practical_exercise_5_ALEKSANDER

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
```

4.1.i. Factorise the variables that need factorising

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

```
# 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"
```

```
# 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.

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 interc
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)
```

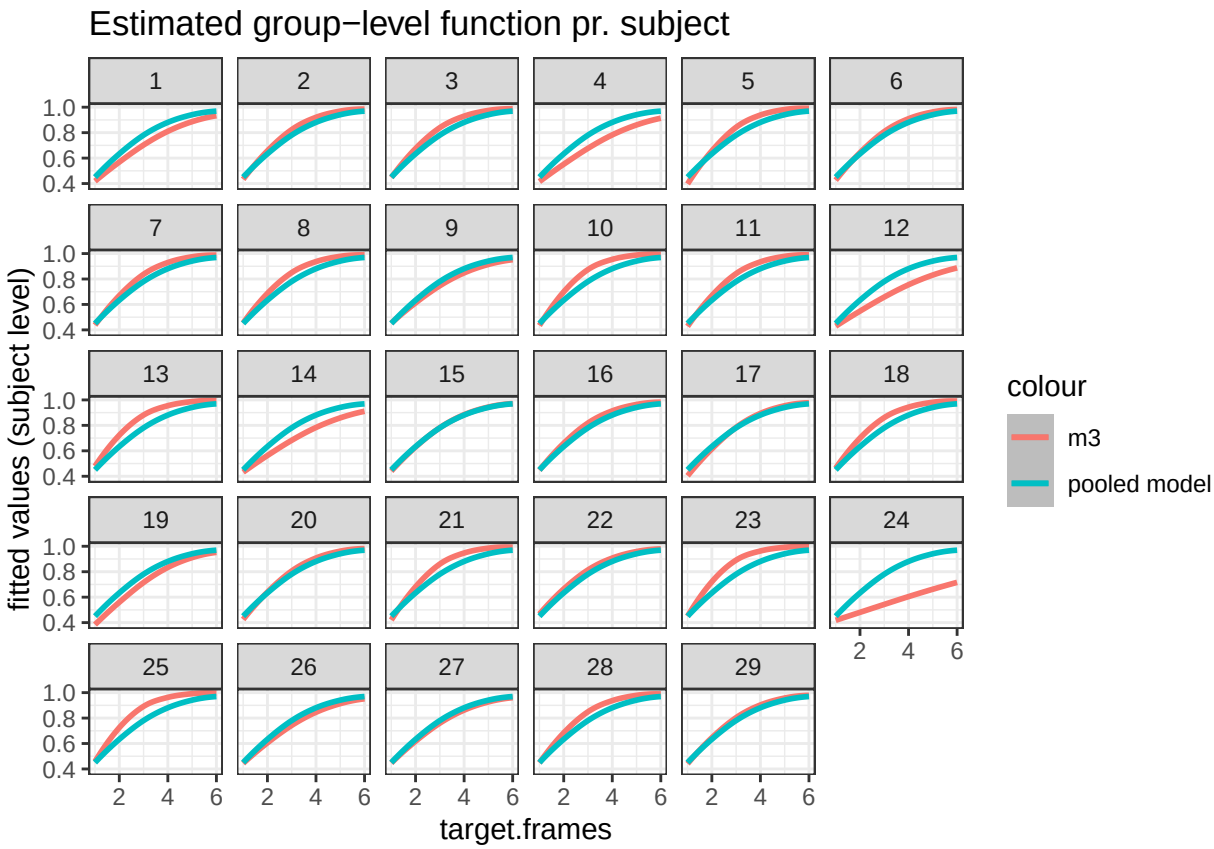
```
## Data: df
## Models:
## m1_partial: correct ~ target.frames + (1 | subject)
## m3: correct ~ target.frames + (target.frames | subject)
##            npar   AIC   BIC logLik deviance  Chisq Df Pr(>Chisq)
## m1_partial    3 21250 21274 -10622    21244
## m3            5 20908 20948 -10449    20898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

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.

```
df %>% # Plot of group-level function per subject
  ggplot() +
  geom_smooth(aes(x = target.frames, y = fitted(m3), color = "m3")) +
  geom_smooth(aes(x = target.frames, y = fitted(m1_pooled), color = "pooled model")) +
  facet_wrap( ~ subject) +
  labs(title = "Estimated group-level function pr. subject") +
  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 pretty bad fit, so i will compare it to the chance of 50%.

```
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 ~ 53% 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, cont

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
```

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

It converges with the bobyqa optimizer.

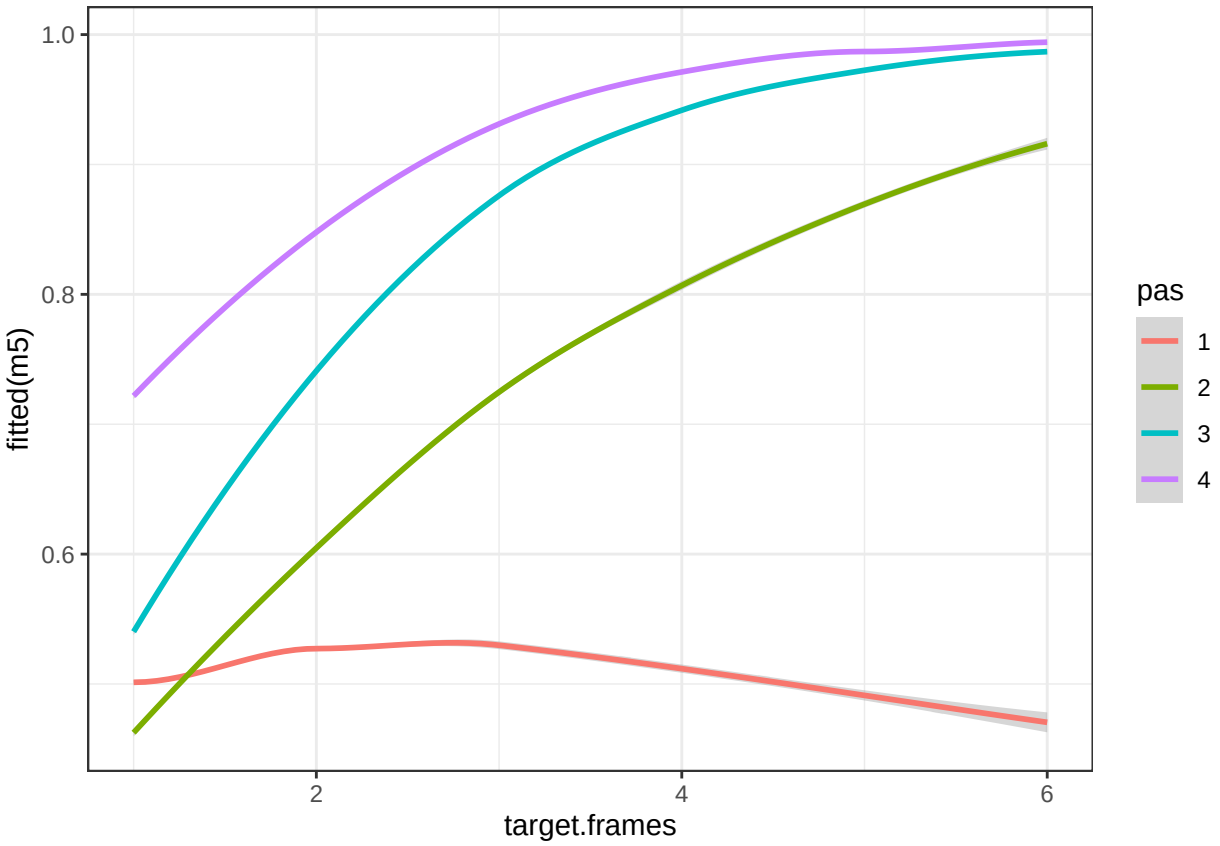
5.3.ii. plot the estimated group-level functions over xlim=c(0, 8) for each of the four PAS-ratings - add this plot to your report (see: 5.2.i) and add a description of your chosen model. Describe how pas affects accuracy together with target duration if at all. Also comment on the estimated functions'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

```
df %>%
  ggplot()+
  geom_smooth(aes(x = target.frames, y = fitted(m5), color = pas), method = "loess")+
  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 the lowest PAS score of 1, increasing target frames both increases accuracy for some subjects and reduces it for others, thus the variance in this data is large. When PAS score increases, we can also see that the distribution of fitted values is reduced, and at PAS rating 3 and 4, large values of target.frames approximates almost 100% probability of having answered correctly.

The fitted values are below 0.5 when target.frames = 0, which doesn't make much sense since chance-level accuracy is 50%. This is a limitation of the model.

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.

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))
```

```
##
##   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  0.44718    0.03473   12.88   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
## 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))
```

```
##
##   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 -0.68618    0.08566   -8.01 1.11e-15 ***
## ---
```

```
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
# testing if accuracy performance increases faster for pas3 than pas2
contrast.vector <- matrix(c(0, -1, 0, 1, 0, 0, 0, 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  -0.6533     0.1366  -4.783 1.73e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
# testing if accuracy performance increases faster for pas4 than pas3
contrast.vector <- matrix(c(0, -1, 0, 0, 1, 0, 0, 0), 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   0.0868     0.2480    0.35    0.726
## (Adjusted p values reported -- single-step method)
```

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

```r
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.i. Now, we will fit the sigmoid for the four PAS ratings for Subject 7

```r
# 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 =
optim_7_pas2 <- optim(data = filter(df_7, pas == "2"), fn = RSS, par = par, method = 'L-BFGS-B', lower =
optim_7_pas3 <- optim(data = filter(df_7, pas == "3"), fn = RSS, par = par, method = 'L-BFGS-B', lower =
optim_7_pas4 <- optim(data = filter(df_7, pas == "4"), fn = RSS, par = par, method = 'L-BFGS-B', lower =

# Showing parameter estimates pr PAS rating
print(optim_7_pas1)
```

```
## $par
## [1] 0.0000000 0.5232894 0.7940608 0.1313708
##
## $value
## [1] 45.26794
##
## $counts
## function gradient
##       52       52
##
## $convergence
```

```
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
print(optim_7_pas2)
```

```
## $par
## [1] 0.53333619 0.61112074 2.00262425 0.06410654
##
## $value
## [1] 31.75556
##
## $counts
## function gradient
##       96       96
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
print(optim_7_pas3)
```

```
## $par
## [1] 0.0000000 0.9259708 1.7354260 0.1305435
##
## $value
## [1] 6.976414
##
## $counts
## function gradient
##       48       48
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

```
print(optim_7_pas4)
```

```
## $par
## [1] 0.2269353 0.9901057 0.9676495 0.4244381
##
## $value
## [1] 5.871991
##
## $counts
## function gradient
##       42       42
```

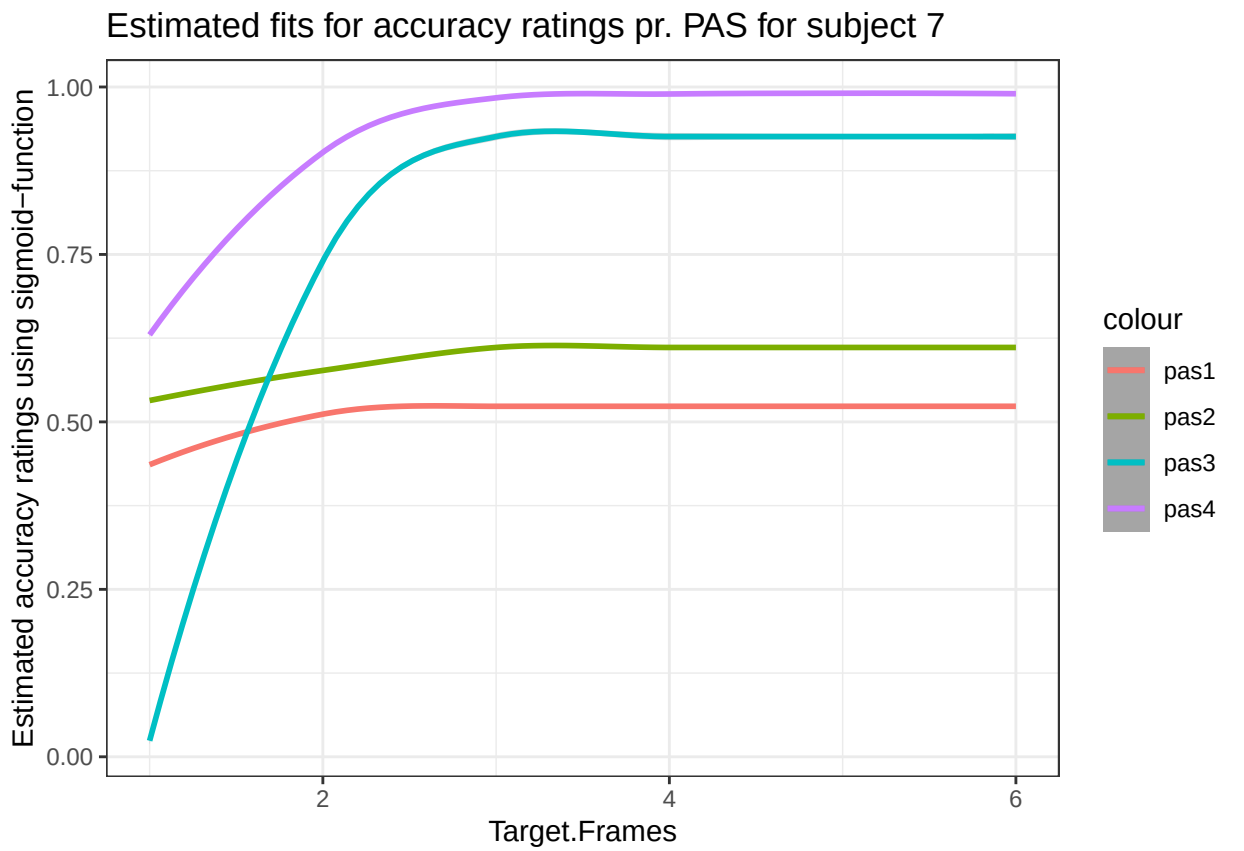
```
##
## $convergence
## [1] 0
##
## $message
## [1] "CONVERGENCE: REL_REDUCTION_OF_F <= FACTR*EPSMCH"
```

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)

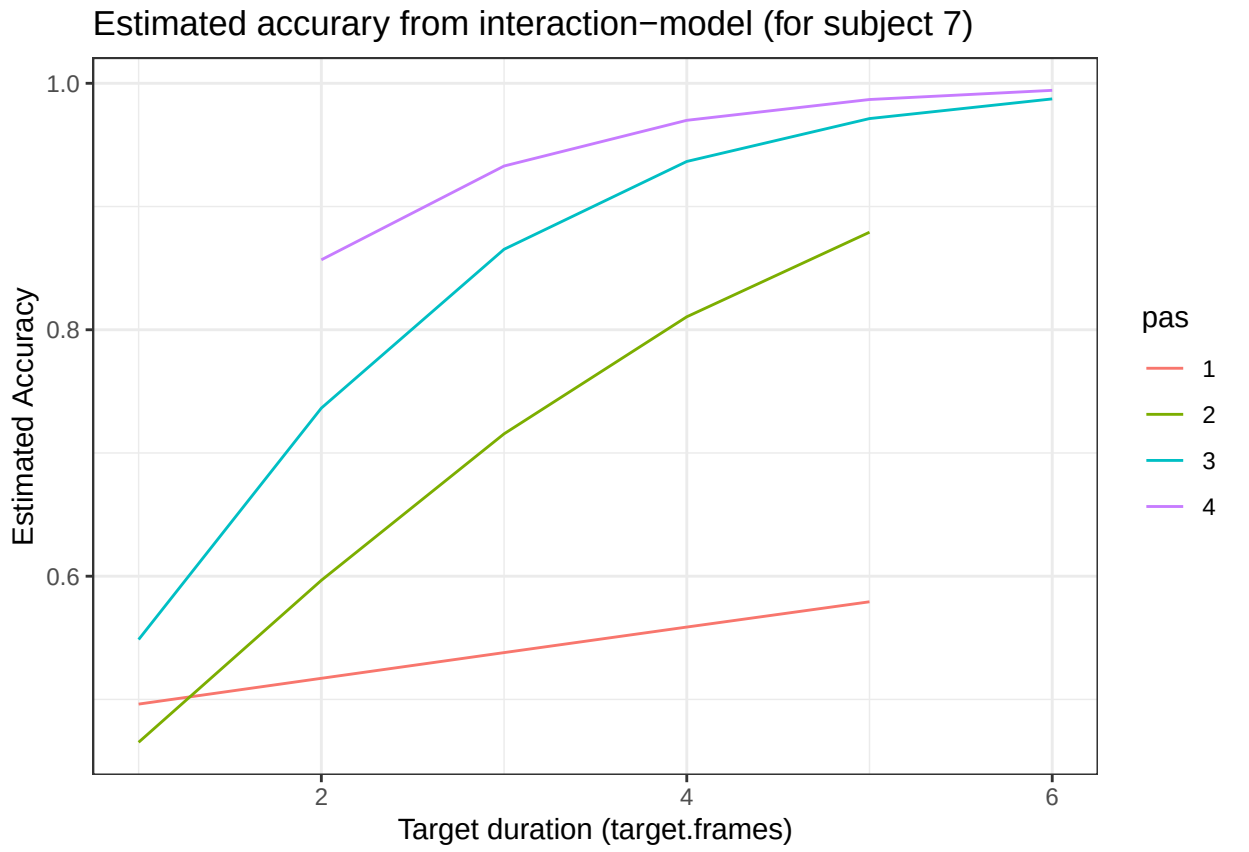
```
# Defining a sigmoid-function that takes parameters from optim-function and x-values from a df.
sigmoid_function <- function(optimfun, x) {
    optimfun$par[1] + ((optimfun$par[2]-optimfun$par[1])/(1+exp(1)^((optimfun$par[3]-x)/optimfun$par[4]))

# 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 TARGET.FRAMES = 0 AND PAS = 4)
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") +
  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. The optim-function 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)

```
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', lo
  optim_pas2 <- optim(data = filter(dataframe, pas == "2"), fn = RSS, par = par, method = 'L-BFGS-B', lo
  optim_pas3 <- optim(data = filter(dataframe, pas == "3"), fn = RSS, par = par, method = 'L-BFGS-B', lo
  optim_pas4 <- optim(data = filter(dataframe, pas == "4"), fn = RSS, par = par, method = 'L-BFGS-B', lo

  # 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]

  # 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)
```

```
## Warning: Unknown or uninitialised column: ‘a_value_pas_2‘.
```

```
## Warning in mean.default(new_df$a_value_pas_2): argument is not numeric or
## logical: returning NA
```

```
b_mean_pas_2 <- mean(new_df$b_value_pas_2)
```

```
## Warning: Unknown or uninitialised column: ‘b_value_pas_2‘.

## Warning in mean.default(new_df$b_value_pas_2): argument is not numeric or
## logical: returning NA
```

```
c_mean_pas_2 <- mean(new_df$c_value_pas_2)
```

```
## Warning: Unknown or uninitialised column: ‘c_value_pas_2‘.

## Warning in mean.default(new_df$c_value_pas_2): argument is not numeric or
## logical: returning NA
```

```
d_mean_pas_2 <- mean(new_df$d_value_pas_2)
```

```
## Warning: Unknown or uninitialised column: ‘d_value_pas_2‘.

## Warning in mean.default(new_df$d_value_pas_2): argument is not numeric or
## logical: returning NA
```

```
a_mean_pas_3 <- mean(new_df$a_value_pas_3)
```

```
## Warning: Unknown or uninitialised column: ‘a_value_pas_3‘.

## Warning in mean.default(new_df$a_value_pas_3): argument is not numeric or
## logical: returning NA
```

```
b_mean_pas_3 <- mean(new_df$b_value_pas_3)
```

```
## Warning: Unknown or uninitialised column: ‘b_value_pas_3‘.

## Warning in mean.default(new_df$b_value_pas_3): argument is not numeric or
## logical: returning NA
```

```
c_mean_pas_3 <- mean(new_df$c_value_pas_3)
```

```
## Warning: Unknown or uninitialised column: ‘c_value_pas_3‘.

## Warning in mean.default(new_df$c_value_pas_3): argument is not numeric or
## logical: returning NA
```

```
d_mean_pas_3 <- mean(new_df$d_value_pas_3)
```

```
## Warning: Unknown or uninitialised column: ‘d_value_pas_3‘.

## Warning in mean.default(new_df$d_value_pas_3): argument is not numeric or
## logical: returning NA
```

```r
a_mean_pas_4 <- mean(new_df$a_value_pas_4)
```

```
## Warning: Unknown or uninitialised column: 'a_value_pas_4'.

## Warning in mean.default(new_df$a_value_pas_4): argument is not numeric or
## logical: returning NA
```

```r
b_mean_pas_4 <- mean(new_df$b_value_pas_4)
```

```
## Warning: Unknown or uninitialised column: 'b_value_pas_4'.

## Warning in mean.default(new_df$b_value_pas_4): argument is not numeric or
## logical: returning NA
```

```r
c_mean_pas_4 <- mean(new_df$c_value_pas_4)
```

```
## Warning: Unknown or uninitialised column: 'c_value_pas_4'.

## Warning in mean.default(new_df$c_value_pas_4): argument is not numeric or
## logical: returning NA
```

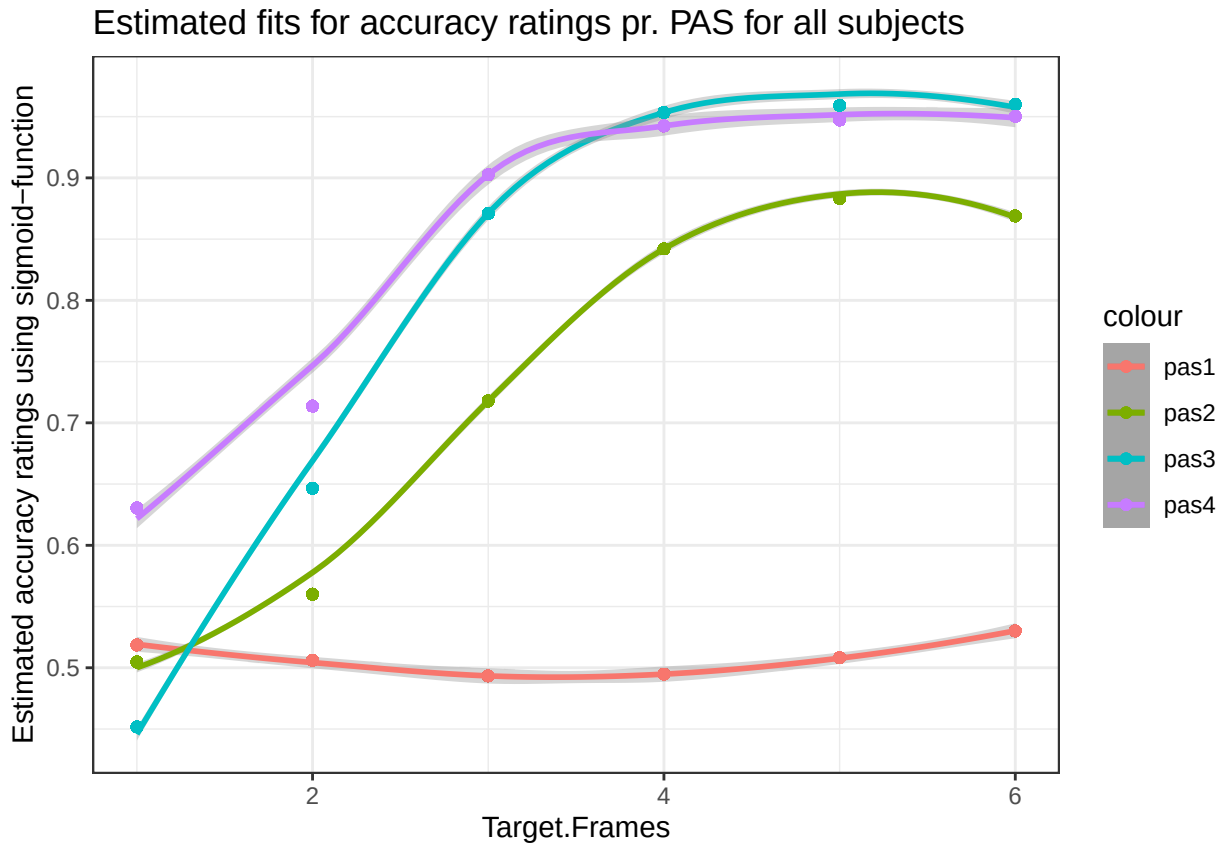
```r
d_mean_pas_4 <- mean(new_df$d_value_pas_4)
```

```
## Warning: Unknown or uninitialised column: 'd_value_pas_4'.

## Warning in mean.default(new_df$d_value_pas_4): argument is not numeric or
## logical: returning NA
```

```r
# 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()
```

```r
# Plotting
new_df %>%
  ggplot() +
  geom_smooth(aes(x = x, y = y_hat_1, color = "pas1"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_2, color = "pas2"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_3, color = "pas3"), method = "loess") +
  geom_smooth(aes(x = x, y = y_hat_4, color = "pas4"), method = "loess") +
  geom_point(aes(x = x, y = y_hat_1_mean_grand, color = "pas1"))+
  geom_point(aes(x = x, y = y_hat_2_mean_grand, color = "pas2"))+
  geom_point(aes(x = x, y = y_hat_3_mean_grand, color = "pas3"))+
  geom_point(aes(x = x, y = y_hat_4_mean_grand, color = "pas4"))+
  labs(title = "Estimated fits for accuracy ratings pr. PAS for all subjects",
       x = "Target.Frames",
       y = "Estimated accuracy ratings using sigmoid-function") +
  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. 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.