Einführung in R

Stephan Goerigk

2022-11-19

Contents

Chapter 1

Packages

You will need to install and load the following packages for these exercises:

```
# install.packages("BayesFactor")
# install.packages("MASS")
# install.packages("devtools") # if not yet installed
# library(devtools)
# install_github("nicebread/BFDA", subdir="package")
#
# library(BFDA)
# library(BayesFactor)
# library(MASS)

#Plus BFDA - see next section
```

```
# install.packages("ggplot2")
# install.packages("lme4")
# install.packages("lmerTest")
# install.packages("dendextend")
# install.packages("tidyverse")
# install.packages("lcmm")
# install.packages("LCTMtools")
# install.packages("cowplot")
# install.packages("emmeans")
# install.packages("psych")
# install.packages("BBmisc")

library(ggplot2)
library(lme4)
library(lmerTest)
library(dendextend)
```

```
library(tidyverse)
library(lcmm)
library(LCTMtools)
library(cowplot)
library(emmeans)
library(psych)
library(BBmisc)
```

Chapter 2

Trial Design

2.1 Exercise 1: Simulating data using BFDA and selecting trial parameters

The Bayes factor design analysis (BFDA) package provides a simple way to simulate and analyse sequential bayesian analyses for trial planning. For more complex analyses you can create your own custom simulations (e.g. see the scripts at https://osf.io/8mxda/ for example)

Please see https://rawgit.com/nicebread/BFDA/master/package/doc/BFDA_manual.html for the BFDA manual and full instructions

2.1.1 Simulate sequential analyses for planning

A first step is to simulate the results of sequential analyses under i) the alternative hypothesis for the target effect size of interest (to find parameters that give you sufficient power), and ii) the null hypothesis (to find parameters that give a suitable false-positive (Type 1) error rate). Once you have suitable parameters for these, you can then simulate a range of other effect sizes to get a better picture of your power at different effect sizes.

As simulations can take some time, for demonstration purposes in this exercise we will i) attempt to find a medium effect size (d = 0.5), as this requires smaller maximum sample sizes, and ii) only do a small number of simulations (500), as we are not too worried about accuracy/precision - for planning an actual trial you would want to do a larger number (e.g. 10,000)

Please run the following simulations as preparation:

```
set.seed(19112022)

sim.H1 <- BFDA.sim(expected.ES=0.5, type="t.between",prior=list("Cauchy",list(prior.lo

sim.H0 <- BFDA.sim(expected.ES=0, type="t.between", prior=list("Cauchy", list(prior.lo

#if you notice these have finished and you have time you could also run some in-betwee

sim.H0.2 <- BFDA.sim(expected.ES=0.2, type="t.between", prior=list("Cauchy", list(prio

sim.H0.4 <- BFDA.sim(expected.ES=0.4, type="t.between", prior=list("Cauchy", list(prio

sim.H0.6 <- BFDA.sim(expected.ES=0.6, type="t.between", prior=list("Cauchy", list(prio

#etc...
```

As an explanation, these simulate sequential Bayes-factor based analyses based on:

a) Effect size of d = 0.5 (expected.ES=0.5) for sim.H1, or d=0 for sim.H0

b) a between-group t-test, e.g. difference between two groups in change in symptoms (type="t.between")

c) use a default Cauchy prior with a rscale value of 0.707 (prior=list("Cauchy", list(prior.location=0, prior.scale=sqrt(2)/2))) [we can't go into priors today but default ones like this work fine! See Stefan et al. (2019) for discussion of using informed priors with BFDA, https://link.springer.com/article/10.3758/s13428-018-01189-8]

d) a minimum sample size (per arm) of 10 (n.min = 10) - note that when trying to find parameters later you can choose higher minimum sample sizes, but not lower

e) a maximum sample size (per arm) of 80 (n.max = 80) - note that when trying out parameters later on you can try lower, but not higher maximum sample sizes

f) a directional (i.e. one-tailed) Bayes Factor (alternative = "greater" and boundary = Inf). We are interested in whether one treatment is superior to another, not whether it is different.

g) we use 500 simulations (B = 500) - fine for getting a feel, but for finalising your study design you would want a larger number e.g. 10,000

h) we carry out the analysis every 5 participants (stepsize = 5). You can repeat the analysis every 1 participant, but this simply takes longer to simulate so we don't do this here.

If you were carrying out large important simulations that took a long time you

would probably want to save them so that you can load them again later without having to re-run the simulations:

```
#Saving a BFDA simulation object:
saveRDS(sim.H1,"sim.H1.d0.5.dd.mm.yy.RDS")
saveRDS(sim.H0,"sim.H1.d0.0.dd.mm.yy.RDS")
#to load again
sim.H1<-readRDS("sim.H1.d0.5.dd.mm.yy.RDS")
sim.H0<-readRDS("sim.H1.d0.0.dd.mm.yy.RDS")
```

2.1.2 Try out different analysis parameters to find a set that fits your requirements

2.1.2.1 Overview

Now that you have a simulated set of sequential BFs you can test out what would happen if you applied different sets of analysis parameters to them.

These are:

- Nmin: The minimum sample size (per arm) at which you start the sequential analyses
- Nmax: The maximum sample size (per arm) at which you drop an arm (if it has not already hit a BF boundary)
- BFfail: A BF threshold for failure (i.e. sufficient evidence for the null hypothesis of non-superiority to the control condition vs. the alternative hypothesis of superiority). This will be a value less than 1 (e.g. 1/3, 1/5, 1/10 etc)
- BFsuccess: A BF threshold for success (i.e. sufficient evidence for the alternative hypothesis of superiority over the control condition vs.the null hypothesis of non-superiority to the control condition). This will be a value greater than 1 (e.g. 3, 5, 10)

We are interested in:

i) power: what proportion of arms hit the BFsuccess threshold when d $>$ 0

ii) false-positive / Type 1 error rate: what proportion of arms hit BF success threshold when d = 0 (or d $<$ 0)

iii) keeping the average sample sizes as low as possible

There are several functions in BFDA to analyse simulation outcomes, but here we will just use the plot function. We will start by using the boundary conditions of the simulations (i.e. n.min = 10 and n.max = 80), and a default set of starting boundaries of BFfail = 1/5 and BFsuccess = 5 (boundary=c(1/5, 5)).

The parameter n.trajectories just tells it how many lines to draw on the graph (representing individual BF trajectories)

```
#First plot for H1
dev.new() #sometimes the plot doesn't work if you don't make a new window first
plot(sim.H1, n.min=10, n.max=80, boundary=c(1/5, 5), n.trajectories = 60)

#Then plot for H0
dev.new() #sometimes the plot doesn't work if you don't make a new window first
plot(sim.H0, n.min=10, n.max=80, boundary=c(1/5, 5), n.trajectories = 60)

#There are other functions in BFDA to explore the simulations/sequential analyses but
```

2.1.2.2 Exercise steps

1. Play around with the parameters (n.min, n.max, the BF boundaries) and see what happens. e.g. increasing n.min will tend to reduce error rates, but means you lose the chance to make decisions so quickly. Remember that the BF boundaries don't have to be symmetrical (e.g. you could use boundary=c(1/3,10) if you weren't so worried about false-negatives but were very concerned about potential false-positives)
2. Try to come up with a set of parameters that gives you 80% power and a false-positive (type 1) error rate of <5%, i.e. for H1, $\geq$ 80% stopping at H1 boundary, for H0, $<$ 5% (i.e. 4%) stopping at h1 boundary.
3. Once you find this you can see if you can improve on these, to reduce the potential sample sizes needed.
4. See what proportion of participants will hit BFsuccess (H1 boundary) for sim.H1 at different sample sizes (by adjusting n.max): the value for which 50% are stopping at the nmax boundary tells you the average sample size you might expect if d=0.5 with these parameters. You can then do the same for sim.H0. It might be that you can find a set of parameters that give you smaller average sample size predictions.
5. If you have simulated other effect sizes you can see what happens for these, e.g. what if you power for d=0.5, but d=0.4?

2.1.2.3 Summarising the simulations for planning

One way to collect this information is into a table, as provided in the examples in Table 2 and Table 3 in the paper by Blackwell et al. (2019) https://journals.sagepub.com/doi/full/10.1177/2167702619858071

Table 2 from the paper is reproduced below. This illustrates a particular set of parameters chosen for a small to medium between-group effect size equivalent

to Cohen's d = 0.4: Nmin = 35 (per arm), Nmax = 125 (per arm), BFfail = 1/4, and a BFsuccess = 5 (and directional default Cauchy prior, rscale parameter = sqrt2/2).

Here we can see that we have a false-positive rate of < 5% (top row), and 81% to find d = 0.4. We can also see that 54% of the time, we would hit BFfail at Nmin when d = 0, and 54% of the time we would stop the trial at n = 50 per arm when d = 0.4 (8% at BFfail, 46% at BFsuccess):

"True" effect size (Cohen's d)

Probability of reaching threshold at each participant number (per group)

Discontinuation threshold

Replacement threshold

n = 35

n = 50

n = 75

n = 100

n = 125

n = 35

n = 50

n = 75

n = 100

n = 125

0 (null)

54

70

81

86

89

1

3

3

4

4

0.1
37
52
62
68
71
4
7
10
12
13
0.2
22
33
41
45
47
8
15
23
28
32
0.3
11
18
22
24
25
16
29
43

52
58
0.4
5
8
10
10
11
28
46
65
75
81
0.5
2
3
4
4
4
43
64
82
91
94
0.6
1
1
1
1
1
60

80

94

97

98

0.7

0

0

0

0

0

75

91

98

100

100

0.8

0

0

0

0

0

88

97

100

100

100

Doing these plots is quite a long-winded way to arrive at such a table, but it at least gives you a good feel for what happens when you change the parameters. If you were doing your own simulations you could write a script to output a table automatically (e.g. see the scripts at https://osf.io/8mxda/ for example)

2.2 Exercise 2: Carry out sequential analysis of (simulated) trial data

2.2.1 Overview and setup

In this exercise you will analyse some (simulated) data using the trial parameters you selected, and find out when you hit the BF boundaries.

First you will need to simulate the trial data - please run the code below (which I'm happy to explain but there's no need to understand it for the exercise - it's just a means to provide you with the data to analyse):

```
#Simulates 100 participants in each arm of trial where:
#There is a correlation of r = 0.35 between pre and post-data
#One arm is not superior to control, the other is
set.seed(19112022)
samples=100
r=0.35
data<-mvrnorm(n=samples,mu=c(1,0.8),Sigma=matrix(c(1,r,r,1),nrow=2),empirical=FALSE)
X = data[,1]
Y = data[,2]
X<-as.integer(X*5+20)
Y<-as.integer(Y*5+20)
data<-mvrnorm(n=samples,mu=c(1,0.15),Sigma=matrix(c(1,r,r,1),nrow=2),empirical=FALSE)
X1 = data[,1]
Y1 = data[,2]
X1<-as.integer(X1*5+20)
Y1<-as.integer(Y1*5+20)
data<-mvrnorm(n=samples,mu=c(1,0.9),Sigma=matrix(c(1,r,r,1),nrow=2),empirical=FALSE)
X2 = data[,1]
Y2 = data[,2]
X2<-as.integer(X2*5+20)
Y2<-as.integer(Y2*5+20)
cdiff<-Y-X
T1diff<-Y1-X1
T2diff<-Y2-X2
predata<-c(X,X1,X2)
postdata<-c(Y,Y1,Y2)
diffdata<-postdata-predata
group<-c(rep("C",samples),rep("Tx1",samples),rep("Tx2",samples))
dseq<-seq(1,samples*3)
pid<-as.character(seq(1,samples*3))
mydata<-(cbind(predata,postdata,diffdata,group))
mydata<-mydata[sample(1:nrow(mydata)), ]
mydata<-data.frame(cbind(pid,dseq,mydata))
```

```
mydata$dseq<-as.integer(mydata$dseq)
mydata$predata<-as.integer(mydata$predata)
mydata$postdata<-as.integer(mydata$postdata)
mydata$diffdata<-as.integer(mydata$diffdata)
```

You now have a dataframe (mydata) with pre and post outcome data for a pretend 3-arm trial of treatments for depression (i.e. a decrease in score on the outcome measure is good). There are three treatment arms, "C" (control condition, e.g. TA), "Tx1" (New treatment 1) and "Tx2" (New treatment 2). One arm is superior to control, one isn't. Each arm includes 100 participants.

Dataframe columns are:

- pid = participant id (a string of a number from 1 to 300)
- dseq = sequence in which they provided outcome data for the trial (numerical, here the same as pid for convenience)
- predata = score on the depression outcome measure at pre-treatment
- postdata = score on the depression outcome measure at post-treatment
- diffdata = post-treatment score minus pre-treatment score
- group = which group (C, Tx1, Tx2)

2.2.2 Exploring the data

Take a look at the data (e.g. with View(mydata)) to get a sense of it. You can then use the functions in the code chunk below to explore the potential outcome of applying sequential Bayesian analyses.

First, run the code below to load the two functions:

```
BFsnapshot<-function(BFdata,n=300,rs=(sqrt(2)/2)){

  if (length(BFdata$pid)<n){
    return("Error: n larger than number of participants");
  }

  tryCatch(
        {
          tdata<-BFdata[1:n,]
          cat("\n\n Total N = ",n,"\n\n")
          cat("Control: n = ",length(tdata[tdata$group=="C",]$pid),", mean change (post
        cat("Tx1: n = ",length(tdata[tdata$group=="Tx1",]$pid),", mean change (post m
        #effect size calculation
        d<-effectsize::cohens_d(tdata[tdata$group=="C",]$diffdata,tdata[tdata$group==
        cat("Effect size vs. control: d=",d$Cohens_d," 95% CIs [",d$CI_low,",",d$CI_hi
        #Calculates directional Bayesian t-test (nullinterval=c(0,Inf))
        BF<-BayesFactor::ttestBF(x=tdata[tdata$group=="C",]$diffdata,y=tdata[tdata$gro
```

```
        cat("BF vs. control: BF=",exp(BF@bayesFactor$bf),"\n\n",sep="")

        cat("Tx2: n = ",length(tdata[tdata$group=="Tx2",]$pid),", mean change (post minus pre-tre
        #effect size calculation
         d<-effectsize::cohens_d(tdata[tdata$group=="C",]$diffdata,tdata[tdata$group=="Tx2",]$dif
         cat("Effect size vs. control: d=",d$Cohens_d," 95% CIs [",d$CI_low,",",d$CI_high,"]\n\n"
         #Calculates directional Bayesian t-test (nullinterval=c(0,Inf))
         BF<-BayesFactor::ttestBF(x=tdata[tdata$group=="C",]$diffdata,y=tdata[tdata$group=="Tx2",
        cat("BF vs. control: BF=",exp(BF@bayesFactor$bf),"\n\n",sep="")

        },
        error=function(cond) {
          return (paste0("Something went wrong! ",cond))
        })
}

seqBFs<-function(BFdata,rs=(sqrt(2)/2)){
  tryCatch(
        {
          alldata<-BFdata[1:length(BFdata$pid),]
          dsTx1<-vector()
          dsTx2<-vector()
          BFsTx1<-vector()
          BFsTx2<-vector()
          nsTx1<-vector()
          nsTx2<-vector()
          is<-vector()
          #start at index number 10 to avoid problems
          for (i in 10:length(alldata$pid)){
          tdata<-alldata[1:i,]
          Ns<-as.data.frame(cbind(table(tdata[1:i,]$group)))
          dsTx1<-c(dsTx1,(effectsize::cohens_d(tdata[tdata$group=="C",]$diffdata,tdata[tdata$grou
          BFsTx1<-c(BFsTx1,exp((BayesFactor::ttestBF(x=tdata[tdata$group=="C",]$diffdata,y=tdata[
            nsTx1<-c(nsTx1,Ns["Tx1",])

            dsTx2<-c(dsTx2,(effectsize::cohens_d(tdata[tdata$group=="C",]$diffdata,tdata[tdata$gr
            BFsTx2<-c(BFsTx2,exp((BayesFactor::ttestBF(x=tdata[tdata$group=="C",]$diffdata,y=tdat
            nsTx2<-c(nsTx2,Ns["Tx2",])
            is<-c(is,i)

        }
          allresults<-data.frame(cbind(is,nsTx1,dsTx1,BFsTx1,nsTx2,dsTx2,BFsTx2))
          return(allresults)
        },
        error=function(cond) {
```

```
          return (paste0("Something went wrong! ",cond))
        })
}
```

You can now use these two functions to explore the effect of doing sequential Bayesian analyses (here for simplicity, t-test on change scores).

2.2.2.1 Function: BFsnapshot()

BFsnapshot() can be used to give you a snapshot of the data at a particular point in time. Arguments are:

- BFdata = the dataframe with the data (i.e. here it is mydata)
- n = the total sample size / sequence number (up to a max of 300) you want to take the snapshot at (e.g. if you put 100 in here, it will give you a snapshot of the analysis outcomes at the point there were 100 total participants in the trial)
- rs = you can ignore unless you want to try change the rscale parameter (sets to sqrt2/2 as default)

e.g. BFsnapshot(mydata,100) would give you analysis output when there are 100 people in the trial

2.2.2.2 Function: seqBFs()

seqBFs() outputs a dataframe with sequential BFs over the course of the simulated trial. You just pass it the dataframe (and can specify a rscale parameter if you want to change this)

trialresults<-seqBFs(mydata)

will give you a dataframe (trialresults), which you can then investigate (e.g. via View(trialresults), or you can try making some plots) to see how the BFs develop over time.

Columns in the dataframe are:

- is = index / total N participants in the trial
- nsTx1 = n in Tx1 arm
- dsTx1 = effect size (cohen's d) for Tx1 vs control
- BFsTx1 = Bayes Factor for Tx1 vs control
- nsTx2 = n in Tx2 arm
- dsTx2 = effect size (cohen's d) for Tx2 vs control
- BFsTx2 = Bayes Factor for Tx2 vs control

2.2.2.3 Exercise task:

See if you can work out what would have happened if you used your chosen trial parameters on this data set. When would you hit a BF boundary for each treatment arm (if at all)?

Chapter 3

Re-analysis of RCT data

3.1 Exercise 1: Symptom Clusters

3.1.1 Simulate Practice Data

First let us simulate some practice data. The data are based on the items of the Hamilton-Anxiety-Rating Scale.

Initially we determine a sample size:

```
n = 350 # number of individuals
```

Then the names of the items:

```
hama_names = c("Anxious Mood",
               "Tension",
               "Fears",
               "Insomnia",
               "Concentration and Memory",
               "Depressed Mood",
               "General somatic symptoms: muscular",
               "General somatic symptoms: sensory",
               "Cardiovascular symptoms",
               "Respiratory symptoms",
               "Gastro-intestinal symptoms",
               "Genito-urinary symptoms",
               "Other autonomic symptoms")
```

And finally mean values and standard deviations as well as a plausible covariance structure:

```
hama_means = c(2.9669421,
               2.8044077,
               2.4559229,
               2.4297521,
               1.3815427,
               1.3071625,
               1.6129477,
               1.5633609,
               1.4531680,
               1.0330579,
               1.7190083,
               0.6694215,
               1.6198347)

hama_means_post = c(2.9669421-2.3,
                    2.8044077-2.4,
                    2.4559229-2.2,
                    2.4297521-0.8,
                    1.3815427-0.7,
                    1.3071625-1.2,
                    1.6129477-0.5,
                    1.5633609-0.4,
                    1.4531680- 0.2,
                    1.0330579-0.4,
                    1.7190083-0.1,
                    0.6694215-0.1,
                    1.6198347-0.8)

hama_sds = c(0.7136179,
             0.8339568,
             1.1314254,
             1.2283532,
             1.2313786,
             1.2454069,
             1.2715661,
             1.2264810,
             1.2583008,
             1.2200659,
             1.2815280,
             1.0691029,
             1.1585213)

hama_cor = read.csv("https://raw.githubusercontent.com/stephangoerigk/DZP_Workshop_Slid
hama_cor = as.matrix(hama_cor[,-1])
```

Now, let us simulate:

```
# Create baseline data
set.seed(123)
data_cluster_bl = round(
  faux::rnorm_multi(n = n,
                    mu = hama_means,
                    sd = hama_sds,
                    r = hama_cor,
                    varnames = hama_names,
                    empirical = F), 2)
data_cluster_bl$id = row.names(data_cluster_bl)
data_cluster_bl$time = 0

# Create post-treatment data

data_cluster_post = round(
  faux::rnorm_multi(n = n,
                    mu = hama_means_post,
                    sd = hama_sds,
                    r = hama_cor,
                    varnames = hama_names,
                    empirical = F), 2)
data_cluster_post$id = row.names(data_cluster_post)
data_cluster_post$time = 1
```

Let us briefly look at the data:

```
psych::describe(data_cluster_bl)
```

```
##                                    vars   n   mean     sd median trimmed    mad
## Anxious Mood                          1 350   2.97   0.71   2.98    2.98   0.67
## Tension                               2 350   2.80   0.83   2.84    2.80   0.85
## Fears                                 3 350   2.43   1.08   2.45    2.45   1.07
## Insomnia                              4 350   2.38   1.21   2.26    2.36   1.20
## Concentration and Memory              5 350   1.43   1.19   1.41    1.44   1.10
## Depressed Mood                        6 350   1.35   1.23   1.40    1.35   1.19
## General somatic symptoms: muscular    7 350   1.68   1.28   1.69    1.70   1.32
## General somatic symptoms: sensory     8 350   1.65   1.15   1.56    1.64   1.14
## Cardiovascular symptoms               9 350   1.38   1.34   1.37    1.37   1.28
## Respiratory symptoms                 10 350   0.98   1.23   0.96    0.98   1.26
## Gastro-intestinal symptoms           11 350   1.69   1.31   1.63    1.65   1.14
## Genito-urinary symptoms              12 350   0.68   1.04   0.64    0.69   1.00
## Other autonomic symptoms             13 350   1.62   1.14   1.63    1.65   1.16
## id*                                  14 350 175.50 101.18 175.50  175.50 129.73
## time                                 15 350   0.00   0.00   0.00    0.00   0.00
##                                     min    max  range  skew kurtosis   se
```

```
## Anxious Mood                          0.62   5.42   4.80 -0.07     0.17 0.04
## Tension                               0.23   5.25   5.02 -0.05     0.03 0.04
## Fears                                -1.38   5.27   6.65 -0.21     0.10 0.06
## Insomnia                             -1.11   6.04   7.15  0.17    -0.03 0.06
## Concentration and Memory             -1.92   4.96   6.88 -0.07     0.12 0.06
## Depressed Mood                       -1.96   5.61   7.57  0.03     0.01 0.07
## General somatic symptoms: muscular -2.39   5.33   7.72 -0.12    -0.32 0.07
## General somatic symptoms: sensory  -1.46   4.59   6.05  0.11    -0.35 0.06
## Cardiovascular symptoms              -2.41   5.23   7.64  0.07    -0.05 0.07
## Respiratory symptoms                 -2.72   4.44   7.16 -0.07    -0.29 0.07
## Gastro-intestinal symptoms           -2.26   6.53   8.79  0.22     0.49 0.07
## Genito-urinary symptoms              -2.13   3.86   5.99  0.01    -0.06 0.06
## Other autonomic symptoms             -2.86   4.55   7.41 -0.29     0.28 0.06
## id*                                   1.00 350.00 349.00  0.00    -1.21 5.41
## time                                  0.00   0.00   0.00   NaN      NaN 0.00
```

Usually we would go on and create a sum score by adding up all the items:

```
data_cluster_bl$sum = rowSums(data_cluster_bl[, 1:13])
data_cluster_post$sum = rowSums(data_cluster_post[, 1:13])
```

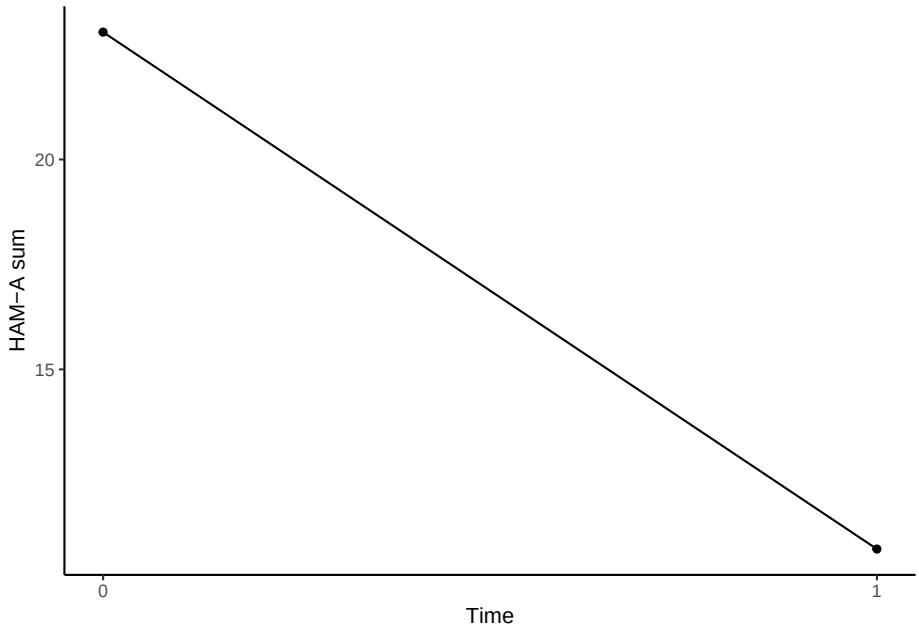
Now let us combine the datasets:

```
data_cluster = rbind(data_cluster_bl, data_cluster_post)
```

3.1.2 Traditional Approach

Plot the data as we usually would using an average group trajectory:

```
ggplot(data = data_cluster, aes(x = time, y = sum)) +
  stat_summary(geom = "line", fun = "mean") +
   stat_summary(geom = "point", fun = "mean") +
  scale_x_continuous(breaks = c(0,1)) +
  labs(y = "HAM-A sum", x = "Time") +
  theme_classic()
```



Analyze using LMM:

```
summary(lmer(sum ~ time + (1|id), data = data_cluster))
```

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

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: sum ~ time + (1 | id)
##    Data: data_cluster
##
## REML criterion at convergence: 4275.9
##
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -3.4157 -0.6507 -0.0258  0.6965  3.2589
##
## Random effects:
##  Groups   Name        Variance  Std.Dev.
##  id       (Intercept) 5.860e-16 2.421e-08
##  Residual             2.635e+01 5.133e+00
## Number of obs: 700, groups:  id, 350
##
## Fixed effects:
##             Estimate Std. Error       df t value Pr(>|t|)
## (Intercept)  23.0323     0.2744 698.0000   83.95   <2e-16 ***
## time        -12.3096     0.3880 698.0000  -31.73   <2e-16 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##      (Intr)
## time -0.707
## optimizer (nloptwrap) convergence code: 0 (OK)
## boundary (singular) fit: see ?isSingular
```

3.1.3 Prepare for Clustering

Let us see if we can find some item clusters in the baseline data to get a more differentiated picture:

For this we first need to drop the id, sum, and time columns:

```
data_cluster_bl = BBmisc::dropNamed(data_cluster_bl, drop = c("time", "id", "sum"))
data_cluster_post = BBmisc::dropNamed(data_cluster_post, drop = c("time", "id", "sum"))
```

As a first step, it makes sense to scale the data, as not all item formats are identical in all scales:

```
data_cluster_bl_s = scale(data_cluster_bl)
```

Next, we will transpose the data, since we want to cluster items into people and not vice versa:

```
data_transposed = t(na.omit(data_cluster_bl_s))
```

Now we will creat a distance matrix to determine the proximity between item responses. The euclidean distance is a commonly used measure for psychometric measures (another one is the manhattan distance).

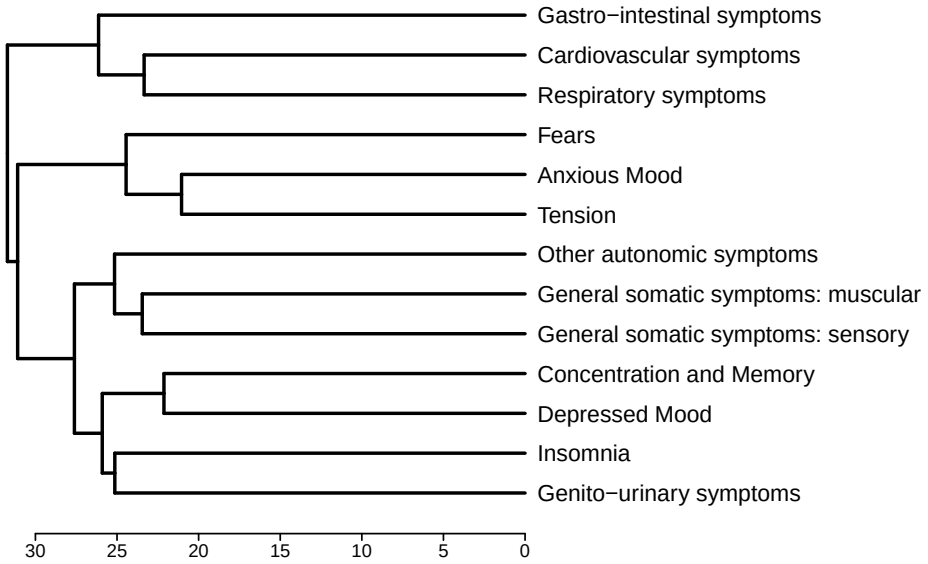
```
d = dist(data_transposed, method = "euclidean")
```

And now, let us cluster the data. We use the `ward.D2` method, this way distances are squared ahead of clustering (no problem with negative data):

```
clust = hclust(d, method = "ward.D2")
```

After the clustering is finished we should inspect the result. A common way to look at clustering solutions is the dendrogram:

```
dend <- as.dendrogram(clust, hang = -1)

labels_cex(dend) = 2
marg = c(4, 4, 10, 35)
par(mar = marg, font = 1, cex = 0.4, cex.axis = 1.7, cex.lab = 2)
plot(rev(dend), horiz = T, edgePar = list(lwd = 2))
```



The earlier two items merge in the dendrogram, the more similar they were scored by the patients. We now have a good idea, which items belong together. However, now we need to decide how many clusters to retain, i.e. where to "cut" our dendrogram. This step is called pruning.

We will use the `cutreeDynamic()` function from the `dynamicTreeCut` package. It has many advantages over traditional methods (e.g. gap statistic, silhouette method) including that it is more sensitive for detection of distinct classes and more stable in bootstrapping procedures.

The argument `minClusterSize` should be set to 1 and the method should be `"hybrid"`.

```
pruned = dynamicTreeCut::cutreeDynamic(clust, distM = as.matrix(d), method = "hybrid", minCluster
```

```
##  ..cutHeight not given, setting it to 31.6  ===>  99% of the (truncated) height range in dendr
##  ..done.
```

```
pruned
```

```
## 1 1 1 2 2 2 4 4 3 3 3 2 4
## 2 2 2 1 1 1 4 4 3 3 3 1 4
```

The `pruned` object includes our final clustering solution (i.e. which item belongs to which cluster).

We should pass the same names to it, that we used for the items:

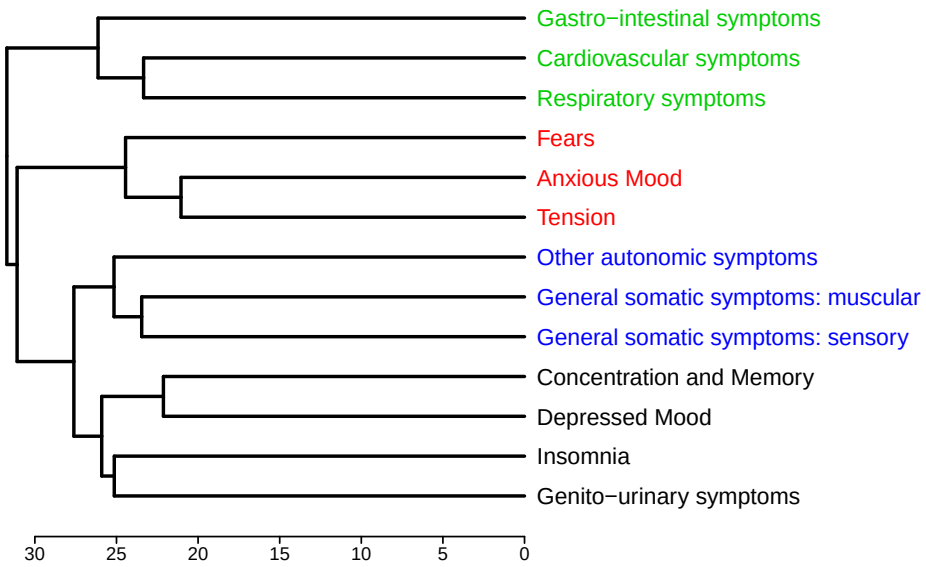
```
names(pruned) = hama_names
pruned
```

```
##                        Anxious Mood                                 Tension
```

```
##                                 2                                 2
##                             Fears                         Insomnia
##                                 2                                 1
##           Concentration and Memory                  Depressed Mood
##                                 1                                 1
## General somatic symptoms: muscular General somatic symptoms: sensory
##                                 4                                 4
##            Cardiovascular symptoms             Respiratory symptoms
##                                 3                                 3
##         Gastro-intestinal symptoms          Genito-urinary symptoms
##                                 3                                 1
##           Other autonomic symptoms
##                                 4
```

Now we can plot our pruned dendrogram. We will indicate class membership using colours:

```
labels_colors(dend) = pruned[c(clust$order)]
labels_cex(dend) = 2
marg = c(4, 4, 10, 35)
par(mar = marg, font = 1, cex = 0.4, cex.axis = 1.7, cex.lab = 2)
plot(rev(dend), horiz = T, edgePar = list(lwd = 2))
```

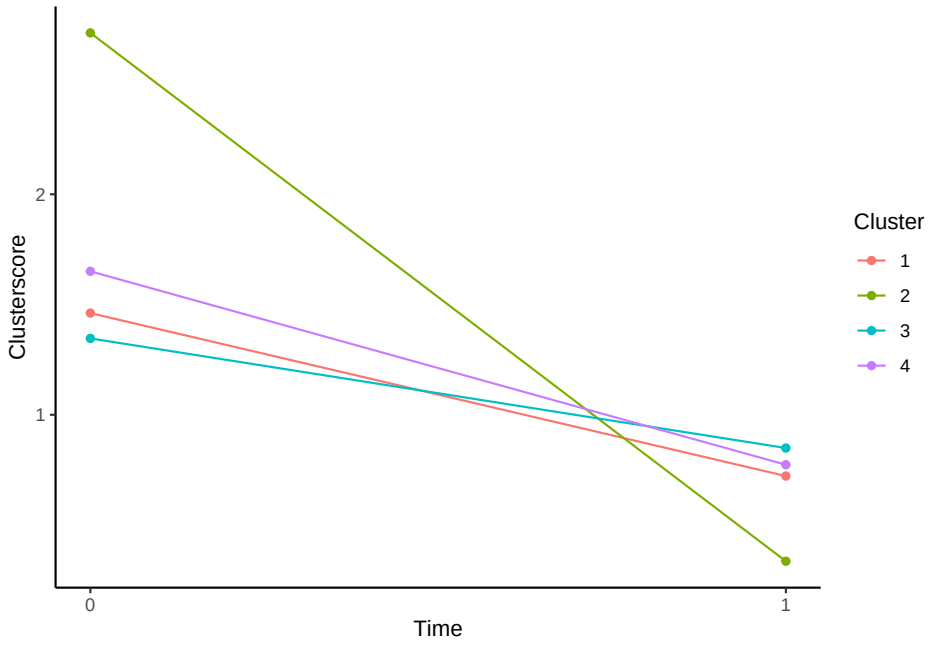



```
data_cluster_bl$id = row.names(data_cluster_bl)
data_cluster_bl$time = 0
data_cluster_post$id = row.names(data_cluster_post)
data_cluster_post$time = 1
```

```
data_cluster_bl$sum_c1 = rowSums(data_cluster_bl[which(pruned == 1)]) / length(which(pruned == 1)
data_cluster_bl$sum_c2 = rowSums(data_cluster_bl[which(pruned == 2)]) / length(which(pruned == 2)
data_cluster_bl$sum_c3 = rowSums(data_cluster_bl[which(pruned == 3)]) / length(which(pruned == 3)
data_cluster_bl$sum_c4 = rowSums(data_cluster_bl[which(pruned == 4)]) / length(which(pruned == 4)

data_cluster_post$sum_c1 = rowSums(data_cluster_post[which(pruned == 1)]) / length(which(pruned =
data_cluster_post$sum_c2 = rowSums(data_cluster_post[which(pruned == 2)]) / length(which(pruned =
data_cluster_post$sum_c3 = rowSums(data_cluster_post[which(pruned == 3)]) / length(which(pruned =
data_cluster_post$sum_c4 = rowSums(data_cluster_post[which(pruned == 4)]) / length(which(pruned =
```

```
data_cluster = rbind(data_cluster_bl, data_cluster_post)
data_cluster_long = multilevel::make.univ(data_cluster, data_cluster[,grep("sum", names(data_clus
data_cluster_long = rename(data_cluster_long, Cluster = TIME)
data_cluster_long$Cluster = data_cluster_long$Cluster +1
data_cluster_long$Cluster = factor(data_cluster_long$Cluster)
```

```
ggplot(data = data_cluster_long, aes(x = time, y = Symptoms, colour = Cluster)) +
  stat_summary(geom = "line", fun = "mean") +
   stat_summary(geom = "point", fun = "mean") +
  scale_x_continuous(breaks = c(0,1)) +
  labs(y = "Clusterscore", x = "Time") +
  theme_classic()
```



Let us model change, but now as a function of cluster:

First let us check the omnibus test:

```
mod = lmer(Symptoms ~ time * Cluster + (1|id), data = data_cluster_long)

anova(mod)
```

```
## Type III Analysis of Variance Table with Satterthwaite's method
##              Sum Sq Mean Sq NumDF DenDF F value    Pr(>F)
## time         889.75  889.75     1  2443 1963.83 < 2.2e-16 ***
## Cluster      424.03  141.34     3  2443  311.97 < 2.2e-16 ***
## time:Cluster 388.60  129.53     3  2443  285.90 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

There is a significant Cluster x Time interaction. Let us probe the interaction effect using pairwise comparisons between the cluster-specific slopes:

```
emmeans::emtrends(mod, specs = pairwise ~ Cluster, var = "time")
```

```
## $emtrends
##  Cluster time.trend     SE   df lower.CL upper.CL
##  1           -0.739 0.0509 2443   -0.839   -0.639
##  2           -2.396 0.0509 2443   -2.496   -2.296
##  3           -0.497 0.0509 2443   -0.597   -0.397
##  4           -0.877 0.0509 2443   -0.977   -0.777
##
## Results are averaged over the levels of: time
## Degrees-of-freedom method: kenward-roger
## Confidence level used: 0.95
##
## $contrasts
##  contrast            estimate    SE   df t.ratio p.value
##  Cluster1 - Cluster2    1.657 0.072 2443  23.029  <.0001
##  Cluster1 - Cluster3   -0.242 0.072 2443  -3.363  0.0043
##  Cluster1 - Cluster4    0.138 0.072 2443   1.920  0.2197
##  Cluster2 - Cluster3   -1.899 0.072 2443 -26.392  <.0001
##  Cluster2 - Cluster4   -1.519 0.072 2443 -21.109  <.0001
##  Cluster3 - Cluster4    0.380 0.072 2443   5.283  <.0001
##
## Results are averaged over the levels of: time
## Degrees-of-freedom method: kenward-roger
## P value adjustment: tukey method for comparing a family of 4 estimates
```

3.2 Exercise 2: Response Trajectories

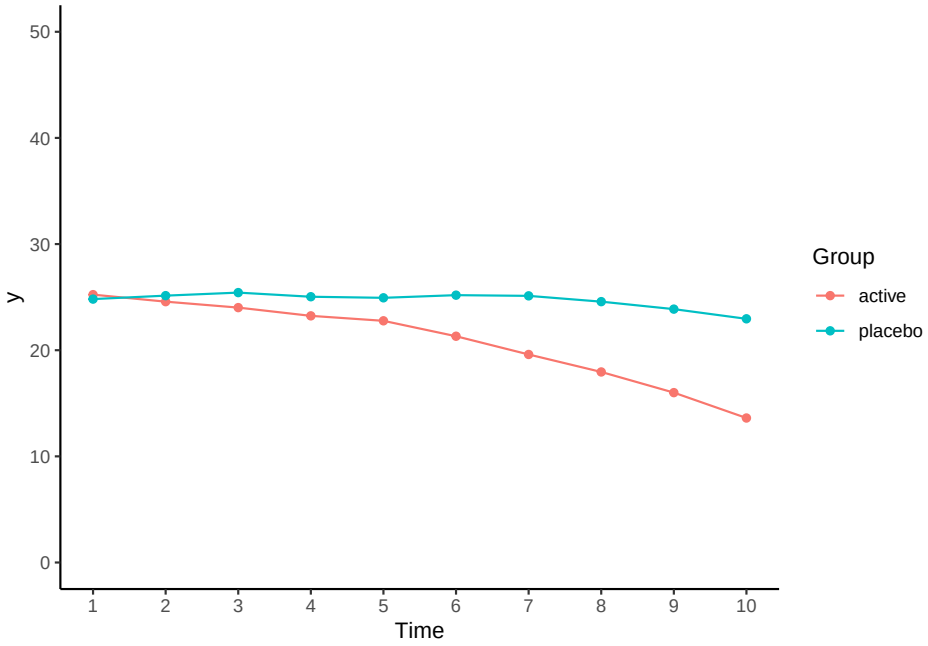
3.2.1 Simulate Practice Data

```
n = 350 # number of individuals
t = 1:10   # number of time periods

df = expand.grid(t = 1:max(t),
                 id = 1:n)
df$group = c(rep("active", nrow(df)/2), rep("placebo", nrow(df)/2))

trajectory = c("Linear response",
               "Deteriorate",
               "Rev. U-shape",
               "Rapid response",
               "No change")

set.seed(123)
for(ch in unique(df$id)){

  if(df$group[df$id == ch][1] == "active"){
    df$trajectory[df$id == ch] = rep(sample(trajectory, size = 1, replace = T, prob = c(.5, .05,
  }
  if(df$group[df$id == ch][1] == "placebo"){
    df$trajectory[df$id == ch] = rep(sample(trajectory, size = 1, replace = T, prob = c(.2, .2, .
  }

  if(df$trajectory[df$id == ch][1] == "No change"){
    df$y[df$id == ch] = 24 + 0*t  + rnorm(nrow(df[df$id == ch,]), 0, 3)
  }
  if(df$trajectory[df$id == ch][1] == "Rev. U-shape"){
    df$y[df$id == ch] = 24 + 8*t - 0.9*t^2 + rnorm(nrow(df[df$id == ch,]), 0, 3)
  }
  if(df$trajectory[df$id == ch][1] == "Linear response"){
    df$y[df$id == ch] = 24 - 1*t  + rnorm(nrow(df[df$id == ch,]), 0, 3)
  }
  if(df$trajectory[df$id == ch][1] == "Deteriorate"){
    df$y[df$id == ch] = 24 + 2*t  + rnorm(nrow(df[df$id == ch,]), 0, 3)
  }
  if(df$trajectory[df$id == ch][1] == "Rapid response"){
    df$y[df$id == ch] = 24 - 10 * log(t) +  rnorm(nrow(df[df$id == ch,]), 0, 3)
  }
}
```

3.2.2 Inspect the Data

Plot the data as we usually would (one trajectory per group)

```
ggplot(data = df, aes(x = t, y = y, colour = group)) +
  stat_summary(geom = "line", fun = "mean") +
  stat_summary(geom = "point", fun = "mean") +
  scale_x_continuous(breaks = t) +
  coord_cartesian(ylim = c(0,50)) +
  labs(x = "Time", colour = "Group") +
  theme_classic()
```



Analyze using linear mixed model:

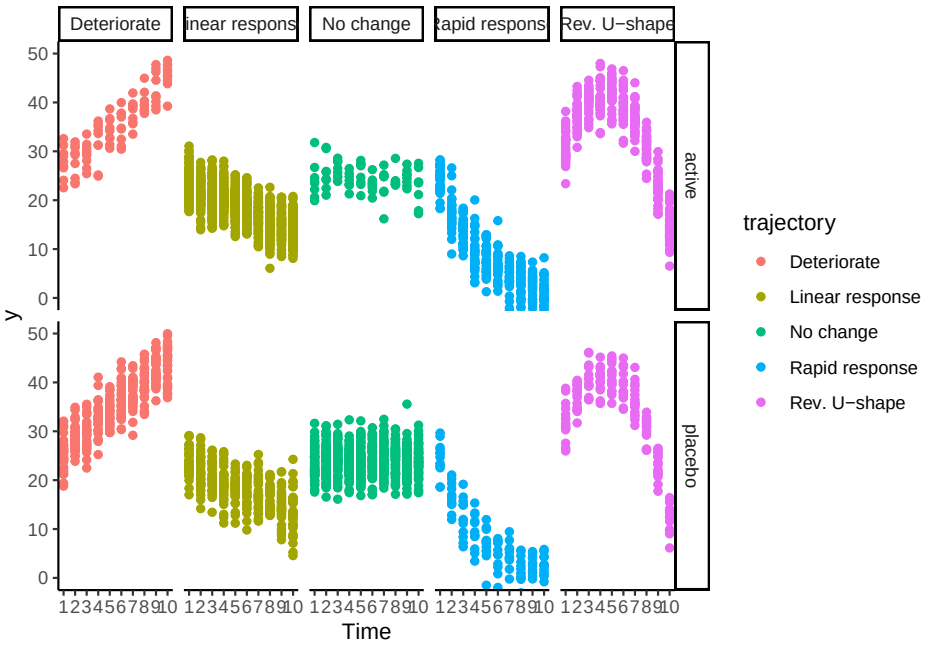
```
library(lme4)
library(lmerTest)

summary(lmer(y ~ t * group + (1|id), data = df))
```

```
## Linear mixed model fit by REML. t-tests use Satterthwaite's method [
## lmerModLmerTest]
## Formula: y ~ t * group + (1 | id)
##    Data: df
##
## REML criterion at convergence: 22778.7
##
```

```
## Scaled residuals:
##     Min      1Q  Median      3Q     Max
## -4.4769 -0.5261 -0.0116  0.5401  3.7271
##
## Random effects:
##  Groups   Name        Variance Std.Dev.
##  id       (Intercept) 61.64    7.851
##  Residual             28.74    5.361
## Number of obs: 3500, groups:  id, 350
##
## Fixed effects:
##                   Estimate Std. Error         df t value Pr(>|t|)
## (Intercept)      27.73980    0.65488  469.59512  42.358   <2e-16 ***
## t                -1.25607    0.04462 3148.00004 -28.152   <2e-16 ***
## groupplacebo     -2.05383    0.92614  469.59513  -2.218   0.0271 *
## t:groupplacebo    1.07806    0.06310 3148.00002  17.085   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Correlation of Fixed Effects:
##             (Intr) t      grpplc
## t           -0.375
## groupplaceb -0.707  0.265
## t:groupplcb  0.265 -0.707 -0.375
```

Plot data on individual change groups (without the mixture model, we usually do not know these in advance):

```
ggplot(data = df, aes(x = t, y = y, colour = trajectory)) +
  geom_point() +
  scale_x_continuous(breaks = t) +
  coord_cartesian(ylim = c(0,50)) +
  facet_grid(cols = vars(trajectory), rows = vars(group)) +
  labs(colour = "trajectory", x = "Time") +
  theme_classic()
```



3.2.3 Create LCLMM

To identify reasonable grouping categories for these individually improving patients, we need to build a latent model.

Let us compute a growth mixtue model aka. latent class linear mixed models (LCLMM). We use the package `lcmm` for this.

- The `fixed` argument is a formula, as we know it from mixed models. We determine a polynomial here (usually quadratic or cubic as this is how most symptoms have been shown to change).
- The `mixture` argument specifies class-specific fixed effects. These are the change parameters the trajectories are defined on
- The `mixture` argument specifies a random argument as in the LMM. May be `~ 1` for random intercepts or `1 + t` for random intercepts and slopes.
- The `ng` argument specifies the number of classes to be extracted. We will learn in a second how the optimal number of classes can be determined.
- The `subject` argument specifies the nesting structure due to the repeated measurements.

```
library(lcmm)
library(LCTMtools)
```

```
mi = lcmm::hlme(fixed = y ~ 1 + t + I(t^2),
           mixture = ~ 1 + t + I(t^2),
           random = ~ 1,
           ng = 5,
           data = df,
           subject = "id")
```

```
## Be patient, hlme is running ...
## The program took 7.48 seconds
```

3.2.4 Inspect the LCLMM Model

Let us check the mixture object `mi`:

```
mi
```

```
## Heterogenous linear mixed model
##      fitted by maximum likelihood method
##
## lcmm::hlme(fixed = y ~ 1 + t + I(t^2), mixture = ~1 + t + I(t^2),
##     random = ~1, subject = "id", ng = 5, data = df)
##
## Statistical Model:
##      Dataset: df
##      Number of subjects: 350
##      Number of observations: 3500
##      Number of latent classes: 5
##      Number of parameters: 21
##
## Iteration process:
##      Convergence criteria satisfied
##      Number of iterations:  20
##      Convergence criteria: parameters= 2.1e-09
##                          : likelihood= 3e-06
##                          : second derivatives= 1.6e-05
##
## Goodness-of-fit statistics:
##      maximum log-likelihood: -9352.41
##      AIC: 18746.83
##      BIC: 18827.84
##
##
```

We can see an overview over our selected parameters and that the model has converged fine. We also get a selection of goodness-of-fit statistics, that we could use for model selection.

We can inspect the model further: The `LCTMtoolkit()` function gives us a convenient print out for the quality of our model and also displays some benchmark for orientation

```
LCTMtoolkit_total = LCTMtoolkit(mi)
```

```
## [1] "class(model) type required to be hlme, lcmm or an imported PROC TRAJ object fr
## $`Class-specific`
##          Class_1  Class_2  Class_3 Class_4 Class_5   Recommendation
## APPA           1    0.999     1.00       1       1 Greater than 0.7
## OCC          Inf 1479.031 73048.39     Inf     Inf   Greater than 5
## Mismatch       0    0.000     0.00       0       0    Close to zero
##
## $`Model-specific`
##                       Model Recommendation
## Entropy               0.463  Close to zero
## Relative_entropy      0.999     Close to 1
## BIC               18827.842              -
## AIC               18746.826              -
```

The `postprob()` function displays posterior classifications (i.e. group membership frequencies) for all extracted classes.

Often we want to define a minimum cutoff for clinical relevance (e.g. min. 5% capture of all patients):

```
postprob_total = lcmm::postprob(mi)
```

```
##
## Posterior classification:
##   class1 class2 class3 class4 class5
## N  45.00 128.00  83.00  52.00     42
## %  12.86  36.57  23.71  14.86     12
##
## Posterior classification table:
##      --> mean of posterior probabilities in each class
##        prob1  prob2  prob3 prob4 prob5
## class1     1 0.0000 0.0000     0     0
## class2     0 0.9988 0.0012     0     0
## class3     0 0.0000 1.0000     0     0
## class4     0 0.0000 0.0000     1     0
## class5     0 0.0000 0.0000     0     1
##
## Posterior probabilities above a threshold (%):
##          class1 class2 class3 class4 class5
## prob>0.7    100 100.00    100    100    100
## prob>0.8    100 100.00    100    100    100
## prob>0.9    100  99.22    100    100    100
```

```
##
```

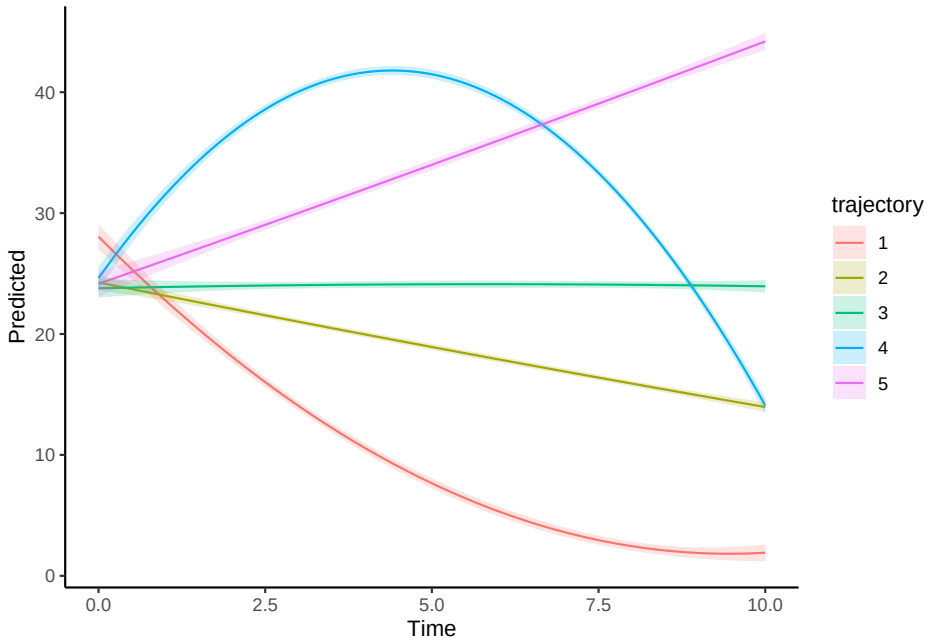
3.2.5 Plot LCLMM Model Predictions

Create custom function for LCLMM plotting:

```
plot_traj = function(mi, data, var.time){
  datnew   <- data.frame(t = seq(0, max(data[, var.time]), length = 100))
  plotpred <- lcmm::predictY(mi, datnew, var.time = var.time, draws = TRUE)

  frame_traj = as.data.frame(expand.grid(Time = plotpred$times$t,
                                         trajectory = unique(mi$pprob$class),
                                         pred = NA,
                                         upper = NA,
                                         lower = NA))

  for(traj in unique(frame_traj$trajectory)){
    for(i in 1:100){
      frame_traj$pred[frame_traj$trajectory == traj][i] = plotpred$pred[,which(grepl(paste0("^Ypr
      frame_traj$upper[frame_traj$trajectory == traj][i] = plotpred$pred[,which(grepl(paste0("^lo
      frame_traj$lower[frame_traj$trajectory == traj][i] = plotpred$pred[,which(grepl(paste0("^up
    }
  }
  frame_traj$trajectory = factor(frame_traj$trajectory)
  return(ggplot(data = frame_traj, aes(x = Time, y = pred, ymin = lower, ymax = upper)) +
    # geom_vline(xintercept = c(5, 10, 14, 17), linetype = "dotted") +
    geom_line(aes(colour = trajectory)) +
      labs(y = "Predicted") +
    geom_ribbon(aes(fill = trajectory), alpha = .2, linetype = "dotted") +
    theme_classic())
}
```

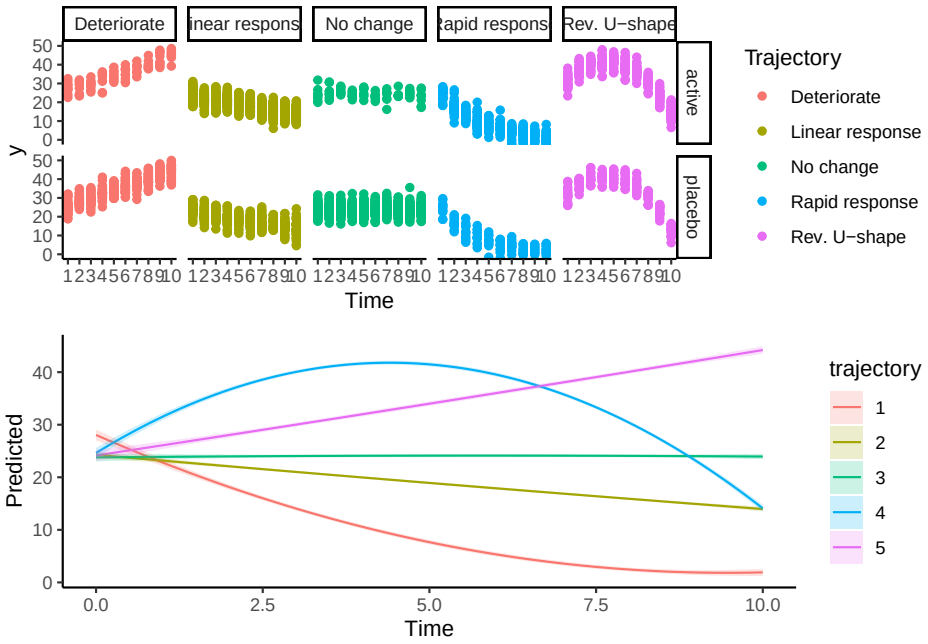
Plot the result:

```
plot_traj(mi, df, "t")
```



Let us check the graph next to the empirical data:

```
cowplot::plot_grid(ggplot(data = df, aes(x = t, y = y, colour = trajectory)) +
                     geom_point() +
                     scale_x_continuous(breaks = t) +
                     coord_cartesian(ylim = c(0,50)) +
                     facet_grid(cols = vars(trajectory), rows = vars(group)) +
                     labs(colour = "Trajectory", x = "Time") +
                     theme_classic(),
                   plot_traj(mi, df, "t"), nrow = 2)
```



3.2.6 Transfer class membership to original dataset

Now we should transfer the determined class to our empirical dataset. Otherwise, we will not be able to run models using the trajectories.

In addition, we extract the certainty, that each person was classified to a category with (the `pprob` variable in the `mi` object). Using these probability values we can weigh later models for categorization uncertainty.

Create custom function for transfer:

```
transfer_class = function(data, mi){
  data$class = NA
  for(ch in unique(data$id)){
    data$class[data$id == ch] = mi$pprob$class[mi$pprob$id == ch]
    data$weight[data$id == ch] = mi$pprob[mi$pprob$id == ch, which(grepl(paste0("prob", as.charac
  }
  data$class = factor(data$class)
  return(data)
}
```

Transfer data to our original dataframe `df`:

```
df = transfer_class(data = df, mi = mi)
```

Create data in wide format:

```
df_wide = as.data.frame(df %>%
  pivot_wider(names_from = t, values_from = y))

levels(df_wide$class) = c( "Rapid response",
                        "Linear response",
                        "No change",
                        "Rev. U-shape",
                        "Deteriorate")
```

3.2.7 Modeling class membership as dependent variable

We can check the dispersion of the trajectory class variable within the 2 treatment groups using `table()`:

```
table(df_wide$class, df_wide$group)
```

```
##
##                    active placebo
##   Rapid response       33      12
##   Linear response      90      38
##   No change             8      75
##   Rev. U-shape         35      17
##   Deteriorate           9      33
```

If we want to use trajectory class membership as the dependent variable, we need to used a logistic-regression model (because `class` is a categorical variable). Since we usually have more than 2 trajectory classes, we will use a multinomial logistic-regression model.

```
multinom = nnet::multinom(class ~ group, data = df_wide, weights = weight)
```

```
## # weights:  15 (8 variable)
## initial  value 563.055666
## iter  10 value 471.814704
## final  value 470.947184
## converged
```

For an omnibus test, we can use a chi-square-likelihood-ratio test:

```
car::Anova(multinom)
```

```
## # weights:  10 (4 variable)
## initial  value 563.055666
## final  value 528.549964
## converged
```

```
## Analysis of Deviance Table (Type II tests)
##
```

```
## Response: class
##       LR Chisq Df Pr(>Chisq)
## group   115.21  4  < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For pairwise comparisons we can use the `emmeans` package:

```
emmeans::lsmeans(multinom, pairwise ~ group | class, adjust="tukey", mode = "prob")
```

```
## $lsmeans
## class = Rapid response:
##  group     prob     SE df lower.CL upper.CL
##  active  0.1887 0.0296  8  0.12049   0.2570
##  placebo 0.0686 0.0191  8  0.02452   0.1126
##
## class = Linear response:
##  group     prob     SE df lower.CL upper.CL
##  active  0.5139 0.0378  8  0.42671   0.6010
##  placebo 0.2171 0.0312  8  0.14527   0.2890
##
## class = No change:
##  group     prob     SE df lower.CL upper.CL
##  active  0.0458 0.0158  8  0.00931   0.0822
##  placebo 0.4286 0.0374  8  0.34230   0.5148
##
## class = Rev. U-shape:
##  group     prob     SE df lower.CL upper.CL
##  active  0.2002 0.0303  8  0.13039   0.2699
##  placebo 0.0971 0.0224  8  0.04552   0.1488
##
## class = Deteriorate:
##  group     prob     SE df lower.CL upper.CL
##  active  0.0515 0.0167  8  0.01294   0.0900
##  placebo 0.1886 0.0296  8  0.12039   0.2568
##
## Confidence level used: 0.95
##
## $contrasts
## class = Rapid response:
##  contrast         estimate     SE df t.ratio p.value
##  active - placebo    0.120 0.0352  8   3.411  0.0092
##
## class = Linear response:
##  contrast         estimate     SE df t.ratio p.value
##  active - placebo    0.297 0.0490  8   6.057  0.0003
##
```

```
## class = No change:
##  contrast         estimate     SE df t.ratio p.value
##  active - placebo   -0.383 0.0406  8  -9.427  <.0001
##
## class = Rev. U-shape:
##  contrast         estimate     SE df t.ratio p.value
##  active - placebo    0.103 0.0376  8   2.737  0.0256
##
## class = Deteriorate:
##  contrast         estimate     SE df t.ratio p.value
##  active - placebo   -0.137 0.0340  8  -4.037  0.0038
```

Estimates are displayed as log-odds, so we have to exponentiate them using `exp()` to get interpretable odds ratios (OR).

3.2.8 Determine the optimal model

In the previous dataset, we have

1. used a qudratic polynomial
2. extracted 5 groups
3. allowed free variation of the intercept as the random effect.

However, usually we operate on a more data-driven approach, i.e. without knowing these parameters in advance.

There are several statistical criteria to determine a mode that optimally describes the data. For LCLMM, the most commonly used ones are:

- Bayesian Information Criterion (BIC)
  - lower values on these information criteria indicate better fitting models
  - models that do not fit better than the baseline model can be dismissed, and a selection of the best fitting models can be carried forward and examined further
- Entropy
  - ranges from 0.00 to 1.00
  - high values of entropy ($> .80$) indicate that individuals are classified with confidence
  - models with higher entropy are favored
- Adjusted Lo-Mendell-Rubin likelihood ratio test
  - corrected likelihood-ratio distribution (a chi-square distribution is inappropriate) to compare models with C and C – 1 unobserved groups
  - likelihood ratio tests compare models that differ only in the number of classes
  - significance test ($p< .05$) indicates that the model with C – 1 classes should be rejected in favor of the model with C classes

3.2.9 Selection loop

To try this, let us run a model selection based on the Bayesian Information Criterion (BIC).

Since we have to test a lot of parameters, we run the model fitting procedure in a loop and save the results in a container:

This procedure can take quite a while, so we'll test it in a smaller dataset:

```
test = read.csv("https://raw.githubusercontent.com/stephangoerigk/DZP_Workshop_Slides/master/test
```

Inspect the data:

```
psych::describe(test)
```

```
##        vars    n   mean     sd median trimmed    mad   min    max  range  skew
## X         1 1680 200.29 113.20  219.4  204.98 150.48  1.00 352.80 351.80 -0.26
## group*    2 1680   1.50   0.50    1.5    1.50   0.74  1.00   2.00   1.00  0.00
## t         3 1680   4.50   2.87    4.5    4.50   3.71  0.00   9.00   9.00  0.00
## stress    4 1677  43.39  30.65   48.0   42.29  39.56 -2.79 101.98 104.77  0.14
## id        5 1680 200.02 113.20  219.0  204.71 150.48  1.00 352.00 351.00 -0.26
##        kurtosis   se
## X         -1.38 2.76
## group*    -2.00 0.01
## t         -1.23 0.07
## stress    -1.07 0.75
## id        -1.38 2.76
```

```
head(test)
```

```
##      X   group t    stress id
## 1 39.0 Healthy 0 80.199113 39
## 2 39.2 Healthy 1 70.188702 39
## 3 39.4 Healthy 2 50.653667 39
## 4 39.6 Healthy 3 16.588692 39
## 5 39.8 Healthy 4 -2.031454 39
## 6 39.1 Healthy 5 -1.285336 39
```

Let us first set up a container:

```
results_total = data.frame(ng = NA,
                           Polynomial = NA,
                           Random = NA,
                           BIC = NA,
                           AIC = NA,
                           loglik = NA)
```

We will run out loop for 2:5 groups,quadratic vs. cubic polynomial and different random effect compositions:

```
set.seed(222)

for(ng in 2:5){
  for(random in c("~ 1", "~ 1 + t")){
    mi_sq <- lcmm::hlme(fixed = stress ~ 1 + t + I(t^2),
                        mixture = ~ 1 + t + I(t^2),
                        random = as.formula(random),
                        ng = ng,
                        nwg = FALSE,
                        idiag = FALSE,
                        data = test,
                        subject = "id")
    mi_cub <- lcmm::hlme(fixed = stress ~ 1 + t + I(t^3),
                         mixture = ~ 1 + t + I(t^3),
                         random = as.formula(random),
                         ng = ng,
                         nwg = FALSE,
                         idiag = FALSE,
                         data = test,
                         subject = "id")

    sq <- c(mi_sq$ng, 2, random, mi_sq$BIC, mi_sq$AIC, mi_sq$loglik)
    cub <- c(mi_cub$ng, 3, random, mi_cub$BIC, mi_cub$AIC, mi_cub$loglik)
    results_total = rbind(results_total, sq)
    results_total = rbind(results_total, cub)
  }
}

results_total = results_total[order(results_total$BIC, decreasing = T),]
```

The solution with the lowest BIC is chosen. Now we can once more fit the LCLMM, only now we used the determined parameters

CAVE: While a statistical determination of model parameters is important, there should always be clinical plausibility checks as well.