```
## Loading required package:  doBy
```

Linear estimates and LS–means

Søren Højsgaard and Ulrich Halekoh

doBy version 4.6-1 as of 2018-03-19

Contents

1 Introduction

1.1 Linear functions of parameters

A linear function of a p–dimensional parameter vector β has the form

$$C = L\beta$$

where L is a $q \times p$ matrix which we call the *Linear Estimate Matrix* of simply *LE-matrix*. The corresponding linear estimate is $\hat{C} = L\hat{\beta}$. A linear hypothesis has the form $H_0 : L\beta = m$ for some q dimensional vector m.

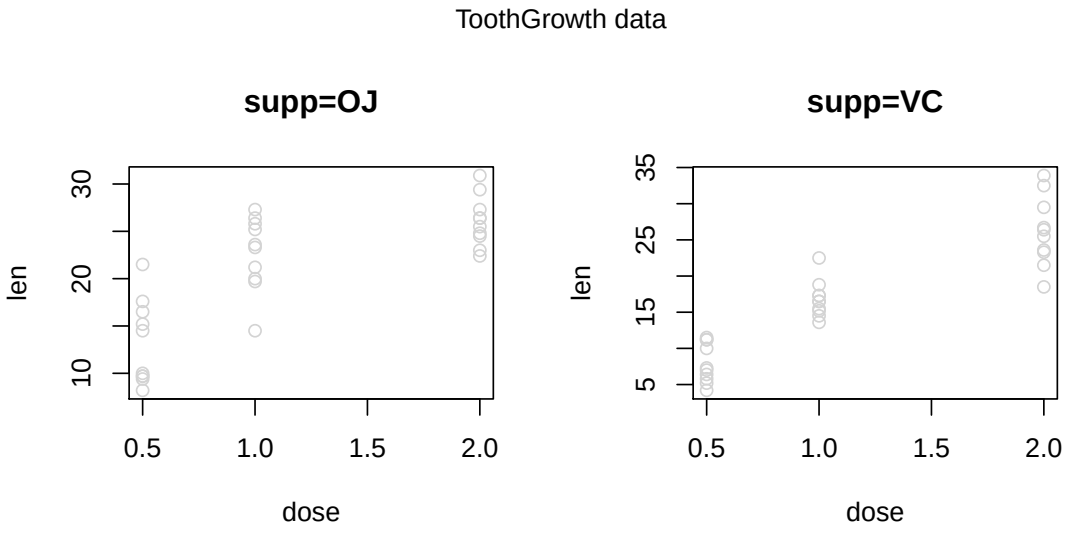

Figure 1: Plot of length against dose for difference sources of vitamin C.

1.2 Tooth growth

The response is the length of odontoblasts cells (cells responsible for tooth growth) in 60 guinea pigs. Each animal received one of three dose levels of vitamin C (0.5, 1, and 2 mg/day) by one of two delivery methods, (orange juice (coded as OJ) or ascorbic acid (a form of vitamin C and (coded as VC)).

```
head(ToothGrowth, 4)

##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5

ftable(xtabs(~ dose + supp, data=ToothGrowth))

##      supp OJ VC
## dose
## 0.5       10 10
## 1         10 10
## 2         10 10
```

The interaction plot suggests a mild interaction which is supported by a formal comparison:

```
ToothGrowth$dose <- factor(ToothGrowth$dose)
head(ToothGrowth)

##    len supp dose
## 1  4.2   VC  0.5
## 2 11.5   VC  0.5
```

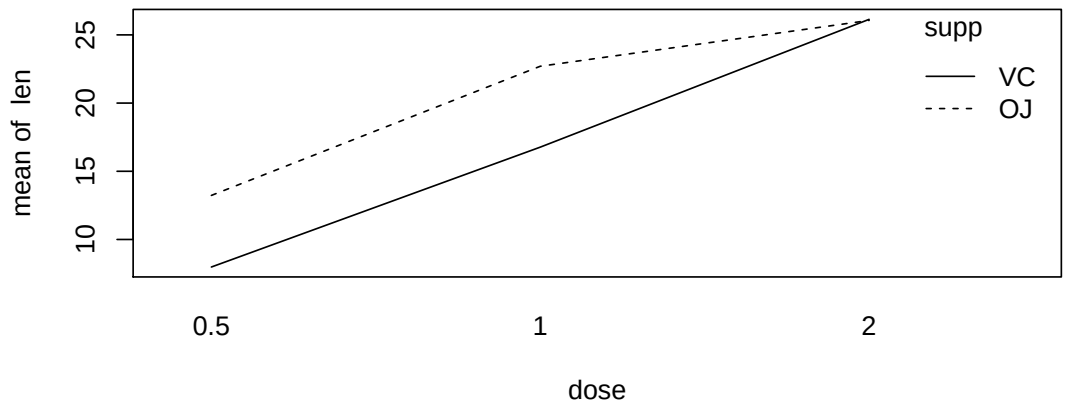

Figure 2: Interaction plot between dose and source of vitamin C.

```
## 3  7.3   VC  0.5
## 4  5.8   VC  0.5
## 5  6.4   VC  0.5
## 6 10.0   VC  0.5

tooth1 <- lm(len ~ dose + supp, data=ToothGrowth)
tooth2 <- lm(len ~ dose * supp, data=ToothGrowth)
anova(tooth1, tooth2)

## Analysis of Variance Table
##
## Model 1: len ~ dose + supp
## Model 2: len ~ dose * supp
##   Res.Df RSS Df Sum of Sq    F Pr(>F)
## 1     56 820
## 2     54 712  2       108 4.11  0.022 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

2 Computing linear estimates

For now, we focus on the additive model:

```
tooth1


##
## Call:
## lm(formula = len ~ dose + supp, data = ToothGrowth)
##
```

```
## Coefficients:
## (Intercept)        dose1        dose2       suppVC
##       12.46         9.13        15.49        -3.70
```

Consider computing the estimated length for each dose of orange juice (OJ): One option: Construct the LE–matrix L directly:

```
L <- matrix(c(1, 0, 0, 0,
              1, 1, 0, 0,
              1, 0, 1, 0), nrow=3, byrow=T)
```

Then do:

```
c1 <- linest(tooth1, L)
c1


## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   12.455  0.988 56.000 12.603       0
## [2,]   21.585  0.988 56.000 21.841       0
## [3,]   27.950  0.988 56.000 28.281       0
```

We can do:

```
summary(c1)

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   12.455  0.988 56.000 12.603       0
## [2,]   21.585  0.988 56.000 21.841       0
## [3,]   27.950  0.988 56.000 28.281       0
##
## Grid:
## NULL
##
## L:
##      [,1] [,2] [,3] [,4]
## [1,]    1    0    0    0
## [2,]    1    1    0    0
## [3,]    1    0    1    0

coef(c1)

##   estimate     se df t.stat   p.value
## 1    12.46 0.9883 56  12.60 5.490e-18
## 2    21.59 0.9883 56  21.84 4.461e-29
## 3    27.95 0.9883 56  28.28 7.627e-35

confint(c1)

##   0.025 0.975
## 1 10.52 14.39
## 2 19.65 23.52
## 3 26.01 29.89
```

3 Automatic generation of L

The matrix L can be generated as follows:

```
L <- LE_matrix(tooth1, effect="dose", at=list(supp="OJ"))
L
```

```
##      (Intercept) dose1 dose2 suppVC
## [1,]           1     0     0      0
## [2,]           1     1     0      0
## [3,]           1     0     1      0
```

3.1 Alternatives

An alternative is to do:

```
c1 <- esticon(tooth1, L)
c1
```

```
##       beta0 Estimate Std.Error t.value     DF Pr(>|t|)  Lower Upper
## [1,]  0.000   12.455     0.988  12.603 56.000    0.000 10.475  14.4
## [2,]  0.000   21.585     0.988  21.841 56.000    0.000 19.605  23.6
## [3,]  0.000   27.950     0.988  28.281 56.000    0.000 25.970  29.9
```

Notice: `esticon` has been in the **doBy** package for many years; `linest` is a newer addition; `esticon` is not actively maintained but remains in **doBy** for historical reasons. Yet another alternative in this case is to generate a new data frame and then invoke predict (but this approach is not generally applicable, see later):

```
nd <- data.frame(dose=c('0.5', '1', '2'), supp='OJ')
nd
```

```
##   dose supp
## 1  0.5   OJ
## 2    1   OJ
## 3    2   OJ
```

```
predict(tooth1, newdata=nd)
```

```
##     1     2     3
## 12.46 21.59 27.95
```

4 Least-squares means (LS–means)

A related question could be: What is the estimated length for each dose if we ignore the source of vitamin C (i.e. whether it is OJ or VC). One approach would be to fit a model in which source does not appear:

```
tooth0 <- update(tooth1, . ~ . - supp)
L0 <- LE_matrix(tooth0, effect="dose")
L0
```

```
##      (Intercept) dose1 dose2
## [1,]           1     0     0
## [2,]           1     1     0
## [3,]           1     0     1
```

```
linest(tooth0, L=L0)
```

```
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.605  0.949 57.000 11.180       0
## [2,]   19.735  0.949 57.000 20.805       0
## [3,]   26.100  0.949 57.000 27.515       0
```

An alternative would be to stick to the original model but compute the estimate for an "average vitamin C source". That would correspond to giving weight 1/2 to each of the two vitamin C source parameters. However, as one of the parameters is already set to zero to obtain identifiability, we obtain the LE–matrix L as

```
L1 <- matrix(c(1, 0, 0, 0.5,
               1, 1, 0, 0.5,
               1, 0, 1, 0.5), nrow=3, byrow=T)
linest(tooth1, L=L1)
```

```
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.605  0.856 56.000 12.391       0
## [2,]   19.735  0.856 56.000 23.058       0
## [3,]   26.100  0.856 56.000 30.495       0
```

Such a particular linear estimate is sometimes called a least-squares mean or an LSmean or a marginal mean. Notice that the parameter estimates under the two approaches are identical. This is is because data is balanced: There are 10 observations per supplementation type. Had data not been balanced, the estimates would in general have been different.

Notice: One may generate L automatically with

```
L1 <- LE_matrix(tooth1, effect="dose")
L1
```

```
##      (Intercept) dose1 dose2 suppVC
## [1,]           1     0     0    0.5
## [2,]           1     1     0    0.5
## [3,]           1     0     1    0.5
```

Notice: One may obtain the LSmean directly as:

```
LSmeans(tooth1, effect="dose")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.605  0.856 56.000 12.391       0
## [2,]   19.735  0.856 56.000 23.058       0
## [3,]   26.100  0.856 56.000 30.495       0
```

which is the same as

```
L <- LE_matrix(tooth1, effect="dose")
le <- linest(tooth1, L=L)
coef(le)
```

For a model with interactions, the LSmeans are

```
LSmeans(tooth2, effect="dose")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.605  0.812 54.000 13.060       0
## [2,]   19.735  0.812 54.000 24.304       0
## [3,]   26.100  0.812 54.000 32.143       0
```

In this case, the LE–matrix is

```
L <- LE_matrix(tooth2, effect="dose")
t(L)

##              [,1] [,2] [,3]
## (Intercept)   1.0  1.0  1.0
## dose1         0.0  1.0  0.0
## dose2         0.0  0.0  1.0
## suppVC        0.5  0.5  0.5
## dose1:suppVC  0.0  0.5  0.0
## dose2:suppVC  0.0  0.0  0.5
```

5 Using the `at=` argument

```
library(ggplot2)
ChickWeight$Diet <- factor(ChickWeight$Diet)
qplot(Time, weight, data=ChickWeight, colour=Chick, facets=~Diet,
      geom=c("point","line"))
```

Consider random regression model:

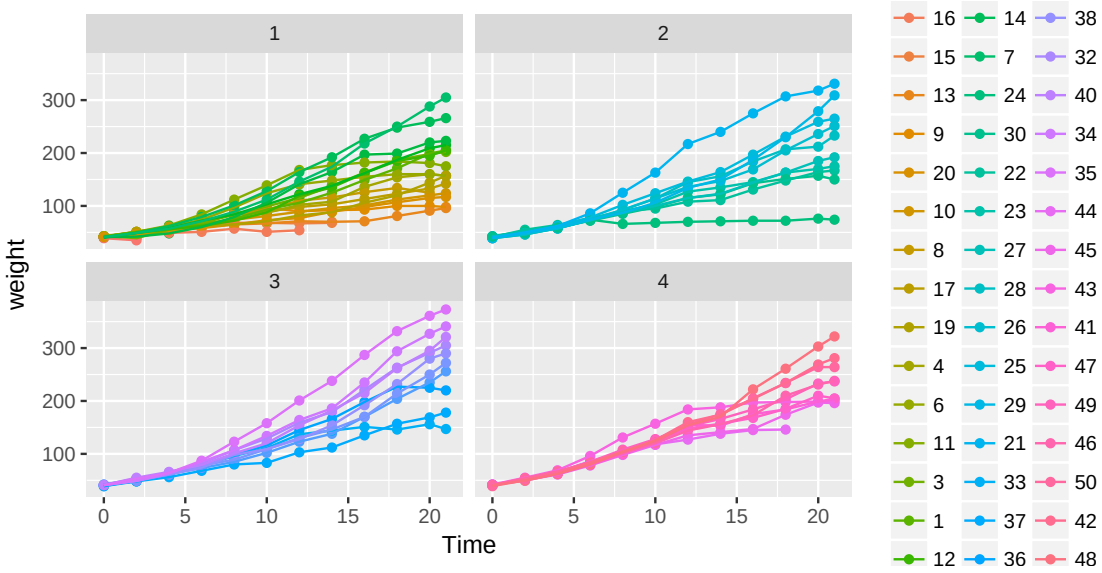

Figure 3: ChickWeight data.

```
library(lme4)

## Loading required package:  Matrix

chick <- lmer(weight ~ Time * Diet + (0 + Time | Chick),
           data=ChickWeight)
coef(summary(chick))

##             Estimate Std. Error t value
## (Intercept)   33.218     1.7697 18.7701
## Time           6.339     0.6103 10.3855
## Diet2         -4.585     3.0047 -1.5258
## Diet3        -14.968     3.0047 -4.9815
## Diet4         -1.454     3.0177 -0.4818
## Time:Diet2     2.271     1.0367  2.1902
## Time:Diet3     5.084     1.0367  4.9043
## Time:Diet4     3.217     1.0377  3.1004
```

The LE–matrix for `Diet` becomes:

```
L <- LE_matrix(chick, effect="Diet")
t(L)

##              [,1]  [,2]  [,3]  [,4]
## (Intercept)  1.00  1.00  1.00  1.00
## Time        10.72 10.72 10.72 10.72
## Diet2        0.00  1.00  0.00  0.00
## Diet3        0.00  0.00  1.00  0.00
## Diet4        0.00  0.00  0.00  1.00
## Time:Diet2   0.00 10.72  0.00  0.00
## Time:Diet3   0.00  0.00 10.72  0.00
```

```
## Time:Diet4   0.00  0.00  0.00 10.72
```

The value of `Time` is by default taken to be the average of that variable. Hence the `LSmeans` is the predicted weight for each diet at that specific point of time. We can consider other points of time with

```
K1 <- LE_matrix(chick, effect="Diet", at=list(Time=1))
t(K1)

##             [,1] [,2] [,3] [,4]
## (Intercept)    1    1    1    1
## Time           1    1    1    1
## Diet2          0    1    0    0
## Diet3          0    0    1    0
## Diet4          0    0    0    1
## Time:Diet2     0    1    0    0
## Time:Diet3     0    0    1    0
## Time:Diet4     0    0    0    1
```

The `LSmeans` for the intercepts is the predictions at `Time=0`. The `LSmeans` for the slopes becomes

```
K0 <- LE_matrix(chick, effect="Diet", at=list(Time=0))
t(K1 - K0)

##             [,1] [,2] [,3] [,4]
## (Intercept)    0    0    0    0
## Time           1    1    1    1
## Diet2          0    0    0    0
## Diet3          0    0    0    0
## Diet4          0    0    0    0
## Time:Diet2     0    1    0    0
## Time:Diet3     0    0    1    0
## Time:Diet4     0    0    0    1

linest(chick, L=K1-K0)

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]    6.339  0.610 49.855 10.383       0
## [2,]    8.609  0.838 48.282 10.273       0
## [3,]   11.423  0.838 48.282 13.631       0
## [4,]    9.556  0.839 48.565 11.386       0
```

We can cook up our own function for comparing trends:

```
LSmeans_trend <- function(object, effect, trend){
    L <- LE_matrix(object, effect=effect, at=as.list(setNames(1, trend))) -
        LE_matrix(object, effect=effect, at=as.list(setNames(0, trend)))
    linest(object, L=L)
}
LSmeans_trend(chick, effect="Diet", trend="Time")
```

```
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]    6.339  0.610 49.855 10.383       0
## [2,]    8.609  0.838 48.282 10.273       0
## [3,]   11.423  0.838 48.282 13.631       0
## [4,]    9.556  0.839 48.565 11.386       0
```

6 Using (transformed) covariates

Consider the following subset of the `CO2` dataset:

```
data(CO2)
CO2 <- transform(CO2, Treat=Treatment, Treatment=NULL)
levels(CO2$Treat) <- c("nchil","chil")
levels(CO2$Type) <- c("Que","Mis")
ftable(xtabs( ~ Plant + Type + Treat, data=CO2), col.vars=2:3)
```

```
##       Type    Que        Mis
##       Treat nchil chil nchil chil
## Plant
## Qn1             7    0     0    0
## Qn2             7    0     0    0
## Qn3             7    0     0    0
## Qc1             0    7     0    0
## Qc3             0    7     0    0
## Qc2             0    7     0    0
## Mn3             0    0     7    0
## Mn2             0    0     7    0
## Mn1             0    0     7    0
## Mc2             0    0     0    7
## Mc3             0    0     0    7
## Mc1             0    0     0    7
```

```
qplot(x=log(conc), y=uptake, data=CO2, color=Treat, facets=~Type)
```

Below, the covariate `conc` is fixed at the average value:

```
co2.lm1 <- lm(uptake ~ conc + Type + Treat, data=CO2)
LSmeans(co2.lm1, effect="Treat")
```

```
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   30.643  0.956 80.000 32.066       0
## [2,]   23.783  0.956 80.000 24.888       0
```

If we use `log(conc)` instead we will get an error when calculating LS–means:

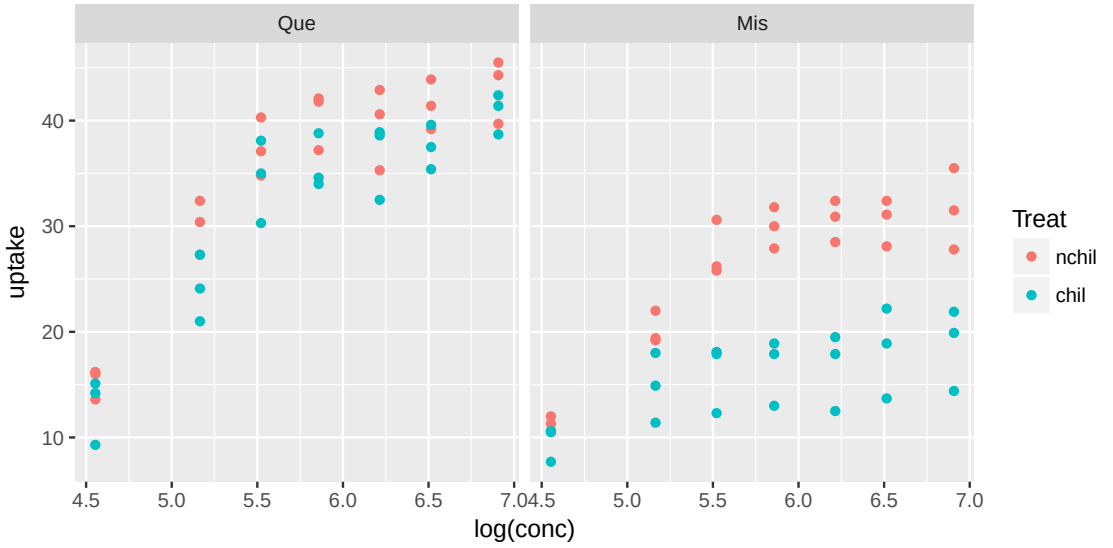

Figure 4: CO2 data

```
co2.lm <- lm(uptake ~ log(conc) + Type + Treat, data=CO2)
LSmeans(co2.lm, effect="Treat")
```

In this case one can do

```
co2.lm2 <- lm(uptake ~ log.conc + Type + Treat,
              data=transform(CO2, log.conc=log(conc)))
LSmeans(co2.lm2, effect="Treat")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   30.643  0.761 80.000 40.261       0
## [2,]   23.783  0.761 80.000 31.248       0
```

This also highlights what is computed: The average of the log of `conc`; not the log of the average of `conc`.

In a similar spirit consider

```
co2.lm3 <- lm(uptake ~ conc + I(conc^2) + Type + Treat, data=CO2)
LSmeans(co2.lm3, effect="Treat")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   34.543  0.982 79.000 35.191       0
## [2,]   27.683  0.982 79.000 28.202       0
```

Above `I(conc^2)` is the average of the squared values of `conc`; not the square of the average of `conc`, cfr. the following.

```
co2.lm4 <- lm(uptake ~ conc + conc2 + Type + Treat, data=
              transform(CO2, conc2=conc^2))
LSmeans(co2.lm4, effect="Treat")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   30.643  0.776 79.000 39.465       0
## [2,]   23.783  0.776 79.000 30.630       0
```

If we want to evaluate the LS–means at `conc=10` then we can do:

```
LSmeans(co2.lm4, effect="Treat", at=list(conc=10, conc2=100))

## Coefficients:
##      estimate    se    df t.stat p.value
## [1,]    14.74  1.70 79.00   8.66       0
## [2,]     7.88  1.70 79.00   4.63       0
```

7 Alternative models

7.1 Generalized linear models

We can calculate LS–means for e.g. a Poisson or a gamma model. Default is that the calculation is calculated on the scale of the linear predictor. However, if we think of LS–means as a prediction on the linear scale one may argue that it can also make sense to transform this prediction to the response scale:

```
tooth.gam <- glm(len ~ dose + supp, family=Gamma, data=ToothGrowth)
LSmeans(tooth.gam, effect="dose", type="link")

## Coefficients:
##      estimate       se       df   t.stat p.value
## [1,]  0.09453  0.00579 56.00000 16.33340       0
## [2,]  0.05111  0.00312 56.00000 16.39673       0
## [3,]  0.03889  0.00238 56.00000 16.36460       0

LSmeans(tooth.gam, effect="dose", type="response")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.578  0.648 56.000 16.333       0
## [2,]   19.565  1.193 56.000 16.397       0
## [3,]   25.711  1.571 56.000 16.365       0
```

7.2 Linear mixed effects model

For the sake of illustration we treat `supp` as a random effect:

```
library(lme4)
tooth.mm <- lmer( len ~ dose  + (1|supp), data=ToothGrowth)
LSmeans(tooth1, effect="dose")

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]   10.605  0.856 56.000 12.391       0
## [2,]   19.735  0.856 56.000 23.058       0
## [3,]   26.100  0.856 56.000 30.495       0

LSmeans(tooth.mm, effect="dose")

## Coefficients:
##      estimate    se    df t.stat p.value
## [1,]    10.61  1.98  1.31   5.36    0.08
## [2,]    19.74  1.98  1.31   9.98    0.03
## [3,]    26.10  1.98  1.31  13.20    0.02
```

Notice here that the estimates themselves identical to those of a linear model (that is not generally the case, but it is so here because data is balanced). In general the estimates are will be very similar but the standard errors are much larger under the mixed model. This comes from that there that `supp` is treated as a random effect.

```
VarCorr(tooth.mm)

##  Groups   Name        Std.Dev.
##  supp     (Intercept) 2.52
##  Residual             3.83
```

Notice that the degrees of freedom by default are adjusted using a Kenward–Roger approximation (provided that **pbkrtest** is installed). Unadjusted degrees of freedom are obtained by setting `adjust.df=FALSE`.

7.3 Generalized estimating equations

Lastly, for gee-type "models" we get

```
library(geepack)
tooth.gee <- geeglm(len ~ dose, id=supp, family=Gamma, data=ToothGrowth)
LSmeans(tooth.gee, effect="dose")

## Coefficients:
##      estimate       se   z.stat p.value
## [1,] 9.43e-02 1.65e-02 5.71e+00       0
## [2,] 5.07e-02 5.38e-03 9.41e+00       0
## [3,] 3.83e-02 4.15e-05 9.23e+02       0

LSmeans(tooth.gee, effect="dose", type="response")

## Coefficients:
```

```
##      estimate       se   z.stat p.value
## [1,]  10.6050   1.8562   5.7134       0
## [2,]  19.7350   2.0966   9.4130       0
## [3,]  26.1000   0.0283 922.7743       0
```

8 Miscellaneous

8.1 Example: Non–estimable linear functions

```
## Make balanced dataset
dat.bal <- expand.grid(list(AA=factor(1:2), BB=factor(1:3), CC=factor(1:3)))
dat.bal$y <- rnorm(nrow(dat.bal))

## Make unbalanced dataset:  'BB' is nested within 'CC' so BB=1
## is only found when CC=1 and BB=2,3 are found in each CC=2,3,4
dat.nst <- dat.bal
dat.nst$CC <-factor(c(1,1,2,2,2,2,1,1,3,3,3,3,1,1,4,4,4,4))
```

```
dat.nst

##    AA BB CC          y
## 1   1  1  1 -1.1507737
## 2   2  1  1 -0.6733382
## 3   1  2  2 -2.0077530
## 4   2  2  2  1.6334920
## 5   1  3  2  0.0008091
## 6   2  3  2  0.2619495
## 7   1  1  1 -1.2697396
## 8   2  1  1 -0.3003146
## 9   1  2  3  0.3868349
## 10  2  2  3  0.8972083
## 11  1  3  3  0.4678174
## 12  2  3  3 -1.4269219
## 13  1  1  1 -1.0965664
## 14  2  1  1  1.1144548
## 15  1  2  4  1.1372470
## 16  2  2  4  0.1341559
## 17  1  3  4  0.1319822
## 18  2  3  4  0.5084049
```

Consider this simulated dataset:

```
head(dat.nst, 4)

##   AA BB CC       y
## 1  1  1  1 -1.1508
## 2  2  1  1 -0.6733
## 3  1  2  2 -2.0078
## 4  2  2  2  1.6335
```

```
ftable(xtabs( ~ AA + BB + CC, data=dat.nst), row.vars="AA")

##    BB 1       2       3
##    CC 1 2 3 4 1 2 3 4 1 2 3 4
## AA
## 1     3 0 0 0 0 1 1 1 0 1 1 1
## 2     3 0 0 0 0 1 1 1 0 1 1 1
```

Data is highly "unbalanced": Whenever `BB=1` then `CC` is always `1`; whenever `BB` is not `1` then `CC` is never `1`. We have

```
mod.nst  <- lm(y ~ AA + BB : CC, data=dat.nst)
coef(summary(mod.nst))

##             Estimate Std. Error  t value Pr(>|t|)
## (Intercept)   0.0119     0.7985  0.01491   0.9884
## AA2           0.6166     0.5050  1.22093   0.2501
## BB1:CC1      -0.8829     0.8747 -1.00938   0.3366
## BB2:CC2      -0.5073     1.0713 -0.47357   0.6460
## BB3:CC2      -0.1888     1.0713 -0.17625   0.8636
## BB2:CC3       0.3218     1.0713  0.30041   0.7700
## BB3:CC3      -0.7997     1.0713 -0.74653   0.4725
## BB2:CC4       0.3155     1.0713  0.29451   0.7744
```

In this case some of the `LSmeans` values are not estimable; for example:

```
lsm.BC <- LSmeans(mod.nst, effect=c("BB", "CC"))
lsm.BC

## Coefficients:
##       estimate     se     df t.stat p.value
##  [1,]   -0.563  0.437 10.000 -1.287    0.23
##  [2,]       NA     NA     NA     NA      NA
##  [3,]       NA     NA     NA     NA      NA
##  [4,]       NA     NA     NA     NA      NA
##  [5,]   -0.187  0.758 10.000 -0.247    0.81
##  [6,]    0.131  0.758 10.000  0.173    0.87
##  [7,]       NA     NA     NA     NA      NA
##  [8,]    0.642  0.758 10.000  0.848    0.42
##  [9,]   -0.480  0.758 10.000 -0.633    0.54
## [10,]       NA     NA     NA     NA      NA
## [11,]    0.636  0.758 10.000  0.839    0.42
## [12,]    0.320  0.758 10.000  0.423    0.68

lsm.BC2 <- LSmeans(mod.nst, effect="BB", at=list(CC=2))
lsm.BC2

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]       NA     NA     NA     NA      NA
## [2,]   -0.187  0.758 10.000 -0.247    0.81
## [3,]    0.131  0.758 10.000  0.173    0.87
```

We describe the situation in Section 8.2 where we focus on `lsm.BC2`.

8.2 Handling non–estimability

The model matrix for the model in Section 8.1 does not have full column rank and therefore not all values are calculated by `LSmeans()`.

```
X <- model.matrix( mod.nst )
Matrix::rankMatrix(X)

## [1] 8
## attr(,"method")
## [1] "tolNorm2"
## attr(,"useGrad")
## [1] FALSE
## attr(,"tol")
## [1] 3.997e-15

dim(X)

## [1] 18 14

as(X, "Matrix")

## 18 x 14 sparse Matrix of class "dgCMatrix"

##    [[ suppressing 14 column names '(Intercept)', 'AA2', 'BB1:CC1' ...  ]]

##
## 1  1 . 1 . . . . . . . . . . .
## 2  1 1 1 . . . . . . . . . . .
## 3  1 . . . . . 1 . . . . . . .
## 4  1 1 . . . . 1 . . . . . . .
## 5  1 . . . . . . 1 . . . . . .
## 6  1 1 . . . . . 1 . . . . . .
## 7  1 . 1 . . . . . . . . . . .
## 8  1 1 1 . . . . . . . . . . .
## 9  1 . . . . . . . . 1 . . . .
## 10 1 1 . . . . . . . 1 . . . .
## 11 1 . . . . . . . . . 1 . . .
## 12 1 1 . . . . . . . . 1 . . .
## 13 1 . 1 . . . . . . . . . . .
## 14 1 1 1 . . . . . . . . . . .
## 15 1 . . . . . . . . . . . 1 .
## 16 1 1 . . . . . . . . . . 1 .
## 17 1 . . . . . . . . . . . . 1
## 18 1 1 . . . . . . . . . . . 1
```

We consider a model, i.e. an n dimensional random vector $y = (y_i)$ for which $\mathbb{E}(y) = \mu = X\beta$ and $\mathbb{C}\text{ov}(y) = V$ where X does not have full column rank We are interested in linear functions of β,

say

$$c = l^\top \beta = \sum_j l_j \beta_j.$$

```
L <- LE_matrix(mod.nst, effect="BB", at=list(CC=2))
t(L)
```

```
##             [,1] [,2] [,3]
## (Intercept)  1.0  1.0  1.0
## AA2          0.5  0.5  0.5
## BB1:CC1      0.0  0.0  0.0
## BB2:CC1      0.0  0.0  0.0
## BB3:CC1      0.0  0.0  0.0
## BB1:CC2      1.0  0.0  0.0
## BB2:CC2      0.0  1.0  0.0
## BB3:CC2      0.0  0.0  1.0
## BB1:CC3      0.0  0.0  0.0
## BB2:CC3      0.0  0.0  0.0
## BB3:CC3      0.0  0.0  0.0
## BB1:CC4      0.0  0.0  0.0
## BB2:CC4      0.0  0.0  0.0
## BB3:CC4      0.0  0.0  0.0
```

```
linest(mod.nst, L=L)
```

```
## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]       NA     NA     NA     NA      NA
## [2,]   -0.187  0.758 10.000 -0.247    0.81
## [3,]    0.131  0.758 10.000  0.173    0.87
```

A least squares estimate of β is

$$\hat{\beta} = GX^\top y$$

where G is a generalized inverse of $X^\top X$. Since the generalized inverse is not unique then neither is the estimate $\hat{\beta}$. Hence $\hat{c} = l^\top \hat{\beta}$ is in general not unique.

One least squares estimate of β and one corresponding linear estimate $L\hat{\beta}$ is:

```
XtXinv <- MASS::ginv(t(X)%*%X)
bhat <- as.numeric(XtXinv %*% t(X) %*% dat.nst$y)
zapsmall(bhat)
```

```
##  [1] -0.2073  0.6166 -0.6637  0.0000  0.0000  0.0000 -0.2882  0.0304  0.0000  0.5410
## [11] -0.5806  0.0000  0.5347  0.2192
```

```
L %*% bhat
```

```
##         [,1]
## [1,]  0.1010
## [2,] -0.1871
## [3,]  0.1314
```

For some values of l (i.e. for some rows of L) the estimate $\hat{c} = l^\top \beta$ is unique (i.e. it does not depend on the choice of generalized inverse). Such linear functions are said to be estimable and can be described as follows:

All we specify with $\mu = X\beta$ is that μ is a vector in the column space $C(X)$ of X. We can only learn about β through $X\beta$ so the only thing we can say something about is linear combinations $\rho^\top X\beta$. Hence we can only say something about $l^\top \beta$ if there exists ρ such that

$$l^\top \beta = \rho^\top X\beta,$$

i.e., if $l = X^\top \rho$ for some ρ, which is if l is in the column space $C(X^\top)$ of $X^\top$. This is the same as saying that l must be perpendicular to all vectors in the null space $N(X)$ of X. To check this, we find a basis B for $N(X)$. This can be done in many ways, for example via a singular value decomposition of X, i.e.

$$X = UDV^\top$$

A basis for $N(X)$ is given by those columns of V that corresponds to zeros on the diagonal of D.

```
S <- svd(X)
B <- S$v[, S$d < 1e-10, drop=FALSE ];
head(B) ## Basis for N(X)

##           [,1]       [,2]       [,3]       [,4]       [,5] [,6]
## [1,]  0.339176 -5.635e-04  9.968e-02 -4.350e-03 -2.274e-03    0
## [2,]  0.000000  1.193e-17 -1.110e-16  1.735e-18  4.337e-19    0
## [3,] -0.339176  5.635e-04 -9.968e-02  4.350e-03  2.274e-03    0
## [4,] -0.272743 -2.494e-01  9.244e-01 -3.167e-03 -9.422e-02    0
## [5,] -0.072691  9.176e-01  2.509e-01 -1.669e-01  2.487e-01    0
## [6,] -0.001889 -9.509e-02  5.169e-02  6.615e-01  7.421e-01    0
```

From

```
rowSums(L %*% B)

## [1]  1.790e+00  1.632e-15 -4.113e-15
```

we conclude that the first row of L is not perpendicular to all vectors in thenull space $N(X)$ whereas the two last rows of L are. Hence these two linear estimates are estimable; their value does not depend on the choice of generalized inverse:

```
lsm.BC2

## Coefficients:
##      estimate     se     df t.stat p.value
## [1,]       NA     NA     NA     NA      NA
## [2,]   -0.187  0.758 10.000 -0.247    0.81
## [3,]    0.131  0.758 10.000  0.173    0.87
```

8.3 Pairwise comparisons

We will just mention that for certain other linear estimates, the matrix L can be generated automatically using `glht()` from the **multcomp** package. For example, pairwise comparisons of all levels of `dose` can be obtained with

```
library("multcomp")

## Loading required package:  mvtnorm
## Loading required package:  survival
## Loading required package:  TH.data
## Loading required package:  MASS
##
## Attaching package:  'TH.data'
## The following object is masked from 'package:MASS':
##
##    geyser

g1 <- glht(tooth1, mcp(dose="Tukey"))
summary( g1 )

##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = len ~ dose + supp, data = ToothGrowth)
##
## Linear Hypotheses:
##               Estimate Std. Error t value Pr(>|t|)
## 1 - 0.5 == 0     9.13       1.21    7.54  < 1e-06 ***
## 2 - 0.5 == 0    15.49       1.21   12.80  < 1e-06 ***
## 2 - 1 == 0       6.37       1.21    5.26  5.6e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

The L matrix is

```
L <- g1$linfct
L

##         (Intercept) dose1 dose2 suppVC
## 1 - 0.5           0     1     0      0
## 2 - 0.5           0     0     1      0
## 2 - 1             0    -1     1      0
## attr(,"type")
## [1] "Tukey"
```

and this matrix can also be supplied to `glht`

```
glht(tooth1, linfct=L)
```