

Fitting linear mixed models in R

Brice Ozenne

Contents

1 Packages	3
2 Illustrative dataset	4
3 Notations	6
4 Recall on Gaussian linear mixed models	8
4.1 Distributional assumption	8
4.2 Linearity	9
4.3 Independence between the mean and the variance	10
5 Using the <code>gls</code> function of the <code>nlme</code> package	11
5.1 Short presentation of the <code>gls</code> function	11
5.2 Specifying the mean structure	12
5.2.1 Without interaction	12
5.2.2 Interactions	15
5.2.3 Baseline adjustment with an interaction	17
5.3 Specifying the variance-covariance matrix	20
5.3.1 Illustration - compound symmetry structure	22
5.3.2 Illustration - unstructured covariance matrix	24
5.4 Technical point: why using <code>as.numeric(time)</code> in <code>corSymm</code>	26
5.5 Prediction	29
5.6 Other methods	32
6 Using the <code>lme</code> function of the <code>nlme</code> package	33
6.1 Short presentation of the <code>lme</code> function	33
6.2 Specifying the mean structure	34
6.3 Specifying the variance-covariance matrix	35
6.3.1 Compound symmetry structure with <code>lme</code>	35
6.3.2 Unstructured covariance matrix with <code>lme</code>	36
6.4 Note on <code>lmer</code> from the <code>lme4</code> package	39
6.4.1 Compound symmetry structure	39
6.4.2 Unstructured residual variance-covariance matrix	40

7	Handling missing values in the response variable	42
7.1	Illustration with <code>gls</code>	42
8	Inference	44
8.1	Univariate Wald tests: model parameters	45
8.2	Univariate Wald tests: linear combination of parameters	46
8.3	Multivariate Wald test (also called F-test)	48
8.4	Inference in small samples	49
8.4.1	Kenward Roger like approximation when using <code>gls</code>	49
8.4.2	Kenward Roger approximation when using <code>lme4</code>	51
8.4.3	Comparisons between <code>lavaSearch2</code> and <code>lmerTest</code>	52
9	References	55
A	Sequential vs. marginal anova	56
A.1	Sequential anova (or type I anova)	56
A.2	Marginal anova without interaction (type II anova)	57
A.3	Marginal anova with interaction (type III anova)	57

1 Packages

Several packages can be used in R to fit mixed models. In this course we will use two packages:

- the **nlme** package: it enables to specify the form of the correlation structure between residuals, to model a potential heteroscedasticity and to consider random effects. It is limited to Gaussian variables but can handle non-linear relationships (e.g. $Y \sim \exp(\beta x)$). In the course we will only consider linear models and mainly use the `gls` function of the nlme package.

References: Pinheiro and Bates (2000), Galecki and Burzykowski (2013)

- the **lme4** package: it is a numerically more efficient alternative to nlme which is recommended for large datasets or when several random effects are considered. Contrary to nlme, the correlation structure between residuals can only be model through random effects. No option for dealing with heteroscedasticity other than using random effects (e.g. random slope). However lme4 enables to model non-Gaussian dependant variables (e.g. binary, Poisson, ...). In the course we will use the `glmer` function of the lme4 package to fit logistic mixed models.

Reference: Bates (2010)

For a broader overview of the available R packages for fitting mixed models see <http://bbolker.github.io/mixedmodels-misc/glmmFAQ.html> (which also includes discussions on the practical use of mixed models) and <http://glmm.wikidot.com/pkg-comparison>.

2 Illustrative dataset

We will use the dataset of the first practical class to illustrate some of the functionalities of the `nlme` package. We import the dataset using (it requires an internet connection):

```
path <- "https://content.sph.harvard.edu/fitzmaur/ala2e/cholesterol-
        data.txt"
dfW.data <- read.table(path, na.string = ".")
```

Before displaying the content of the data, we add names to the columns of the dataset and convert the `group` and `id` into factor variables:

```
colnames(dfW.data) <- c("group", "id", "y0", "y1", "y2", "y3", "y4")
dfW.data$group <- factor(dfW.data$group,
                           levels = 1:2, labels = c("T", "C"))
dfW.data$group <- relevel(dfW.data$group, ref = "C")
dfW.data$id <- as.factor(dfW.data$id)

str(dfW.data)
```

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

The dataset contains a total of 103 patients, divided in two groups: a control group and a treatment group, for which the level of cholesterol is measured at inclusion and after 6, 12, 20, and 24 months. A treatment is given to the patients in the treatment group shortly after inclusion while the patients in the control group receive a placebo.

To be able to use the `gls` function we will convert the dataset into the long format. First we load the package `reshape2`:

```
library(reshape2)
```

and then use `melt`:

```
dfL.data <- melt(dfW.data, id.vars = c("group", "id"),
                  value.name = "cholesterol", variable.name = "time")
dfL.data$time <- as.factor(gsub("y", "visit", dfL.data$time))
dfL.data$time <- relevel(dfL.data$time, ref = "visit0")
```

and order it by individual and time:

```
dfL.data <- dfL.data[order(dfL.data$id,dfL.data$time),]  
str(dfL.data)
```

```
'data.frame': 515 obs. of 4 variables:  
$ group : Factor w/ 2 levels "C","T": 2 2 2 2 2 2 2 2 2 ...  
$ id : Factor w/ 103 levels "1","2","3","4",...: 1 1 1 1 1 2 2 2 2 ...  
$ time : Factor w/ 5 levels "visit0","visit1",...: 1 2 3 4 5 1 2 3 4 5 ...  
$ cholesterol: int 178 246 295 228 274 254 260 278 245 340 ...
```

Finally we will create a treatment variable, that is equivalent to the group variable except it takes value "none" at baseline:

```
dfL.data$treatment <- as.character(dfL.data$group)  
dfL.data[dfL.data$time == "visit0", "treatment"] <- "none"  
dfL.data$treatment <- factor(dfL.data$treatment,  
                               levels = c("none", "C", "T"),  
                               labels = c("none", "pl", "tr"))
```

Indeed:

```
table(dfL.data$treatment, dfL.data$time, dfL.data$group)
```

, , = C

	visit0	visit1	visit2	visit3	visit4
none	41	0	0	0	0
pl	0	41	41	41	41
tr	0	0	0	0	0

, , = T

	visit0	visit1	visit2	visit3	visit4
none	62	0	0	0	0
pl	0	0	0	0	0
tr	0	62	62	62	62

3 Notations

We consider

- a set of individuals, indexed by $i \in \{1, \dots, n\}$ e.g. $n = 103$
- a set of measurement times, indexed by $j \in \{1, \dots, T\}$ e.g. $T = 5$
- a response variable Y e.g. the cholesterol level
- a set of explanatory variables X e.g. the treatment, the time

Since X and Y are measured for several individuals and at several times, we will denote by Y_{ij} (X_{ij}) the value of the response variable (resp. value of the explanatory variables) at the $j - th$ time for the $i - th$ individual. For instance in our example Y_{11} equals:

```
dfL.data[dfL.data$id=="1" & dfL.data$time == "visit0",
         "cholesterol"]
```

[1] 178

and X_{23} equals:

```
dfL.data[dfL.data$id=="2" & dfL.data$time == "visit3",
         c("group","time")]
```

```
group   time
311     T visit3
```

To shorten the notations, we will define for a given individual i :

- the vector of responses $\mathbf{Y}_i = (Y_{i1}, Y_{i2}, Y_{i3}, Y_{i4}, Y_{i5})$. For instance in our example \mathbf{Y}_1 equals:

```
dfL.data[dfL.data$id=="1", "cholesterol"]
```

[1] 178 246 295 228 274

- the vector of explanatory variables $\mathbf{X}_i = (X_{i1}, X_{i2}, X_{i3}, X_{i4}, X_{i5})$ (we say vector here even though in practice it is a matrix). For instance in our example \mathbf{X}_1 equals:

```
dfL.data[dfL.data$id=="2", c("group", "time")]
```

```
group   time
2       T visit0
105    T visit1
208    T visit2
311    T visit3
414    T visit4
```

Finally we will note by \mathcal{X}_i the design matrix for individual i . The design matrix is very similar to the vector of covariates except that discrete variables (i.e. character or factor variables in **R**) are converted into binary indicators (denoted by $\mathbb{1}_.$ in the following). For instance compare \mathbf{X}_1 to \mathcal{X}_1 :

```
model.matrix(~ group + time, data = dfL.data[dfL.data$id=="1",])
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4
1           1     1       0       0       0       0
104          1     1       1       0       0       0
207          1     1       0       1       0       0
310          1     1       0       0       1       0
413          1     1       0       0       0       1
attr("assign")
[1] 0 1 2 2 2 2
attr("contrasts")
attr("contrasts")$group
[1] "contr.treatment"

attr("contrasts")$time
[1] "contr.treatment"
```

The column `groupT` is the indicator of whether `group` variable for individual i at the $j - th$ time equals T and will be written $\mathbb{1}_{group_{ij}=T}$. The column `timevisit2` is the indicator of whether `time` equals `visit2` and will be written $\mathbb{1}_{time_{ij}=visit2}$. So in this example the design matrix can be written in a vector form as:

$$\mathcal{X}_i = \left(\mathbf{1}, \mathbb{1}_{group_{ij}=T}, \mathbb{1}_{time_{ij}=visit1}, \mathbb{1}_{time_{ij}=visit2}, \mathbb{1}_{time_{ij}=visit3}, \mathbb{1}_{time_{ij}=visit4} \right)$$

where $\mathbf{1}$ denotes a vector of ones. Note that by default **R** adds an intercept, i.e. a column of 1 to the design matrix (it can be removed by adding `-1` or `0` in the formula).

4 Recall on Gaussian linear mixed models

A Gaussian linear mixed model (LMM) models the relationship between X and Y under three assumptions. These assumptions could be relaxed using more "advanced" LMMs but we won't discuss them in this document. We also only consider here the case of discrete observation times.

4.1 Distributional assumption

In LMM the vector of response variables of any individual **follows a multivariate normal distribution** whose moments (mean μ , variance Σ) depends on the vector of explanatory variable \mathbf{X}_i of the individual:

$$\begin{bmatrix} Y_{i1} \\ Y_{i2} \\ Y_{i3} \\ Y_{i4} \\ Y_{i5} \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \mu_1(X_{i1}) \\ \mu_2(X_{i2}) \\ \mu_3(X_{i3}) \\ \mu_4(X_{i4}) \\ \mu_5(X_{i5}) \end{bmatrix}, \begin{bmatrix} \sigma_{11}(X_{i1}) & \sigma_{12}(X_{i1}, X_{i2}) & \sigma_{13}(X_{i1}, X_{i3}) & \sigma_{14}(X_{i1}, X_{i4}) & \sigma_{15}(X_{i1}, X_{i5}) \\ \sigma_{12}(X_{i1}, X_{i2}) & \sigma_{22}(X_{i2}) & \sigma_{23}(X_{i2}, X_{i3}) & \sigma_{24}(X_{i2}, X_{i4}) & \sigma_{25}(X_{i2}, X_{i5}) \\ \sigma_{13}(X_{i1}, X_{i3}) & \sigma_{23}(X_{i2}, X_{i3}) & \sigma_{33}(X_{i3}) & \sigma_{34}(X_{i3}, X_{i4}) & \sigma_{35}(X_{i3}, X_{i5}) \\ \sigma_{14}(X_{i1}, X_{i4}) & \sigma_{24}(X_{i2}, X_{i4}) & \sigma_{34}(X_{i3}, X_{i4}) & \sigma_{44}(X_{i4}) & \sigma_{45}(X_{i4}, X_{i5}) \\ \sigma_{15}(X_{i1}, X_{i5}) & \sigma_{25}(X_{i2}, X_{i5}) & \sigma_{35}(X_{i3}, X_{i5}) & \sigma_{45}(X_{i4}, X_{i5}) & \sigma_{55}(X_{i5}) \end{bmatrix} \right)$$

or in a more concise notation:

$$\mathbf{Y}_i \sim \mathcal{N}_T(\mu(\mathbf{X}_i), \Sigma(\mathbf{X}_i))$$

REMARK 1: (special case) Gaussian linear model (function `lm` in **R**) :
Gaussian linear model are special cases of LMM. For instance the model:

```
e.lm <- lm(cholesterol ~ time, data = dfL.data, na.action = na.exclude)
```

can be written:

$$\begin{bmatrix} Y_{i1} \\ Y_{i2} \\ Y_{i3} \\ Y_{i4} \\ Y_{i5} \end{bmatrix} \sim \mathcal{N} \left(\begin{bmatrix} \alpha \\ \alpha + \beta_1 \\ \alpha + \beta_2 \\ \alpha + \beta_3 \\ \alpha + \beta_4 \end{bmatrix}, \begin{bmatrix} \sigma^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2 \end{bmatrix} \right)$$

where $\alpha, \beta_1, \beta_2, \beta_3, \beta_4, \sigma^2$ equal:

```
c(coef(e.lm), sigma = sigma(e.lm)^2)
```

```
(Intercept) timevisit1 timevisit2 timevisit3 timevisit4 sigma
229.96117    14.63107   19.09260   27.19073   25.90840 1991.60962
```

4.2 Linearity

In LMM, the mean $\mu(\mathbf{X}_i)$ is constrained to be a **linear function** of the model coefficients:

$$\mu(\mathbf{X}_i) = \mathcal{X}_i\beta$$

We recall that here \mathcal{X}_i denotes the design matrix for individual i (and not the vector of explanatory variables that is denoted by \mathbf{X}_i , even though in practice they are similar - but not identical).

REMARK 1: (continued) :

We have seen previously that in the model:

```
e.lm <- lm(cholesterol ~ time, data = dfL.data, na.action = na.exclude)
```

the modeled mean for any individual equals α at the first time (i.e. $\mu_1(X_{i1}) = \alpha$), $\alpha + \beta_1$ at the second time ($\mu_2(X_{i2}) = \alpha + \beta_1$), $\alpha + \beta_2$ at the third time ($\mu_3(X_{i3}) = \alpha + \beta_2$), $\alpha + \beta_3$ at the fourth time (i.e. $\mu_4(X_{i4}) = \alpha + \beta_3$), and $\alpha + \beta_4$ at the fifth time (i.e. $\mu_5(X_{i5}) = \alpha + \beta_4$). So we expressed the modeled mean for individual i at the j -th time as:

$$\mu_j(X_{ij}) = \alpha + \beta_2 \mathbb{1}_{time_{ij}=visit1} + \beta_3 \mathbb{1}_{time_{ij}=visit2} + \beta_4 \mathbb{1}_{time_{ij}=visit3} + \beta_5 \mathbb{1}_{time_{ij}=visit4}$$

which can be shorten into:

$$\mu_j(X_{ij}) = \mathcal{X}_{ij}\beta$$

where $\beta = (\alpha, \beta_1, \beta_2, \beta_3, \beta_4)$ and $\mathcal{X}_{ij} = (1, \mathbb{1}_{time_{ij}=visit1}, \mathbb{1}_{time_{ij}=visit2}, \mathbb{1}_{time_{ij}=visit3}, \mathbb{1}_{time_{ij}=visit4})$. Here $\mathcal{X}_{ij}\beta$ implicitly denotes the vector product between \mathcal{X}_{ij} and β . This corresponds in **R** to the matrix product:

```
Xb <- model.matrix(~time, data = dfL.data) %*% coef(e.lm)
```

Indeed:

```
mu <- fitted(e.lm)
range(mu-Xb, na.rm = TRUE)
```

```
[1] 1.136868e-13 4.376943e-12
```

REMARK 2: the first and second assumptions imply that the model can be written:

$$\mathbf{Y}_i = \mathcal{X}_i\beta + \boldsymbol{\varepsilon}_i, \text{ where } \boldsymbol{\varepsilon}_i \sim \mathcal{N}(0, \Sigma(\mathbf{X}_i))$$

4.3 Independence between the mean and the variance

In LMM, the variance may also depends on the the explanatory variables in a possibly non linear way:

$$\Sigma(\mathbf{X}_i) = f(\mathbf{X}_i, \gamma)$$

where γ denotes a set of parameters that **are different** from β . f denotes an arbitrary function that ensures that Σ is positive definite. We usually refer to:

- β as the mean parameters.
- γ as the variance parameters.
- f as the covariance structure.

Equivalently when we wrote:

$$\Sigma(\mathbf{X}_i) = \begin{bmatrix} \sigma_{11}(X_{i1}) & \sigma_{12}(X_{i1}, X_{i2}) & \sigma_{13}(X_{i1}, X_{i3}) & \sigma_{14}(X_{i1}, X_{i4}) & \sigma_{15}(X_{i1}, X_{i5}) \\ \sigma_{12}(X_{i1}, X_{i2}) & \sigma_{22}(X_{i2}) & \sigma_{23}(X_{i2}, X_{i3}) & \sigma_{24}(X_{i2}, X_{i4}) & \sigma_{25}(X_{i2}, X_{i5}) \\ \sigma_{13}(X_{i1}, X_{i3}) & \sigma_{23}(X_{i2}, X_{i3}) & \sigma_{33}(X_{i3}) & \sigma_{34}(X_{i3}, X_{i4}) & \sigma_{35}(X_{i3}, X_{i5}) \\ \sigma_{14}(X_{i1}, X_{i4}) & \sigma_{24}(X_{i2}, X_{i4}) & \sigma_{34}(X_{i3}, X_{i4}) & \sigma_{44}(X_{i4}) & \sigma_{45}(X_{i4}, X_{i5}) \\ \sigma_{15}(X_{i1}, X_{i5}) & \sigma_{25}(X_{i2}, X_{i5}) & \sigma_{35}(X_{i3}, X_{i5}) & \sigma_{45}(X_{i4}, X_{i5}) & \sigma_{55}(X_{i5}) \end{bmatrix}$$

we can see that we decompose f in many scalar functions σ_{ij} that specify the variance (diagonal term) and the covariance (off diagonal terms).

REMARK 1: (continued) :

In the model:

```
e.lm <- lm(cholesterol ~ time, data = dfL.data, na.action = na.exclude)
```

σ_{ij} equals 0 when $i \neq j$ and equals a constant (usually denoted σ^2) when $i = j$. This gives:

$$\Sigma(\mathbf{X}_i) = \begin{bmatrix} \sigma^2 & 0 & 0 & 0 & 0 \\ 0 & \sigma^2 & 0 & 0 & 0 \\ 0 & 0 & \sigma^2 & 0 & 0 \\ 0 & 0 & 0 & \sigma^2 & 0 \\ 0 & 0 & 0 & 0 & \sigma^2 \end{bmatrix}$$

5 Using the `gls` function of the `nlme` package

5.1 Short presentation of the `gls` function

To access the `gls` function we first need to load the `nlme` package:

```
library(nlme)
```

Then we can have a look to the arguments of the `gls` function:

```
args(gls)
```

```
function (model, data = sys.frame(sys.parent()), correlation = NULL,
         weights = NULL, subset, method = c("REML", "ML"), na.action = na.fail,
         control = list(), verbose = FALSE)
NULL
```

In this class we will use the arguments:

- `data`: to specify the dataset we are using.
- `model`: a formula, e.g. `cholesterol ~ group` where
 - the left hand side (here `cholesterol`) indicates the response variable \mathbf{Y}_i
 - the right hand side (here `group`) defines the model for the mean i.e. in $\mu(\mathbf{X}_i) = \beta \mathcal{X}_i$ what are the explanatory variables that we are using (by default **R** includes an intercept).

In our example, the model would be $\mu(\mathbf{X}_i) = \alpha + \beta_G \mathbf{1}_{group_i=T}$ where $\mathbf{1}_{group=T}$ is binary variable taking value 1 if the individual i belongs to the treatment group and 0 otherwise.

- `correlation`, `weights`: defines the covariance structure, i.e. f .
- `method` defines which objective function should maximized in order to estimate the model parameters (β, γ) .
- `na.action` indicates how to deal with missing values.

5.2 Specifying the mean structure

In this section we focus on the model for the mean and, to simplify the syntax, will not specify any model for the variance. This is in general wrong, but the purpose of the section is not to discuss the modeling strategy but only to explain the meaning of the estimated mean parameters (denoted $\hat{\beta}$). To simplify we also only consider categorical explanatory variables.

5.2.1 Without interaction

We first consider a model without interaction:

```
e.gls <- gls(model = cholesterol ~ group + time,
               data = dfL.data,
               na.action = na.omit)
```

and we obtain the following estimated mean parameters:

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

	Value	Std.Error	t-value	p-value
(Intercept)	230.4553567	5.103917	45.1526449	1.791075e-167
groupT	-0.8209958	4.291058	-0.1913271	8.483574e-01
timevisit1	14.6310680	6.225472	2.3501940	1.920343e-02
timevisit2	19.0839419	6.390793	2.9861618	2.982377e-03
timevisit3	27.1538055	6.684372	4.0622825	5.752077e-05
timevisit4	25.8663511	6.953669	3.7198134	2.251071e-04

The column **Value** contains the estimated coefficients. To interpret them we first write down the model for individual i at visit j :

$$Y_{ij} = \alpha + \beta_G \mathbf{1}_{group_i=T} + \beta_{v1} \mathbf{1}_{j=1} + \beta_{v2} \mathbf{1}_{j=2} + \beta_{v3} \mathbf{1}_{j=3} + \beta_{v4} \mathbf{1}_{j=4} + \varepsilon_{ij} \quad (1)$$

$$\varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2) \quad (2)$$

where $\mathbf{1}_{group=T}$ is the indicator variable taking value 1 when the individual is in the treatment group and 0 otherwise. $\mathbf{1}_{visit=1}$ is the indicator variable taking value 1 for the first visit and 0 otherwise. And so on for $\mathbf{1}_{visit=2} \dots \mathbf{1}_{visit=4}$. Note that the design matrix is a matrix whose i -th line combines one (for the intercept) with all the indicator variables for individual i :

$$X_i = [1, \mathbf{1}_{group_i=T}, \mathbf{1}_{j=1}, \mathbf{1}_{j=2}, \mathbf{1}_{j=3}, \mathbf{1}_{j=4}]$$

We can obtain the design matrix using the function `model.matrix` in **R**. For instance for the four observations:

```
dfL.data[c(1:2,396:397),]
```

	group	id	time	cholesterol	treatment
1	T	1	visit0	178	none
104	T	1	visit1	246	tr
80	C	80	visit0	294	none
183	C	80	visit1	313	pl

we get the following design matrix:

```
X <- model.matrix(~ group + time, data = dfL.data[c(1:2,396:397),])
X
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4
1           1     1       0       0       0       0
104         1     1       1       0       0       0
80          1     0       0       0       0       0
183         1     0       1       0       0       0
attr("assign")
[1] 0 1 2 2 2 2
attr("contrasts")
attr("contrasts")$group
[1] "contr.treatment"

attr("contrasts")$time
[1] "contr.treatment"
```

Imagine we are interested in the treatment effect at baseline. Under some assumptions (e.g. no unmeasured confounders), this effect can be written in mathematical term as:

$$\mathbb{E}[Y_{ij} | \text{group} = T, \text{visit} = 0] - \mathbb{E}[Y_{ij} | \text{group} = C, \text{visit} = 0]$$

This is very convenient because if we can express each term as a function of the model parameters, we then know which parameter or combination of parameters to test. Recalling that the residuals have mean 0 (from (2)), i.e. $\mathbb{E}[\varepsilon_{ij}] = 0$, and are independent of the covariates, we get $\mathbb{E}[\varepsilon_{ij} | \text{group} = T, \text{visit} = 0] = \mathbb{E}[\varepsilon_{ij} | \text{group} = C, \text{visit} = 0] = 0$. Applying the conditional expectation to equation (1) (i.e. removing the residual

term and replacing the indicator variable by their value), we get:

$$\begin{aligned}\mathbb{E}[Y_{ij} | group = T, visit = 0] &= \alpha + \beta_G * 1 + \beta_{v1} * 0 + \beta_{v2} * 0 + \beta_{v3} * 0 + \beta_{v4} * 0 + 0 \\ &= \alpha + \beta_G \\ \mathbb{E}[Y_{ij} | group = C, visit = 0] &= \alpha + \beta_G * 0 + \beta_{v1} * 0 + \beta_{v2} * 0 + \beta_{v3} * 0 + \beta_{v4} * 0 + 0 \\ &= \alpha\end{aligned}$$

So, as we could have expected, the treatment effect equals:

$$\mathbb{E}[Y_{ij} | group = T, visit = 0] - \mathbb{E}[Y_{ij} | group = C, visit = 0] = \alpha + \beta_G - \alpha = \beta_G$$

We can also see that from the contrast matrix, by subtracting the row corresponding to the patient from the control group at baseline to the row corresponding to the patient from the treatment group at baseline:

```
X[1,]-X[3,]
```

(Intercept)	groupT	timevisit1	timevisit2	timevisit3	timevisit4
0	1	0	0	0	0

We see that the difference is only for the indicator $\mathbf{1}_{group_i=T}$. So when doing the matrix product with $\hat{\beta}$ (i.e. $\mathcal{X}_i\beta = \sum_{k=1}^p \mathcal{X}_{ip}\beta_p$):

```
coef(e.gls)
```

(Intercept)	groupT	timevisit1	timevisit2	timevisit3	timevisit4
230.4553567	-0.8209958	14.6310680	19.0839419	27.1538055	25.8663511

we obtain:

```
sum( (X[1,]-X[3,]) * coef(e.gls) )
```

```
[1] -0.8209958
```

which is exactly the estimated β_G .

5.2.2 Interactions

We can also apply this approach in presence of interaction. We consider the following model:

```
e.glsI <- gls(model = cholesterol ~ group * time,
               data = dfL.data,
               na.action = na.omit)
```

and we obtain the following estimated mean parameters (with 4 more parameters compared to the case without interaction):

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

	Value	Std.Error	t-value	p-value
(Intercept)	235.926829	6.993836	33.7335369	9.861822e-124
groupT	-9.910700	9.014429	-1.0994263	2.721875e-01
timevisit1	7.243902	9.890778	0.7323896	4.643235e-01
timevisit2	8.836329	10.084101	0.8762634	3.813683e-01
timevisit3	21.673171	10.305955	2.1029756	3.603917e-02
timevisit4	21.557042	10.658617	2.0224990	4.373343e-02
groupT:timevisit1	12.272227	12.748328	0.9626538	3.362536e-01
groupT:timevisit2	17.165724	13.057498	1.3146258	1.893252e-01
groupT:timevisit3	9.106155	13.569731	0.6710638	5.025344e-01
groupT:timevisit4	6.979461	14.097080	0.4950998	6.207785e-01

We now write down the model we have fitted for individual i at visit j :

$$Y_{ij} = \alpha + \beta_G \mathbf{1}_{group_i=T} + \beta_{v1} \mathbf{1}_{j=1} + \beta_{v2} \mathbf{1}_{j=2} + \beta_{v3} \mathbf{1}_{j=3} + \beta_{v4} \mathbf{1}_{j=4} \\ + \beta_{I1} \mathbf{1}_{group_i=T} \mathbf{1}_{j=1} + \beta_{I2} \mathbf{1}_{group_i=T} \mathbf{1}_{j=2} + \beta_{I3} \mathbf{1}_{group_i=T} \mathbf{1}_{j=3} + \beta_{I4} \mathbf{1}_{group_i=T} \mathbf{1}_{j=4} \\ + \varepsilon_{ij}, \quad \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2) \quad (3)$$

The design matrix is again a matrix whose i -th line combines one (for the intercept) with all the indicator variables for individual i :

$$X_i = [1, \mathbf{1}_{group_i=T}, \mathbf{1}_{j=1}, \mathbf{1}_{j=2}, \mathbf{1}_{j=3}, \mathbf{1}_{j=4}, \mathbf{1}_{group_i=T} \mathbf{1}_{j=1}, \mathbf{1}_{group_i=T} \mathbf{1}_{j=2}, \mathbf{1}_{group_i=T} \mathbf{1}_{j=3}, \mathbf{1}_{group_i=T} \mathbf{1}_{j=4}]$$

As before we can extract the design matrix for the four observations:

```
dfL.data[c(1:2,396:397),]
```

group	id	time	cholesterol	treatment
1	T	1	visit0	178
104	T	1	visit1	246
80	C	80	visit0	294
183	C	80	visit1	313

using `model.matrix`:

```
X <- model.matrix(~ group * time, data = dfL.data[c(1:2,396:397),])
X
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4 groupT:timevisit1
1           1     1       0       0       0       0             0
104         1     1       1       0       0       0             1
80          1     0       0       0       0       0             0
183         1     0       1       0       0       0             0
groupT:timevisit2 groupT:timevisit3 groupT:timevisit4
1           0       0       0
104         0       0       0
80          0       0       0
183         0       0       0
attr("assign")
[1] 0 1 2 2 2 2 3 3 3 3
attr("contrasts")
attr("contrasts")$group
[1] "contr.treatment"

attr("contrasts")$time
[1] "contr.treatment"
```

Imagine we are interested in the treatment effect at visit 1. Under some assumptions (e.g. no unmeasured confounders), this effect can be written in mathematical term as:

$$\mathbb{E}[Y_{ij} | \text{group} = T, \text{visit} = 1] - \mathbb{E}[Y_{ij} | \text{group} = C, \text{visit} = 1]$$

As before we can compute each of the terms:

$$\begin{aligned}\mathbb{E}[Y_{ij} | \text{group} = T, \text{visit} = 1] &= \alpha + \beta_G * 1 + \beta_{v1} * 1 + \beta_{v2} * 0 + \beta_{v3} * 0 + \beta_{v4} * 0 \\ &\quad + \beta_{I1} * 1 * 1 + \beta_{I2} * 1 * 0 + \beta_{I3} * 1 * 0 + \beta_{I4} * 1 * 0 \\ &\quad + 0 \\ &= \alpha + \beta_G + \beta_{v1} + \beta_{I1} \\ \mathbb{E}[Y_{ij} | \text{group} = C, \text{visit} = 1] &= \alpha + \beta_G * 0 + \beta_{v1} * 1 + \beta_{v2} * 0 + \beta_{v3} * 0 + \beta_{v4} * 0 \\ &\quad + \beta_{I1} * 0 * 1 + \beta_{I2} * 0 * 0 + \beta_{I3} * 0 * 0 + \beta_{I4} * 0 * 0 \\ &\quad + 0 \\ &= \alpha + \beta_{v1}\end{aligned}$$

So we now get that the treatment effect at visit 4 is:

$$\begin{aligned}\mathbb{E}[Y_{ij} | \text{group} = T, \text{visit} = 1] - \mathbb{E}[Y_{ij} | \text{group} = C, \text{visit} = 1] &= (\alpha + \beta_G + \beta_{v1} + \beta_{I1}) - (\alpha + \beta_{v1}) \\ &= \beta_G + \beta_{I1}\end{aligned}$$

Once more, we can also see that from the contrast matrix, by subtracting the row corresponding to the patient from the control group at visit 1 to the row corresponding to the patient from the treatment group at visit 1:

```
x[2,] - x[4,]
```

(Intercept)	groupT	timevisit1	timevisit2	timevisit3
0	1	0	0	0
timevisit4	groupT:timevisit1	groupT:timevisit2	groupT:timevisit3	groupT:timevisit4
0	1	0	0	0

5.2.3 Baseline adjustment with an interaction

In some studies we want to put constraints on the treatment effect. For instance if the treatment is only administered at the second visit, we want to set the group difference at the first visit to 0. This can be done by first defining the appropriate treatment variable:

```
dfL.data[dfL.data$id %in% c("1", "101"), c("time", "group", "treatment")]
```

	time	group	treatment
1	visit0	T	none
104	visit1	T	tr
207	visit2	T	tr
310	visit3	T	tr
413	visit4	T	tr
101	visit0	C	none
204	visit1	C	pl
307	visit2	C	pl
410	visit3	C	pl
513	visit4	C	pl

and then instead of using the * to specify the interaction:

```
try(gls(model = cholesterol ~ treatment * time,
        data = dfL.data,
        na.action = na.omit))
```

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

one can use the following syntax:

```
e.gls <- gls(cholesterol ~ I(droplevels(interaction(treatment,time))),  
               data = dfL.data,  
               na.action = na.omit)  
logLik(e.gls)
```

'log Lik.' -2304.06 (df=10)

or, equivalently first define the interaction variable:

```
dfL.data$ii <- droplevels(interaction(dfL.data$treatment,  
                                         dfL.data$time))  
table(dfL.data$ii, dfL.data$time, dfL.data$group)
```

, , = C

	visit0	visit1	visit2	visit3	visit4
none.visit0	41	0	0	0	0
pl.visit1	0	41	0	0	0
tr.visit1	0	0	0	0	0
pl.visit2	0	0	41	0	0
tr.visit2	0	0	0	0	0
pl.visit3	0	0	0	41	0
tr.visit3	0	0	0	0	0
pl.visit4	0	0	0	0	41
tr.visit4	0	0	0	0	0

, , = T

	visit0	visit1	visit2	visit3	visit4
none.visit0	62	0	0	0	0
pl.visit1	0	0	0	0	0
tr.visit1	0	62	0	0	0
pl.visit2	0	0	0	0	0
tr.visit2	0	0	62	0	0
pl.visit3	0	0	0	0	0
tr.visit3	0	0	0	62	0
pl.visit4	0	0	0	0	0
tr.visit4	0	0	0	0	62

and then call the `gls` model:

```
e.gls <- gls(cholesterol ~ ii,
               data = dfL.data,
               na.action = na.omit)
logLik(e.gls)
```

```
'log Lik.' -2304.06 (df=10)
```

REMARK: one drawback of this approach is that when calling `anova` to display the F-test:

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

```
Denom. DF: 438
      numDF   F-value p-value
(Intercept)     1 2714.7169  <.0001
ii              8    2.8401  0.0044
```

the interaction mixes time effects and treatment effects while we usually want a separate test for those. To test the treatment effect alone, the easiest solution is to fit a model without treatment effect:

```
e.gls0 <- gls(cholesterol ~ time,
                data = dfL.data,
                na.action = na.omit)
logLik(e.gls0)
```

```
'log Lik.' -2317.244 (df=6)
```

and then to use a likelihood ratio test to compare both models:

```
anova(update(e.gls, method = "ML"),
      update(e.gls0, method = "ML"))
```

	Model	df	AIC	BIC	logLik	Test	L.Ratio
update(e.gls, method = "ML")	1	10	4678.475	4719.501	-2329.238		
update(e.gls0, method = "ML")	2	6	4671.227	4695.842	-2329.613	1 vs 2	0.7520657
				p-value			
update(e.gls, method = "ML")							
update(e.gls0, method = "ML")				0.9448			

Note that to perform the likelihood ratio test, the models need to be estimated by maximum likelihood (ML) (and not by restricted maximum likelihood, default option in `gls`). This is why we used `update` to re-estimate the model with ML when calling `anova`.

5.3 Specifying the variance-covariance matrix

In this section we aim at choosing the structure of the variance-covariance matrix Σ of the residuals:

$$\varepsilon_i \sim \mathcal{N}(0, \Sigma)$$

To simplify we will consider the case of 5 repetitions as in our example. So Σ is a matrix with 5 rows and 5 columns. We also know that it is symmetric so its most general form is:

$$\Sigma = \begin{bmatrix} \sigma_{11}^2 & \sigma_{12}^2 & \sigma_{13}^2 & \sigma_{14}^2 & \sigma_{15}^2 \\ \sigma_{12}^2 & \sigma_{22}^2 & \sigma_{23}^2 & \sigma_{24}^2 & \sigma_{25}^2 \\ \sigma_{13}^2 & \sigma_{23}^2 & \sigma_{33}^2 & \sigma_{34}^2 & \sigma_{35}^2 \\ \sigma_{14}^2 & \sigma_{24}^2 & \sigma_{34}^2 & \sigma_{44}^2 & \sigma_{45}^2 \\ \sigma_{15}^2 & \sigma_{25}^2 & \sigma_{35}^2 & \sigma_{45}^2 & \sigma_{55}^2 \end{bmatrix} = \begin{bmatrix} \sigma_{11}^2 & . & . & . & . \\ \sigma_{12}^2 & \sigma_{22}^2 & . & . & . \\ \sigma_{13}^2 & \sigma_{23}^2 & \sigma_{33}^2 & . & . \\ \sigma_{14}^2 & \sigma_{24}^2 & \sigma_{34}^2 & \sigma_{44}^2 & . \\ \sigma_{15}^2 & \sigma_{25}^2 & \sigma_{35}^2 & \sigma_{45}^2 & \sigma_{55}^2 \end{bmatrix}$$

where the notation on the right do not show the repeated terms and display in a different color the diagonal term and the off-diagonal terms. The diagonal terms tell about the variance of the residual at each time, e.g. $\sigma_{11}^2 = \sigma_{22}^2$ indicates that the residual variability is the same at time 1 and 2. The off-diagonal terms tell about the covariance of the residual between timepoints. It is often easier to think in term of correlation instead of covariance so we will re-parametrize Σ :

$$\Sigma = \begin{bmatrix} \sigma_1^2 & . & . & . & . \\ \rho_{12}\sigma_1\sigma_2 & \sigma_2^2 & . & . & . \\ \rho_{13}\sigma_1\sigma_3 & \rho_{23}\sigma_2\sigma_3 & \sigma_3^2 & . & . \\ \rho_{14}\sigma_1\sigma_4 & \rho_{24}\sigma_2\sigma_4 & \rho_{34}\sigma_3\sigma_4 & \sigma_4^2 & . \\ \rho_{15}\sigma_1\sigma_5 & \rho_{25}\sigma_2\sigma_5 & \rho_{35}\sigma_3\sigma_5 & \rho_{45}\sigma_4\sigma_5 & \sigma_5^2 \end{bmatrix}$$

So that now we have 5 orange parameters that model the variance at each time and 10 correlation parameters that model the correlation between the residuals. In practice `gls` uses a slightly different parametrisation:

$$\Sigma = \sigma^2 \begin{bmatrix} 1 & . & . & . & . \\ \rho_{12}k_2 & k_2^2 & . & . & . \\ \rho_{13}k_3 & \rho_{23}k_2k_3 & k_3^2 & . & . \\ \rho_{14}k_4 & \rho_{24}k_2k_4 & \rho_{34}k_3k_4 & k_4^2 & . \\ \rho_{15}k_5 & \rho_{25}k_2k_5 & \rho_{35}k_3k_5 & \rho_{45}k_4k_5 & k_5^2 \end{bmatrix}$$

where there is a "baseline" variance level, σ^2 and the parameters k_2^2, \dots, k_5^2 reflect the relative increase or decrease in variance over time.

Now it should be logical that `gls` uses two arguments to specify Σ :

- **correlation:** specifies the form of **the correlation terms**. It will typically be: `corCompSymm(form = ~ 1 | subject)` or `corSymm(form = ~ visit | subject)` and is composed of three parts:
 - the structure of the correlation matrix. We will use either `corCompSymm` that forces all the correlation coefficients to be equal or `corSymm` that puts no constrain. See `?corClasses` for other available structures.
 - the position variable. This is usually not needed when using `corCompSymm` (i.e. `~1`). When using `corSymm` it is used to indicate to which repetition belong the observation (e.g. first visit or second visit, i.e. `~ visit` where `visit` is a numeric variable). In other structures it can be used to specify explanatory variables (e.g. geographical position) but we won't use that in the class.
 - the grouping variable (.e.g `|subject`). This indicates that observations are correlated within a group and therefore jointly model all observations belonging to the same group (here to the same subject).
- **weight:** specifies the form of **the variance terms**. It will typically be: `varIdent(form = ~ 1 | visit)` and is also composed of three parts:
 - the structure of the variance terms. We will only use `varIdent` that uses a different variance parameter at each repetition time.
 - the position variable. This is usually not needed when using `varIdent` (i.e. `~1`).
 - the grouping variable. This variable indicate to which repetition belong the observation (e.g. `|visit`).

5.3.1 Illustration - compound symmetry structure

We first use a compound symmetry structure. This corresponds to the following residual covariance matrix:

$$\Sigma = \sigma^2 \begin{bmatrix} 1 & . & . & . & . \\ . & 1 & . & . & . \\ . & . & 1 & . & . \\ . & . & . & 1 & . \\ . & . & . & . & 1 \end{bmatrix}$$

This is achieved specifying `correlation` equals to `corCompSymm(form =~ 1|id)` when calling `gls`:

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

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

When calling `summary`, we get the estimated correlation parameter (ρ):

```
summary(e.glsCS$modelStruct)
```

Correlation Structure: Compound symmetry

Formula: ~1 | id

Parameter estimate(s):

Rho

0.7144955

and the square root of the variance parameter (σ^2):

```
sigma(e.glsCS)
```

[1] 44.3852

We can extract the residual variance-covariance matrix for the first individual using `getVarCov`:

```
Sigma <- getVarCov(e.glsCS, individual = 1)
Sigma
```

```
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
```

and rescale it into a correlation matrix using `cov2cor`:

```
cov2cor(Sigma)
```

```
Marginal variance covariance matrix
 [,1]   [,2]   [,3]   [,4]   [,5]
[1,] 1.0000 0.7145 0.7145 0.7145 0.7145
[2,] 0.7145 1.0000 0.7145 0.7145 0.7145
[3,] 0.7145 0.7145 1.0000 0.7145 0.7145
[4,] 0.7145 0.7145 0.7145 1.0000 0.7145
[5,] 0.7145 0.7145 0.7145 0.7145 1.0000
Standard Deviations: 1 1 1 1 1
```

5.3.2 Illustration - unstructured covariance matrix

We now use an unstructured covariance matrix. This corresponds covariance matrix has already been shown in the previous sections:

$$\Sigma = \sigma^2 \begin{bmatrix} 1 & . & . & . & . \\ \rho_{12}k_2 & k_2^2 & . & . & . \\ \rho_{13}k_3 & \rho_{23}k_2k_3 & k_3^2 & . & . \\ \rho_{14}k_4 & \rho_{24}k_2k_4 & \rho_{34}k_3k_4 & k_4^2 & . \\ \rho_{15}k_5 & \rho_{25}k_2k_5 & \rho_{35}k_3k_5 & \rho_{45}k_4k_5 & k_5^2 \end{bmatrix}$$

This is achieved specifying `correlation` equals to `corSymm(form =~ as.numeric(time)|id)` and `weights` equals to `varIdent(form =~ 1|time)` when calling `gls`:

```
e.glsUN <- gls(model = cholesterol ~ group * time,
                 data = dfL.data,
                 correlation = corSymm(form =~ as.numeric(time)|id),
                 weights = varIdent(form =~ 1|time),
                 na.action = na.omit)
logLik(e.glsUN)
```

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

When calling `summary`, we get the estimated correlation parameter ($\rho_{12}, \dots, \rho_{45}$)

```
summary(e.glsUN$modelStruct$corStruct)
```

```
Correlation Structure: General
Formula: ~as.numeric(time) | id
Parameter estimate(s):
Correlation:
      1     2     3     4
2 0.770
3 0.732 0.775
4 0.737 0.796 0.725
5 0.589 0.669 0.680 0.626
```

the inflation factors for the variance (k_2^2, \dots, k_5^2):

```
summary(e.glsUN$modelStruct$varStruct)
```

```
Variance function:
Structure: Different standard deviations per stratum
Formula: ~1 | time
Parameter estimates:
  visit0    visit1    visit2    visit3    visit4
1.0000000 0.9322230 0.8798470 0.8962107 1.0303536
```

and the square root of the variance parameter (σ^2):

```
sigma(e.glsUN)
```

```
[1] 46.76062
```

As before we can extract the residual variance-covariance matrix for the first individual using `getVarCov`:

```
Sigma <- getVarCov(e.glsUN, individual = 1)  
Sigma
```

```
Marginal variance covariance matrix  
[,1] [,2] [,3] [,4] [,5]  
[1,] 2186.6 1570.0 1407.4 1444.5 1326.1  
[2,] 1570.0 1900.2 1390.5 1454.9 1406.0  
[3,] 1407.4 1390.5 1692.7 1250.1 1347.6  
[4,] 1444.5 1454.9 1250.1 1756.2 1264.1  
[5,] 1326.1 1406.0 1347.6 1264.1 2321.3  
Standard Deviations: 46.761 43.591 41.142 41.907 48.18
```

and rescale it into a correlation matrix using `cov2cor`:

```
cov2cor(Sigma)
```

```
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
```

We can also check that σ^2 equals:

```
Sigma[1,1]
```

```
[1] 2186.555
```

and $1, k_2^2, \dots, k_5^2$ equals:

```
sqrt(diag(Sigma)/Sigma[1,1])
```

```
[1] 1.0000000 0.9322230 0.8798470 0.8962107 1.0303536
```

5.4 Technical point: why using `as.numeric(time)` in `corSymm`

Using `~as.numeric(time)|id` in `corSymm` may seem a weird syntax. As explained before `as.numeric(time)` is here to indicate the time repetition. If we don't and use instead `~1|id`:

```
eWrong.glsUN <- gls(model = cholesterol ~ group * time,
                      data = dfL.data,
                      correlation = corSymm(form = ~ 1|id),
                      weights = varIdent(form = ~ 1|time),
                      na.action = na.omit)

logLik(eWrong.glsUN)
```

'log Lik.' -2132.294 (df=25)

we get different estimates:

```
coef(eWrong.glsUN) - coef(e.glsUN)
```

	(Intercept)	groupT	timevisit1	timevisit2	timevisit3
-5.684342e-13	2.324097e-01	-7.727152e-14	-2.113024e-03	5.462693e-04	
timevisit4	groupT:timevisit1	groupT:timevisit2	groupT:timevisit3	groupT:timevisit4	
-1.461798e-03	-5.471536e-02	-1.286151e-01	-2.202927e-01	-3.671284e-01	

This is because the covariance structure for some individuals with missing data, e.g.:

```
dfL.data[306:310,]
```

	group	id	time	cholesterol	treatment	tXt	treatmenttime
62	T	62	visit0	193	none none.visit0	none	
165	T	62	visit1	189	tr tr.visit1	visit1	
268	T	62	visit2	NA	tr tr.visit2	visit2	
371	T	62	visit3	232	tr tr.visit3	visit3	
474	T	62	visit4	211	tr tr.visit4	visit4	

is incorrect:

```
cov2cor(getVarCov(eWrong.glsUN, individual = 62))
```

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

Comparing it to the estimated correlation coefficients:

```
summary(eWrong.glsUN$modelStruc$corStruct)
```

```
Correlation Structure: General  
Formula: ~1 | id  
Parameter estimate(s):  
Correlation:  
      1   2   3   4  
2 0.770  
3 0.732 0.773  
4 0.738 0.800 0.726  
5 0.586 0.665 0.678 0.625
```

we can see that it is the correlation between time 0,1,2,3,4 that has been modeled while the observed visit times are 0,1,3,4. This is because, by default, `gls` assumes the missing values occurred at the end of the follow-up and not in the middle.

Another way to see that is to define an autoregressive correlation structure:

```
SigmaAR <- corAR1(value = 0.3, form = ~1|id)
```

If there was no missing value the full correlation matrix would be:

```
SigmaAR.62 <- Initialize(SigmaAR, data = dfL.data[306:310,])  
corMatrix(SigmaAR.62)
```

```
[,1] [,2] [,3] [,4] [,5]  
[1,] 1.0000 0.300 0.09 0.027 0.0081  
[2,] 0.3000 1.000 0.30 0.090 0.0270  
[3,] 0.0900 0.300 1.00 0.300 0.0900  
[4,] 0.0270 0.090 0.30 1.000 0.3000  
[5,] 0.0081 0.027 0.09 0.300 1.0000
```

If we omit row 3 we obtain:

```
SigmaAR.62na <- Initialize(SigmaAR,  
                           data = dfL.data[c(306:307,309:310),])  
corMatrix(SigmaAR.62na)
```

```
[,1] [,2] [,3] [,4]  
[1,] 1.000 0.30 0.09 0.027  
[2,] 0.300 1.00 0.30 0.090  
[3,] 0.090 0.30 1.00 0.300  
[4,] 0.027 0.09 0.30 1.000
```

which is incorrect.

We should have:

```
SigmaAR2 <- corAR1(value = 0.3, form = ~as.numeric(time)|id)
SigmaAR2.62na <- Initialize(SigmaAR2,
                                data = dfL.data[c(306:307,309:310),])
corMatrix(SigmaAR2.62na)
```

```
[,1] [,2] [,3] [,4]
[1,] 1.0000 0.300 0.027 0.0081
[2,] 0.3000 1.000 0.090 0.0270
[3,] 0.0270 0.090 1.000 0.3000
[4,] 0.0081 0.027 0.300 1.0000
```

5.5 Prediction

When performing predictions, the safest way is to build a `data.frame` that precisely matches the format of the dataset used to train the model. Care must be taken in presence of factors since the `predict` function requires to have the same levels as the one used when fitting the model. For instance when considering the following model:

```
e.gls <- gls(model = cholesterol ~ group + time,
               data = dfL.data,
               correlation = corSymm(form =~ as.numeric(time)|id),
               weights = varIdent(form =~ 1|time),
               na.action = na.omit)
logLik(e.gls)
```

'log Lik.' -2147.318 (df=21)

that has been fitted using the dataset:

```
str(dfL.data)
```

```
'data.frame':      515 obs. of  5 variables:
 $ group      : Factor w/ 2 levels "C","T": 2 2 2 2 2 2 2 2 2 ...
 $ id         : Factor w/ 103 levels "1","2","3","4",...: 1 1 1 1 1 2 2 2 2 ...
 $ time        : Factor w/ 5 levels "visit0","visit1",...: 1 2 3 4 5 1 2 3 4 5 ...
 $ cholesterol: int  178 246 295 228 274 254 260 278 245 340 ...
 $ treatment   : Factor w/ 3 levels "none","pl","tr": 1 3 3 3 3 1 3 3 3 3 ...
```

we need to re-create a dataset with the same factor levels for `time` and `group`:

```
newdf <- expand.grid(time = c("visit0","visit1","visit2"),
                      group = c("C","T"))
newdf$time <- factor(newdf$time, levels = levels(dfL.data$time))
newdf$group <- factor(newdf$group, levels = levels(dfL.data$group))
newdf
```

	time	group
1	visit0	C
2	visit1	C
3	visit2	C
4	visit0	T
5	visit1	T
6	visit2	T

to apply the `predict` function:

```
newdf$pred <- predict(e.gls, newdata = newdf)
```

Standard errors for the predictions can be obtain using the package AICcmodavg:

```
library(AICcmodavg)
```

and calling the function `predictSE.gls`:

```
newdf$pred.se <- predictSE.gls(e.gls, newdata = newdf)$se.fit
```

Note that this corresponds to:

- creating the new design matrix:

```
newX <- model.matrix(~ group + time, data = newdf)
```

- computing the variance-covariance matrix of the predictions:

```
vcovPred <- newX %*% vcov(e.gls) %*% t(newX)
```

- taking the square root of the diagonal to extract the standard error of the predictions:

```
sePred <- sqrt(diag(vcovPred))
```

We can check that this matches the results of `predictSE.gls`:

```
newdf$pred.se - sePred
```

Erreur : objet 'sePred' introuvable

To get 95% confidence intervals, we first compute the critical quantiles:

```
df.gls <- e.gls$dim$N - e.gls$dim$p
alpha <- 0.05
quantileLower <- qt(alpha/2, df = df.gls)
quantileUpper <- qt(1 - alpha/2, df = df.gls)
```

and then apply the standard formula:

```
newdf$ciLower <- newdf$pred + quantileLower * newdf$pred.se
newdf$ciUpper <- newdf$pred + quantileUpper * newdf$pred.se
newdf
```

	time	group	pred	pred.se	ciLower	ciUpper
1	visit0	C	229.7555	6.580639	216.8222	242.6888
2	visit1	C	244.3865	6.349382	231.9077	256.8653
3	visit2	C	248.3912	6.238909	236.1295	260.6529
4	visit0	T	230.0972	5.566373	219.1573	241.0371
5	visit1	T	244.7283	5.290969	234.3296	255.1269
6	visit2	T	248.7329	5.178308	238.5557	258.9101

Note: these confidence intervals rely on asymptotic results and may not be very accurate in small samples. In particular the estimation of the degree of freedom is a very crude approximation. As mentioned in the previous section, other technics (e.g. Satterthwaite approximation, bootstrap resampling) will usually have better coverage in small samples.

5.6 Other methods

The list of available methods for `gls` objects can be obtained using:

```
methods(class = "gls")
```

```
[1] ACF                  AICc                 anova                augPred  
[5] checkParms          coef                  compare2              comparePred  
[9] conditionalMoment   createContrast      deviance              extractAIC  
[13] extractData         fitted                formula               getData  
[17] getGroups           getGroupsFormula getResponse            getVarCov  
[21] getVarCov2          iid2                  intervals             leverage2  
[25] logLik               multComp              nobs                  plot  
[29] predict              predictSE              print                 qnorm  
[33] residuals           score2                sCorrect              sCorrect<-  
[37] sigma                summary               summary2              terms  
[41] update               useBIC                Variogram            vcov  
[45] vcov2  
see '?methods' for accessing help and source code
```

The list may vary depending on which packages have been loaded (**R** searches among the loaded packages all methods referring to `gls`).

6 Using the `lme` function of the `nlme` package

6.1 Short presentation of the `lme` function

To access the `gls` function we first need to load the `nlme` package:

```
library(nlme)
```

Then we can have a look to the arguments of the `lme` function:

```
args(lme)
```

```
function (fixed, data = sys.frame(sys.parent()), random, correlation = NULL,
         weights = NULL, subset, method = c("REML", "ML"), na.action = na.fail,
         control = list(), contrasts = NULL, keep.data = TRUE)
NULL
```

Compared to `gls` (see section 5.1), there is an additional argument `random` that we will use instead of `correlation` and `weight`. It defines the covariance structure:

- `random =~ 1|id` corresponds to a compound symmetry structure
- `random =~ time|id` corresponds to a unstructured covariance matrix

There is some redundancy between the arguments. For instance:

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

```
'log Lik.' -2147.318 (df=22)
```

is equivalent to:

```
e.lmeBis <- lme(cholesterol ~ group + time,
                  data = dfL.data,
                  random =~ time | id,
                  correlation = corSymm(form =~ as.numeric(time)|id),
                  weights = varIdent(form =~ 1|time),
                  na.action = na.omit)
logLik(e.lmeBis)
```

```
'log Lik.' -2147.318 (df=36)
```

and the extra 14 parameters specified by `correlation` and `weights` are not used. This is not always the case since for instance:

```
e.lmeTer <- lme(cholesterol ~ group + time,
                  data = dfL.data,
                  random =~ 1 | id,
                  correlation = corSymm(form =~ as.numeric(time) | id),
                  weights = varIdent(form =~ 1 | time),
                  na.action = na.omit)

logLik(e.lmeTer)
```

```
'log Lik.' -2147.318 (df=22)
```

differs from

```
e.lme0 <- lme(cholesterol ~ group + time,
                 data = dfL.data,
                 random =~ 1 | id,
                 na.action = na.omit)

logLik(e.lme0)
```

```
'log Lik.' -2160.3 (df=8)
```

6.2 Specifying the mean structure

Similar to the `gls` function, see section 5.2.

6.3 Specifying the variance-covariance matrix

In this section we will investigate the difference between:

```
e.lme0 <- lme(cholesterol ~ group + time,
               data = dfL.data,
               random =~ 1 | id,
               na.action = na.omit)
```

and

```
e.lme <- lme(cholesterol ~ group + time,
               data = dfL.data,
               random =~ time | id,
               na.action = na.omit)
```

and see that the first model corresponds to a compound symmetry structure while the second has an unstructured residual covariance matrix.

6.3.1 Compound symmetry structure with `lme`

We can write the first model as:

$$Y_{ij} = \mu_j(X_{ij}) + u_i + \varepsilon_{ij}, \text{ where } u_i \sim \mathcal{N}(0, \tau), \varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2), \text{ and } u_i \perp\!\!\!\perp \varepsilon_{ij}$$

where $\mu_j(X_{ij})$ is the mean structure corresponding to `~group + time`, u_i the random effect corresponding to `~1|id` and ε_{ij} the residual. So for fixed X :

$$\begin{aligned} \text{Cov}[Y_{ij}, Y_{ij'}] &= \text{Cov}[\mu_j(X_{ij}) + u_i + \varepsilon_{ij}, \mu_j(X_{ij'}) + u_i + \varepsilon_{ij'}] \\ &= \text{Cov}[u_i + \varepsilon_{ij}, u_i + \varepsilon_{ij'}] \\ &= \text{Cov}[u_i, u_i] + \text{Cov}[u_i, \varepsilon_{ij'}] + \text{Cov}[\varepsilon_{ij}, u_i] + \text{Cov}[\varepsilon_{ij}, \varepsilon_{ij'}] \\ &= \text{Var}[u_i] + 0 + 0 + \text{Cov}[\varepsilon_{ij}, \varepsilon_{ij'}] \\ &= \begin{cases} \tau + \sigma^2 & \text{if } j = j' \\ \tau & \text{if } j \neq j' \end{cases} \end{aligned}$$

where σ^2 equals:

```
sigma(e.lme0)^2
```

[1] 567.1223

and τ equals (coefficient (Intercept)):

```
getVarCov(e.lme0, type = "random")
```

```
Random effects variance covariance matrix
(Intercept)
(Intercept) 1405.1
Standard Deviations: 37.485
```

In this corresponds to the covariance matrix:

$$\Sigma = \begin{bmatrix} \sigma^2 + \tau & . & . & . & . \\ . & \sigma^2 + \tau & . & . & . \\ . & . & \sigma^2 + \tau & . & . \\ . & . & . & \sigma^2 + \tau & . \\ . & . & . & . & \sigma^2 + \tau \end{bmatrix}$$

This is indeed what we observed when using getVarCov:

```
getVarCov(e.lme0, type = "marginal")
```

```
id 1
Marginal variance covariance matrix
      1     2     3     4     5
1 1972.2 1405.1 1405.1 1405.1 1405.1
2 1405.1 1972.2 1405.1 1405.1 1405.1
3 1405.1 1405.1 1972.2 1405.1 1405.1
4 1405.1 1405.1 1405.1 1972.2 1405.1
5 1405.1 1405.1 1405.1 1405.1 1972.2
Standard Deviations: 44.41 44.41 44.41 44.41 44.41
```

6.3.2 Unstructured covariance matrix with lme

We can write the second model as:

$$Y_{ij} = \mu_j(X_{ij}) + u_{i1}\mathbf{1}_{j=1} + u_{i2}\mathbf{1}_{j=2} + u_{i3}\mathbf{1}_{j=3} + u_{i4}\mathbf{1}_{j=4} + u_{i5}\mathbf{1}_{j=5} + \varepsilon_{ij}$$

Here the vector of random effects $(u_{i1}, u_{i2}, u_{i3}, u_{i4}, u_{i5})$ is independent of ε_{ij} and follows a joint normal distribution centered around 0 with covariance matrix:

$$\Sigma_u = \begin{bmatrix} \tau_{11} & . & . & . & . \\ \tau_{12} & \tau_{22} & . & . & . \\ \tau_{13} & \tau_{23} & \tau_{33} & . & . \\ \tau_{14} & \tau_{24} & \tau_{34} & \tau_{44} & . \\ \tau_{15} & \tau_{25} & \tau_{35} & \tau_{45} & \tau_{55} \end{bmatrix}$$

Unfortunately `getVarCov` does not return directly this matrix:

```
D <- getVarCov(e.lme, type = "random")
D
```

```
Random effects variance covariance matrix
              (Intercept) timevisit1 timevisit2 timevisit3 timevisit4
(Intercept)    2031.10   -466.84   -636.12   -576.96   -695.15
timevisit1     -466.84    625.70    468.31    459.97    534.15
timevisit2     -636.12    468.31    769.93    420.19    647.78
timevisit3     -576.96    459.97    420.19    698.20    511.50
timevisit4     -695.15    534.15    647.78    511.50   1496.50
Standard Deviations: 45.068 25.014 27.748 26.424 38.684
```

(we know that the observations are positively correlated so the off-diagonal coefficients should be positives). To reconstruct the matrix Σ_u we first need to re-create the design matrix corresponding to the random effects:

```
Z <- model.matrix(e.lme$modelStruct$reStruc,
                   data = dfL.data[dfL.data$id=="1",])
attr(Z, "ncols") <- NULL
attr(Z, "nams") <- NULL
attr(Z, "contr") <- NULL
Z
```

```
(Intercept) timevisit1 timevisit2 timevisit3 timevisit4
1            1          0          0          0          0
104           1          1          0          0          0
207           1          0          1          0          0
310           1          0          0          1          0
413           1          0          0          0          1
```

Then we obtain Σ_u using the matrix product:

```
Z %*% D %*% t(Z)
```

```
      1      104      207      310      413
1  2031.087 1564.247 1394.967 1454.127 1335.941
104 1564.247 1723.112 1396.437 1447.257 1403.249
207 1394.967 1396.437 1528.773 1238.197 1347.597
310 1454.127 1447.257 1238.197 1575.370 1270.478
413 1335.941 1403.249 1347.597 1270.478 2137.275
```

Recalling that we still assume that $\varepsilon_{ij} \sim \mathcal{N}(0, \sigma^2)$ where σ^2 equals:

```
sigma(e.lme)^2
```

[1] 174.1248

we have for a fixed X :

$$\begin{aligned}\mathbb{C}ov [Y_{ij}, Y_{ij'}] &= \mathbb{C}ov [\mu_j(X_{ij}) + u_{ij} + \varepsilon_{ij}, \mu_j(X_{ij'}) + u_{ij'} + \varepsilon_{ij'}] \\ &= \mathbb{C}ov [u_{ij} + \varepsilon_{ij}, u_{ij'} + \varepsilon_{ij'}] \\ &= \mathbb{C}ov [u_{ij}, u_{ij'}] + \mathbb{C}ov [u_{ij}, \varepsilon_{ij'}] + \mathbb{C}ov [\varepsilon_{ij}, u_{ij'}] + \mathbb{C}ov [\varepsilon_{ij}, \varepsilon_{ij'}] \\ &= \tau_{jj'} + 0 + 0 + \mathbb{C}ov [\varepsilon_{ij}, \varepsilon_{ij'}] \\ &= \begin{cases} \tau_{jj'} + \sigma^2 & \text{if } j = j' \\ \tau_{jj'} & \text{if } j \neq j' \end{cases}\end{aligned}$$

This is indeed what we observed when using `getVarCov`:

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

```
id 1
Marginal variance covariance matrix
      1       2       3       4       5
1 2205.2 1564.2 1395.0 1454.1 1335.9
2 1564.2 1897.2 1396.4 1447.3 1403.2
3 1395.0 1396.4 1702.9 1238.2 1347.6
4 1454.1 1447.3 1238.2 1749.5 1270.5
5 1335.9 1403.2 1347.6 1270.5 2311.4
Standard Deviations: 46.96 43.557 41.266 41.827 48.077
```

6.4 Note on `lmer` from the `lme4` package

To extract the residual covariance matrices from objects generated by `lmer` we will adapt the code proposed by <https://stat.ethz.ch/pipermail/r-sig-mixed-models/2008q1/000558.html>:

```
extractSigmaLMER <- function(model, indiv = 1){  
  vec.group <- getME(model, "flist")[[1]]  
  index.indiv <- which(vec.group == levels(vec.group)[indiv])  
  tau.corr <- as.matrix(getME(model, "A") [index.indiv, index.indiv])  
  Id <- diag(1, length(index.indiv), length(index.indiv))  
  out <- sigma(model)^2 * (crossprod(tau.corr) + Id)  
  return(out)  
}
```

6.4.1 Compound symmetry structure

The `lmer` function works similarly to the `lme` function, except that the `random` argument is integrated into the formula, e.g. for a compound symmetry structure:

```
e.lmer <- lmer(cholesterol ~ group + time + (1 | id),  
                 data = dfL.data,  
                 na.action = na.omit)  
logLik(e.lmer)
```

```
'log Lik.' -2160.3 (df=8)
```

This matches the results of `lme`:

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

```
'log Lik.' -2160.3 (df=8)
```

regarding the log-likelihood but also for the estimate of the mean parameters:

```
fixef(e.lme)-fixef(e.lmer)
```

```
(Intercept)      groupT    timevisit1   timevisit2   timevisit3   timevisit4  
-6.906504e-09  1.148485e-08 -2.760459e-12  5.226319e-09  8.829215e-09  4.864336e-09
```

or the residual variance-covariance matrix:

```
extractSigmaLME(e.lmer) - getVarCov(e.lme, type = "marginal")[[1]]
```

	1	104	207	310	413
1	0.0001055656 0.0001180995 0.0001180995 0.0001180995 0.0001180995				
104	0.0001180995 0.0001055656 0.0001180995 0.0001180995 0.0001180995				
207	0.0001180995 0.0001180995 0.0001055656 0.0001180995 0.0001180995				
310	0.0001180995 0.0001180995 0.0001180995 0.0001055656 0.0001180995				
413	0.0001180995 0.0001180995 0.0001180995 0.0001180995 0.0001055656				

which is very small compared to the magnitude of the residual variance.

6.4.2 Unstructured residual variance-covariance matrix

When estimating a mixed model with a unstructured covariance matrix, e.g.

```
try(lmer(cholesterol ~ group + time + (time | id),
         data = dfL.data,
         na.action = na.omit))
```

```
Error : number of observations (=447) <= number of random effects (=515) for term (time | i
```

it fails when we only have one measurement per `id` and `time`. This is because the `lmer` function run automatic checks warning the user of a possible misspecification of the model. However in this example, the model is correctly specified so we have to disable the check:

```
e.lmer <- lmer(cholesterol ~ group + time + (time | id),
                data = dfL.data,
                na.action = na.omit,
                control = lmerControl(check.nobs.vs.nRE = "ignore"))
logLik(e.lmer)
```

```
singular fit
Warning message:
Model failed to converge with 1 negative eigenvalue: -9.4e-03
'log Lik.' -2147.32 (df=22)
```

Even though `lmer` obtains the same log-likelihood as `lme`:

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

```
'log Lik.' -2147.318 (df=22)
```

it outputs warnings that the estimation procedure did not converge properly. Indeed:

```
eigen(e.lmer@optinfo$derivs$Hessian)$values
```

```
[1] 175.194483547 118.111426128 91.690825734 75.628389400 65.032550385 57.956280658  
[7] 52.473854202 46.555990555 33.927560869 24.163133167 15.013979020 10.968749182  
[13] 6.042927796 2.879039656 -0.009246451
```

We can see that the estimated mean parameters are slightly different from `lme`:

```
fixef(e.lme) - fixef(e.lmer)
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4  
-1.044225e-02 1.734761e-02 4.476419e-13 -2.025123e-03 -1.049226e-04 1.514414e-02
```

as well as the residual variance covariance matrix:

```
extractSigmaLMER(e.lmer) - getVarCov(e.lme, type = "marginal")[[1]]
```

```
1 104 207 310 413  
1 -0.09240821 0.2253390 -0.1348845 -0.2691045 -0.200672  
104 0.22533901 -0.6295047 -0.5927819 0.1974928 3.068481  
207 -0.13488446 -0.5927819 0.6769815 0.5910536 -2.687127  
310 -0.26910451 0.1974928 0.5910536 0.0254745 -3.109014  
413 -0.20067204 3.0684807 -2.6871268 -3.1090138 -1.493004
```

7 Handling missing values in the response variable

The argument `na.action` indicates to `gls` / `lme` how to deal with missing values:

- `na.action=na.fail` (default option) leads to an error in presence of missing values.
- `na.action=na.omit` deals with missing values by removing the corresponding lines in the dataset.
- `na.action=na.exclude` ignores the missing values. Compared to `na.omit` this means that the extractors (like `fitted` or `residuals`) will output the same number of observations compared to the original dataset.
- `na.action=na.pass` will continue the execution of the function without any change. If the function cannot manage missing values, it will lead to an error.

7.1 Illustration with `gls`

We recall the the dataset has:

```
NROW(dfL.data)
```

```
[1] 515
```

rows and contains

```
sum(!is.na(dfL.data$cholesterol))
```

```
[1] 447
```

cholesterol mesurements without missing data.

`na.fail`:

```
e.glsUN.fail <- try(gls(model = cholesterol ~ group * time,
                           data = dfL.data,
                           correlation = corSymm(form =~ 1|id),
                           weights = varIdent(form =~ 1|time),
                           na.action = na.fail))
```

```
Error in na.fail.default(structure(list(id = structure(c(1L, 1L, 1L, 1L, :  
missing values in object
```

```
na.omit:
```

```
e.glsUN.omit <- gls(model = cholesterol ~ group * time,
                      data = dfL.data,
                      correlation = corSymm(form =~ 1|id),
                      weights = varIdent(form =~ 1|time),
                      na.action = na.omit)
length(residuals(e.glsUN.omit))
```

[1] 447

```
na.exclude:
```

```
e.glsUN.exclude <- gls(model = cholesterol ~ group * time,
                         data = dfL.data,
                         correlation = corSymm(form =~ 1|id),
                         weights = varIdent(form =~ 1|time),
                         na.action = na.exclude)
length(residuals(e.glsUN.exclude))
```

[1] 515

```
na.pass:
```

```
e.glsUN.pass <- try(gls(model = cholesterol ~ group * time,
                           data = dfL.data,
                           correlation = corSymm(form =~ 1|id),
                           weights = varIdent(form =~ 1|time),
                           na.action = na.pass))
```

Error in array(c(X, y), c(N, ncol(X) + 1L), list(row.names(dataMod), c(colnames(X), :
length of 'dimnames' [1] not equal to array extent

8 Inference

In this section we will look at inference for `gls` models with the following example:

```
e.gls <- gls(cholesterol ~ group + time,  
               data = dfL.data,  
               correlation = corSymm(form =~ as.numeric(time)|id),  
               weights = varIdent(form =~ 1|time),  
               na.action = na.omit)  
  
logLik(e.gls)
```

'log Lik.' -2147.318 (df=21)

but the same methods could be used for `lme` models, e.g.:

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

'log Lik.' -2147.318 (df=22)

8.1 Univariate Wald tests: model parameters

To obtain p-values corresponding to testing whether a given model parameter is 0, we can call the summary function:

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

	Value	Std.Error	t-value	p-value
(Intercept)	229.7554614	6.580639	34.91385520	5.156557e-129
groupT	0.3417334	7.773553	0.04396103	9.649554e-01
timevisit1	14.6310680	3.075035	4.75801728	2.653785e-06
timevisit2	18.6356997	3.388140	5.50027401	6.435953e-08
timevisit3	26.3411516	3.436471	7.66517440	1.148252e-13
timevisit4	25.6064656	4.831008	5.30043937	1.829541e-07

Confidence intervals can be obtained by calling `intervals`:

```
intervals(e.gls, which = "coef")
```

Approximate 95% confidence intervals

Coefficients:

	lower	est.	upper
(Intercept)	216.822152	229.7554614	242.68877
groupT	-14.936080	0.3417334	15.61955
timevisit1	8.587525	14.6310680	20.67461
timevisit2	11.976792	18.6356997	25.29461
timevisit3	19.587256	26.3411516	33.09505
timevisit4	16.111806	25.6064656	35.10113

`attr(,"label")`

[1] "Coefficients:"

For `lme` models the argument `which` should be fixed to `fixed` instead:

```
intervals(e.lme, which = "fixed")
```

Approximate 95% confidence intervals

Fixed effects:

	lower	est.	upper
(Intercept)	216.811486	229.7554734	242.69946
groupT	-15.079092	0.3417135	15.76252
timevisit1	8.582571	14.6310680	20.67956
timevisit2	11.971388	18.6356937	25.30000
timevisit3	19.581788	26.3411636	33.10054
timevisit4	16.104114	25.6064781	35.10884

`attr(,"label")`

[1] "Fixed effects:"

8.2 Univariate Wald tests: linear combinaison of parameters

We will use the package multcomp to test linear combinaisons of coefficients:

```
library(multcomp)
```

We first need to define a contrast matrix to indicate which test we want to perform. We first create the contrast matrix with as many rows as tests and as many columns as model parameters. For instance for one test:

```
name.coef <- names(coef(e.gls))
n.coef <- length(name.coef)
C <- matrix(0, nrow = 1, ncol = n.coef,
            dimnames = list("timevisit3-timevisit4", name.coef))
C
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4
timevisit3-timevisit4          0      0      0      0      0      0
```

We want to test whether the change in cholesterol from baseline to visit 3 is the same as the change in cholesterol from baseline to visit 4, i.e. $\beta_{v4} - \beta_{v3} = 0$. So we fill accordingly the constraint matrix:

```
C[1,"timevisit3"] <- -1
C[1,"timevisit4"] <- 1
C
```

```
(Intercept) groupT timevisit1 timevisit2 timevisit3 timevisit4
timevisit3-timevisit4          0      0      0      0      -1      1
```

We now call glht:

```
e.glht <- glht(e.gls, linfct = C)
e.glht
```

General Linear Hypotheses

Linear Hypotheses:

	Estimate
timevisit3-timevisit4 == 0	-0.7347

and use `summary` to obtain p-values:

```
summary(e.glht)
```

Simultaneous Tests for General Linear Hypotheses

Fit: `gls(model = cholesterol ~ group + time, data = dfL.data, correlation = corSymm(form = id), weights = varIdent(form = ~1 | time), na.action = na.omit)`

Linear Hypotheses:

	Estimate	Std. Error	z value	Pr(> z)
<code>timevisit3-timevisit4 == 0</code>	-0.7347	4.6025	-0.16	0.873

(Adjusted p values reported -- single-step method)

and `confint` to obtain confidence intervals:

```
confint(e.glht)
```

Simultaneous Confidence Intervals

Fit: `gls(model = cholesterol ~ group + time, data = dfL.data, correlation = corSymm(form = id), weights = varIdent(form = ~1 | time), na.action = na.omit)`

Quantile = 1.96

95% family-wise confidence level

Linear Hypotheses:

	Estimate	lwr	upr
<code>timevisit3-timevisit4 == 0</code>	-0.7347	-9.7554	8.2861

Note that by default, `summary` and `confint` used on `glht` object display p-values and confidence intervals adjusted for multiple comparisons when doing several tests (i.e. the contrast matrix contains several rows).

8.3 Multivariate Wald test (also called F-test)

The `anova` function can be used to test whether any of the level of a categorical variable has an effect. For instance in:

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

```
Denom. DF: 441
      numDF   F-value p-value
(Intercept)     1 1218.9773 <.0001
group           1    0.0019  0.965
time            4   16.4851 <.0001
```

the line `time` corresponds to testing the null hypothesis: $\beta_{v1} = 0$ or $\beta_{v2} = 0$ or $\beta_{v3} = 0$ or $\beta_{v4} = 0$. Note that we could also have used a contrast matrix:

```
C <- matrix(0, nrow = 4, ncol = n.coef,
            dimnames = list(paste0("timevisit", 1:4), name.coef))
C["timevisit1", "timevisit1"] <- 1
C["timevisit2", "timevisit2"] <- 1
C["timevisit3", "timevisit3"] <- 1
C["timevisit4", "timevisit4"] <- 1
C
```

	(Intercept)	groupT	timevisit1	timevisit2	timevisit3	timevisit4
timevisit1	0	0	1	0	0	0
timevisit2	0	0	0	1	0	0
timevisit3	0	0	0	0	1	0
timevisit4	0	0	0	0	0	1

to perform the F-test:

```
anova(e.lme, L = C)
```

```
F-test for linear combination(s)
      timevisit1 timevisit2 timevisit3 timevisit4
timevisit1      1      0      0      0
timevisit2      0      1      0      0
timevisit3      0      0      1      0
timevisit4      0      0      0      1
      numDF denDF   F-value p-value
1       4    340 16.48534 <.0001
```

Note that when calling `anova` without contrast matrix setting the argument `type` to "marginal" is important since otherwise `anova` outputs the result for a sequential anova (see appendix A for more details).

8.4 Inference in small samples

The p-values and confidence intervals that we have seen in the previous sub-section rely on asymptotic results. They may not be very accurate in small samples. Small depends both on the number of observations (n) and the number of model parameters (p): when $n - p$ is large then asymptotic results can be trusted. Otherwise resampling techniques (permutations, bootstrap) are often more reliable.

`gls` and `lme` use two corrections to improve the behavior of the p-values/confidence interval in small samples:

- REML estimation (Restricted Maximum Likelihood) instead of ML estimation (Maximum Likelihood). This is the default `method` option. This greatly reduce the bias of the variance estimates in small samples and is the analogue of using $\frac{1}{n-p} \sum_{i=1}^n \varepsilon_i^2$ instead of $\frac{1}{n} \sum_{i=1}^n \varepsilon_i^2$ when estimating the variance of the residuals.
- Modeling the distribution of the model parameters using a Student's t-distribution instead of a normal distribution. Unfortunately `gls` and `lme` use a very crude approximation of the degrees of freedom.

8.4.1 Kenward Roger like approximation when using `gls`

A better approximation of the degree of freedom can be obtained using `lavaSearch2`. It has been implemented for maximum likelihood (ML) estimation and only for 2 type of covariance structure: compound symmetry and unstructured. We first need to re-estimate the model using ML:

```
e.glsUN <- gls(cholesterol ~ group + time,
                  data = dfL.data,
                  correlation = corSymm(form =~ as.numeric(time)|id),
                  weights = varIdent(form =~ 1|time),
                  na.action = na.omit,
                  method = "ML")

e.glsCS <- gls(model = cholesterol ~ group + time,
                  data = dfL.data,
                  correlation = corCompSymm(form =~ 1|id),
                  na.action = na.omit, method = "ML")
```

We then load the package `lavaSearch2`:

```
library(lavaSearch2)
```

[Optional] To estimate the degrees of freedom once for all we use (otherwise they are recomputed at each call of `summary2`):

```
sCorrect(e.glsUN) <- TRUE
sCorrect(e.glsCS) <- TRUE
```

and then display degrees of freedom using `summary2`:

```
eS.glsUN <- summary2(e.glsUN)$tTable  
eS.glsUN
```

	Value	Std.Error	t-value	p-value	df
(Intercept)	229.7564242	6.581234	34.91083997	0.000000e+00	120.67094
groupT	0.3401341	7.774946	0.04374745	9.651928e-01	100.20797
timevisit1	14.6310680	3.075032	4.75802141	6.468861e-06	101.84406
timevisit2	18.6360287	3.387863	5.50082192	2.995929e-07	98.38702
timevisit3	26.3422067	3.435589	7.66745098	1.792566e-11	91.66521
timevisit4	25.6067428	4.830066	5.30153039	9.572590e-07	81.70679

For a compound symmetry structure:

```
eS.glsCS <- summary2(e.glsCS)$tTable  
eS.glsCS
```

	Value	Std.Error	t-value	p-value	df
(Intercept)	230.970133	6.465397	35.7240458	0.000000e+00	124.4700
groupT	-1.676188	7.908753	-0.2119409	8.325791e-01	101.2702
timevisit1	14.631068	3.315299	4.4131974	1.371287e-05	338.4192
timevisit2	18.572590	3.437637	5.4027202	1.234016e-07	341.1960
timevisit3	26.398734	3.637217	7.2579481	2.639666e-12	343.5816
timevisit4	25.447831	3.810489	6.6783634	9.682832e-11	345.0494

It is also possible to perform F-test with `lavaSearch2` using the method `compare2`, e.g.:

```
compare2(e.glsUN, par = paste0("timevisit",1:4))
```

- Wald test -

Null Hypothesis:
[timevisit1] = 0
[timevisit2] = 0
[timevisit3] = 0
[timevisit4] = 0

data:
F-statistic = 16.496, df1 = 4, df2 = 88.75, p-value = 3.861e-10
sample estimates:

	Estimate	Std.Err	df	2.5%	97.5%
[timevisit1]	= 0	14.63107	3.075032	101.84406	8.531645 20.73049
[timevisit2]	= 0	18.63603	3.387863	98.38702	11.913256 25.35880
[timevisit3]	= 0	26.34221	3.435589	91.66521	19.518499 33.16591
[timevisit4]	= 0	25.60674	4.830066	81.70679	15.997686 35.21580

8.4.2 Kenward Roger approximation when using lme4

It is also possible to use the `lme4` package in conjunction with the `lmeTest` package to obtain appropriate inference in small sample size. First we load the packages:

```
library(lme4)
library(lmerTest)
```

Then instead of using `gls` to fit the model, we use `lmer`. For a compound symmetry structure:

```
e.lmerCS <- lmer(cholesterol ~ group + time + (1|id),
                  data = dfL.data,
                  na.action = na.omit)
logLik(e.lmerCS)
```

'log Lik.' -2160.3 (df=8)

and for a unstructured covariance matrix:

```
e.lmerUN <- lmer(cholesterol ~ group + time + (time|id),
                  data = dfL.data,
                  na.action = na.omit,
                  control = lmerControl(check.nobs.vs.nRE = "ignore"))
logLik(e.lmerUN)
```

Warning messages:

```
1: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
  unable to evaluate scaled gradient
2: In checkConv(attr(opt, "derivs"), opt$par, ctrl = control$checkConv, :
  Model failed to converge: degenerate Hessian with 1 negative eigenvalues
3: Model failed to converge with 1 negative eigenvalue: -9.4e-03
'log Lik.' -2147.32 (df=22)
```

Then calling `summary` automatically display standard errors and p-value with an appropriate small sample correction:

```
eS.lmerCS <- summary(e.lmerCS)$coefficients
eS.lmerCS
```

	Estimate	Std. Error	df	t value	Pr(> t)
(Intercept)	230.970716	6.465730	124.4835	35.7222978	2.788565e-67
groupT	-1.677156	7.907752	101.2479	-0.2120902	8.324631e-01
timevisit1	14.631068	3.318446	340.7304	4.4090121	1.393875e-05
timevisit2	18.572149	3.439567	342.9874	5.3995604	1.250298e-07
timevisit3	26.397990	3.637691	344.8045	7.2567994	2.642904e-12
timevisit4	25.447421	3.808877	345.5232	6.6810833	9.508265e-11

```
eS.lmerUN <- summary(e.lmerUN)$coefficients
eS.lmerUN
```

	Estimate	Std. Error	df	t value	Pr(> t)
(Intercept)	229.7659158	6.580373	120.24807	34.91685345	8.722113e-65
groupT	0.3243657	7.773061	101.29547	0.04172947	9.667965e-01
timevisit1	14.6310680	3.073188	101.71042	4.76087702	6.403317e-06
timevisit2	18.6377188	3.389571	102.40600	5.49854834	2.825368e-07
timevisit3	26.3412685	3.437004	93.62130	7.66401965	1.640685e-11
timevisit4	25.5913340	4.829082	83.82463	5.29942008	9.237008e-07

8.4.3 Comparisons between lavaSearch2 and lmerTest

Overall, we recommand the of the package `lmerTest` over `lavaSearch2` for 3 reasons:

- the package `lmerTest` implements the Kenward-Roger approximation which is widely used, tested and known approximation. The package `lavaSearch2` implements a recent adaptation of the Kenward-Roger approximation to maximum likelihood estimation, initially used in latent variable models.
- the package `lmerTest` is older, probably much more widely used and therefore probably more reliable.
- the package `lmerTest` has been developped specifically for working with `lme4` while the package `lavaSearch2` has been developped for `lava` and only works with `nlme` for specific models.

Nevertheless the two packages leads to very similar results when using a unstructured covariance matrix:

```
diff <- list(estimate = data.frame(
  gls = eS.glsUN[,"Value"],
  lmer = eS.lmerUN[,"Estimate"],
  difference = eS.glsUN[,"Value"]-eS.lmerUN[,"Estimate"]
),
  se = data.frame(
  gls = eS.glsUN[,"Std.Error"],
  lmer = eS.lmerUN[,"Std. Error"],
  difference = eS.glsUN[,"Std.Error"]-eS.lmerUN[,"Std.
Error"]),
  df = data.frame(
  gls = eS.glsUN[,"df"],
  lmer = eS.lmerUN[,"df"],
  difference = eS.glsUN[,"df"]-eS.lmerUN[,"df"]
))
diff
```

```

$estimate
      gls      lmer   difference
(Intercept) 229.7564242 229.7659158 -9.491608e-03
groupT       0.3401341  0.3243657  1.576832e-02
timevisit1   14.6310680 14.6310680  3.748113e-13
timevisit2   18.6360287 18.6377188 -1.690066e-03
timevisit3   26.3422067 26.3412685  9.382159e-04
timevisit4   25.6067428 25.5913340  1.540877e-02

$se
      gls      lmer   difference
(Intercept) 6.581234 6.580373  0.0008616049
groupT       7.774946 7.773061  0.0018855202
timevisit1   3.075032 3.073188  0.0018444297
timevisit2   3.387863 3.389571 -0.0017082026
timevisit3   3.435589 3.437004 -0.0014157626
timevisit4   4.830066 4.829082  0.0009842334

$df
      gls      lmer difference
(Intercept) 120.67094 120.24807  0.4228668
groupT       100.20797 101.29547 -1.0874968
timevisit1   101.84406 101.71042  0.1336384
timevisit2   98.38702 102.40600 -4.0189796
timevisit3   91.66521 93.62130 -1.9560942
timevisit4   81.70679 83.82463 -2.1178453

```

or a compound symmetry structure:

```

diff <- list(estimate = data.frame(
  gls = eS.glsCS[, "Value"],
  lmer = eS.lmerCS[, "Estimate"],
  difference = eS.glsCS[, "Value"]-eS.lmerCS[, "Estimate"
]),
  se = data.frame(
  gls = eS.glsCS[, "Std.Error"],
  lmer = eS.lmerCS[, "Std. Error"],
  difference = eS.glsCS[, "Std.Error"]-eS.lmerCS[, "Std.
Error"]),
  df = data.frame(
  gls = eS.glsCS[, "df"],
  lmer = eS.lmerCS[, "df"],
  difference = eS.glsCS[, "df"]-eS.lmerCS[, "df"]
))

```

```
diff
```

```
$estimate
      gls     lmer   difference
(Intercept) 230.970133 230.970716  5.826382e-04
groupT       -1.676188 -1.677156 -9.679312e-04
timevisit1    14.631068 14.631068  2.627232e-12
timevisit2    18.572590 18.572149 -4.404172e-04
timevisit3    26.398734 26.397990 -7.437908e-04
timevisit4    25.447831 25.447421 -4.098973e-04

$se
      gls     lmer   difference
(Intercept) 6.465397 6.465730  0.0003326729
groupT       7.908753 7.907752 -0.0010012280
timevisit1   3.315299 3.318446  0.0031470920
timevisit2   3.437637 3.439567  0.0019301025
timevisit3   3.637217 3.637691  0.0004732225
timevisit4   3.810489 3.808877 -0.0016125864

$df
      gls     lmer   difference
(Intercept) 124.4700 124.4835  0.01345532
groupT       101.2702 101.2479 -0.02226745
timevisit1   338.4192 340.7304  2.31127726
timevisit2   341.1960 342.9874  1.79142699
timevisit3   343.5816 344.8045  1.22288323
timevisit4   345.0494 345.5232  0.47385896
```

9 References

- Bates, D. M. (2010). *lme4: Mixed-effects modeling with R*.
- Galecki, A. and Burzykowski, T. (2013). *Linear Mixed-Effects Models Using R: A Step-by-Step Approach*. Springer.
- Pinheiro, J. C. and Bates, D. M. (2000). *Mixed Effects Models in S and S-PLUS*.

A Sequential vs. marginal anova

We are interested in testing the effect of a variable A, a variable B, and their interaction AB. As an example we consider the models:

```
e.lm <- gls(cholesterol ~ group - 1 + time,  
             data = dfL.data,  
             na.action = na.omit)  
e.lmI <- gls(cholesterol ~ group -1 + time + group:time,  
              data = dfL.data,  
              na.action = na.omit)
```

and denote by $\text{SS}(X)$ the explained sum of squares by the variable X.

A.1 Sequential anova (or type I anova)

This corresponds to the default output of `anova`:

```
anova(e.lmI)
```

```
Denom. DF: 437  
          numDF   F-value p-value  
group        2  6750.314  <.0001  
time         4     5.476  0.0003  
group:time    4     0.478  0.7522
```

Each line, respectively, corresponds to testing:

- $\text{SS}(\text{group})$
- $\text{SS}(\text{time} | \text{group}) = \text{SS}(\text{time}, \text{group}) - \text{SS}(\text{group})$
- $\text{SS}(\text{group:time} | \text{time}, \text{group}) = \text{SS}(\text{group}, \text{time}, \text{group:time}) - \text{SS}(\text{time}, \text{group})$

i.e. it will give different results depending on which main effect is considered first since, for instance, the first test tests the first factor without controlling for the other factor(s).

A.2 Marginal anova without interaction (type II anova)

This corresponds to applying `anova` to a model without interaction, with the argument `type` set to "marginal":

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

```
Denom. DF: 441
      numDF   F-value p-value
group      2 1364.4895 <.0001
time       4     5.5022 2e-04
```

Each line, respectively, corresponds to testing:

- $\text{SS}(\text{group} \mid \text{time})$
- $\text{SS}(\text{time} \mid \text{group})$

i.e. the order of the main effects does not matter, they are tested controlling for the other factor(s).

A.3 Marginal anova with interaction (type III anova)

This corresponds applying `anova` to a model with interaction, with the argument `type` set to "marginal":

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

```
Denom. DF: 437
      numDF   F-value p-value
group      2 1358.6097 <.0001
time       4     1.6403 0.1631
group:time  4     0.4776 0.7522
```

Each line, respectively, corresponds to testing:

- $\text{SS}(\text{group} \mid \text{time}, \text{group:time})$
- $\text{SS}(\text{time} \mid \text{group}, \text{group:time})$
- $\text{SS}(\text{group:time} \mid \text{time}, \text{group})$

i.e. the order of the main effect does not matter, they are tested controlling for the other factor(s).