

# Linear estimates and LS-means in the **doBy** package

Søren Højsgaard and Ulrich Halekoh

**doBy** version 4.6.11 as of 2021-11-29

---

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Linear functions of parameters . . . . .	1
1.2	Tooth growth . . . . .	1
<b>2</b>	<b>Computing linear estimates</b>	<b>3</b>
2.1	Automatic generation of $L$ . . . . .	4
2.2	Alternatives - <code>esticon()</code> . . . . .	4
<b>3</b>	<b>Least-squares means (LS-means)</b>	<b>5</b>
3.1	Using the <code>at=</code> argument . . . . .	6
3.2	Ambiguous specification when using the <code>effect</code> and <code>at</code> arguments . . . . .	8
3.3	Using (transformed) covariates . . . . .	8
<b>4</b>	<b>Alternative models</b>	<b>10</b>
4.1	Generalized linear models . . . . .	10
4.2	Linear mixed effects model . . . . .	11
4.3	Generalized estimating equations . . . . .	11
<b>5</b>	<b>Miscellaneous</b>	<b>12</b>
5.1	Example: Non-estimable linear functions . . . . .	12
5.2	Handling non-estimability . . . . .	13
5.3	Pairwise comparisons . . . . .	16
<b>6</b>	<b>LSmeans (population means, marginal means)</b>	<b>17</b>
6.1	A simulated dataset . . . . .	17
6.2	What are these quantities . . . . .	17
6.3	Using <code>POPMATRIX</code> and <code>POPMEANS</code> . . . . .	18

# 1 Introduction

## 1.1 Linear functions of parameters

A linear function of a  $p$ -dimensional parameter vector  $\beta$  has the form

$$C = L\beta$$

where  $L$  is a  $q \times p$  matrix which we call the *Linear Estimate Matrix* or 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  $m$ .

## 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

> ToothGrowth %>% interaction_plot(len ~ dose + supp)
```

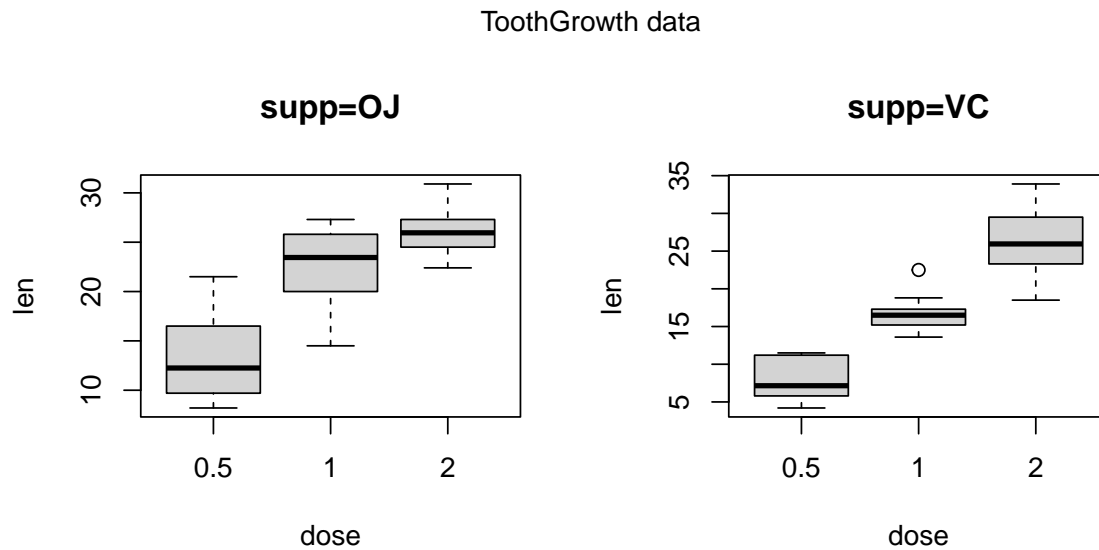
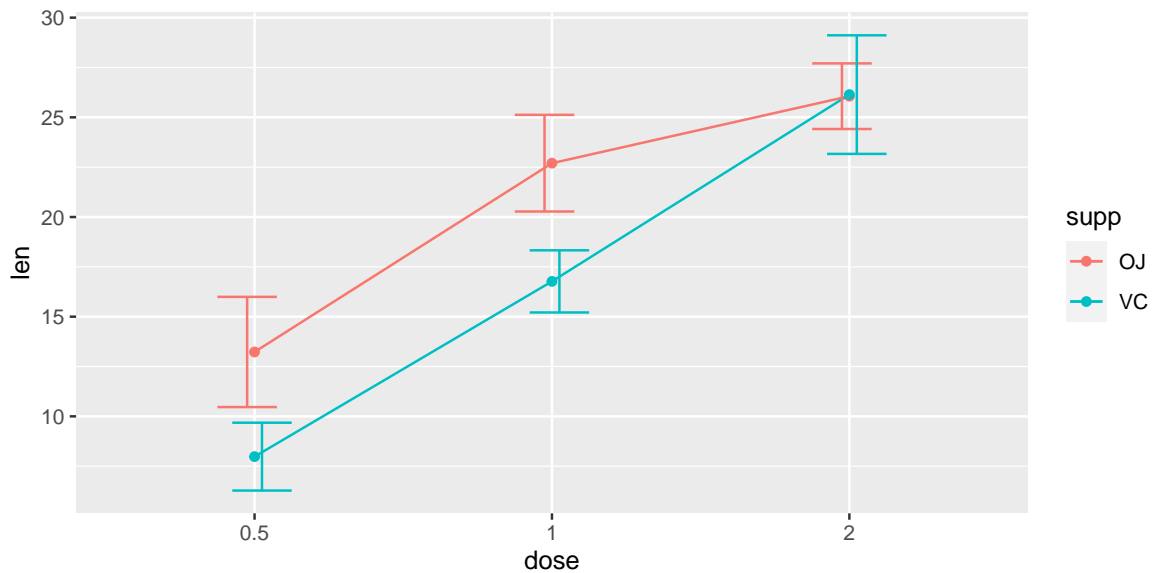


Figure 1: Plot of length against dose for difference sources of vitamin C.



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
## 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 std.error statistic      df p.value
## [1,]  12.455    0.988   12.603 56.000      0
## [2,]  21.585    0.988   21.841 56.000      0
## [3,]  27.950    0.988   28.281 56.000      0

```

We can do:

```

> summary(c1)

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,]  12.455    0.988   12.603 56.000      0
## [2,]  21.585    0.988   21.841 56.000      0
## [3,]  27.950    0.988   28.281 56.000      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 std.error statistic df    p.value
## 1      12.46     0.9883      12.60 56 5.490e-18
## 2      21.59     0.9883      21.84 56 4.461e-29
## 3      27.95     0.9883      28.28 56 7.627e-35

> confint(c1)

##      0.025 0.975
## 1 10.48 14.43
## 2 19.61 23.56
## 3 25.97 29.93
```

## 2.1 Automatic generation of $L$

The matrix  $L$  can be generated as follows:

```
> L <- LE_matrix(tooth1, effect="dose", at=list(supp="0J"))
> L

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

## 2.2 Alternatives - `esticon()`

An alternative is to do:

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

##      estimate std.error statistic p.value  beta0 df
## [1,]  12.455     0.988    12.603   0.000  0.000 56
## [2,]  21.585     0.988    21.841   0.000  0.000 56
## [3,]  27.950     0.988    28.281   0.000  0.000 56
```

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='0J')
> nd

##      dose supp
## 1  0.5    0J
## 2    1    0J
## 3    2    0J
```

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

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

### 3 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 std.error statistic      df p.value
## [1,]   10.605    0.949    11.180 57.000      0
## [2,]   19.735    0.949    20.805 57.000      0
## [3,]   26.100    0.949    27.515 57.000      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 std.error statistic      df p.value
## [1,]   10.605    0.856    12.391 56.000      0
## [2,]   19.735    0.856    23.058 56.000      0
## [3,]   26.100    0.856    30.495 56.000      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 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 std.error statistic      df p.value
## [1,]   10.605    0.856   12.391 56.000      0
## [2,]   19.735    0.856   23.058 56.000      0
## [3,]   26.100    0.856   30.495 56.000      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 std.error statistic      df p.value
## [1,]   10.605    0.812   13.060 54.000      0
## [2,]   19.735    0.812   24.304 54.000      0
## [3,]   26.100    0.812   32.143 54.000      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
```

### 3.1 Using the at= argument

```
> library(ggplot2)
> ChickWeight$Diet <- factor(ChickWeight$Diet)
> qplot(Time, weight, data=ChickWeight, colour=Chick, facets=~Diet,
        geom=c("point", "line")) + theme(legend.position="none")
```

Consider random regression model:

```
> library(lme4)
> 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
```

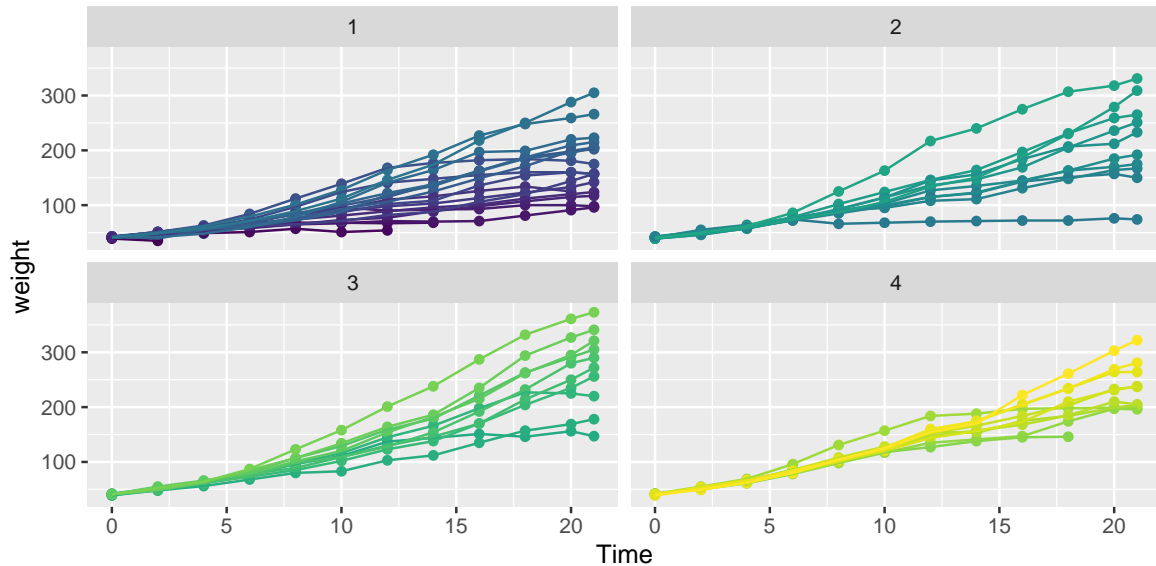


Figure 2: ChickWeight data.

```
## 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 std.error statistic      df p.value
## [1,]    6.339    0.610   10.383 49.855      0
## [2,]    8.609    0.838   10.273 48.282      0
## [3,]   11.423    0.838   13.631 48.282      0
## [4,]    9.556    0.839   11.386 48.565      0
```

We can create 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 std.error statistic      df p.value
## [1,]    6.339    0.610   10.383 49.855      0
## [2,]    8.609    0.838   10.273 48.282      0
## [3,]   11.423    0.838   13.631 48.282      0
## [4,]    9.556    0.839   11.386 48.565      0
```

## 3.2 Ambiguous specification when using the effect and at arguments

## 3.3 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
```

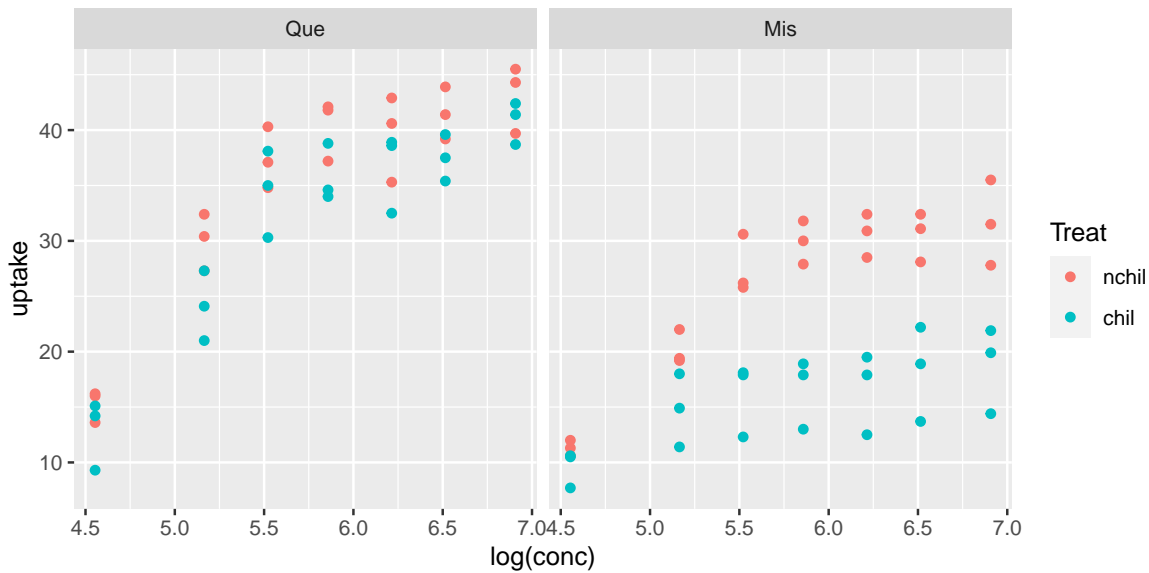


Figure 3: CO2 data

```
## 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=C02, color=Treat, facets=~Type)
```

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

```
> co2.lm1 <- lm(uptake ~ conc + Type + Treat, data=C02)
> LSmeans(co2.lm1, effect="Treat")

## Coefficients:
##      estimate std.error statistic    df p.value
## [1,]   30.643    0.956   32.066 80.000      0
## [2,]   23.783    0.956   24.888 80.000      0
```

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

```
> co2.lm <- lm(uptake ~ log(conc) + Type + Treat, data=C02)
> LSmeans(co2.lm, effect="Treat")
```

In this case one can do

```
> co2.lm2 <- lm(uptake ~ log.conc + Type + Treat,
  data=transform(C02, log.conc=log(conc)))
> LSmeans(co2.lm2, effect="Treat")

## Coefficients:
```

```
##      estimate std.error statistic      df p.value
## [1,]   30.643    0.761   40.261 80.000      0
## [2,]   23.783    0.761   31.248 80.000      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=C02)
> LSmeans(co2.lm3, effect="Treat")

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,]   34.543    0.982   35.191 79.000      0
## [2,]   27.683    0.982   28.202 79.000      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(C02, conc2=conc^2))
> LSmeans(co2.lm4, effect="Treat")

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,]   30.643    0.776   39.465 79.000      0
## [2,]   23.783    0.776   30.630 79.000      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 std.error statistic      df p.value
## [1,]   14.74    1.70    8.66 79.00      0
## [2,]    7.88    1.70    4.63 79.00      0
```

## 4 Alternative models

### 4.1 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 std.error statistic p.value
## [1,]  0.09453   0.00579   16.33340      0
## [2,]  0.05111   0.00312   16.39673      0
## [3,]  0.03889   0.00238   16.36460      0

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

## Coefficients:
```

```
##      estimate std.error statistic p.value
## [1,]  0.09453   0.00579  16.33340      0
## [2,]  0.05111   0.00312  16.39673      0
## [3,]  0.03889   0.00238  16.36460      0
```

## 4.2 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 std.error statistic      df p.value
## [1,]   10.605    0.856   12.391 56.000      0
## [2,]   19.735    0.856   23.058 56.000      0
## [3,]   26.100    0.856   30.495 56.000      0

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

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,]    10.61     1.98     5.36  1.31  0.08
## [2,]    19.74     1.98     9.98  1.31  0.03
## [3,]    26.10     1.98    13.20  1.31  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`.

## 4.3 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 std.error statistic 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 std.error statistic 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
```

## 5 Miscellaneous

### 5.1 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 -0.6565
## 2    2  1  1  0.5437
## 3    1  2  2 -0.4038
## 4    2  2  2  1.0040
## 5    1  3  2 -0.4669
## 6    2  3  2 -0.9808
## 7    1  1  1  0.3871
## 8    2  1  1  0.7652
## 9    1  2  3  1.6487
## 10   2  2  3  0.3072
## 11   1  3  3  1.0357
## 12   2  3  3 -0.6332
## 13   1  1  1 -0.5037
## 14   2  1  1  0.2796
## 15   1  2  4  0.1614
## 16   2  2  4  0.9293
## 17   1  3  4 -0.1734
## 18   2  3  4  1.0296
```

Consider this simulated dataset:

```
> head(dat.nst, 4)

##      AA BB CC      y
## 1    1  1  1 -0.6565
## 2    2  1  1  0.5437
## 3    1  2  2 -0.4038
## 4    2  2  2  1.0040

> 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)   0.3050     0.5698   0.5353  0.6041
## AA2           0.2462     0.3604   0.6832  0.5100
## BB1:CC1      -0.2922     0.6242  -0.4681  0.6497
## BB2:CC2      -0.1280     0.7645  -0.1675  0.8703
## BB3:CC2      -1.1519     0.7645  -1.5069  0.1628
## BB2:CC3       0.5499     0.7645   0.7193  0.4884
## BB3:CC3      -0.2269     0.7645  -0.2967  0.7727
## BB2:CC4       0.1173     0.7645   0.1534  0.8811
```

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 std.error statistic    df p.value
## [1,]    0.136    0.312     0.435 10.000    0.67
## [2,]      NA      NA      NA    NA      NA
## [3,]      NA      NA      NA    NA      NA
## [4,]      NA      NA      NA    NA      NA
## [5,]    0.300    0.541     0.555 10.000    0.59
## [6,]   -0.724    0.541    -1.339 10.000    0.21
## [7,]      NA      NA      NA    NA      NA
## [8,]    0.978    0.541     1.809 10.000    0.10
## [9,]    0.201    0.541     0.372 10.000    0.72
## [10,]     NA      NA      NA    NA      NA
## [11,]    0.545    0.541     1.009 10.000    0.34
## [12,]    0.428    0.541     0.792 10.000    0.45

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

## Coefficients:
##      estimate std.error statistic    df p.value
## [1,]      NA      NA      NA    NA      NA
## [2,]    0.300    0.541     0.555 10.000    0.59
## [3,]   -0.724    0.541    -1.339 10.000    0.21
```

We describe the situation in Section 5.2 where we focus on lsm.BC2.

## 5.2 Handling non-estimability

The model matrix for the model in Section 5.1 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  $\text{Cov}(y) = V$  where  $X$  does not have full column rank We are interested in linear functions of  $\beta$ , 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 std.error statistic      df p.value
## [1,]      NA      NA      NA      NA      NA
## [2,]    0.300    0.541    0.555 10.000    0.59
## [3,]   -0.724    0.541   -1.339 10.000    0.21
```

A least squares estimate of  $\beta$  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  $\beta$  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]  0.1254  0.2462 -0.1126  0.0000  0.0000  0.0000  0.0516 -0.9723  0.0000  0.7295
## [11] -0.0472  0.0000  0.2969  0.1796

> L %*% bhat

##      [,1]
## [1,]  0.2485
## [2,]  0.3001
## [3,] -0.7238
```

For some values of  $l$  (i.e. for some rows of  $L$ ) the estimate  $\hat{c} = l^\top \hat{\beta}$  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  $\mu$  is a vector in the column space  $C(X)$  of  $X$ . We can only learn about  $\beta$  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  $\rho$  such that

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

i.e., if  $l = X^\top \rho$  for some  $\rho$ , 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 the null 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 std.error statistic      df p.value
## [1,]      NA      NA      NA      NA      NA
## [2,]    0.300    0.541    0.555 10.000    0.59
## [3,]   -0.724    0.541   -1.339 10.000    0.21
```

### 5.3 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")
> 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     6.36      1.21   5.26 5.5e-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)
```

## 6 LSmeans (population means, marginal means)

### 6.1 A simulated dataset

In the following sections we consider these data:

```
> library(doby)
> dd <- expand.grid(A=factor(1:3),B=factor(1:3),C=factor(1:2))
> dd$y <- rnorm(nrow(dd))
> dd$x <- rnorm(nrow(dd))^2
> dd$z <- rnorm(nrow(dd))
> head(dd,10)
```

	A	B	C	y	x	z
## 1	1	1	1	-1.49546	0.78288	1.0808
## 2	2	1	1	-0.37917	1.42919	0.3328
## 3	3	1	1	0.30257	0.03586	1.4263
## 4	1	2	1	0.23700	0.20167	0.5502
## 5	2	2	1	-0.14892	0.89591	0.1955
## 6	3	2	1	1.13943	0.04060	1.0272
## 7	1	3	1	-1.20054	0.09584	0.1798
## 8	2	3	1	0.65590	0.19273	-0.9473
## 9	3	3	1	0.04295	1.09377	0.2569
## 10	1	1	2	-1.86757	0.13544	-1.3944

Consider the additive model

$$y_i = \beta_0 + \beta_{A(i)}^1 + \beta_{B(i)}^2 + \beta_{C(i)}^3 + e_i \quad (1)$$

where  $e_i \sim N(0, \sigma^2)$ . We fit this model:

```
> mm <- lm(y~A+B+C, data=dd)
> coef(mm)
```

	(Intercept)	A2	A3	B2	B3	C2
##	-1.3719	0.5688	1.2390	1.3907	0.6351	0.2500

Notice that the parameters corresponding to the factor levels A1, B1 and C2 are set to zero to ensure identifiability of the remaining parameters.

### 6.2 What are these quantities

LSmeans, population means and marginal means are used synonymously in the literature. These quantities are a special kind of contrasts as defined in Section 1.1. LSmeans seems to be the most widely used term, so we shall adopt this terms here too.

The model (1) is a model for the conditional mean  $\mathbb{E}(y|A, B, C)$ . Sometimes one is interested in quantities like  $\mathbb{E}(y|A)$ . This quantity can not formally be found unless  $B$  and  $C$  are random variables such that we may find  $\mathbb{E}(y|A)$  by integration. However, suppose that  $A$  is a treatment of main interest,  $B$  is a blocking factor and  $C$  represents days on which the experiment was carried out. Then it is tempting to average  $\mathbb{E}(y|A, B, C)$  over  $B$  and  $C$  (average over block and day) and think of this average as  $\mathbb{E}(y|A)$ .

The population mean for  $A = 1$  is

$$\beta^0 + \beta_{A1}^1 + \frac{1}{3}(\beta_{B1}^2 + \beta_{B2}^2 + \beta_{B3}^2) + \frac{1}{2}(\beta_{C1}^3 + \beta_{C2}^3) \quad (2)$$

Recall that the parameters corresponding to the factor levels A1, B1 and C2 are set to zero to ensure identifiability of the remaining parameters. Therefore we may also write the population mean for  $A = 1$  as

$$\beta^0 + \frac{1}{3}(\beta_{B2}^2 + \beta_{B3}^2) + \frac{1}{2}(\beta_{C2}^3) \quad (3)$$

This quantity can be estimated as:

@

```
> w <- c(1, 0, 0, 1/3, 1/3, 1/2)
> coef(mm)*w

## (Intercept)      A2      A3      B2      B3      C2
##      -1.3719      0.0000      0.0000      0.4636      0.2117      0.1250

> sum(coef(mm)*w)

## [1] -0.5716
```

We may find the population mean for all three levels of  $A$  as

```
> W <- matrix(c(1, 0, 0, 1/3, 1/3, 1/2,
                1, 1, 0, 1/3, 1/3, 1/2,
                1, 0, 1, 1/3, 1/3, 1/2), nr=3, byrow=TRUE)
```

Notice that the matrix  $W$  is based on that the first level of  $A$  is set as the reference level. If the reference level is changed then so must  $W$  be.

Given that one has specified  $W$ , we can use the `esticon()` function in the `doBy` as illustrated below:

```
> esticon(mm, W)

##      estimate std.error statistic  p.value  beta0 df
## [1,] -0.57163   0.24404  -2.34237  0.03723  0.00000 12
## [2,] -0.00284   0.24404  -0.01163  0.99091  0.00000 12
## [3,]  0.66741   0.24404   2.73481  0.01810  0.00000 12

> popMatrix <- LE_matrix
> popMeans <- LSmeans
```

### 6.3 Using POPMATRIX and POPMEANS

Writing the matrix  $W$  is somewhat tedious and hence error prone. In addition, there is a potential risk of getting the wrong answer if the the reference level of a factor has been

changed. The POPMATRIX function provides an automated way of generating such matrices. The above W matrix is constructed by

```
> pma <- popMatrix(mm, effect='A')
> summary(pma)

##      (Intercept) A2 A3      B2      B3 C2
## [1,]           1  0  0 0.3333 0.3333 0.5
## [2,]           1  1  0 0.3333 0.3333 0.5
## [3,]           1  0  1 0.3333 0.3333 0.5
## at:
## NULL
## grid:
##   A
## 1 1
## 2 2
## 3 3
```

The **effect** argument requires to calculate the population means for each level of *A* aggregating across the levels of the other variables in the data.

The POPMEANS function is simply a wrapper around first a call to POPMATRIX followed by a call to (by default) **esticon()**:

```
> pme <- popMeans(mm, effect='A')
> pme

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,] -0.57163   0.24404  -2.34237 12.00000    0.04
## [2,] -0.00284   0.24404  -0.01163 12.00000    0.99
## [3,]  0.66741   0.24404   2.73481 12.00000    0.02
```

More details about how the matrix was constructed is provided by the **summary()** function:

```
> summary(pme)

## Coefficients:
##      estimate std.error statistic      df p.value
## [1,] -0.57163   0.24404  -2.34237 12.00000    0.04
## [2,] -0.00284   0.24404  -0.01163 12.00000    0.99
## [3,]  0.66741   0.24404   2.73481 12.00000    0.02
##
## Grid:
##   A
## 1 1
## 2 2
## 3 3
##
## L:
##      (Intercept) A2 A3      B2      B3 C2
## [1,]           1  0  0 0.3333 0.3333 0.5
## [2,]           1  1  0 0.3333 0.3333 0.5
## [3,]           1  0  1 0.3333 0.3333 0.5
```

As an additional example we may do:

```
> popMatrix(mm, effect=c('A', 'C'))

##      (Intercept) A2 A3      B2      B3 C2
## [1,]           1  0  0 0.3333 0.3333  0
```

```
## [2,]      1  1  0 0.3333 0.3333  0
## [3,]      1  0  1 0.3333 0.3333  0
## [4,]      1  0  0 0.3333 0.3333  1
## [5,]      1  1  0 0.3333 0.3333  1
## [6,]      1  0  1 0.3333 0.3333  1
```

This gives the matrix for calculating the estimate for each combination of A and C when averaging over B.

Omitting **effect** as in

```
> popMatrix(mm)

##      (Intercept)      A2      A3      B2      B3  C2
## [1,]           1 0.3333 0.3333 0.3333 0.3333 0.5

> popMeans(mm)

## Coefficients:
##      estimate std.error statistic    df p.value
## [1,]    0.031    0.141    0.220 12.000    0.83
```

gives the “total average”.