensor 73
data_73 = data[:, 73, :]

# in y I find the indices for each different pas rating and assign them to new lists
pas_1 = np.where(y == 1)
pas_2 = np.where(y == 2)
pas_3 = np.where(y == 3)
pas_4 = np.where(y == 4)

# finding the average brain activation in sensor 73 seperated by pas-rating
sens_73_pas1_avg = np.mean(data_73[pas_1], axis = 0)
sens_73_pas2_avg = np.mean(data_73[pas_2], axis = 0)
sens_73_pas3_avg = np.mean(data_73[pas_3], axis = 0)
sens_73_pas4_avg = np.mean(data_73[pas_4], axis = 0)

# plotting this baby
plt.figure
plt.plot(times, sens_73_pas1_avg)
plt.plot(times, sens_73_pas2_avg)
plt.plot(times, sens_73_pas3_avg)
plt.plot(times, sens_73_pas4_avg)
plt.axvline()
plt.axhline()
plt.xlabel(" time")
plt.ylabel("magnetic field")
plt.title("Average magnetic field for each PAS-rating (sensor 73)")
plt.legend(["pas 1", "pas 2", "pas 3", "pas 4"])
plt.show
```

Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



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?

- I would expect the peaks at 250 to be ordered by PAS-rating with PAS-4 at the top (largest magnetic field). It seems that PAS-1 is the lowest as expected, by PAS-2 (which is described as a "weak glimpse") has a higher peak than PAS-3 and PAS-4 (which are quite similiar). This pattern is somewhat similar in the other peaks at 450-750 ms, though with some variability.

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

In [ ]:
```
# subsetting
data_1_2 = data[y < 3,:,:]
y_1_2 = y[y < 3]
```

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 [ ]:
```
# repetition as rows, and sensor and time as columns
X_1_2 = data_1_2.reshape(214, -1)
X_1_2.shape
```

Out[ ]:
```
(214, 25602)
```

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

In [ ]:
```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_1_2_scaled = 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
```

```
logR = LogisticRegression(penalty='none') # no regularisation

logR.fit(X_1_2_scaled, y_1_2)
```

Out[ ]:
```
LogisticRegression(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 [ ]:
```
print(logR.score(X_1_2_scaled, y_1_2))
```

```
1.0
```

The score was perfect, so it seems we are overfitting. We aren't testing the model on new data, which is probably one of the reasons.

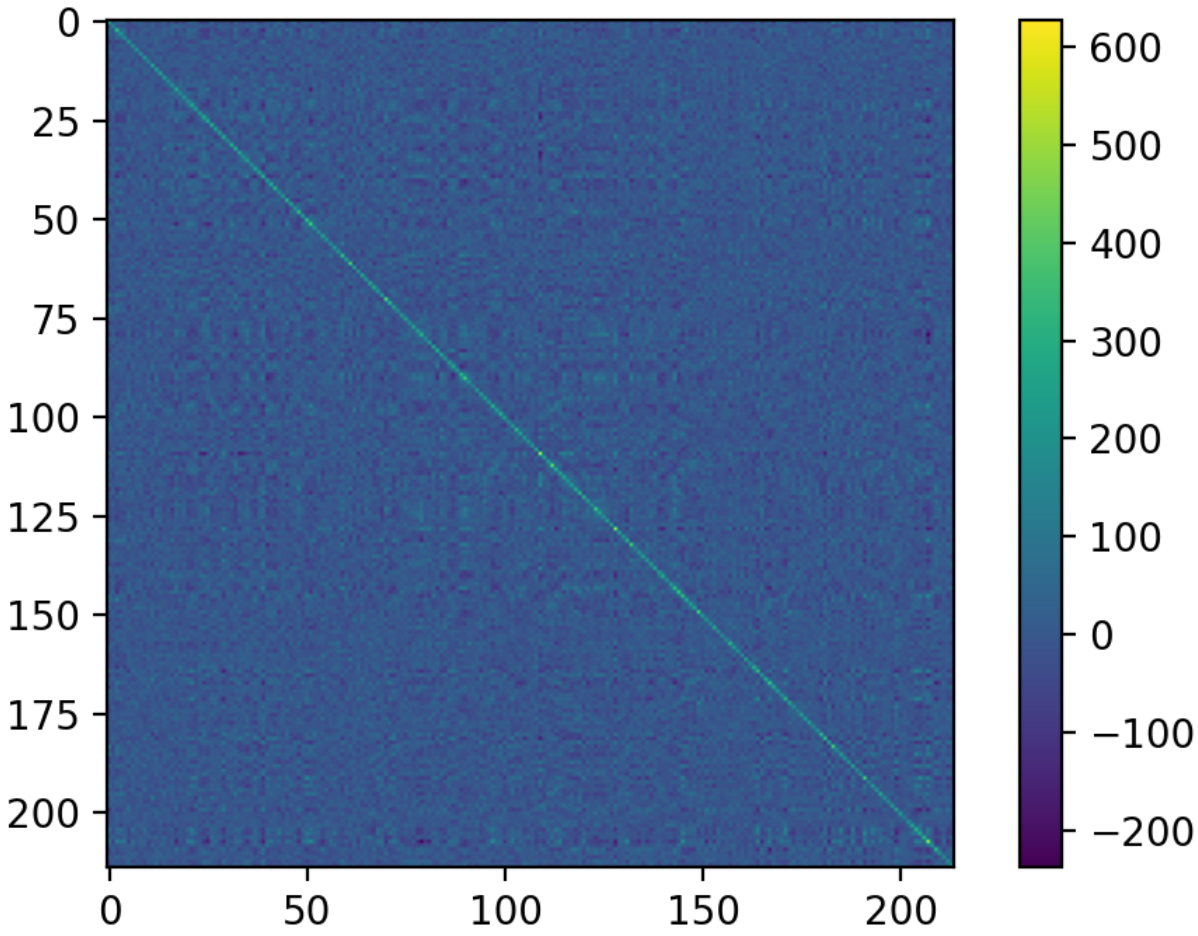
vi. Now apply the *L1* penalty instead - how many of the coefficients (`.coef_`) are non-zero after this?

In [ ]:
```
# Fitting
logR = LogisticRegression(penalty="l1", solver='liblinear', random_state=1) # With regularization
fit1 = logR.fit(X_1_2_scaled, y_1_2)
print(fit1)

# Finding nonzero coefs
print(np.sum(fit1.coef_ != 0), "coefficients are non-zero.")
```

```
LogisticRegression(penalty='l1', random_state=1, solver='liblinear')
282 coefficients are non-zero.
```

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 [ ]:
```
coefs = logR.coef_.flatten()
non_zero = coefs != 0
X_reduced = X_1_2_scaled[:, non_zero]

# Non-zero coefficients covariance matrix
coef_covar = X_reduced @ np.transpose(X_reduced)

plt.close("all")
plt.figure()
plt.imshow(coef_covar)
plt.colorbar()
plt.show()
```



We see less covariance overall, the diagonal is very clear and defined. There is still covariance of the non-zero coefficients, but it is better than the sensor covariance matrix in 1.1.iii.

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

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 [ ]:
```
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

# Making equal data with function
y_1_2 = np.array(y_1_2) # Has to be array instead of list, thats why
data_1_2_equal, y_1_2_equal = equalize_targets_binary(data_1_2, y_1_2) # Assigning new data
X_1_2_equal = data_1_2_equal.reshape(198, -1)
X_1_2_equal = sc.fit_transform(X_1_2_equal)
```

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

In [ ]:
```
cv = StratifiedKFold()

logR = LogisticRegression()
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))
```

```
0.5402564102564102
```

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 [ ]:
```
# With C = 1e5
cv = StratifiedKFold()

logR = LogisticRegression(C=1e5, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))

# With C = 1e1
cv = StratifiedKFold()

logR = LogisticRegression(C=1e1, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))

# With C = 1e-5
cv = StratifiedKFold()

logR = LogisticRegression(C=1e-5, penalty="l2")
logR.fit(X_1_2_equal, y_1_2_equal)

scores = cross_val_score(logR, X_1_2_equal, y_1_2_equal, cv=5)
print(np.mean(scores))
```

```
0.5353846153846155
0.5252564102564102
0.5956410256410256
```

In [ ]:
```
# Difference between models
0.5956410256410256-0.5252564102564102
```

```
0.07038461538461538
```

Out[ ]:

The best model (C=1e-5) is on average about 7% better at prediciting than with C=1e1.

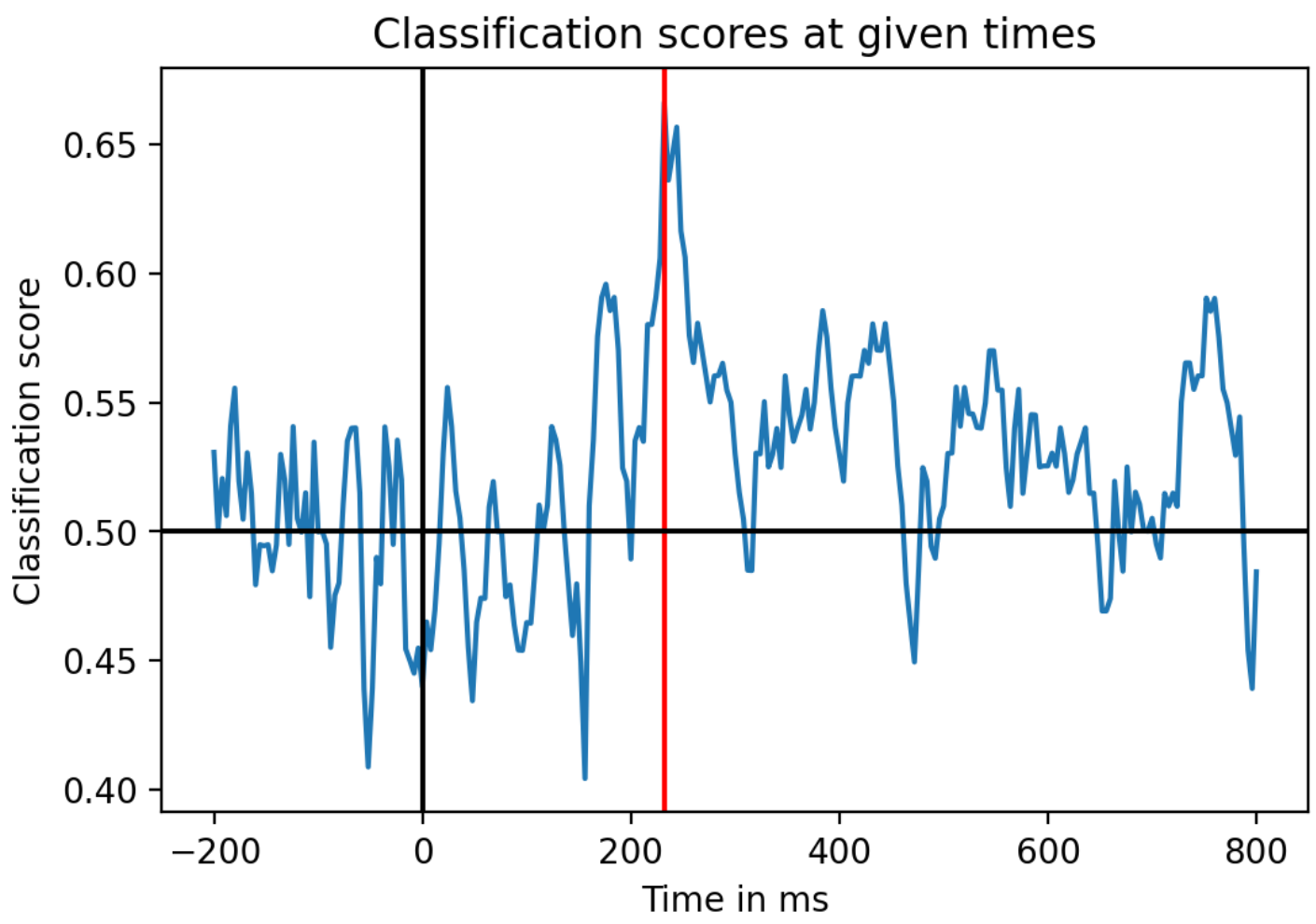
v. Instead of fitting a model on all `n_sensors * n_samples` features, fit a logistic regression (same kind as in Exercise 2.2.iv (use the `C` that resulted in the best prediction)) for each time sample and use the same cross-validation as in Exercise 2.2.iii. What are the time points where classification 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 [ ]:
```
# I need 251 models.
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-5, penalty="l2", solver = "liblinear")
cv_scores = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_2_equal[:,:,i])
    logR.fit(t, y_1_2_equal)
    scores = cross_val_score(logR, t, y_1_2_equal, cv=5)
    cv_scores.append(np.mean(scores))
```

In [ ]:
```
# Picking highest score
print(np.amax(cv_scores)) #  What is the max value?
print(np.argmax(cv_scores)) # Where is the max value? (Indeci)
```

```
0.6661538461538462
108
```

In [ ]:
```
plt.figure()
plt.plot(times, cv_scores)
plt.axvline(x = 0, color = "black")
plt.axvline(times[108], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

print(times[108]) # Where classification is best

# Classification is best at 232 ms.
```




```
232
```

Classification is best between 200-250 and 300-450 ms.

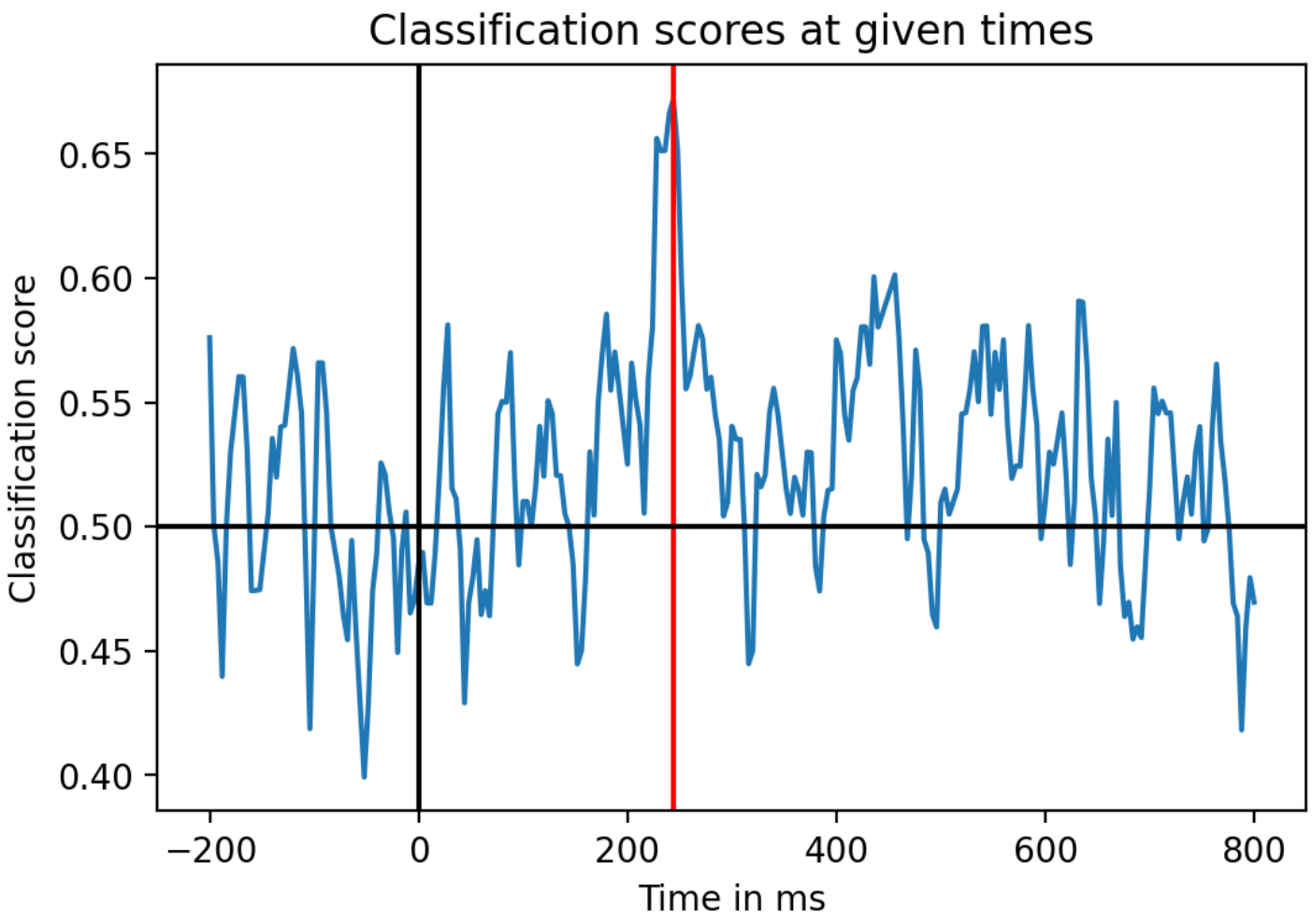
vi. Now do the same, but with L1 regression - set `C=1e-1` - what are the time points when classification is best? (make a plot)?

In [ ]:
```
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-1, penalty="l1", solver = "liblinear")
cv_scores = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_2_equal[:,:,i])
    logR.fit(t, y_1_2_equal)
    scores = cross_val_score(logR, t, y_1_2_equal, cv=5)
    cv_scores.append(np.mean(scores))
```

In [ ]:
```
# Picking highest score
print(np.amax(cv_scores))
print(np.argmax(cv_scores)) # Indeci with highest classification
```

```
0.6720512820512822
111
```

In [ ]:
```
plt.figure()
plt.plot(times, cv_scores)
plt.axvline(x = 0, color = "black")
plt.axvline(times[111], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

times[111]
```



Out[ ]:
```
244
```

Classification is best at 244 ms.

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 [ ]:
```
# Prepare array
data_1_4 = data[(y != 2) & (y != 3),:,:]
print(data_1_4.shape)
```

```
(359, 102, 251)
```

In [ ]:
```
# Prepare target vector
y_1_4 = []
```

```
for i in range(len(y)):
    if y[i] == 1:
        y_1_4.append(1)
    if y[i] == 4:
        y_1_4.append(4)

# repetition as rows, and sensor and time as columns
X_1_4 = data_1_4.reshape(359, -1)
X_1_4.shape
```

Out[ ]:
```
(359, 25602)
```

In [ ]:
```
# Scaling data
X_1_4_scaled = sc.fit_transform(X_1_4)
```

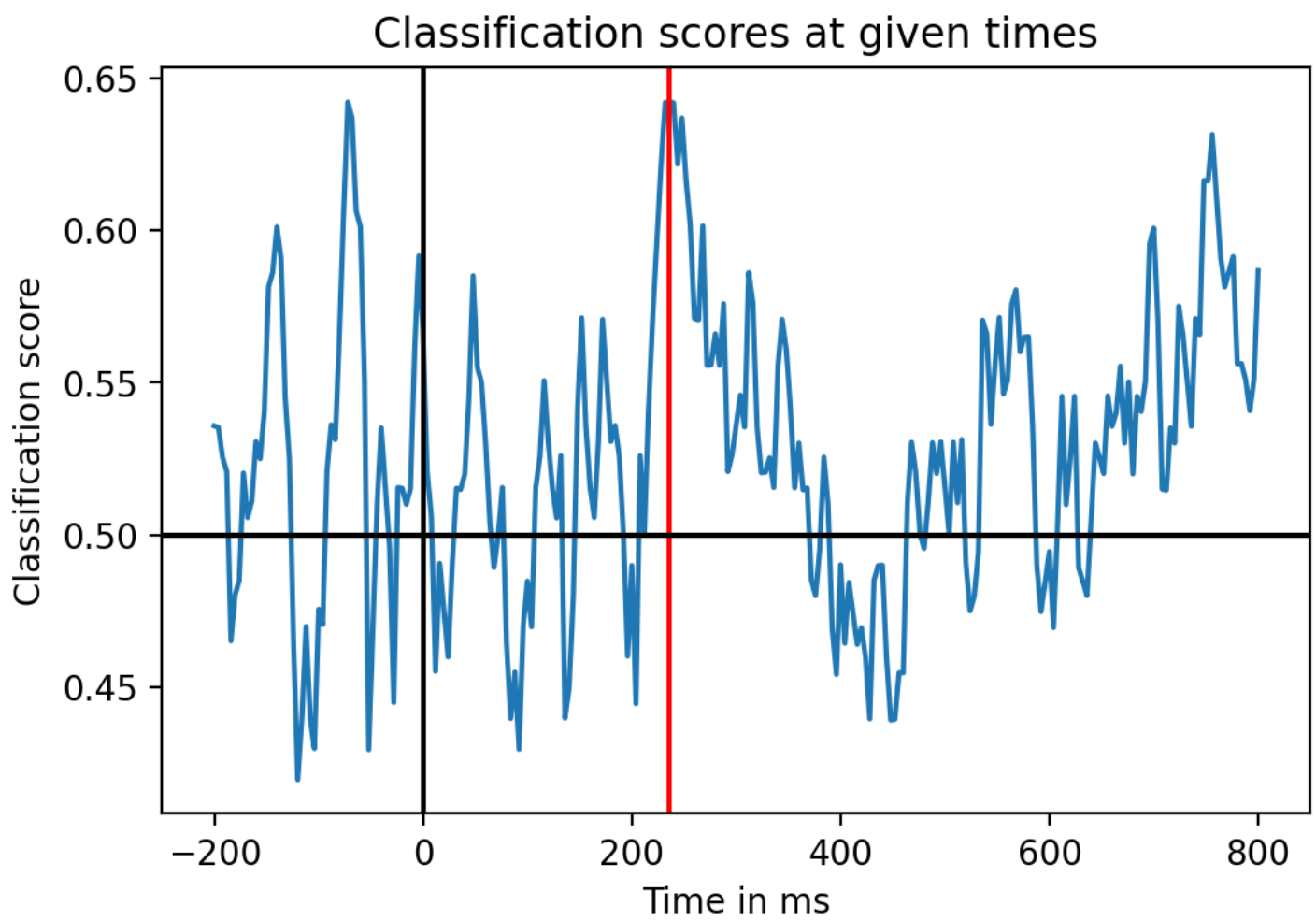
In [ ]:
```
# Making equal data with function
y_1_4 = np.array(y_1_4) # Has to be array instead of list, thats why
data_1_4_equal, y_1_4_equal = equalize_targets_binary(data_1_4, y_1_4) # Assigning new data
X_1_4_equal = data_1_4_equal.reshape(198, -1)
X_1_4_equal = sc.fit_transform(X_1_4_equal)
```

In [ ]:
```
cv = StratifiedKFold()
logR = LogisticRegression(C=1e-1, penalty="l1", solver = "liblinear")
cv_scores_1_4 = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_1_4_equal[:,:,i])
    logR.fit(t, y_1_4_equal)
    scores = cross_val_score(logR, t, y_1_4_equal, cv=5)
    cv_scores_1_4.append(np.mean(scores))
```

In [ ]:
```
# Picking highest score
print(np.amax(cv_scores_1_4))
print(np.argmax(cv_scores_1_4)) # Indeci with highest classification
```

```
0.6423076923076924
109
```

In [ ]:
```
# Plotting
plt.figure()
plt.plot(times, cv_scores_1_4)
plt.axvline(x = 0, color = "black")
plt.axvline(times[109], color = "red")
plt.axhline(y = 0.5, color = "black") # Chance level is 50% for binary classification
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()

times[109]
```



Out[ ]: 236

Classification is best at 236 ms.

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?

It appears that it is more succesful to classify 1 vs 2 compared to 1 vs 4, which is quite surprising as PAS-score can be viewed as a pseudo-variable where scores of 1 should be more similar to 2 than to 4, which is also apparent from the plot of PAS-ratings and magnetic field from sensor 73. The difference between the two max-scores is only about 3%, which isn't huge. I would say that pairwise classification for PAS is somewhat possible in the 200-400 ms range, but the scores are still low.

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 [ ]:
```
# Define function
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=False)
    third_choice = np.random.choice(indices[2], size=min_count, replace=False)
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=False)

    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
```

In [ ]:
```
# Making data equal
data_equal, y_equal = equalize_targets(data, y)
```

```
print(data_equal.shape)
print(y_equal.shape)
```

```
(396, 102, 251)
(396,)
```

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 [ ]:
```
# Import
from sklearn.svm import SVC

# Making classes
svm_linear = SVC(kernel="linear")
svm_radial = SVC(kernel="rbf")

# Converting to 2d array
X_equal = data_equal.reshape(396, -1)
print(X_equal.shape)

# Scaling data
X_equal_scaled = sc.fit_transform(X_equal)

# Fitting
scores_svm_linear = cross_val_score(svm_linear, X_equal_scaled, y_equal, cv=cv)
print("The linear basis classifier score is", np.mean(scores_svm_linear))

# Fitting
scores_svm_radial = cross_val_score(svm_radial, X_equal_scaled, y_equal, cv=cv)
print("The radial basis classifier score is", np.mean(scores_svm_radial))
```

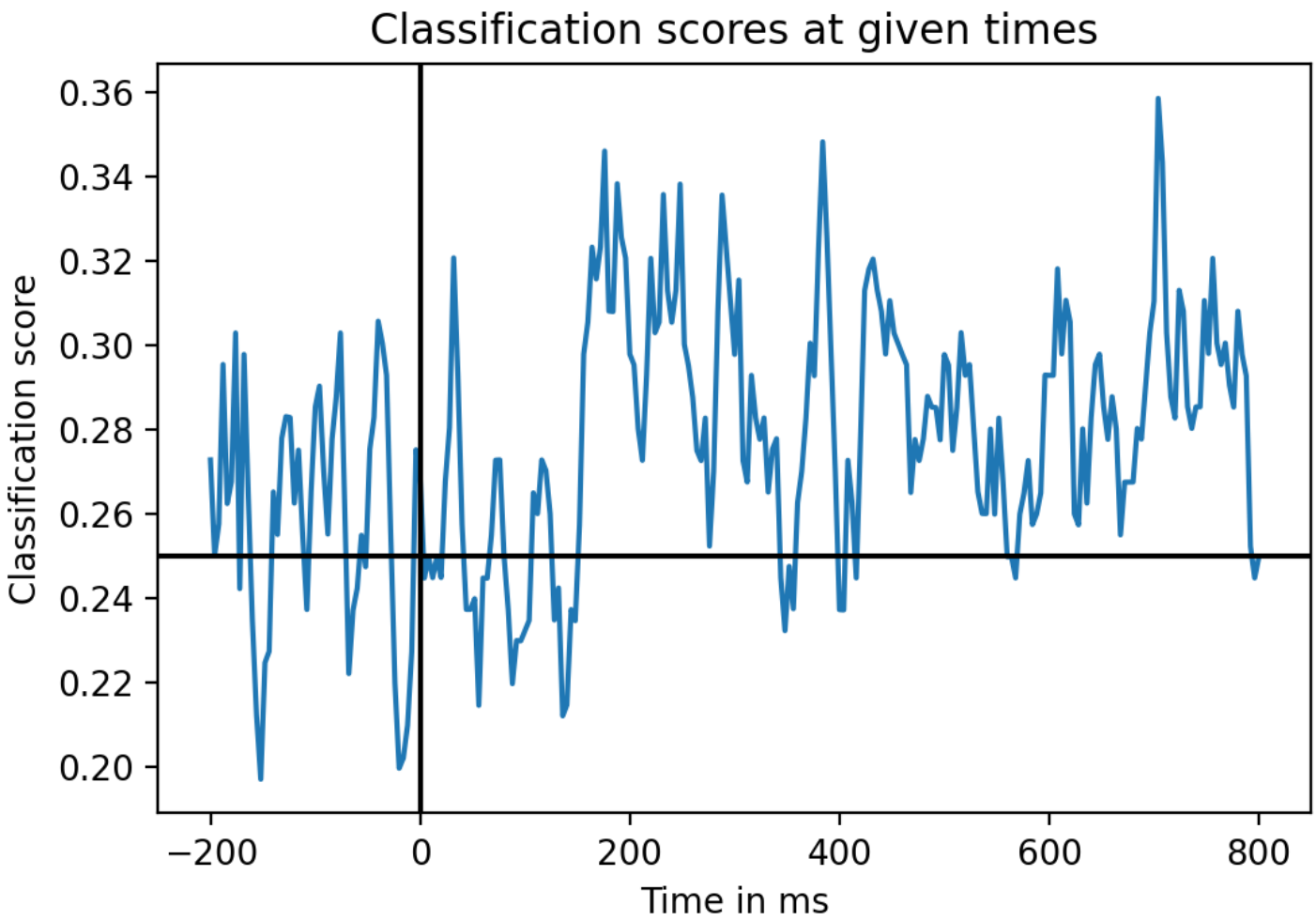
```
(396, 25602)
The linear basis classifier score is 0.2928164556962025
The radial basis classifier score is 0.3333544303797468
```

It appears that the radial classifier is best with mean scores of 0,33. Though chance level is at 0,25, so this is a pretty bad performance.

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 [ ]:
```
# I need 251 models.
cv = StratifiedKFold()
svm_radial = SVC(kernel="rbf")
cv_scores_svm = []

# Subsetting time
for i in range(251):
    t = sc.fit_transform(data_equal[:,:,i])
    logR.fit(t, y_equal)
    scores = cross_val_score(svm_radial, t, y_equal, cv=5)
    cv_scores_svm.append(np.mean(scores))
```

In [ ]:
```
# Plotting
plt.figure()
plt.plot(times, cv_scores_svm)
plt.axvline(x = 0, color = "black")
plt.axhline(y = 0.25, color = "black") # Chance level is 25% for classification with 4 classes
plt.xlabel("Time in ms")
plt.ylabel("Classification score")
plt.title("Classification scores at given times")
plt.show()
```



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

- It is better than chance level at this time, with classification scores around 0.30-0.34.

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`

In [ ]:
```
# Import
from sklearn.model_selection import train_test_split

# Splitting data
X_train, X_test, y_train, y_test = train_test_split(X_equal, y_equal, test_size=0.30)
```

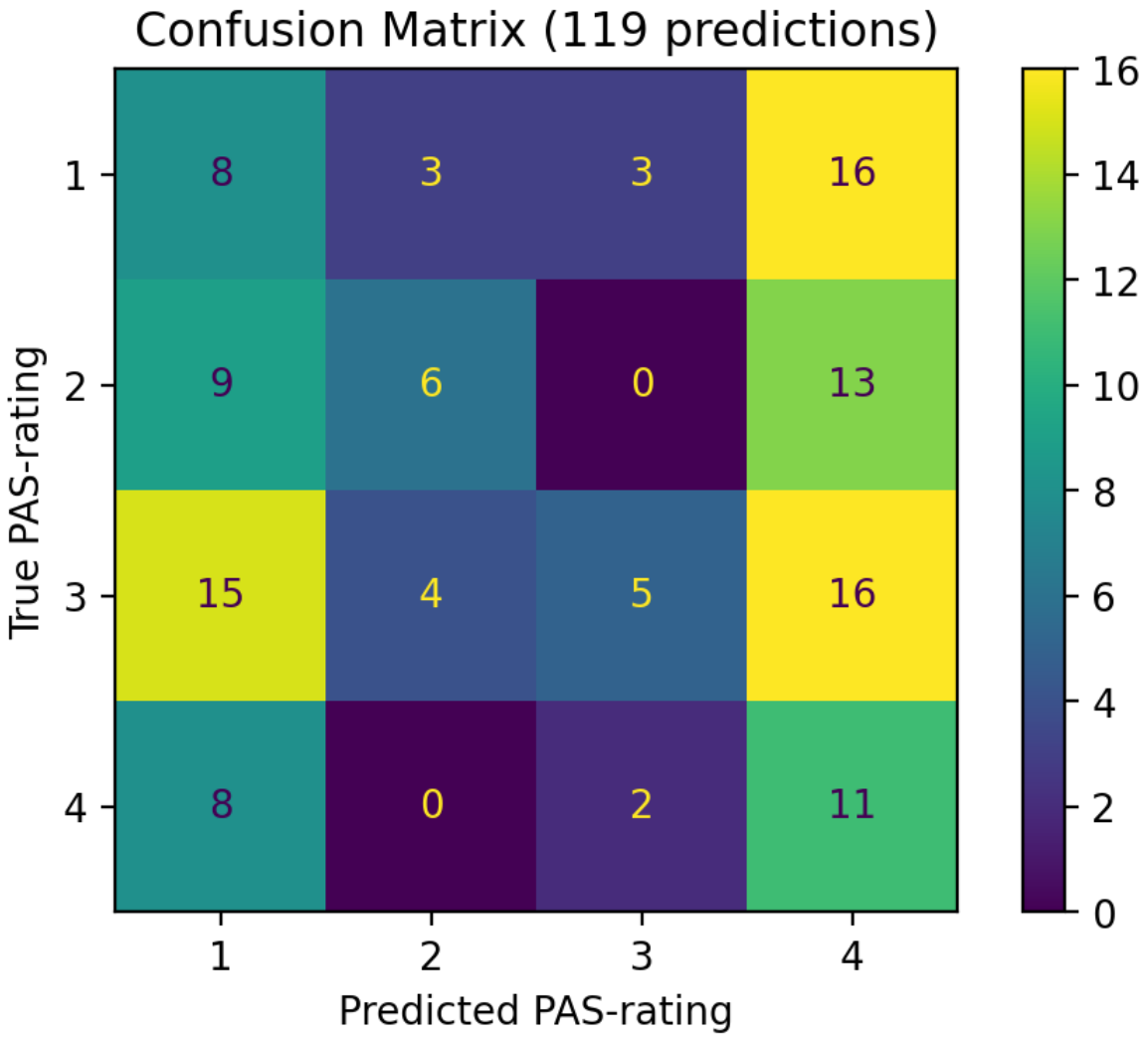
i. Use the kernel that resulted in the best classification in Exercise 3.1.ii and `fit` the training set and `predict` on the test set. This time your features are the number of sensors multiplied by the number of samples.

In [ ]:
```
# Setting up class
svm_radial = SVC(kernel="rbf")

# Fitting
svm_radial.fit(X_train, y_train)

# Predicting
predictions = svm_radial.predict(X_test)
print(predictions)
```

```
[2 4 4 1 4 1 4 4 4 4 1 3 4 4 2 1 2 4 1 4 4 4 1 2 4 4 4 1 4 1 1 2 1 1 3 4 4
 2 4 1 1 2 1 4 4 1 2 2 4 4 1 4 1 3 1 3 4 1 4 4 4 1 4 1 1 3 1 4 1 1 1 2 4 4
 4 4 4 4 3 4 1 1 4 4 3 1 4 1 1 1 4 1 2 4 1 4 4 1 4 4 3 1 1 4 4 4 4 1 2 3 4 4
 1 4 3 4 4 2 1 4]
```

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 [ ]:
```
# Import
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay as cmd

# Plot
plt.close("all")
cmd.from_estimator(svm_radial, X_test, y_test) # Plotting given estimator, test data and true labels.
plt.xlabel("Predicted PAS-rating")
plt.ylabel("True PAS-rating")
plt.title("Confusion Matrix (119 predictions)")
plt.show()
```



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?

- It appears that all PAS-ratings were often classified as PAS 4, but also quite often as 1, and is therefore biased towards these. It also means that it often correctly classifies PAS 1 and 4, but at the expense of misclassifying PAS 2 and 3. I would also assume that PAS ratings would more often be misclassified as the adjacent ratings, e.g. PAS 2 would more often be misclassified as 1 or 3 than 4. This doesn't seem to be the case though.

Assignment 4, Methods 3, 2021, autumn semester

Aleksander Moeslund Wael, 02/12/2021

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

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

Importing packages

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import Colormap
plt.rcParams['figure.dpi'] = 200 # HIGH DPI PLOTS PLEASE
import warnings
warnings.filterwarnings('ignore')
```

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 [ ]:
```
data = np.load("megmag_data.npy")
y = np.load("pas_vector.npy")
```

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

In [ ]:
```
# Define function
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=False)
    third_choice = np.random.choice(indices[2], size=min_count, replace=False)
    fourth_choice = np.random.choice(indices[3], size=min_count, replace=False)

    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

# Equalize targets
data_eq, y_eq = 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)
print(np.where(times==248)) #248 is at index 112

# Data reduction
data_eq_112 = data_eq[:,:,112]
```

```
(array([112], dtype=int64),)
```

iv. Scale the data using `StandardScaler`

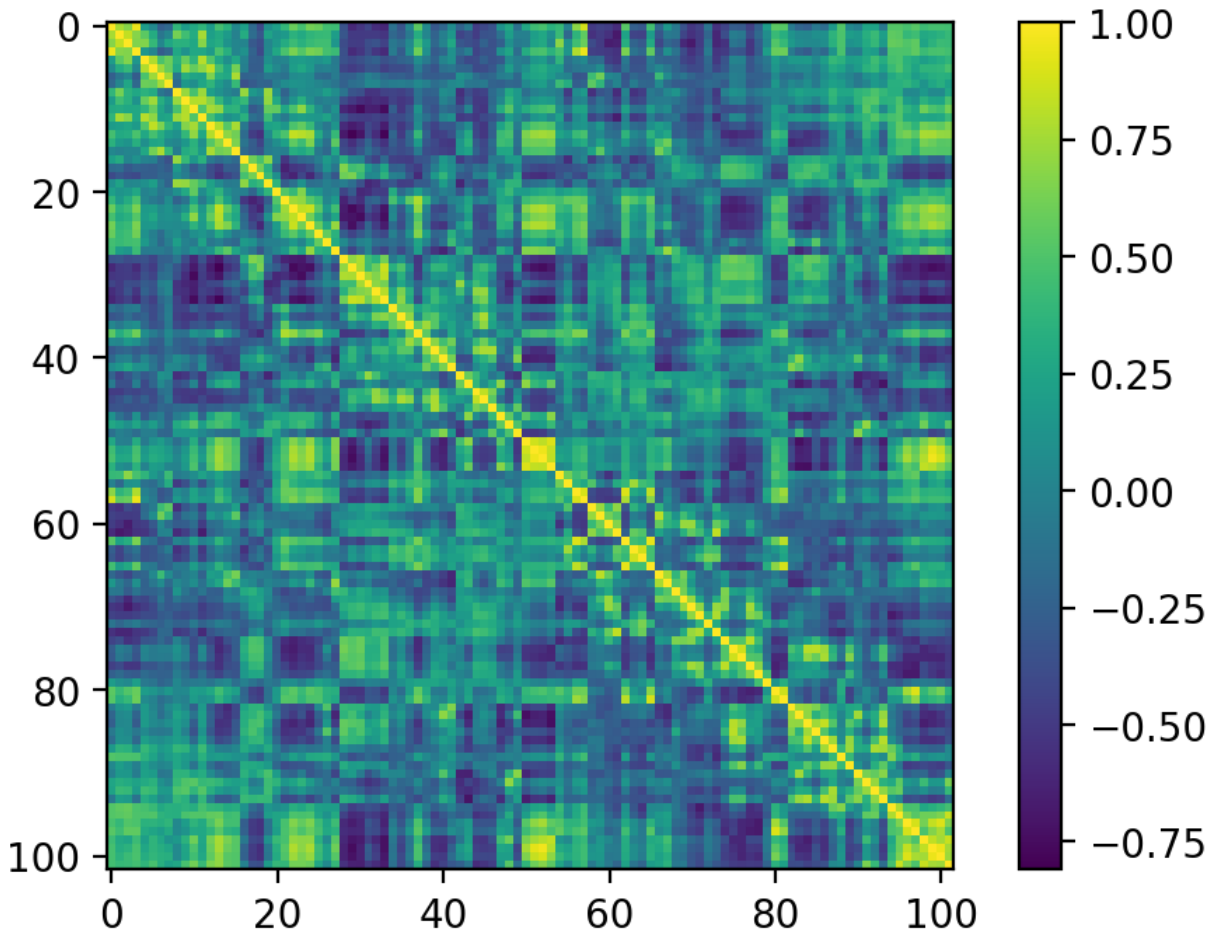
In [ ]:
```
from sklearn.preprocessing import StandardScaler
```

```
sc = StandardScaler()
data_eq_scaled = sc.fit_transform(data_eq_112)
```

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 [ ]:
```
# Making cov matrix
covariance_matrix = np.cov(data_eq_scaled.T)

# Plotting
plt.imshow(covariance_matrix)
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1a4dc3604c0>
```



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

- There is quite some covariance between the sensors, which might imply that sensors are picking up the same signals. It might also apply that sensors pick up different signals that have the same strength, at the same time.

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

In [ ]:
```
print("Value obtained from matrix_rank function is", np.linalg.matrix_rank(covariance_matrix))
```

```
Value obtained from matrix_rank function is 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 [ ]:
```
# Creating two arrays
eigenvalues, eigenvectors = np.linalg.eig(covariance_matrix)
eigenvalues = np.real(eigenvalues)
eigenvectors = np.real(eigenvectors)
```

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 [ ]:
```
eigenvalues = np.abs(eigenvalues)
```

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 [ ]:
```
# Sorting indicies based on values from lowest to highest
eigenvalues_index = np.argsort(eigenvalues)

# Flipping indici order to get highest to lowest (np.flip)
sorted_indices = np.flip(eigenvalues_index)
```

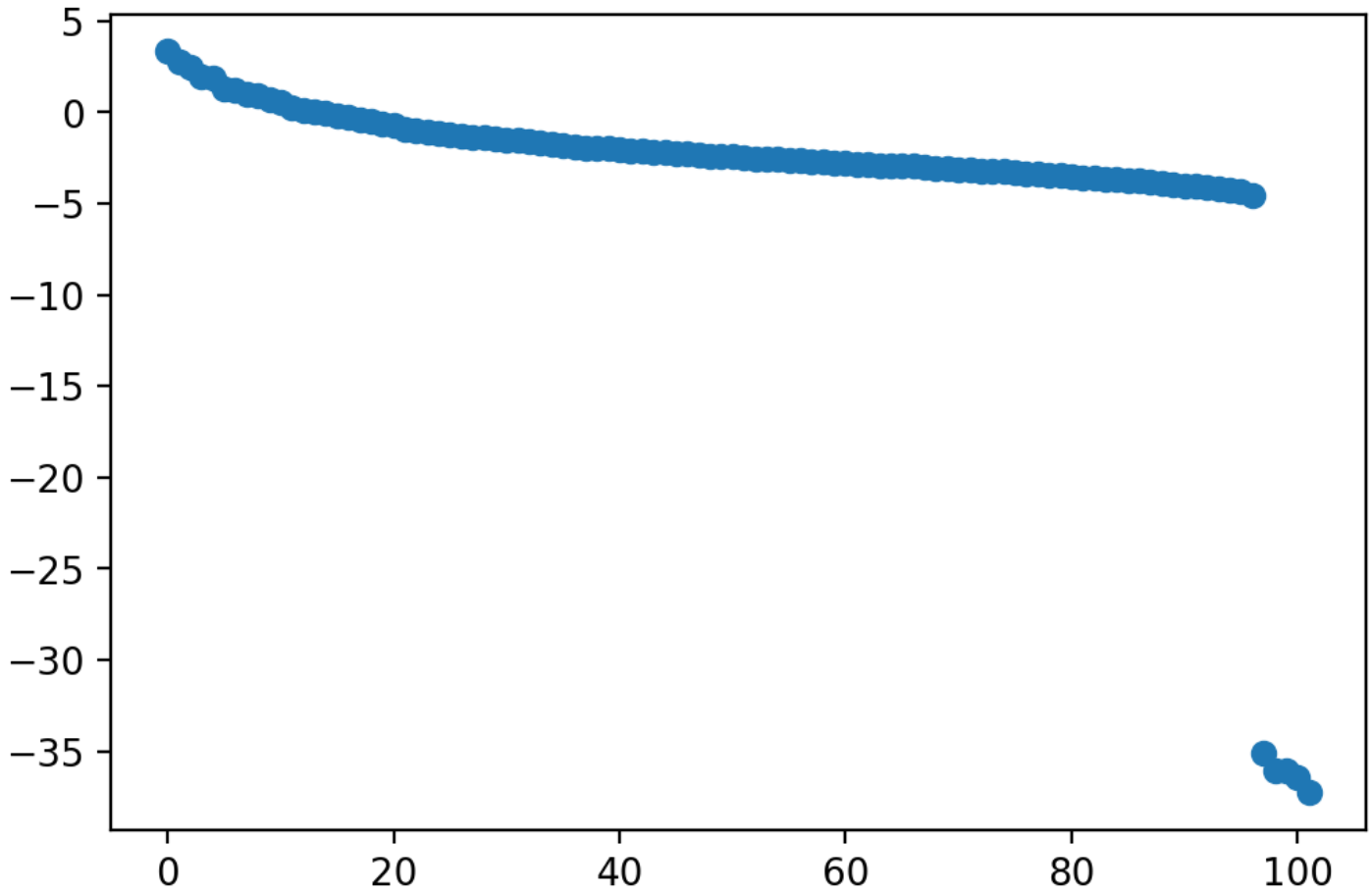
iii. Finally, create arrays of sorted eigenvalues and eigenvectors using the `sorted_indices` array just created. For the eigenvalues, it should like this `eigenvalues = eigenvalues[sorted_indices]` and for the eigenvectors: `eigenvectors = eigenvectors[:, sorted_indices]`

In [ ]:
```
eigenvalues = eigenvalues[sorted_indices]
eigenvectors = eigenvectors[:, sorted_indices]
```

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 [ ]:
```
# Plotting
plt.close("all")
plt.plot(np.log(eigenvalues), 'o')
```

Out[ ]:
```
[<matplotlib.lines.Line2D at 0x1a4dc451a60>]
```



There are five eigenvalues that are very very low, meaning that they explain very little of the variance in the data. Maybe this is why the rank is 97 - the 5 values are so small that they are below the tolerance threshold, and are therefore 0 when using the matrix_rank function.

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

In [ ]:
```
W = eigenvectors
```

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 [ ]:
```
Z = data_eq_scaled @ W
```

In [ ]:
```
np.isclose((Z @ np.transpose(W)), data_eq_scaled)
```

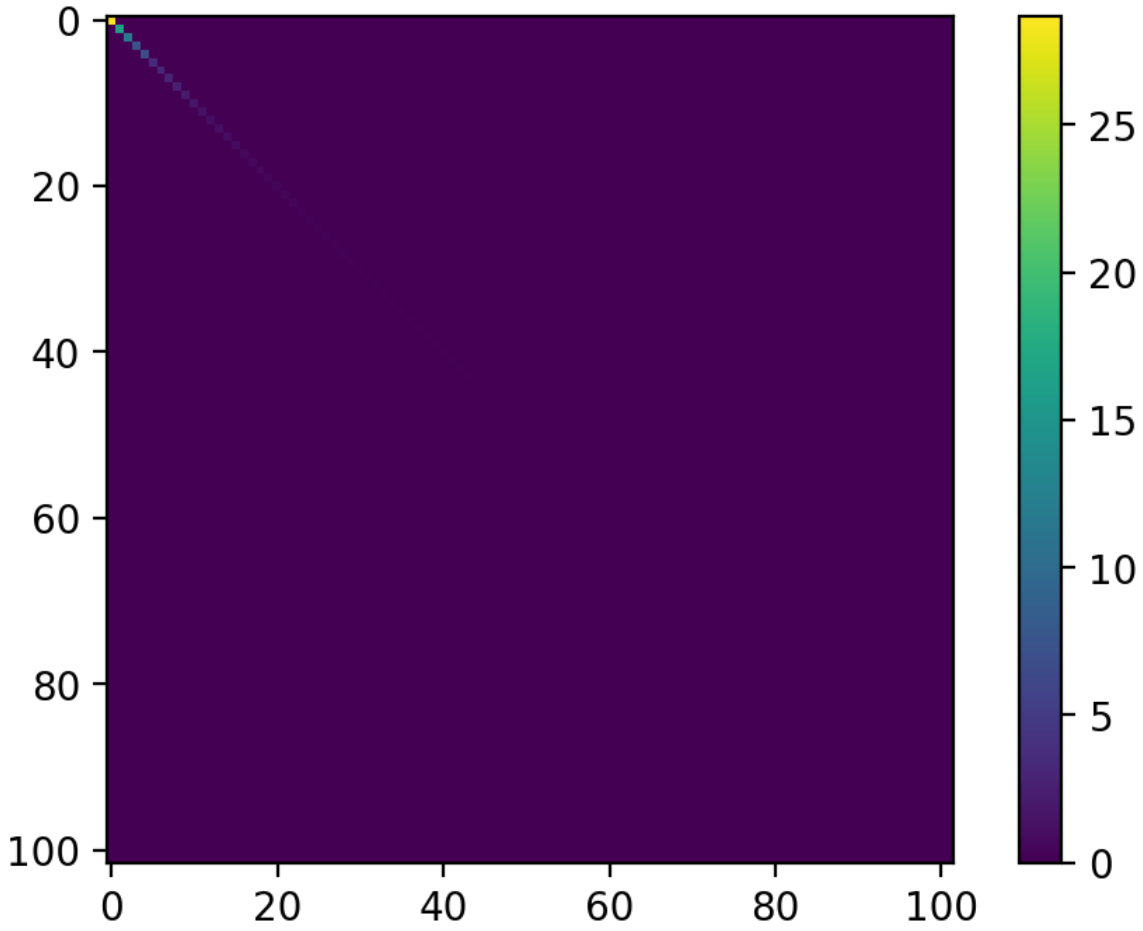
Out[ ]:
```
array([[ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],
       ...,
       [ True,  True,  True, ...,  True,  True,  True],
       [ True,  True,  True, ...,  True,  True,  True],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

It was correct since all values are equal (or close) to eachother in the two X-matrices.

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

In [ ]:
```
# Create covariance matrix
Z_covariance_matrix = np.cov(Z.T)

# Plotting
plt.imshow(Z_covariance_matrix)
plt.colorbar()
```

Out[ ]:
```
<matplotlib.colorbar.Colorbar at 0x1a4dcaa9ee0>
```



- The covariance has completely disappeared on the off-diagonal, and on the diagonal we see the new features (the prinicipal components) that we have created. Since they're sorted by value, the most explanatory principal component is 0, then 1, then 2 etc. This is exactly what we want from our PCs, reducing the dimensionality of the features so we're left with new individually explanatory features.

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

i. First, run standard logistic regression (no regularization) based on $Z_{n\times k}$ 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 $Z_{n\times k}$ and y. This is an in-sample validation. Use the solver `newton-cg` if the default solver doesn't converge

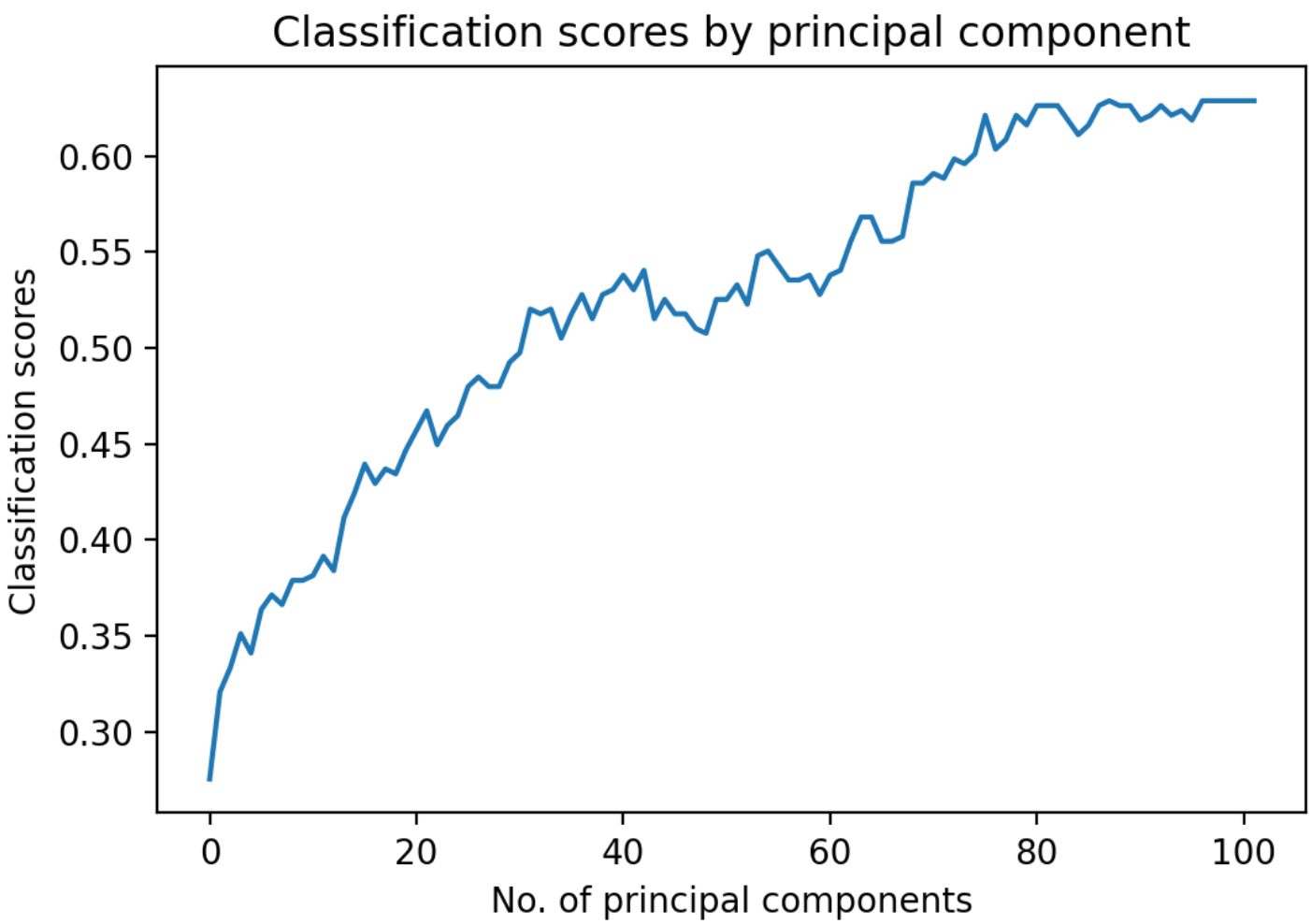
In [ ]:
```
# Fitting
from sklearn.linear_model import LogisticRegression
logR = LogisticRegression(penalty='none') # no regularisation

# Empty array
sample_scores = []

# Subsetting time
for i in range(102):
    logR.fit(Z[:,0:i+1], y_eq)
    score = logR.score(Z[:,0:i+1], y_eq)
    sample_scores.append(score)
```

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 [ ]:
```
# Plotting
plt.close("all")
plt.figure()
plt.plot(sample_scores)
plt.ylabel("Classification scores")
```

```
plt.xlabel("No. of principal components")
plt.title("Classification scores by principal component")
```

Out[ ]:
```
Text(0.5, 1.0, 'Classification scores by principal component')
```



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

- The last five components do not add any value to the classification (or explain any variance), as mentioned previously when commenting on the eigenvalues. The scores are therefore identical whether you add the terms or not.

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

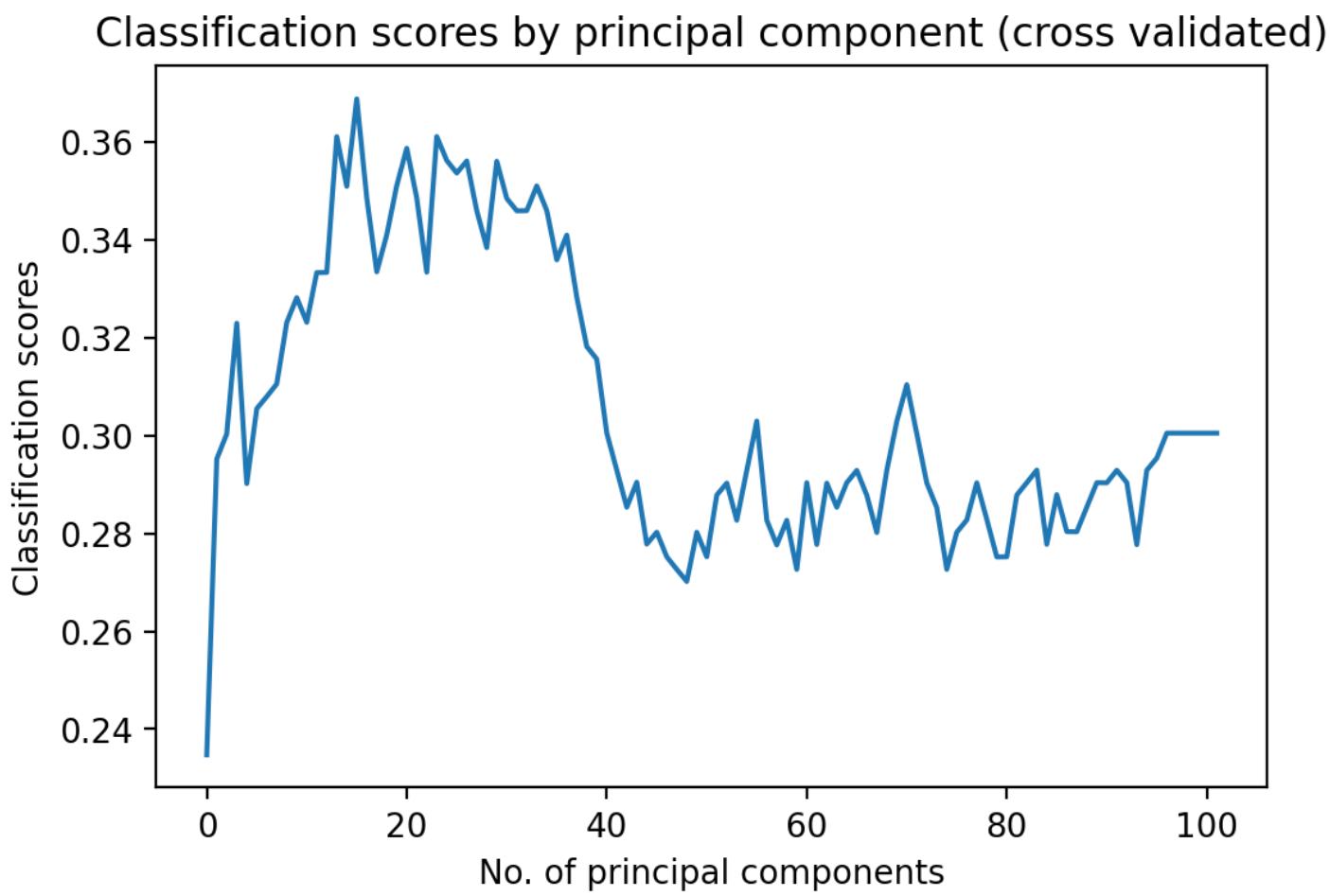
i. Define the variable: `cv = StratifiedKFold()` and run `cross_val_score` (remember to set the `cv` argument to your created `cv` variable). Use the same `estimator` in `cross_val_score` as in Exercise 2.1.i. Find the mean score over the 5 folds (the default of `StratifiedKFold`) for each k, $k = [1,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

In [ ]:
```
# Fitting
from sklearn.linear_model import LogisticRegression
logR = LogisticRegression(penalty='none') # no regularisation

# Preparing cross validation
from sklearn.model_selection import cross_val_score, StratifiedKFold
cv = StratifiedKFold()

# Empty array
sample_scores_cv = []

# Subsetting time
for i in range(102):
    logR.fit(Z[:,0:i+1], y_eq)
    scores = cross_val_score(logR, Z[:,0:i+1], y_eq, cv=5)
    sample_scores_cv.append(np.mean(scores))
```

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 [ ]:
```
# Plotting
plt.close("all")
plt.figure()
plt.plot(sample_scores_cv)
plt.ylabel("Classification scores")
plt.xlabel("No. of principal components")
plt.title("Classification scores by principal component (cross validated)")
```

Out[ ]: Text(0.5, 1.0, 'Classification scores by principal component (cross validated)')



We see that, when going outside of the sample, the model is penalized for complexity because it increaslingly models sample-specific variance (or noise) which doesn't benefit the model when using when given new data.

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

In [ ]:
```
print("The greatest classification score is", round(np.amax(sample_scores_cv), 4))
print("The number of principal components that result in the best score is", np.argmax(sample_scores_cv)+1)
```

```
The greatest classification score is 0.3688
The number of principal components that result in the best score is 16
```

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

In [ ]:
```
print("The greatest classification score is", round((sample_scores_cv[15] - sample_scores_cv[101])*100, 4), "%", "better than the full dimensi
```

```
The greatest classification score is 6.8323 % better than the full dimensional classification.
```

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.

- In the first one, which is in-sample validation, we don't really penalize overfitting. We get better classification results by modelling the noise in the data when we add PCs. So when we cross validate, we make sure that the PCs needed actually generalize and explain the signal and not just the noise.

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 [ ]:
```
# PCA with k_max_accuracy components (16)
from sklearn.decomposition import PCA
pca = PCA(n_components=16)
scores_pca_k = []

# Subsetting time
for i in range(251):
    std = sc.fit_transform(data_eq[:,:,i])
    pcs = pca.fit_transform(std)
    logR.fit(pcs, y_eq)
    scores = cross_val_score(logR, pcs, y_eq, cv=5)
    scores_pca_k.append(np.mean(scores))

# PCA with all dimensions
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=102)
scores_pca_d = []

# Subsetting time
for i in range(251):
    std = sc.fit_transform(data_eq[:,:,i])
    pcs = pca.fit_transform(std)
    logR.fit(pcs, y_eq)
    scores = cross_val_score(logR, pcs, y_eq, cv=5)
    scores_pca_d.append(np.mean(scores))
```

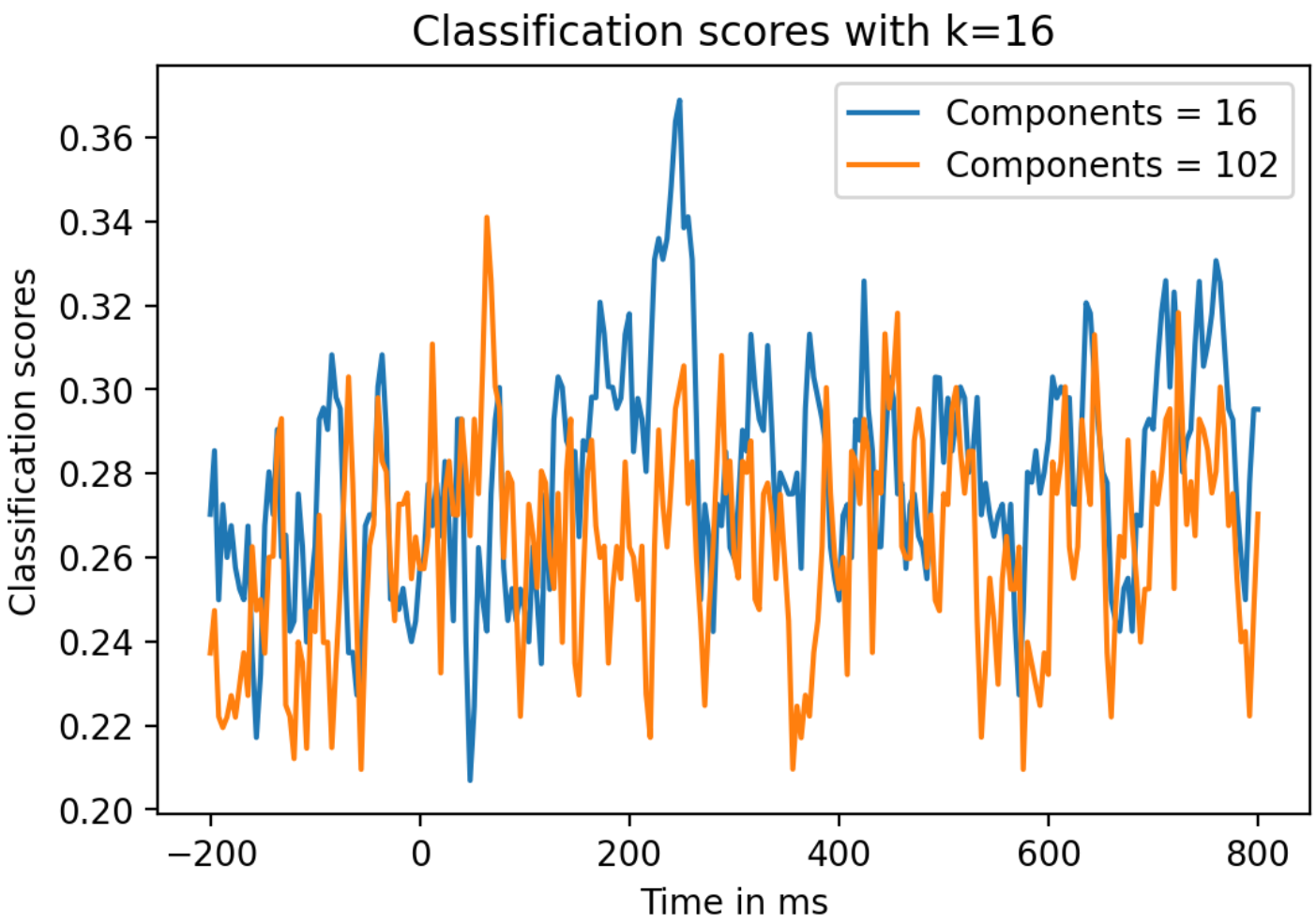
ii. Plot the classification accuracies for each time sample for the analysis with PCA and for the one without in the same plot. Have time (ms) on the *x*-axis and classification accuracy on the *y*-axis

In [ ]:

```
# Plotting
plt.close("all")
plt.figure()
plt.plot(times, scores_pca_k, label="Components = 16")
plt.plot(times, scores_pca_d, label="Components = 102")
plt.legend()
plt.ylabel("Classification scores")
plt.xlabel("Time in ms")
plt.title("Classification scores with k=16")
```

Out[ ]:

```
Text(0.5, 1.0, 'Classification scores with k=16')
```



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

- As mentioned before, using all principal components, you will likely overfit your data. This means that, even when cross validating, some of the PCs explain very little OR explain noise, which isn't favorable. So, at the point where it's easier to classify PAS (i.e. were there is more of a signal), which happens after the stimulus onset, it becomes clear that complexity is penalized heavier, and the 16-component model performs better here.