Solution NCGS exercise

Course repeated measurements - R exercise class 1

November 20, 2018

Contents

NOTE: This document contains an example of R code and related software outputs that answers the questions of the NCGS exercise. The focus here is on the implementation using the R software and not on the interpretation - we refer to the SAS solution for a more detailed discussion of the results.

The solution is written using as much as possible basic R packages such as `base`, `stats`, `graphics`, `nlme`. The benefit is that any **R** should be able to read the code (i.e. knowledge of specific specific packages is not required). However in some cases, it makes the code overly complicated. Alternative code based on `data.table` and `ggplot2` is given in appendix.

Load the packages that will be necessary for the analysis:

```
library(reshape2)    # for converting data.frame from wide to long format
library(nlme)    # implementation of models for repeated measurements (e.g. gls, lme)
```

1 Question 1: Import the dataset and describe it

1.1 Import data

Download the file `ncgs.R` from the course webpage http://publicifsv.sund.ku.dk/~jufo/RepeatedMeasures2018.html and put it into a directory that we will call `Exercise1`.

Set the working directory to `Exercise1` using `setwd`. Check that the file `ncgs.R` exists in this directory using `file.exists`:

```
file.exists("ncgs.R")
```

```
[1] TRUE
```

We are now ready to load the data. Since the file `ncgs.R` is an **R** file generating the data, we just need to source this file. Note that for now we have an empty **R** session:

```
ls()
```

```
character(0)
```

Let's now source `ncgs.R` using `source`:

```
source("ncgs.R")
```

We can now see that two objects have been created:

```
ls()
```

```
[1] "ncgs" "raw"
```

We are interested in the `ncgs` object. In fact the object `raw` is the same as the `ncgs` object, one converted from the matrix format to the data.frame format:

```
identical(ncgs,as.data.frame(raw))
```

```
[1] TRUE
```

Data.frame can handle columns with different types of variables (e.g. integer, character, double) so they are usually more convenient to work with. The drawback is that numerically they are less efficient (i.e. the computation time of data.frame operations is higher) but this is often neglectable.

1.2 Working with `data.frame` objects

There are several ways to get information about a dataset. First you can extract its dimensions using the method `dim`:

```
dim(ncgs)
```

```
[1] 103   7
```

The dataset contains 103 lines and 7 columns. Use `names` to extract the name of the columns (i.e. of the variables):

```
names(ncgs)
```

```
[1] "group" "id"    "y0"    "y1"    "y2"    "y3"    "y4"
```

Now use `str` to summarize the content of each column:

```
str(ncgs)
```

```
'data.frame':        103 obs. of  7 variables:
 $ group: num  1 1 1 1 1 1 1 1 1 1 ...
 $ id   : num  1 2 3 4 5 6 7 8 9 10 ...
 $ y0   : num  178 254 185 219 205 182 310 191 245 229 ...
 $ y1   : num  246 260 232 268 232 213 334 204 270 200 ...
 $ y2   : num  295 278 215 241 265 173 290 227 209 238 ...
 $ y3   : num  228 245 220 260 242 200 286 228 255 259 ...
 $ y4   : num  274 340 292 320 230 193 248 196 213 221 ...
```

Since the dataset only contains numbers R has converted all columns to the integer type (`int` in the software output). This is not appropriate since the patient id and the group are categorical variables (coded with integers). We can change that by converted these columns to factor:

```
ncgs$group <- factor(ncgs$group, levels = 1:2, labels = c("T", "C"))
ncgs$id <- as.factor(ncgs$id)
str(ncgs)
```

```
'data.frame':        103 obs. of  7 variables:
 $ group: Factor w/ 2 levels "T","C": 1 1 1 1 1 1 1 1 1 1 ...
 $ id   : Factor w/ 103 levels "1","2","3","4",..: 1 2 3 4 5 6 7 8 9 10 ...
 $ y0   : num  178 254 185 219 205 182 310 191 245 229 ...
 $ y1   : num  246 260 232 268 232 213 334 204 270 200 ...
 $ y2   : num  295 278 215 241 265 173 290 227 209 238 ...
 $ y3   : num  228 245 220 260 242 200 286 228 255 259 ...
 $ y4   : num  274 340 292 320 230 193 248 196 213 221 ...
```

The method `head` output the first lines of the dataset:

```
head(ncgs)
```

```
  group id  y0  y1  y2  y3  y4
1     T  1 178 246 295 228 274
2     T  2 254 260 278 245 340
3     T  3 185 232 215 220 292
4     T  4 219 268 241 260 320
5     T  5 205 232 265 242 230
6     T  6 182 213 173 200 193
```

Since each line contains all the observations for each patient, the data is in the wide format.

Finally it is often a good idea to check whether there are any missing values in the dataset. Missing values are called `NA` in R. The function `is.na` checks the presence of missing values:

```
is.na(NA)
is.na(1)
```

```
[1] TRUE
[1] FALSE
```

To return the number of missing values by variable, one can use the function `colSums` which will sum by column the binary indicators of missingness:

```
colSums(is.na(ncgs))
```

```
group    id    y0    y1    y2    y3    y4
    0     0     0     0    10    24    34
```

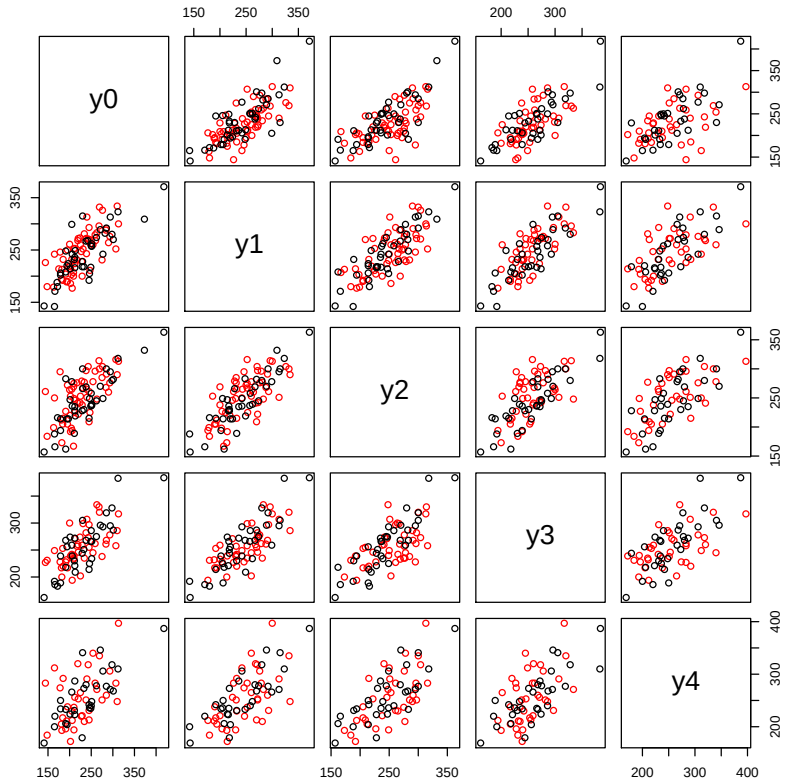
So there are no missing value in the first four columns while the column `y2` contains 10 missing values, column `y3` contains 24 missing values, and colum `y4` contains 34 missing values.

2 Question 2: Descriptive statistics

2.1 Investigating marginal distribution

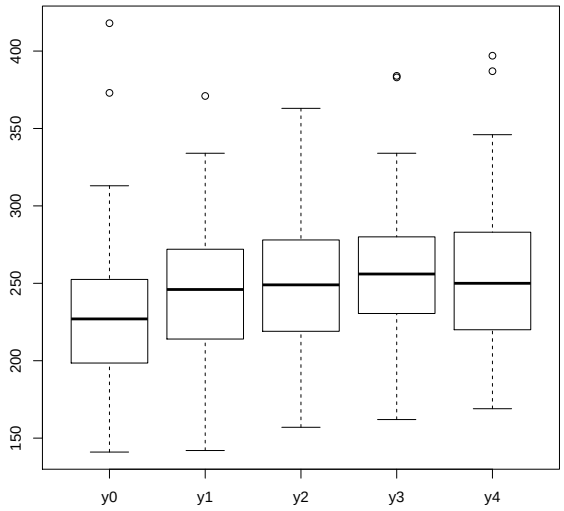
The basic display function in R is the `plot` function. When applied to a `data.frame` object it displays the bivariates association between variables:

```
vec.colors <- ifelse(ncgs$group=="T","red","black")
plot(ncgs[,c("y0","y1","y2","y3","y4")], col = vec.colors)
```



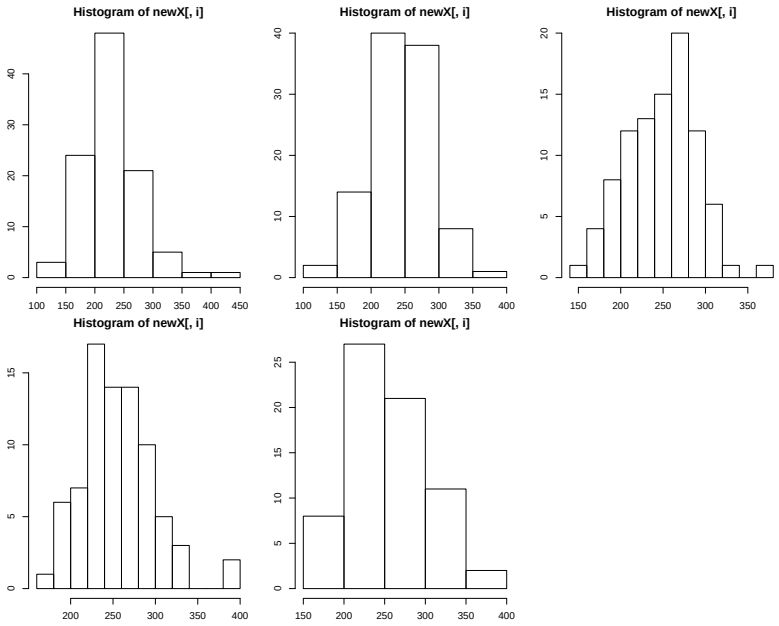
Here measurements corresponding to patients from the high dose group are displayed in red and those from the placebo group are displayed in black. The `boxplot` function can be used to display boxplots:

```
boxplot(ncgs[,c("y0","y1","y2","y3","y4")])
```



To draw histogram for several variables, first divide the graphical window and then apply the histogram function to each column of the `data.frame`:

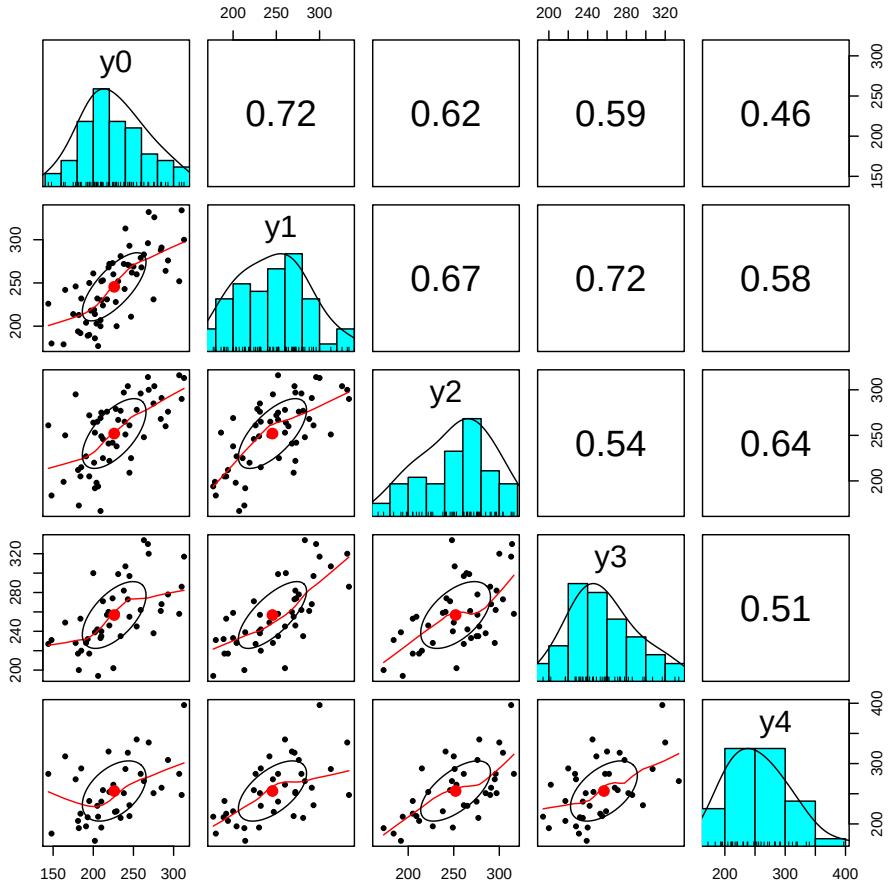
```
par(mfrow = c(2,3), mar = rep(2,4))
ls.hist <- apply(ncgs[,c("y0","y1","y2","y3","y4")],2,hist) # 2 indicates by column
```



See appendix B for the `ggplot2` syntax.

Specific types of graphical display can be obtained using functions from additional packages. For instance we can use the psych package to display histograms and scatterplots for the high dose group:

```
psych::pairs.panels(ncgs[ncgs$group=="T",c("y0","y1","y2","y3","y4")])
```



2.2 Trend in mean and variance

Mean:

```
ncgs.mean <- aggregate(cbind(y0,y1,y2,y3,y4) ~ group,
                       ncgs,
                       mean)
ncgs.mean
```

```
  group       y0       y1       y2       y3       y4
1     T 226.7778 249.6111 252.6111 253.1389 256.7222
2     C 236.6452 243.3226 244.5484 261.9032 257.4839
```

Variance:

```
ncgs.var <- aggregate(cbind(y0,y1,y2,y3,y4) ~ group,
                      ncgs,
                      var)
ncgs.var
```

```
  group       y0       y1       y2       y3       y4
1     T 1962.463 1715.216 1553.902 1147.609 2545.692
2     C 3080.437 2755.492 2267.723 2666.957 2439.191
```

See appendix A for an alternative to `aggregate` that uses `data.table`.

2.3 Computation of the correlation matrix

To compute the correlation for each treatment group we first need to subset the dataset by group and extract the columns corresponding to the cholesterol measurements at the different times. When using a `data.frame` object, the first argument in the bracket enable to subset by row while the second indicates which columns should be picked. Therefore:

```
ncgs.T <- ncgs[ncgs$group=="T",c("y0","y1","y2","y3","y4")]
```

selects the rows corresponding to the patients who received the high dose treamtent and only extract the columns of the cholesterol measurements. The correlation matrix can be computed using the `cor` function.

```
cor(ncgs.T, use = "pairwise.complete.obs")
```

```
          y0        y1        y2        y3        y4
y0 1.0000000 0.7203380 0.6226680 0.5907770 0.4581925
y1 0.7203380 1.0000000 0.6695283 0.7153099 0.5832976
y2 0.6226680 0.6695283 1.0000000 0.5374287 0.6363163
y3 0.5907770 0.7153099 0.5374287 1.0000000 0.5140974
y4 0.4581925 0.5832976 0.6363163 0.5140974 1.0000000
```

Note that one needs to specify to R how to deal with the missing values (argument `use`, see the section details in the documentation of the `cor` function). The same can be done for the placebo group:

```
ncgs.C <- ncgs[ncgs$group=="C",c("y0","y1","y2","y3","y4")]
cor(ncgs.C, use = "pairwise.complete.obs")
```

```
          y0        y1        y2        y3        y4
y0 1.0000000 0.8161257 0.8323153 0.8442542 0.7612846
y1 0.8161257 1.0000000 0.8874039 0.8688476 0.8191013
y2 0.8323153 0.8874039 1.0000000 0.8779475 0.7776534
y3 0.8442542 0.8688476 0.8779475 1.0000000 0.7892396
y4 0.7612846 0.8191013 0.7776534 0.7892396 1.0000000
```

3 Question 3: Conversion from the wide format to the long format

The `melt` function enable to convert `data.frame` objects from the wide format to the long format:

```
ncgsL <- melt(ncgs,
              id.vars = c("id","group"),
              measure.vars = c("y0","y1","y2","y3","y4"),
              variable.name = "time",
              value.name = "cholesterol")
head(ncgsL)
```

```
  id group time cholesterol
1  1     T   y0         178
2  2     T   y0         254
3  3     T   y0         185
4  4     T   y0         219
5  5     T   y0         205
6  6     T   y0         182
```

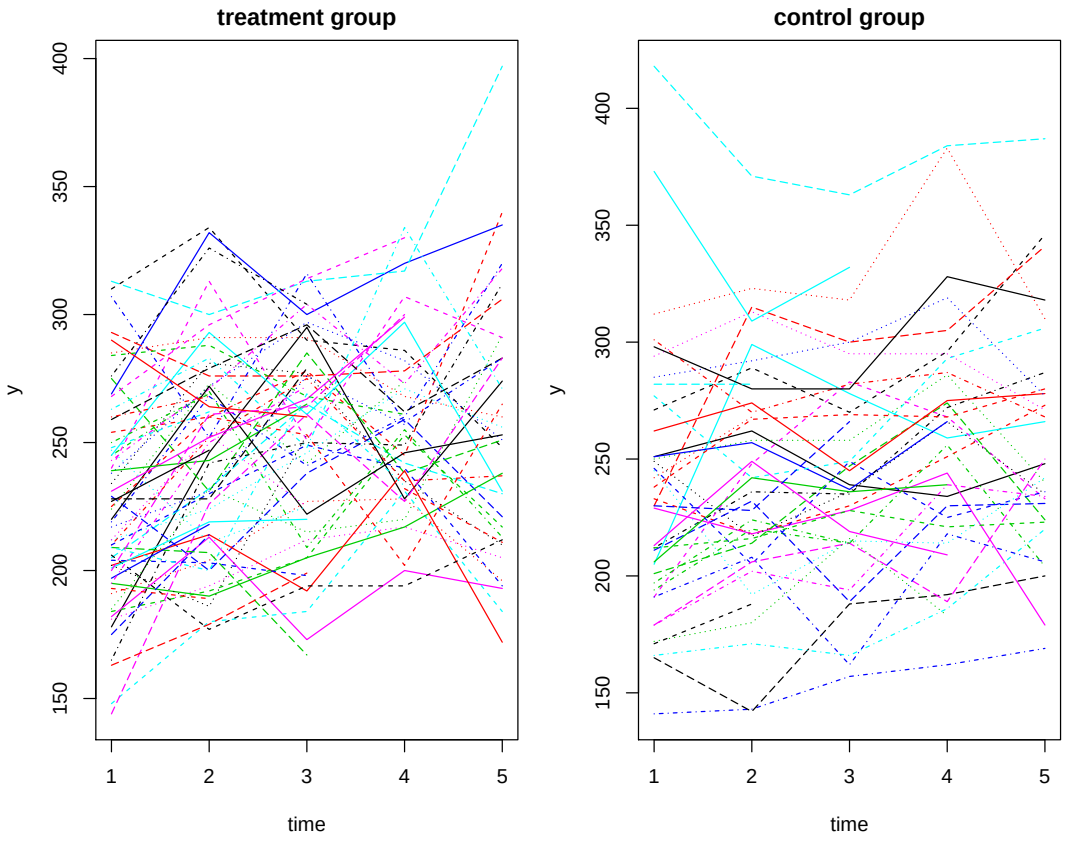
The argument `id.vars` specifies the variable names that are kept constant over the repetitions while the argument `measure.vars` indicates the name of the columns containing the different measurements.

4 Question 4: Spaghetti plots

The `matplot` method can be used to display spaguetti plot. For this we need to have the data in the wide format, with a separate dataset for each group. We can re-use the dataset `ncgs.T` and `ncgs.C` defined in Question 2:

```
par(mfrow = c(1,2), mar = c(4,4,2,1))
matplot(t(ncgs.T),
        type = "l", ylab = "y", xlab = "time", main = "treatment group")

matplot(t(ncgs.C),
        type = "l", ylab = "y", xlab = "time", main = "control group")
```



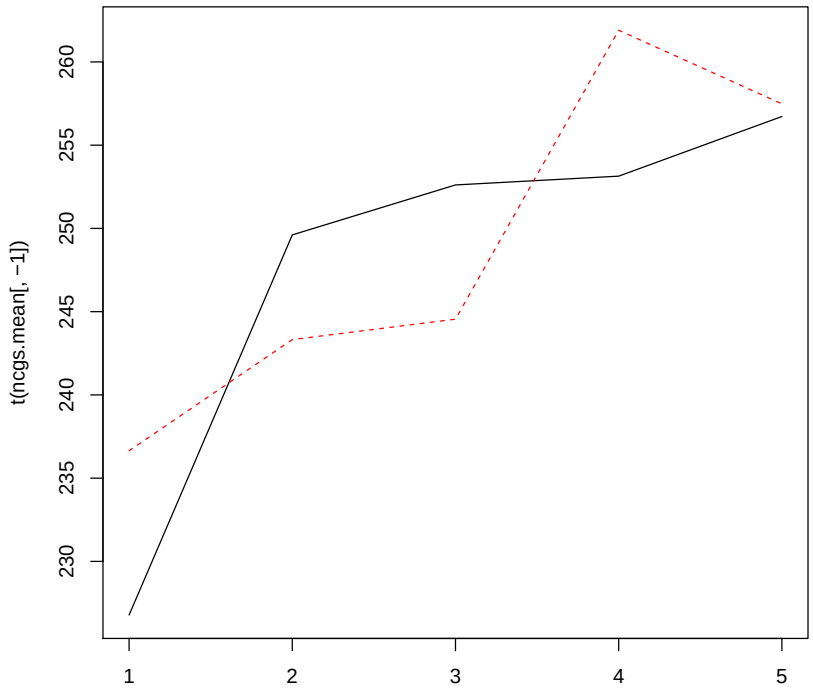
Note that the method `t(.)` is used to convert lines into columns (and columns into lines, i.e. transpose) such that the time is along the x-axis. See appendix B for the `ggplot2` syntax.

5 Question 5: Mean plot

5.1 Quick way using the `matplot` method

We already have computed the means in Question 2. A quick way to obtain a graphical display of the mean (or variance) is using `matplot`:

```
matplot(t(ncgs.mean[,-1]), type = "l")
```



When calling `matplot` we removed the first column and transpose the table so that the x-axis represents time and the y-axis represent the levels of serum cholesterol.

5.2 A nicer display using the `plot` method

Otherwise we can extract the mean for each group:

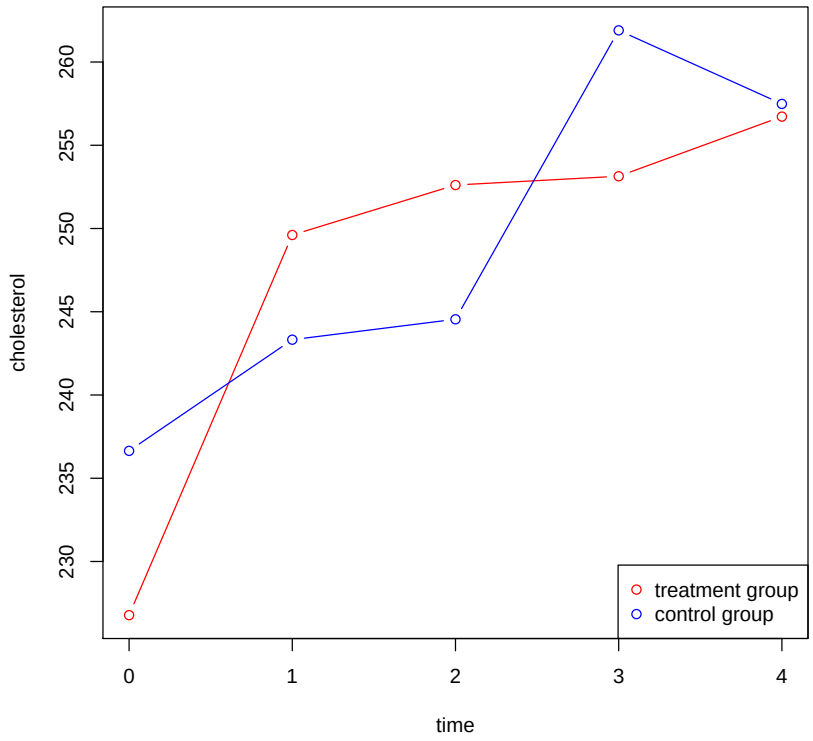
```
vec.meanT <- ncgs.mean[ncgs.mean$group == "T",-1]
vec.meanC <- ncgs.mean[ncgs.mean$group == "C",-1]
```

and the compute the minimum and maximum value:

```
y.rangeObs <- range(ncgs.mean[,-1])
```

We can then create the graphical display using the `plot` function.

```
plot(0:4, vec.meanT, col = "red", type = "b",
     ylim = y.rangeObs, xlab = "time", ylab = "cholesterol")
points(0:4, vec.meanC, col = "blue", type = "b")
legend("bottomright", pch = 21,
       legend = c("treatment group","control group"), col = c("red","blue"))
```



See appendix B for the `ggplot2` syntax.

6 Question 6: Mixed model (ignoring randomization)

Note: for pedagogical reasons we use a mixed model with a simpler structure for the residual covariance (compound symmetry structure instead of unstructured) compared to the SAS solution. See appendix C for the model matching the SAS correction.

Before fitting a mixed model we specify the reference category for the categorical variables:

```
ncgsL$group <- relevel(ncgsL$group,"C")
ncgsL$time <- relevel(ncgsL$time,"y0")
```

6.1 `lme` vs. `gls`

We can fit a random intercept model with constant variance:

- using `lme`:

```
e.lme <- lme(cholesterol ~ group*time,
             random =~1|id,
             data = ncgsL, na.action = na.omit)
logLik(e.lme)
```

```
'log Lik.' -2145.864 (df=12)
```

- using `gls`

```
e.gls <- gls(cholesterol ~ group*time,
             correlation = corCompSymm(form =~ 1|id),
             data = ncgsL, na.action = na.omit)
logLik(e.gls)
```

```
'log Lik.' -2145.864 (df=12)
```

The two models are equivalent as indicated by the log-likelihood. We can also compare the estimated parameters for the mean:

```
coef(e.gls)-fixef(e.lme)
```

```
 (Intercept)        groupT        timey1        timey2        timey3
3.979039e-13 -1.225686e-13  3.819167e-14  5.335057e-10 -1.298881e-08
      timey4 groupT:timey1 groupT:timey2 groupT:timey3 groupT:timey4
1.894868e-09  2.184919e-13  4.388696e-09  3.378876e-08  2.538938e-09
```

and the estimated variance-covariance matrices:

```
getVarCov(e.gls, individuals = 1)
```

```
Marginal variance covariance matrix
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 1970.0 1407.6 1407.6 1407.6 1407.6
[2,] 1407.6 1970.0 1407.6 1407.6 1407.6
[3,] 1407.6 1407.6 1970.0 1407.6 1407.6
[4,] 1407.6 1407.6 1407.6 1970.0 1407.6
[5,] 1407.6 1407.6 1407.6 1407.6 1970.0
  Standard Deviations: 44.385 44.385 44.385 44.385 44.385
```

```
getVarCov(e.lme, individuals = 1, type = "marginal")
```

```
id 1
Marginal variance covariance matrix
       1      2      3      4      5
1 1970.0 1407.6 1407.6 1407.6 1407.6
2 1407.6 1970.0 1407.6 1407.6 1407.6
3 1407.6 1407.6 1970.0 1407.6 1407.6
4 1407.6 1407.6 1407.6 1970.0 1407.6
5 1407.6 1407.6 1407.6 1407.6 1970.0
  Standard Deviations: 44.385 44.385 44.385 44.385 44.385
```

We will continue with the `gls` model.

6.2 Estimated mean change

We can extract the estimates and standard errors for the mean parameters using the `summary` function:

```
summary(e.gls)$tTable
```

```
                  Value Std.Error    t-value       p-value
(Intercept)   235.926829  6.931803 34.0354206 5.863900e-125
groupT         -9.910700  8.934474 -1.1092651  2.679259e-01
timey1          7.243902  5.238026  1.3829451  1.673879e-01
timey2          8.789252  5.382499  1.6329314  1.032037e-01
timey3         23.316037  5.532919  4.2140576  3.048868e-05
timey4         20.792687  5.758305  3.6109043  3.405148e-04
groupT:timey1  12.272227  6.751347  1.8177449  6.978774e-02
groupT:timey2  16.425710  6.978203  2.3538595  1.902153e-02
groupT:timey3   4.797929  7.324612  0.6550421  5.127853e-01
groupT:timey4   7.594387  7.659611  0.9914848  3.219976e-01
```

Note that the computation of the degree of freedom in `nlme` rely on a very crude approximation. This can be problematic when the sample size is small compared to the number of parameters (incorrect control of the type 1 error).

The `intervals` method can be used to obtain confidence intervals:

```
intervals(e.gls)[["coef"]]
```

```
                    lower       est.     upper
(Intercept)   222.3030129 235.926829 249.55065
groupT        -27.4705805  -9.910700   7.64918
timey1         -3.0509524   7.243902  17.53876
timey2         -1.7895514   8.789252  19.36805
timey3         12.4415986  23.316037  34.19048
timey4          9.4752730  20.792687  32.11010
groupT:timey1  -0.9969201  12.272227  25.54137
groupT:timey2   2.7106980  16.425710  30.14072
groupT:timey3  -9.5979169   4.797929  19.19378
groupT:timey4  -7.4598679   7.594387  22.64864
attr(,"label")
[1] "Coefficients:"
```

6.3 F-tests

The `anova` method outputs the F-tests testing the overall effect of each variable:

```
anova(e.gls, type = "marginal")
```

```
Denom. DF: 437
            numDF   F-value p-value
(Intercept)     1 1158.4099  <.0001
group           1    1.2305  0.2679
time            4    5.9153  0.0001
group:time      4    1.6837  0.1527
```

6.4 Predicted response profiles

The `predict` method enables to compute the predicted means for each group. We first construct an auxiliary dataset containing the value of the group and time at which the predictions should be made:

```
df.grid <- expand.grid(group = c("C","T"),
                       time = c("y0","y1","y2","y3","y4"))
df.grid$time.num <- as.numeric(gsub("y","",df.grid$time))
df.grid
```

```
  group time time.num
1     C   y0        0
2     T   y0        0
3     C   y1        1
4     T   y1        1
```

```
5      C   y2        2
6      T   y2        2
7      C   y3        3
8      T   y3        3
9      C   y4        4
10     T   y4        4
```

Then we use the `predict` method:

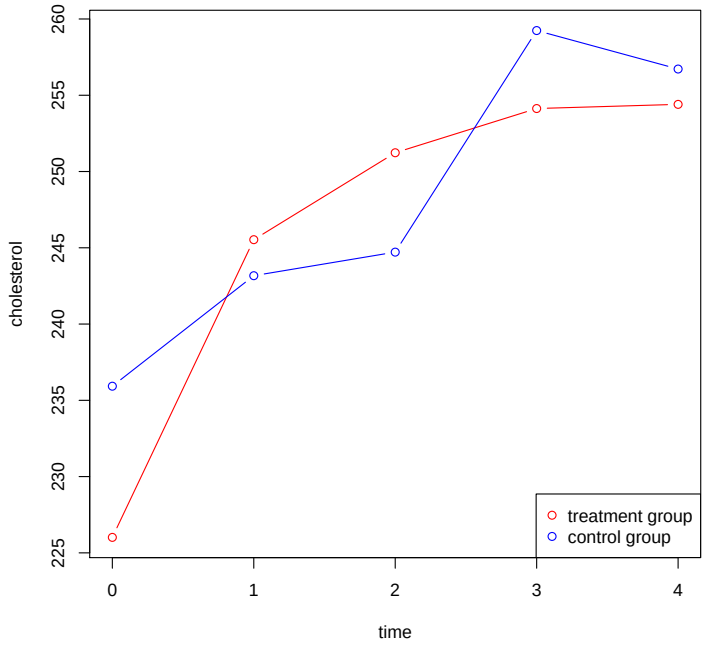
```
df.grid$response.profile <- predict(e.gls, newdata = df.grid)
```

Before displaying the response profiles we compute their range:

```
y.rangeProfile <- range(df.grid$response.profile)
```

The syntax to display the response profile with `plot` is the following:

```
plot(x = df.grid[df.grid$group=="T","time.num"],
     y = df.grid[df.grid$group=="T","response.profile"],
     col = "red", type = "b", ylim = y.rangeProfile,
     xlab = "time", ylab = "cholesterol")
points(x = df.grid[df.grid$group=="C","time.num"],
       y = df.grid[df.grid$group=="C","response.profile"],
       col = "blue", type = "b")
legend("bottomright", pch = 21, legend = c("treatment group","control group"),
       col = c("red","blue"))
```

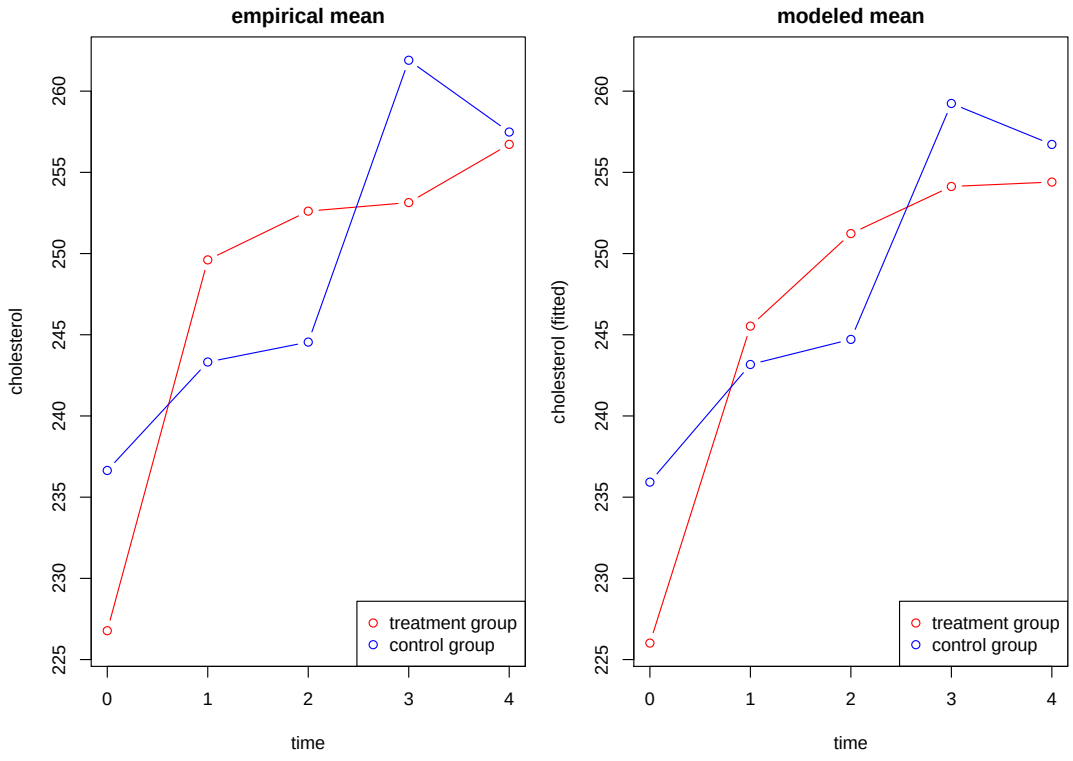


See appendix B for the syntax using `ggplot2`. We can also plot side by side the empircal mean and the response profiles using `par`:

```
y.rangeBoth <- range(c(y.rangeProfile,y.rangeObs))
par(mfrow = c(1,2), mar = c(4,2,2,1))

plot(x = 0:4, y = vec.meanT, col = "red", type = "b", ylim = y.rangeBoth,
     xlab = "time", ylab = "cholesterol", main = "empirical mean")
points(x = 0:4, y = vec.meanC, col = "blue", type = "b")
legend("bottomright", pch = 21, legend = c("treatment group","control group"),
       col = c("red","blue"))

plot(x = df.grid[df.grid$group=="T","time.num"],
     y = df.grid[df.grid$group=="T","response.profile"],
     col = "red", type = "b", ylim = y.rangeBoth,
     xlab = "time", ylab = "cholesterol (fitted)", main = "modeled mean")
points(x = df.grid[df.grid$group=="C","time.num"],
       y = df.grid[df.grid$group=="C","response.profile"], col = "blue", type = "b")
legend("bottomright", pch = 21, legend = c("treatment group","control group"),
       col = c("red","blue"))
```



7 Question 7: Mixed model (accounting for randomization)

7.1 Defining the treatment variable

To force the model to have the same fitted value at baseline for both group, we define the variable `treatment`. This variable should take value:

- `"none"` at baseline. This corresponds to the following observations:

```
index.baseline <- which(ncgsL$time=="y0")
```

- `"high dose"` in the treatment group after baseline. This corresponds to the following observations:

```
index.HD <- setdiff(which(ncgsL$group=="T"), index.baseline)
```

- `"placebo"` in the placebo group after baseline. This corresponds to the following observations:

```
index.Pl <- setdiff(which(ncgsL$group=="C"), index.baseline)
```

We can now use the indexes to define the variable treatment in our dataset:

```
ncgsL$treatment <- as.character(NA)
ncgsL[index.baseline,"treatment"] <- "none"
ncgsL[index.HD,"treatment"] <- "high dose"
ncgsL[index.Pl,"treatment"] <- "placebo"
```

and check that we did it correctly:

```
table(ncgsL$time,ncgsL$treatment)
```

```
   high dose none placebo
y0         0  103       0
y1        62    0      41
y2        62    0      41
y3        62    0      41
y4        62    0      41
```

7.2 Fitting the model

We can then call `gls` with this new variable:

```
e.glsConstrain <- try(gls(cholesterol ~ time*treatment,
                      correlation = corCompSymm(form =~ 1|id),
                      data = ncgsL, na.action = na.omit))
```

```
Error in glsEstimate(object, control = control) :
  computed "gls" fit is singular, rank 10
```

This returns an error since the design matrix is singular. This is because `gls` is creating interaction terms for each combination of time and treatment:

```
X <- model.matrix(cholesterol ~ time*treatment, data = ncgsL)
colnames(X)
```

```
 [1] "(Intercept)"             "timey1"
 [3] "timey2"                  "timey3"
 [5] "timey4"                  "treatmentnone"
 [7] "treatmentplacebo"        "timey1:treatmentnone"
 [9] "timey2:treatmentnone"    "timey3:treatmentnone"
[11] "timey4:treatmentnone"    "timey1:treatmentplacebo"
[13] "timey2:treatmentplacebo" "timey3:treatmentplacebo"
[15] "timey4:treatmentplacebo"
```

while we only need 4 interaction terms. We can force `gls` to guess what is the correct design matrix (i.e. drop one or more variables) setting the argument `control` to `glsControl(singular.ok = TRUE)`:

```
e.glsConstrain0 <- gls(cholesterol ~ time*treatment,
                       correlation = corCompSymm(form =~ 1|id),
                       data = ncgsL, na.action = na.omit,
                       control = glsControl(singular.ok = TRUE)
                       )
logLik(e.glsConstrain0)
```

```
'log Lik.' -2275.714 (df=17)
```

But this is not recommanded since in this example it gives an incorrect covariance parameters.

```
getVarCov(e.glsConstrain0, type = "marginal")
```

```
Marginal variance covariance matrix
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 2034.3    0.0    0.0    0.0    0.0
[2,]    0.0 2034.3    0.0    0.0    0.0
[3,]    0.0    0.0 2034.3    0.0    0.0
[4,]    0.0    0.0    0.0 2034.3    0.0
[5,]    0.0    0.0    0.0    0.0 2034.3
  Standard Deviations: 45.103 45.103 45.103 45.103 45.103
```

Instead we will define a new variable:

```
ncgsL$timeXtreatment <- "none"
ncgsL$timeXtreatment[index.HD] <- as.character(ncgsL$time)[index.HD]
ncgsL$timeXtreatment <- factor(ncgsL$timeXtreatment)
```

This variable equals "none" except after baseline in the treated group where it equals time (i.e. y1, y2, y3, or y4):

```
table(ncgsL$time,ncgsL$timeXtreatment,ncgsL$group)
```

```
, ,  = C


    none y1 y2 y3 y4
y0    41  0  0  0  0
y1    41  0  0  0  0
y2    41  0  0  0  0
y3    41  0  0  0  0
y4    41  0  0  0  0

, ,  = T


    none y1 y2 y3 y4
y0    62  0  0  0  0
y1     0 62  0  0  0
y2     0  0 62  0  0
y3     0  0  0 62  0
y4     0  0  0  0 62
```

```
ncgsL$timeXtreatment <- relevel(ncgsL$timeXtreatment,"none")
e.glsConstrain <- gls(cholesterol ~ time + timeXtreatment,
                      correlation = corCompSymm(form =~ 1|id),
                      data = ncgsL, na.action = na.omit
                      )
logLik(e.glsConstrain)
```

```
'log Lik.' -2149.588 (df=11)
```

7.3 Estimated mean change

We can now extract the estimated coefficients:

```
summary(e.glsConstrain)$tTable
```

```
                     Value Std.Error    t-value       p-value
(Intercept)      229.961165  4.376513 52.5443817 3.163007e-191
timey1             8.944961  5.008598  1.7859213  7.480369e-02
timey2            10.490296  5.159534  2.0331867  4.263611e-02
timey3            25.017435  5.316296  4.7058015  3.396545e-06
timey4            22.493696  5.550527  4.0525336  5.994834e-05
timeXtreatmenty1   9.446275  6.252044  1.5109098  1.315325e-01
timeXtreatmenty2  13.599643  6.496415  2.0934075  3.688734e-02
timeXtreatmenty3   1.971095  6.867241  0.2870286  7.742262e-01
timeXtreatmenty4   4.768369  7.223545  0.6601148  5.095271e-01
```

the confidence intervals:

```
intervals(e.glsConstrain)[["coef"]]
```

```
                      lower       est.     upper
(Intercept)      221.3595890 229.961165 238.56274
timey1            -0.8989111   8.944961  18.78883
timey2             0.3497745  10.490296  20.63082
timey3            14.5688135  25.017435  35.46606
timey4            11.5847191  22.493696  33.40267
timeXtreatmenty1  -2.8414606   9.446275  21.73401
timeXtreatmenty2   0.8316229  13.599643  26.36766
timeXtreatmenty3 -11.5257464   1.971095  15.46794
timeXtreatmenty4  -9.4287489   4.768369  18.96549
attr(,"label")
[1] "Coefficients:"
```

7.4 F-tests

```
anova(e.glsConstrain)
```

```
Denom. DF: 438
               numDF  F-value p-value
(Intercept)        1 3996.242  <.0001
time               4   17.563  <.0001
timeXtreatment     4    1.388  0.2373
```

7.5 Predicted response profiles

We can re-use the code shown in question 5 to compute and display the predicted values. We first add the variable `timeXtreatment` to the auxiliary dataset:

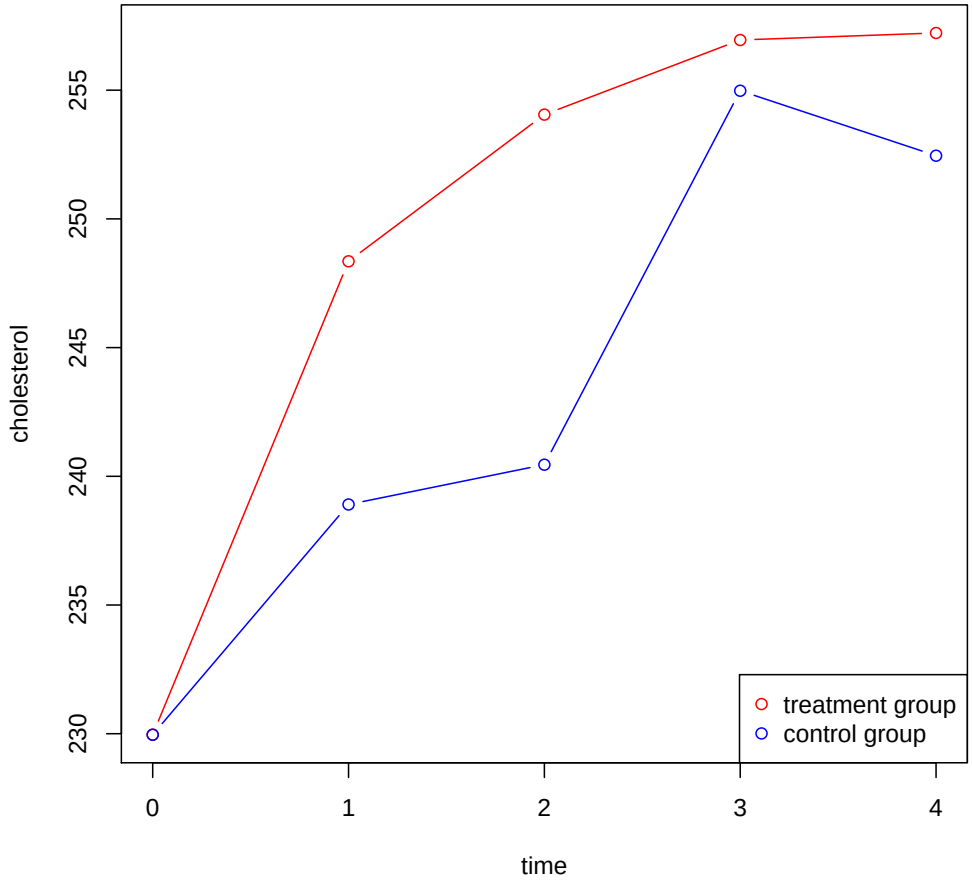
```
index.HD_grid <- which(df.grid$group=="T" & df.grid$time!="y0")
df.grid$timeXtreatment <- "none"
df.grid$timeXtreatment[index.HD_grid] <- as.character(df.grid$time)[index.HD_grid]
df.grid$timeXtreatment <- factor(df.grid$timeXtreatment)
```

and then call the `predict` method:

```
df.grid$response.profileConstrain <- predict(e.glsConstrain, newdata = df.grid)
```

to display the profiles:

```
y.rangeProfile <- range(df.grid$response.profileC)
plot(x = df.grid[df.grid$group=="T","time.num"],
     y = df.grid[df.grid$group=="T","response.profileConstrain"],
     col = "red", type = "b", ylim = y.rangeProfile,
     xlab = "time", ylab = "cholesterol")
points(x = df.grid[df.grid$group=="C","time.num"],
       y = df.grid[df.grid$group=="C","response.profileConstrain"],
       col = "blue", type = "b")
legend("bottomright", pch = 21, legend = c("treatment group","control group"),
       col = c("red","blue"))
```


cholesterol
230
235
240
245
250
255
0
1
2
3
4
time
treatment group
control group

A Using `data.table`

```
library(data.table)
```

Convert `ncgs` into a `data.table` object:

```
dt.ncgs <- as.data.table(ncgs)
```

A.1 Question 2

```
dt.ncgs[,.(y0 = mean(y0, na.rm = TRUE),
           y1 = mean(y1, na.rm = TRUE),
           y2 = mean(y2, na.rm = TRUE),
           y3 = mean(y3, na.rm = TRUE),
           y4 = mean(y4, na.rm = TRUE)),
        by = "group"]
```

```
   group       y0       y1       y2       y3       y4
1:     T 226.0161 245.5323 252.0182 256.7955 254.5526
2:     C 235.9268 243.1707 244.7632 257.6000 257.4839
```

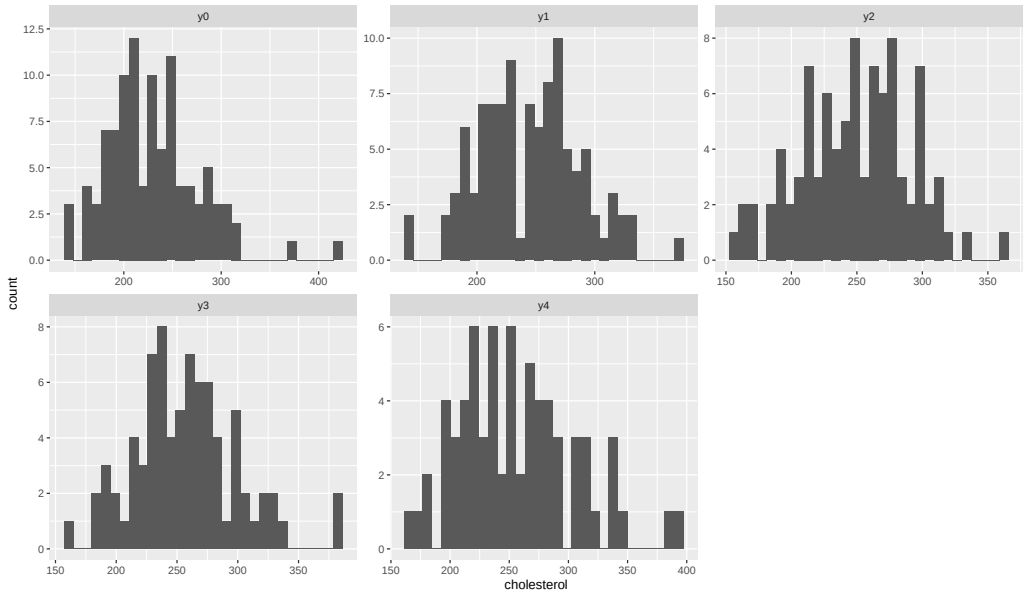
B Using ggplot2

```
library(ggplot2)
```

B.1 Question 2

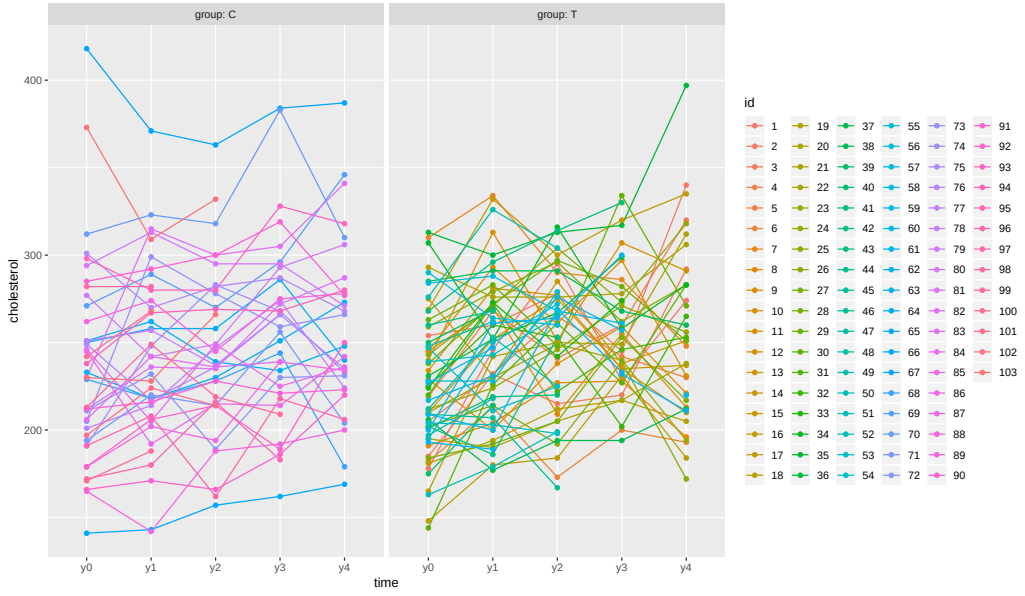
```
gg.hist <- ggplot(ncgsL, aes(x = cholesterol))
gg.hist <- gg.hist + geom_histogram()
gg.hist <- gg.hist + facet_wrap(~time, scales = "free")
gg.hist
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
Warning message:
Removed 68 rows containing non-finite values (stat_bin).
```



B.2 Question 3

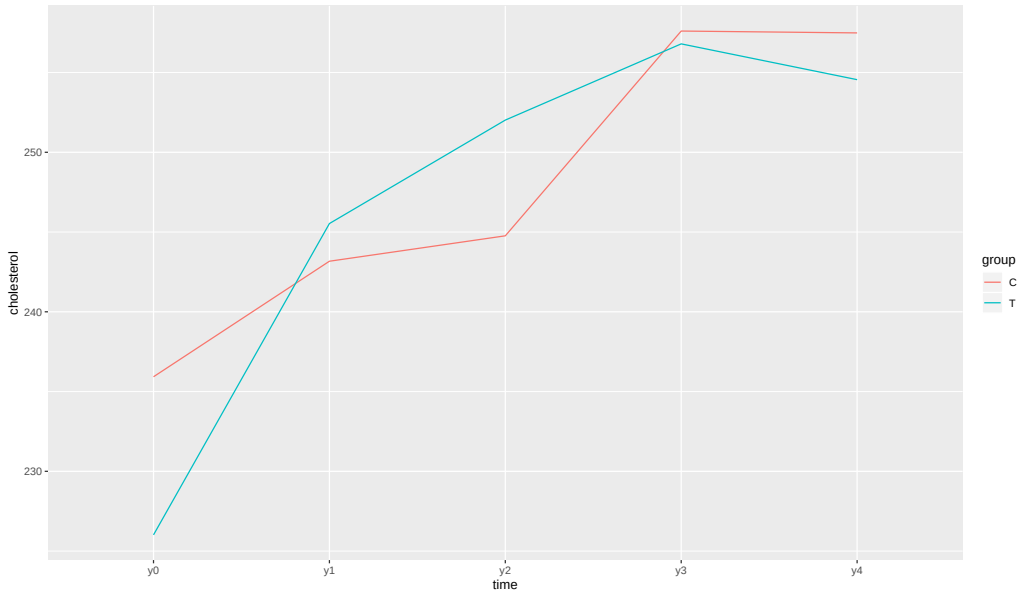
```
gg.spaguetti <- ggplot(ncgsL, aes(x = time, y = cholesterol, group = id, color = id))
gg.spaguetti <- gg.spaguetti + geom_line() + geom_point()
gg.spaguetti <- gg.spaguetti + facet_grid(~group, labeller = label_both)
gg.spaguetti
```



B.3 Question 5

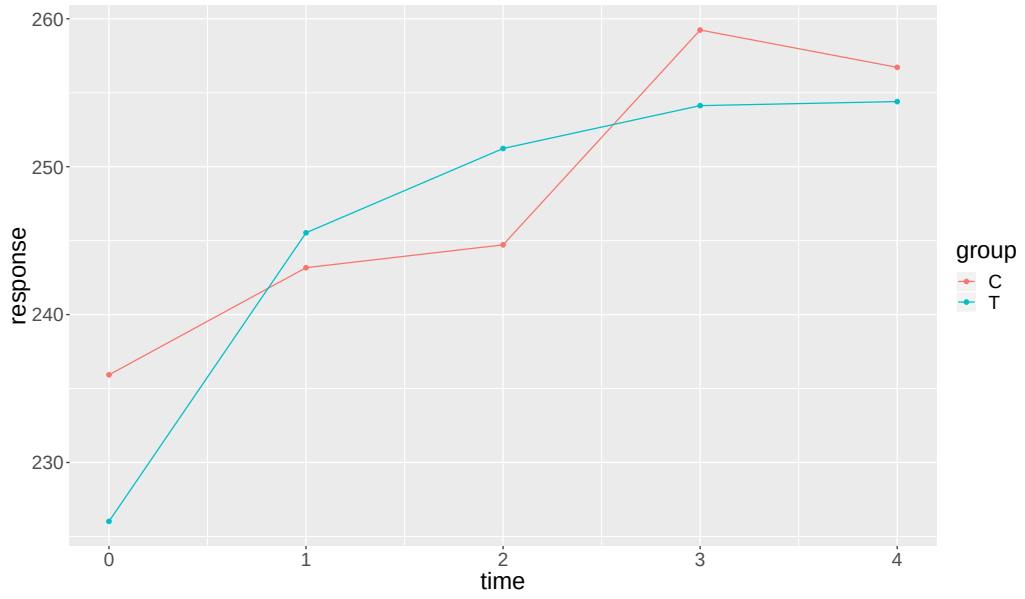
```
gg.mean <- ggplot(ncgsL, aes(x = time, y = cholesterol, group = group, color = group))
gg.mean <- gg.mean + stat_summary(geom = "line", fun.y = mean)
gg.mean
```

```
Warning message:
Removed 68 rows containing non-finite values (stat_summary).
```



B.4 Question 6

```
gg.predict <- ggplot(df.grid, aes(x = time.num, y = response.profile,
                                  group = group, color = group))
gg.predict <- gg.predict + geom_point() + geom_line()
gg.predict <- gg.predict + xlab("time") + ylab("response")
gg.predict
```



C Mixed model with unstructure covariance matrix

When answering question 6, we assumed a constant variance of the cholesterol measurements at all timepoints (diagonal terms) as well as a constant correlation between timepoints (extra-diagonal terms):

```
getVarCov(e.gls, individual = 1)
```

```
Marginal variance covariance matrix
       [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 1970.0 1407.6 1407.6 1407.6 1407.6
[2,] 1407.6 1970.0 1407.6 1407.6 1407.6
[3,] 1407.6 1407.6 1970.0 1407.6 1407.6
[4,] 1407.6 1407.6 1407.6 1970.0 1407.6
[5,] 1407.6 1407.6 1407.6 1407.6 1970.0
  Standard Deviations: 44.385 44.385 44.385 44.385 44.385
```

To relax these assumptions we will use `weight` argument and specify an unstructured correlation matrix in the `correlation` argument:

```
e.glsUN0 <- gls(cholesterol ~ group*time,
                correlation = corSymm(form= ~1|id),
                weights = varIdent(form= ~1|time),
                data = ncgsL, na.action = na.omit)
cov2cor(getVarCov(e.glsUN0, individual = 1))
```

```
Marginal variance covariance matrix
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,] 1.00000 0.77040 0.73176 0.73792 0.58586
[2,] 0.77040 1.00000 0.77348 0.79961 0.66514
[3,] 0.73176 0.77348 1.00000 0.72649 0.67778
[4,] 0.73792 0.79961 0.72649 1.00000 0.62490
[5,] 0.58586 0.66514 0.67778 0.62490 1.00000
  Standard Deviations: 1 1 1 1 1
```

Unfortunately, due to the presence of missing values the model is misspecified. To see that consider the individual 61:

```
ncgsL[ncgsL$id==61,]
```

```
    id group time cholesterol treatment timeXtreatment
61  61     T   y0         227      none           none
164 61     T   y1         247 high dose             y1
267 61     T   y2          NA high dose             y2
370 61     T   y3          NA high dose             y3
473 61     T   y4         220 high dose             y4
```

This individual has only measurements at time 0,1, and 4. But when we display the variance-covariance matrix between the cholesterol measurements for this individual we see:

```
cov2cor(getVarCov(e.glsUN0, individual = 61))
```

```
Marginal variance covariance matrix
        [,1]    [,2]    [,3]
[1,] 1.00000 0.77040 0.73176
[2,] 0.77040 1.00000 0.77348
[3,] 0.73176 0.77348 1.00000
  Standard Deviations: 1 1 1
```

It appears that `gls` has attributed the correlation coefficients of time 0, 1, and 2 which is incorrect. This is because we have not specified in the argument `correlation` at which time which observation was measured (by default `gls` assumes chronological order). To solve that we define a variable `time.num` indexing the times:

```
ncgsL$time.num <- as.numeric(ncgsL$time)

e.glsUN <- gls(cholesterol ~ group*time,
               correlation = corSymm(form= ~time.num|id),
               weights = varIdent(form= ~1|time),
               data = ncgsL, na.action = na.omit)

logLik(e.glsUN)
```

```
'log Lik.' -2132.54 (df=25)
```

We now check the variance-covariance matrices for two individuals:

```
list("1" = cov2cor(getVarCov(e.glsUN, individual = 1)),
     "61" = cov2cor(getVarCov(e.glsUN, individual = 61)))
```

```
$`1`
Marginal variance covariance matrix
        [,1]    [,2]    [,3]    [,4]    [,5]
[1,] 1.00000 0.77022 0.73158 0.73712 0.58863
[2,] 0.77022 1.00000 0.77530 0.79643 0.66944
[3,] 0.73158 0.77530 1.00000 0.72504 0.67985
[4,] 0.73712 0.79643 0.72504 1.00000 0.62609
[5,] 0.58863 0.66944 0.67985 0.62609 1.00000
  Standard Deviations: 1 1 1 1 1

$`61`
Marginal variance covariance matrix
        [,1]    [,2]    [,3]
[1,] 1.00000 0.77022 0.58863
[2,] 0.77022 1.00000 0.66944
[3,] 0.58863 0.66944 1.00000
  Standard Deviations: 1 1 1
```

This looks better! We now get the following estimates:

```
summary(e.glsUN)$tTable
```

```
                  Value Std.Error    t-value       p-value
(Intercept)   235.926829  7.302781 32.3064341 7.157418e-118
groupT         -9.910700  9.412632 -1.0529149  2.929618e-01
timey1          7.243902  4.805468  1.5074291  1.324232e-01
timey2          8.848318  5.207268  1.6992246  8.998854e-02
timey3         23.102788  5.297447  4.3611170  1.615699e-05
timey4         21.123766  7.369940  2.8662059  4.354883e-03
groupT:timey1  12.272227  6.193818  1.9813669  4.817644e-02
groupT:timey2  16.417543  6.743418  2.4346027  1.530749e-02
groupT:timey3   4.976989  6.982010  0.7128304  4.763312e-01
groupT:timey4   6.903112  9.791182  0.7050336  4.811650e-01
```