Using **lsmeans**

Russell V. Lenth
The University of Iowa

March 8, 2017

Abstract

Least-squares means are predictions from a linear model, or averages thereof. They are useful in the analysis of experimental data for summarizing the effects of factors, and for testing linear contrasts among predictions. The **lsmeans** package provides a simple way of obtaining least-squares means and contrasts thereof. It supports many models fitted by R core packages (as well as a few key contributed ones) that fit linear or mixed models, and provides a simple way of extending it to cover more model classes.

1 Introduction

least-squares means (LS means for short) for a linear model are simply predictions—or averages thereof—over a regular grid of predictor settings which I call the *reference grid*. They date back at least to Harvey (1960) and his associated computer program LSML (Harvey, 1977) and the contributed SAS procedure named **HARVEY** (Harvey, 1976). Later, they were incorporated via `LSMEANS` statements for various linear model procedures such as **GLM** in the regular SAS releases. See also Goodnight and Harvey (1997) and SAS Institute Inc. (2012) for more information about the SAS implementation.

In simple analysis-of-covariance models, LS means are the same as covariate-adjusted means. In unbalanced factorial experiments, LS means for each factor mimic the main-effects means but are adjusted for imbalance. The latter interpretation is quite similar to the "unweighted means" method for unbalanced data, as presented in old design books.

LS means are not always well understood, in part because the term itself is confusing. Searle *et al.* (1980) discusses exactly how they are defined for various factorial, nested, and covariance models. Searle *et al.* suggest the term "predicted marginal means" (or PMMs) as a better descriptor. However, the term "least-squares means" was already well established in the SAS software, and it has stuck.

The most important things to remember are:

- LS means are computed relative to a *reference grid*.
- Once the reference grid is established, LS means are simply predictions on this grid, or marginal averages of a table of these predictions.

A user who understands these points will know what is being computed, and thus can judge whether or not LS means are appropriate for the analysis.

2 The reference grid

Since the reference grid is fundamental, it is our starting point. For each predictor in the model, we define a set of one or more *reference levels*. The reference grid is then the set of all combinations of reference levels. If not specified explicitly, the default reference levels are obtained as follows:

- For each predictor that is a factor, its reference levels are the unique levels of that factor.
- Each numeric predictor has just one reference level—its mean over the dataset.

So the reference grid depends on both the model and the dataset.

2.1 Example: Orange sales

To illustrate, consider the `oranges` data provided with **lsmeans**. This dataset has sales of two varieties of oranges (response variables `sales1` and `sales2`) at 6 stores (factor `store`), over a period of 6 days (factor `day`). The prices of the oranges (covariates `price1` and `price2`) fluctuate in the different stores and the different days. There is just one observation on each store on each day.

For starters, let's consider an additive covariance model for sales of the first variety, with the two factors and both `price1` and `price2` as covariates (since the price of the other variety could also affect sales).

```
R> library("lsmeans")
R> oranges.lm1 <- lm(sales1 ~ price1 + price2 + day + store, data = oranges)
R> anova(oranges.lm1)

Analysis of Variance Table

Response: sales1
          Df Sum Sq Mean Sq F value   Pr(>F)
price1     1  516.6   516.6  29.100 1.76e-05
price2     1   62.7    62.7   3.533  0.07287
day        5  422.2    84.4   4.757  0.00395
store      5  223.8    44.8   2.522  0.05835
Residuals 23  408.3    17.8
```

The `ref.grid` function in **lsmeans** may be used to establish the reference grid. Here is the default one:

```
R> ( oranges.rg1 <- ref.grid(oranges.lm1) )

'ref.grid' object with variables:
    price1 = 51.222
    price2 = 48.556
    day = 1, 2, 3, 4, 5, 6
    store = 1, 2, 3, 4, 5, 6
```

As outlined above, the two covariates `price1` and `price2` have their means as their sole reference level; and the two factors have their levels as reference levels. The reference grid thus consists of the $1 \times 1 \times 6 \times 6 = 36$ combinations of these reference levels. LS means are based on predictions on this reference grid, which we can obtain using `predict` or `summary`:

```
R> summary(oranges.rg1)
```

```
  price1  price2 day store prediction      SE df
 51.2222 48.5556 1   1        2.91841 2.71756 23
 51.2222 48.5556 2   1        3.84880 2.70134 23
 51.2222 48.5556 3   1       11.01857 2.53456 23
 51.2222 48.5556 4   1        6.09629 2.65137 23
 51.2222 48.5556 5   1       12.79580 2.44460 23
 51.2222 48.5556 6   1        8.74878 2.78618 23
 51.2222 48.5556 1   2        4.96147 2.37774 23
 51.2222 48.5556 2   2        5.89187 2.33558 23
 51.2222 48.5556 3   2       13.06163 2.41645 23
 51.2222 48.5556 4   2        8.13935 2.35219 23
 51.2222 48.5556 5   2       14.83886 2.46615 23
 51.2222 48.5556 6   2       10.79184 2.33760 23
 51.2222 48.5556 1   3        3.20089 2.37774 23
 51.2222 48.5556 2   3        4.13128 2.33558 23
 51.2222 48.5556 3   3       11.30105 2.41645 23
 51.2222 48.5556 4   3        6.37876 2.35219 23
 51.2222 48.5556 5   3       13.07828 2.46615 23
 51.2222 48.5556 6   3        9.03126 2.33760 23
 51.2222 48.5556 1   4        6.19876 2.36367 23
 51.2222 48.5556 2   4        7.12915 2.35219 23
 51.2222 48.5556 3   4       14.29891 2.43168 23
 51.2222 48.5556 4   4        9.37663 2.38865 23
 51.2222 48.5556 5   4       16.07614 2.51909 23
 51.2222 48.5556 6   4       12.02912 2.36469 23
 51.2222 48.5556 1   5        5.54322 2.36312 23
 51.2222 48.5556 2   5        6.47361 2.33067 23
 51.2222 48.5556 3   5       13.64337 2.36367 23
 51.2222 48.5556 4   5        8.72109 2.33760 23
 51.2222 48.5556 5   5       15.42060 2.39554 23
 51.2222 48.5556 6   5       11.37358 2.35232 23
 51.2222 48.5556 1   6       10.56374 2.36668 23
 51.2222 48.5556 2   6       11.49413 2.33925 23
 51.2222 48.5556 3   6       18.66390 2.34784 23
 51.2222 48.5556 4   6       13.74161 2.34130 23
 51.2222 48.5556 5   6       20.44113 2.37034 23
 51.2222 48.5556 6   6       16.39411 2.37054 23
```

2.2 LS means as marginal averages over the reference grid

The ANOVA indicates there is a significant `day` effect after adjusting for the covariates, so we might want to do a follow-up analysis that involves comparing the days. The `lsmeans` function provides a starting point:

```
R> lsmeans(oranges.rg1, "day")   ## or lsmeans(oranges.lm1, "day")
```

```
 day   lsmean      SE df lower.CL upper.CL
 1    5.56442 1.76808 23  1.90686  9.22197
```

```
 2    6.49481 1.72896 23  2.91818 10.07143
 3   13.66457 1.75150 23 10.04131 17.28783
 4    8.74229 1.73392 23  5.15540 12.32918
 5   15.44180 1.78581 23 11.74758 19.13603
 6   11.39478 1.76673 23  7.74003 15.04953

Results are averaged over the levels of: store
Confidence level used: 0.95
```

These results, as indicated in the annotation in the output, are in fact the averages of the predictions shown earlier, for each day, over the 6 stores. The above LS means (often called "adjusted means") are not the same as the overall means for each day:

```
R> with(oranges, tapply(sales1, day, mean))

      1       2       3       4       5       6
7.87275 7.10060 13.75860 8.04247 12.92460 11.60365
```

These unadjusted means are not comparable with one another because they are affected by the differing `price1` and `price2` values on each day, whereas the LS means are comparable because they use predictions at uniform `price1` and `price2` values.

Note that one may call `lsmeans` with either the reference grid or the model. If the model is given, then the first thing it does is create the reference grid; so if the reference grid is already available, as in this example, it's more efficient to make use of it.

For users who dislike the term "LS means," there is also a `pmmeans` function (for predicted marginal means) which is an alias for `lsmeans` but relabels the `lsmean` column in the summary.

2.3 Altering the reference grid

The wixcodeat argument may be used to override defaults in the reference grid. The user may specify this argument either in a `ref.grid` call or an `lsmeans` call; and should specify a `list` with named sets of reference levels. Here is a silly example:

```
R> lsmeans(oranges.lm1, "day", at = list(price1 = 50,
       price2 = c(40,60), day = c("2","3","4")) )

 day   lsmean      SE df lower.CL upper.CL
 2    7.72470 1.73517 23  4.13524  11.3142
 3   14.89446 1.75104 23 11.27217  18.5168
 4    9.97218 1.76613 23  6.31866  13.6257

Results are averaged over the levels of: price2, store
Confidence level used: 0.95
```

Here, we restricted the results to three of the days, and used different prices. One possible surprise is that the predictions are averaged over the two `price2` values. That is because `price2` is no longer a single reference level, and we average over the levels of all factors not used to split-out the LS means. This is probably not what we want.[1] To get separate sets of predictions for each

[1]The *default* reference grid produces LS means exactly as described in Searle *et al.* (1980). However, an altered reference grid containing more than one value of a covariate, such as in this example, departs from (or generalizes, if you please) their definition by averaging with equal weights over those covariate levels. It is not a good idea here, but there is an example later in this vignette where it makes sense.

price2, one must specify it as another factor or as a by factor in the lsmeans call (we will save the result for later discussion):

```
R> org.lsm <- lsmeans(oranges.lm1, "day", by = "price2",
       at = list(price1 = 50, price2 = c(40,60), day = c("2","3","4")) )
R> org.lsm

price2 = 40:
 day   lsmean      SE df lower.CL upper.CL
 2    6.23623 1.88711 23  2.33245  10.1400
 3   13.40599 2.11938 23  9.02173  17.7903
 4    8.48371 1.86651 23  4.62254  12.3449

price2 = 60:
 day   lsmean      SE df lower.CL upper.CL
 2    9.21317 2.10945 23  4.84944  13.5769
 3   16.38293 1.90522 23 12.44169  20.3242
 4   11.46065 2.17805 23  6.95500  15.9663

Results are averaged over the levels of: store
Confidence level used: 0.95
```

Note: We could have obtained the same results using any of these:

```
R> lsmeans(oranges.lm1, ~ day | price, at = ... )         # Ex 1
R> lsmeans(oranges.lm1, c("day","price2"), at = ... )     # Ex 2
R> lsmeans(oranges.lm1, ~ day * price, at = ... )         # Ex 3
```

Ex 1 illustrates the formula method for specifying factors, which is more compact. The | character replaces the by specification. Ex 2 and Ex 3 produce the same results, but their results are displayed as one table (with columns for day and price) rather than as two separate tables.

3 Working with the results

3.1 Objects

The ref.grid function produces an object of class "ref.grid", and the lsmeans function produces an object of class "lsmobj", which is a subclass of "ref.grid". There is really no practical difference between these two classes except for their show methods—what is displayed by default—and the fact that an "lsmobj" is not (necessarily) a true reference grid as defined earlier in this article. Let's use the str function to examine the "lsmobj" object just produced:

```
R> str(org.lsm)

'lsmobj' object with variables:
    day = 2, 3, 4
    price2 = 40, 60
```

We no longer see the reference levels for all predictors in the model—only the levels of day and price2. These *act* like reference levels, but they do not define the reference grid upon which the predictions are based.

3.2 Summaries

There are several methods for "ref.grid" (and hence also for "lsmobj") objects. One already seen is summary. It has a number of arguments—see its help page. In the following call, we summarize days.lsm differently than before. We will also save the object produced by summary for further discussion.

```
R> ( org.sum <- summary(org.lsm, infer = c(TRUE,TRUE),
                        level = .90, adjust = "bon", by = "day") )

day = 2:
 price2   lsmean      SE df lower.CL upper.CL t.ratio p.value
     40  6.23623 1.88711 23  2.33245  10.1400   3.305  0.0062
     60  9.21317 2.10945 23  4.84944  13.5769   4.368  0.0005

day = 3:
 price2   lsmean      SE df lower.CL upper.CL t.ratio p.value
     40 13.40599 2.11938 23  9.02173  17.7903   6.325  <.0001
     60 16.38293 1.90522 23 12.44169  20.3242   8.599  <.0001

day = 4:
 price2   lsmean      SE df lower.CL upper.CL t.ratio p.value
     40  8.48371 1.86651 23  4.62254  12.3449   4.545  0.0003
     60 11.46065 2.17805 23  6.95500  15.9663   5.262  <.0001

Results are averaged over the levels of: store
Confidence level used: 0.9
Conf-level adjustment: bonferroni method for 2 estimates
P value adjustment: bonferroni method for 2 tests
```

The infer argument causes both confidence intervals and tests to be produced; the default confidence level of .95 was overridden; a Bonferroni adjustment was applied to both the intervals and the P values; and the tables are organized the opposite way from what we saw before.

What kind of object was produced by summary? Let's see:

```
R> class(org.sum)

[1] "summary.ref.grid" "data.frame"
```

The "summary.ref.grid" class is an extension of "data.frame". It includes some attributes that, among other things, cause additional messages to appear when the object is displayed. But it can also be used as a "data.frame" if the user just wants to use the results computationally. For example, suppose we want to convert the LS means from dollars to Russian rubles (at the July 13, 2014 exchange rate):

```
R> transform(org.sum, lsrubles = lsmean * 34.2)

  day price2   lsmean      SE df lower.CL upper.CL t.ratio     p.value lsrubles
1   2     40  6.23623 1.88711 23  2.33245  10.1400 3.30465 6.19070e-03  213.279
2   3     40 13.40599 2.11938 23  9.02173  17.7903 6.32544 3.74162e-06  458.485
3   4     40  8.48371 1.86651 23  4.62254  12.3449 4.54523 2.89206e-04  290.143
4   2     60  9.21317 2.10945 23  4.84944  13.5769 4.36757 4.50464e-04  315.090
5   3     60 16.38293 1.90522 23 12.44169  20.3242 8.59899 2.43159e-08  560.296
6   4     60 11.46065 2.17805 23  6.95500  15.9663 5.26188 4.88377e-05  391.954
```

Observe also that the summary is just one data frame with six rows, rather than a collection of three data frames; and it contains a column for all reference variables, including any `by` variables.

Besides `str` and `summary`, there is also a `confint` method, which is the same as `summary` with `infer=c(TRUE,FALSE)`, and a `test` method (same as `summary` with `infer=c(FALSE,TRUE)`, by default). The `test` method may in addition be called with `joint=TRUE` to obtain a joint test that all or some of the linear functions are equal to zero or some other value.

There is also an `update` method which may be used for changing the object's display settings. For example:

```
R> org.lsm2 <- update(org.lsm, by.vars = NULL, level = .99)
R> org.lsm2

 day price2   lsmean      SE df  lower.CL upper.CL
 2       40  6.23623 1.88711 23  0.938488  11.5340
 3       40 13.40599 2.11938 23  7.456193  19.3558
 4       40  8.48371 1.86651 23  3.243791  13.7236
 2       60  9.21317 2.10945 23  3.291240  15.1351
 3       60 16.38293 1.90522 23 11.034351  21.7315
 4       60 11.46065 2.17805 23  5.346122  17.5752

Results are averaged over the levels of: store
Confidence level used: 0.99
```

3.3 Plots

Confidence intervals for LS means may be displayed graphically, using the `plot` method. For example:

```
R> plot(org.lsm, by = "price2")
```

The resulting display is shown in Figure 1. This function requires that the **lattice** package be installed. Additional graphical presentations are covered later in this vignette.

4 Contrasts and comparisons

4.1 Contrasts in general

Often, people want to do pairwise comparisons of LS means, or compute other contrasts among them. This is the purpose of the `contrast` function, which uses a `"ref.grid"` or `"lsmobj"` object as input. There are several standard contrast families such as `"pairwise"`, `"trt.vs.ctrl"`, and `"poly"`. In the following command, we request `"eff"` contrasts, which are differences between each mean and the overall mean:

```
R> contrast(org.lsm, method = "eff")

price2 = 40:
 contrast estimate      SE df t.ratio p.value
 2 effect -3.13908 1.41529 23  -2.218  0.0551
 3 effect  4.03068 1.44275 23   2.794  0.0310
 4 effect -0.89160 1.42135 23  -0.627  0.5366
```

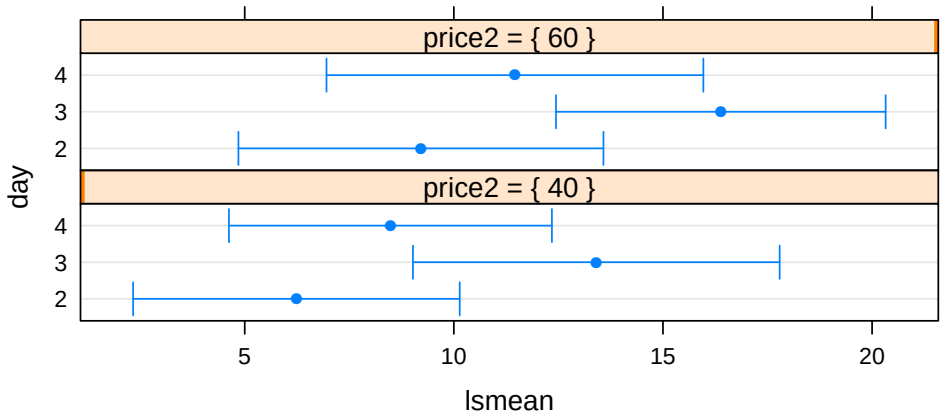

Figure 1: Confidence intervals for LS means in the `oranges` example.

```
price2 = 60:
 contrast estimate      SE df t.ratio p.value
 2 effect -3.13908 1.41529 23  -2.218  0.0551
 3 effect  4.03068 1.44275 23   2.794  0.0310
 4 effect -0.89160 1.42135 23  -0.627  0.5366

Results are averaged over the levels of: store
P value adjustment: fdr method for 3 tests
```

Note that this preserves the `by` specification from before, and obtains the effects for each group. In this example, since it is an additive model, we obtain exactly the same results in each group. This isn't wrong, it's just redundant.

Another popular method is Dunnett-style contrasts, where a particular LS mean is compared with each of the others. This is done using `"trt.vs.ctrl"`. In the following, we obtain (again) the LS means for days, and compare each with the average of the LS means on day 5 and 6.

```
R> days.lsm <- lsmeans(oranges.rg1, "day")
R> ( days_contr.lsm <- contrast(days.lsm, "trt.vs.ctrl", ref = c(5,6)) )

 contrast      estimate      SE df t.ratio p.value
 1 - avg(5,6) -7.853877 2.19424 23  -3.579  0.0058
 2 - avg(5,6) -6.923486 2.12734 23  -3.255  0.0125
 3 - avg(5,6)  0.246279 2.15553 23   0.114  0.9979
 4 - avg(5,6) -4.676003 2.11076 23  -2.215  0.1184

Results are averaged over the levels of: store
P value adjustment: dunnettx method for 4 tests
```

For convenience, `"trt.vs.ctrl1"` and `"trt.vs.ctrlk"` methods are provided for use in lieu of `ref` for comparing with the first and the last LS means. The `"dunnettx"` adjustment is a good

approximation to the exact Dunnett P value adjustment. If the exact adjustment is desired, use `adjust = "mvt"`; but this can take a lot of computing time when there are several tests.

Note that by default, `lsmeans` results are displayed with confidence intervals while `contrast` results are displayed with t tests. One can easily override this; for example,

```
R> confint(contrast(days.lsm, "trt.vs.ctrlk"))
```

(Results not shown.)

In the above examples, a default multiplicity adjustment is determined from the contrast method. This may be overridden by adding an `adjust` argument.

4.2 Pairwise comparisons

Often, users want pairwise comparisons among the LS means. These may be obtained by specifying `"pairwise"` or `"revpairwise"` as the `method` argument in the call to `contrast`. For group labels A, B, C, `"pairwise"` generates the comparisons $A - B, A - C, B - C$ while `"revpairwise"` generates $B - A, C - A, C - B$. As a convenience, a `pairs` method is provided that calls `contrast` with `method="pairwise"`:

```
R> pairs(org.lsm)
```

```
price2 = 40:
 contrast estimate      SE df t.ratio p.value
 2 - 3    -7.16976 2.47970 23  -2.891  0.0216
 2 - 4    -2.24748 2.44234 23  -0.920  0.6333
 3 - 4     4.92228 2.49007 23   1.977  0.1406

price2 = 60:
 contrast estimate      SE df t.ratio p.value
 2 - 3    -7.16976 2.47970 23  -2.891  0.0216
 2 - 4    -2.24748 2.44234 23  -0.920  0.6333
 3 - 4     4.92228 2.49007 23   1.977  0.1406

Results are averaged over the levels of: store
P value adjustment: tukey method for comparing a family of 3 estimates
```

There is also a `cld` (compact letter display) method that lists the LS means along with grouping symbols for pairwise contrasts. It requires the **multcompView** package (Graves *et al.*, 2012) to be installed.

```
R> cld(days.lsm, alpha = .10)
```

```
 day   lsmean      SE df lower.CL upper.CL .group
 1    5.56442 1.76808 23  1.90686  9.22197  1
 2    6.49481 1.72896 23  2.91818 10.07143  1
 4    8.74229 1.73392 23  5.15540 12.32918  12
 6   11.39478 1.76673 23  7.74003 15.04953  12
 3   13.66457 1.75150 23 10.04131 17.28783   2
 5   15.44180 1.78581 23 11.74758 19.13603   2
```

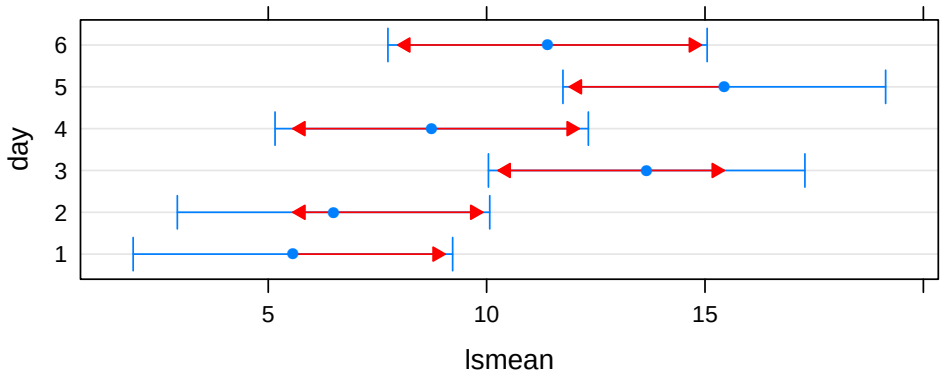

Figure 2: Graphical comparisons of the LS means for `days`.

```
Results are averaged over the levels of: store
Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 6 estimates
significance level used: alpha = 0.1
```

Two LS means that share one or more of the same grouping symbols are not significantly different at the stated value of `alpha`, after applying the multiplicity adjustment (in this case Tukey's HSD). By default, the LS means are ordered in this display, but this may be overridden with the argument `sort=FALSE`. `cld` returns a `"summary.ref.grid"` object, not an `lsmobj`.

Another way to display pairwise comparisons is via the `comparisons` argument of `plot`. When this is set to `TRUE`, arrows are added to the plot, with lengths set so that the amount by which they overlap (or don't overlap) matches as closely as possible to the amounts by which corresponding confidence intervals for differences cover (or don't cover) the value zero. This does not always work, and if there are discrepancies, a message is printed. But it usually works as long as the standard errors of differences are not too discrepant.

```
R> plot(days.lsm, comparisons = TRUE, alpha = .10)
```

Figure 2 shows the result. Note that the pairs of means having overlapping arrows are the same as those grouped together in the `cld` display. However, these comparison arrows show more about the degree of significance in the comparisons. The lowest and highest LS mean have arrows pointing only inward, as the others are not needed. If the confidence intervals and arrows together look too cluttered, one can add the argument `intervals = FALSE`, then only the arrows will be displayed.

4.3 Multiplicity adjustments—changing the family

You may have noticed that in the preceding examples where *P*-value adjustments were implemented, those adjustments were made *separately* for each sub-table when a `by` variable is active. Some users prefer that all the adjusted tests together as one family—or even combine more than one family of tests into one family for purposes of adjustment. This may be done using the `rbind` method (similar to using `rbind` to combine matrices.

On the flip side, perhaps we want to exclude some tests. This may be used using the operator: simply specify the row indexes of the tests to include.

To illustrate, consider the previously obtained `org.lsm` object. In `pairs(org.lsm)`, we obtain the same results twice (as seen above) because the model is additive. For the same reason, if we change the `by` variable to `"day"`, we'll obtain three copies of the same comparison of the two `price2`s. If we want to consider the three `day` comparisons and the one `price2` comparison together as one family of four tests, we can do:

```
R> rbind(pairs(org.lsm)[1:3], pairs(org.lsm, by = "day")[1])
```

```
 contrast price2 day estimate      SE df t.ratio p.value
 2 - 3    40     .   -7.16976 2.47970 23  -2.891  0.0292
 2 - 4    40     .   -2.24748 2.44234 23  -0.920  0.7585
 3 - 4    40     .    4.92228 2.49007 23   1.977  0.1875
 40 - 60  .      2   -2.97694 1.99466 23  -1.492  0.4056

Results are averaged over some or all of the levels of: store
P value adjustment: mvt method for 4 tests
```

Note that by default, the `"mvt"` adjustment level is used; for complicated families like this, ordinary Tukey and Dunnett adjustments are usually not appropriate.

We arrived at this point by a circuitous path. In the additive model, the above conditional results are the same as the marginal ones:

```
R> rbind(pairs(lsmeans(org.lsm, "day")), pairs(lsmeans(org.lsm, "price2")))
```

```
 contrast estimate      SE df t.ratio p.value
 2 - 3    -7.16976 2.47970 23  -2.891  0.0292
 2 - 4    -2.24748 2.44234 23  -0.920  0.7585
 3 - 4     4.92228 2.49007 23   1.977  0.1875
 40 - 60  -2.97694 1.99466 23  -1.492  0.4057

Results are averaged over some or all of the levels of: store, price2, day
P value adjustment: mvt method for 4 tests
```

5 Multivariate models

The `oranges` data has two response variables. Let's try a multivariate model for predicting the sales of the two varieties of oranges, and see what we get if we call `ref.grid`:

```
R> oranges.mlm <- lm(cbind(sales1,sales2) ~ price1 + price2 + day + store,
                    data = oranges)
R> ref.grid(oranges.mlm)
```

```
'ref.grid' object with variables:
    price1 = 51.222
    price2 = 48.556
    day = 1, 2, 3, 4, 5, 6
    store = 1, 2, 3, 4, 5, 6
    rep.meas = multivariate response levels: sales1, sales2
```

What happens is that the multivariate response is treated like an additional factor, by default named `rep.meas`. In turn, it can be used to specify levels for LS means. Here we rename the multivariate response to `"variety"` and obtain `day` means (and a compact letter display for comparisons thereof) for each `variety`:

```
R> org.mlsm <- lsmeans(oranges.mlm, ~ day | variety, mult.name = "variety")
R> cld(org.mlsm, sort = FALSE)

variety = sales1:
 day   lsmean      SE df  lower.CL upper.CL .group
 1    5.56442 1.76808 23  1.906856  9.22197  1
 2    6.49481 1.72896 23  2.918183 10.07143  12
 3   13.66457 1.75150 23 10.041308 17.28783   23
 4    8.74229 1.73392 23  5.155403 12.32918  123
 5   15.44180 1.78581 23 11.747576 19.13603    3
 6   11.39478 1.76673 23  7.740031 15.04953  123

variety = sales2:
 day   lsmean      SE df  lower.CL upper.CL .group
 1    7.71566 2.32649 23  2.902962 12.52836  12
 2    3.97645 2.27500 23 -0.729758  8.68265  1
 3   16.59781 2.30467 23 11.830240 21.36539   2
 4   11.04454 2.28153 23  6.324832 15.76425  12
 5   14.99079 2.34981 23 10.129837 19.85174   2
 6   12.04878 2.32470 23  7.239777 16.85779  12

Results are averaged over the levels of: store
Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 6 estimates
significance level used: alpha = 0.05
```

6 Contrasts of contrasts (interaction contrasts)

With the preceding model, we might want to compare the two varieties on each day:

```
R> org.vardiff <- update(pairs(org.mlsm, by = "day"), by = NULL)
```

The results (not yet shown) will comprise the six `sales1-sales2` differences, one for each day. The two `by` specifications seems odd, but the one in `pairs` specifies doing a separate comparison for each day, and the one in `update` asks that we convert it to one table with six rows, rather than 6 tables with one row each. Now, let's compare these differences to see if they vary from day to day.

```
R> cld(org.vardiff)

 contrast        day  estimate      SE df t.ratio p.value .group
 sales1 - sales2 3   -2.933243 2.69411 23  -1.089  0.2875  1
 sales1 - sales2 4   -2.302251 2.66706 23  -0.863  0.3969  1
 sales1 - sales2 1   -2.151248 2.71961 23  -0.791  0.4370  1
 sales1 - sales2 6   -0.654002 2.71752 23  -0.241  0.8120  1
```

```
 sales1 - sales2 5    0.451016 2.74688 23   0.164  0.8710  1
 sales1 - sales2 2    2.518361 2.65943 23   0.947  0.3535  1

Results are averaged over the levels of: store
P value adjustment: tukey method for comparing a family of 6 estimates
significance level used: alpha = 0.05
```

There is little evidence of variety differences, nor that these differences vary from day to day.

A newer feature of the `contrast` function is the optional `interaction` argument, which may be used to specify interaction contrasts by naming which contrast to use for each variable (in the order of appearance in the grid). In a similar example to the above, suppose we want to compare each polynomial contrast in `day` between the two varieties:

```
R> org.icon <- contrast(org.mlsm, interaction = c("poly", "pairwise"))
R> org.icon

 day_poly  variety_pairwise  estimate      SE df t.ratio p.value
 linear    sales1 - sales2    1.91519 22.9766 23   0.083  0.9343
 quadratic sales1 - sales2    3.94635 24.7950 23   0.159  0.8749
 cubic     sales1 - sales2   19.43368 36.0788 23   0.539  0.5953
 quartic   sales1 - sales2  -22.18437 14.1180 23  -1.571  0.1298
 degree 5  sales1 - sales2   18.14389 43.6461 23   0.416  0.6815

Results are averaged over the levels of: store
```

Exactly what contrasts are being generated can become somewhat confusing, especially where interaction contrasts are concerned. The `coef` method helps with this; it returns a `data.frame` with the grid of factor levels that were contrasted, along with the contrast coefficients that were used:

```
R> coef(org.icon)

   day variety c.1 c.2 c.3 c.4 c.5
1    1  sales1  -5   5  -5   1  -1
2    2  sales1  -3  -1   7  -3   5
3    3  sales1  -1  -4   4   2 -10
4    4  sales1   1  -4  -4   2  10
5    5  sales1   3  -1  -7  -3  -5
6    6  sales1   5   5   5   1   1
7    1  sales2   5  -5   5  -1   1
8    2  sales2   3   1  -7   3  -5
9    3  sales2   1   4  -4  -2  10
10   4  sales2  -1   4   4  -2 -10
11   5  sales2  -3   1   7   3   5
12   6  sales2  -5  -5  -5  -1  -1
```

We can see more clearly that each contrast is the difference of a polynomial contrast on the first six rows of `org.mlsm`, minus that same contrast of the last six rows. (Note: `coef` is only useful for objects generated by `contrast` or `pairs`; if called on some other `ref.grid` object, it simply returns `NULL`.)

7 Interfacing with multcomp

The **multcomp** package (Hothorn *et al.*, 2013) supports more options for simultaneous inference than are available in **lsmeans**. Its `glht` (general linear hypothesis testing) function and associated `"glht"` class are similar in some ways to `lsmeans` and `"lsmobj"` objects, respectively. So **lsmeans** provides an `as.glht` function to do the conversion.

To illustrate, let us convert the `days_contr.lsm` object (produced earlier) to a `glht` object, and use it to obtain adjusted P values under Westfall's adjustment procedure (not available in **lsmeans**):

```
R> library("multcomp")
R> days.glht <- as.glht(days_contr.lsm)
R> summary(days.glht, test = adjusted("Westfall"))

         Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:
                   Estimate Std. Error t value Pr(>|t|)
1 - avg(5,6) == 0   -7.854      2.194   -3.58   0.0062
2 - avg(5,6) == 0   -6.923      2.127   -3.25   0.0099
3 - avg(5,6) == 0    0.246      2.156    0.11   0.9100
4 - avg(5,6) == 0   -4.676      2.111   -2.22   0.0697
(Adjusted p values reported -- Westfall method)
```

In addition, **lsmeans** provides an `lsm` function (or its alias, `pmm`) that may be called from within a call to `glht`. Thus, another way to obtain the same `glht` object is to use

```
R> days.glht1 <- glht(oranges.lm1,
                      lsm("day", contr = "trt.vs.ctrl", ref = c(5,6)))
```

By the way, the following two statements will produce the same results:

```
R> summary(days_contr.lsm, adjust = "mvt")
R> summary(days.glht)
```

That is, the `"mvt"` adjust method in **lsmeans** is the same as the default single-step P value adjustment in **multcomp**.

One additional detail: If there is a `by` variable in effect, `glht` or `as.glht` returns a `list` of `glht` objects—one for each `by` level. There are courtesy `coef`, `confint`, `plot`, `summary`, and `vcov` methods for this `"glht.list"` class to make things a bit more user-friendly. Recall the earlier example result `org.lsm`, which contains information for LS means for three `days` at each of two values of `price2`. Suppose we are interested in pairwise comparisons of these LS means, by `price2`. If we call

```
R> summary(as.glht(pairs(org.lsm)))
```

(results not displayed) we will obtain two `glht` objects with three contrasts each, so that the results shown will incorporate multiplicity adjustments for each family of three contrasts. If, on the other hand, we want to consider those six contrasts as one family, use

```
R> summary(as.glht(pairs(org.lsm), by = NULL))
```

... and note (look carefully at the parentheses) that this is *not* the same as

```
R> summary(as.glht(pairs(org.lsm, by = NULL)))
```

which removes the `by` grouping *before* the pairwise comparisons are generated, thus yielding $\binom{6}{2} = 15$ contrasts instead of just six.

8 A new example: Oat yields

Orange-sales illustrations are probably getting tiresome. To illustrate some new features, let's turn to a new example. The `Oats` dataset in the **nlme** package (Pinheiro *et al.*, 2013) has the results of a split-plot experiment discussed in Yates (1935). The experiment was conducted on six blocks (factor `Block`). Each block was divided into three plots, which were randomized to three varieties (factor `Variety`) of oats. Each plot was divided into subplots and randomized to four levels of nitrogen (variable `nitro`). The response, `yield`, was measured once on each subplot after a suitable growing period.

We will fit a model using the `lmer` function in the **lme4** package (Bates *et al.*, 2013). This will be a mixed model with random intercepts for `Block` and `Block:Variety` (which identifies the plots). A logarithmic transformation is applied to the response variable (mostly for illustration purposes, though it does produce a good fit to the data). Note that `nitro` is stored as a numeric variable, but we want to consider it as a factor in this initial model.

```
R> data("Oats", package = "nlme")
R> library("lme4")
R> Oats.lmer <- lmer(log(yield) ~ Variety*factor(nitro) + (1|Block/Variety),
                     data = Oats)
R> anova(Oats.lmer)

Analysis of Variance Table
                     Df Sum Sq Mean Sq F value
Variety               2 0.0750  0.0375   2.008
factor(nitro)         3 2.1350  0.7117  38.110
Variety:factor(nitro) 6 0.0451  0.0075   0.402
```

Apparently, the interaction is not needed. But perhaps we can further simplify the model by using only a linear or quadratic trend in `nitro`. We can find out by looking at polynomial contrasts:

```
R> contrast(lsmeans(Oats.lmer, "nitro"), "poly")

NOTE: Results may be misleading due to involvement in interactions

 contrast     estimate        SE df t.ratio p.value
 linear     1.50565129 0.1440469 45  10.453  <.0001
 quadratic -0.14510997 0.0644197 45  -2.253  0.0292
 cubic      0.00273198 0.1440469 45   0.019  0.9850

Results are averaged over the levels of: Variety
Results are given on the log (not the response) scale.
```

(A message is issued when we average over predictors that interact with those that delineate the LS means. In this case, it is not a serious problem because the interaction is weak.) Both the linear and quadratic contrasts are pretty significant. All this suggests fitting an additive model where `nitro` is included as a numeric predictor with a quadratic trend.

```
R> Oats.lmer2 <- lmer(log(yield) ~ Variety + poly(nitro,2)
                                  + (1|Block/Variety),  data = Oats)
```

Remember that `nitro` is now used as a quantitative predictor. But for comparing with the previous model, we want to see predictions at the four unique `nitro` values rather than at the average of `nitro`. This may be done using `at` as illustrated earlier, or a shortcut is to specify `cov.reduce` as `FALSE`, which tells `ref.grid` to use all the unique values of numeric predictors.

```
R> Oats.lsm2 <- lsmeans(Oats.lmer2, ~ nitro | Variety, cov.reduce = FALSE)
```

The results are displayed as an export table (see Section 9.1) in Table 1, page 17. These LS means follow the same quadratic trend for each variety, but with different intercepts.[2]

Fractional degrees of freedom are displayed in these results. These are obtained by default using the Satterthwaite method, using routines in the **lmerTest** package (Kuznetsova *et al.*, 2016). Adding the argument `mode = "kenward-roger"` to the `lsmeans` call will cause the degrees of freedom to be computed using instead the Kenward-Roger (K-R) method from the **pbkrtest** package (Halekoh and Højsgaard, 2013), which also implements, as a side-effect, a bias adjustment in the estimated covariances (and hence standard errors). The K-R method is probably preferable, but it requires a lot more computation, and hence is no longer the default. A third option is to specify `mode = "asymptotic"`, for which all the degrees of freedom are set to `NA`—producing z tests rather than t tests. You may change the default via `lsm.options(lmer.df = "desired default")`. These `mode` settings are partially matched, so `mode = "k"` is actually good enough.

9 Additional display methods

9.1 Export tables

The **lsmeans** package provides an `xtable` method (Dahl, 2016) that works with `lsmobj`, `ref.grid`, and `summary.ref.grid` objects. (It actually uses the `xtableList` interface; see the documentation for details.) This is quite useful if you want a nicely formatted table, especially using `Sweave` or `knitr`. To illustrate, we display the `Oats.lsm2` object just created.

```
R> library("xtable")
R> xtbl <- xtable(Oats.lsm2, caption = "Example using \\texttt{xtable}",
       label = "xtable:example")
R> print(xtbl, table.placement = "t")


R> cat("See Table~\\ref{xtable:example}.\n")
```

See Table 1.

[2]This is the promised example where our generalization of Searle *et al.* (1980)'s definition of LS means makes sense. Suppose we want to compare the LS means for `Variety` with those in the original model `Oats.lmer` where `nitro` was a factor, we want to average equally over the four `nitro` levels, even though `nitro` is a covariate in this second model.

nitro	lsmean	SE	df	lower.CL	upper.CL
Variety = Golden Rain					
0.0000	4.3546	0.0770	11.77	4.1864	4.5228
0.2000	4.5777	0.0745	10.34	4.4124	4.7430
0.4000	4.7283	0.0745	10.34	4.5629	4.8936
0.6000	4.8063	0.0770	11.77	4.6381	4.9745
Variety = Marvellous					
0.0000	4.4122	0.0770	11.77	4.2440	4.5804
0.2000	4.6353	0.0745	10.34	4.4700	4.8007
0.4000	4.7859	0.0745	10.34	4.6206	4.9513
0.6000	4.8639	0.0770	11.77	4.6957	5.0321
Variety = Victory					
0.0000	4.2751	0.0770	11.77	4.1069	4.4434
0.2000	4.4983	0.0745	10.34	4.3329	4.6636
0.4000	4.6488	0.0745	10.34	4.4835	4.8142
0.6000	4.7268	0.0770	11.77	4.5586	4.8951

Degrees-of-freedom method: satterthwaite

Results are given on the log (not the response) scale.

Confidence level used: 0.95

Table 1: Example using `xtable`

9.2 Displaying LS means graphically

We have already seen the use of the `plot` function to display confidence intervals and/or comparison arrows. The **lsmeans** package also includes a function `lsmip` that displays predictions in an interaction-plot-like manner. It uses a formula of the form

```
curve.factors ~ x.factors | by.factors
```

This function also requires the **lattice** package. In the above formula, `curve.factors` specifies factor(s) used to delineate one displayed curve from another (i.e., groups in **lattice**'s parlance). `x.factors` are those whose levels are plotted on the horizontal axis. And `by.factors`, if present, break the plots into panels.

To illustrate, consider the first model we fitted to the `Oats` data. Let's do a graphical comparison of the two models we have fitted to the `Oats` data.

```
R> lsmip(Oats.lmer, Variety ~ nitro, ylab = "Observed log(yield)")
R> lsmip(Oats.lsm2, Variety ~ nitro, ylab = "Predicted log(yield)")
```

The plots are shown in Figure 3. Note that the first model fits the cell means perfectly, so its plot is truly an interaction plot of the data. The other displays the parabolic trends we fitted in the revised model.

10 Transformations

10.1 Automatic support for transformations

When a transformation or link function is used in fitting a model, `ref.grid` (also called by `lsmeans`) stores that information in the returned object, as seen in this example:

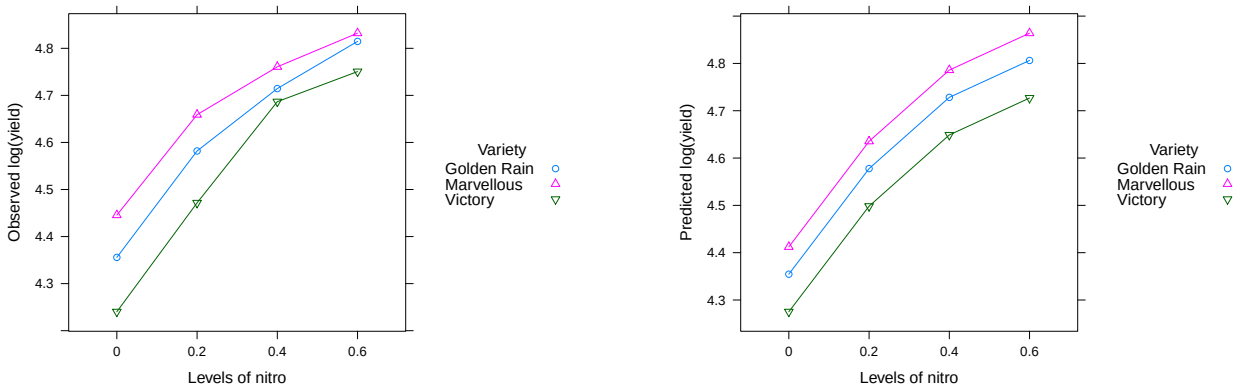

Figure 3: Interaction plots for the cell means and the fitted model, `Oats` example.

```
R> str(Oats.lsm2)

'lsmobj' object with variables:
    nitro = 0.0, 0.2, 0.4, 0.6
    Variety = Golden Rain, Marvellous, Victory
Transformation: "log"
```

This allows us to conveniently unravel the transformation, via the `type` argument in `summary` or related functions such as `lsmip` and `predict`. Here are the predicted yields for (as opposed to predicted log yields) for the polynomial model:

```
R> summary(Oats.lsm2, type = "response")

Variety = Golden Rain:
 nitro response      SE    df lower.CL upper.CL
   0.0  77.8340 5.99577 11.77  65.7834  92.0921
   0.2  97.2902 7.25165 10.34  82.4632 114.7831
   0.4 113.0989 8.42998 10.34  95.8627 133.4343
   0.6 122.2751 9.41920 11.77 103.3439 144.6742

Variety = Marvellous:
 nitro response      SE    df lower.CL upper.CL
   0.0  82.4529 6.35158 11.77  69.6871  97.5571
   0.2 103.0637 7.68199 10.34  87.3568 121.5946
   0.4 119.8106 8.93024 10.34 101.5515 141.3527
   0.6 129.5313 9.97816 11.77 109.4766 153.2596

Variety = Victory:
 nitro response      SE    df lower.CL upper.CL
   0.0  71.8907 5.53794 11.77  60.7602  85.0601
   0.2  89.8612 6.69793 10.34  76.1664 106.0184
   0.4 104.4629 7.78628 10.34  88.5428 123.2454
   0.6 112.9383 8.69996 11.77  95.4527 133.6271
```

```
Degrees-of-freedom method: satterthwaite
Confidence level used: 0.95
Intervals are back-transformed from the log scale
```

It is important to realize that the statistical inferences are all done *before* reversing the transformation. Thus, t ratios are based on the linear predictors and will differ from those computed using the printed estimates and standard errors. Likewise, confidence intervals are computed on the linear-predictor scale, then the endpoints are back-transformed.

This kind of automatic support is available for most popular response transformations such as `log`, `log10`, and even transformations like `asin(sqrt())` and `sqrt(y)+sqrt(y+1)`. The Details section for `help("make.tran")` provides a complete list. It is also possible to support custom transformations via the `tran` argument in the `update` method—see its help page.

10.2 Using `make.tran`

The `make.tran` function provides support for yet more popular types of transformations, particularly those that require specifying one or more parameters. Examples are general power transformations, Box-Cox transformations, and transformations with a shifted origin such as `log(y + a)`. Details may of course be found via `help("make.tran")`. The function returns a `list` of functions, compatible with what is returned by `make.link` in the **stats** package. The latter is intended primarily for use with generalized linear models, and `make.tran` extends such capabilities to other response transformations.

There are two basic ways to use `make.tran`: retrospectively on an existing model, and prospectively in fitting a new model. Here is an example of retrospective use, where the $\log(y + 5)$ transformation was used. This transformation is not auto-detected.

```
R> Oats.log1 <- lmer(log(yield + 5) ~ Variety + factor(nitro)
                     + (1|Block/Variety), data = Oats)
R> ( Oats.rg1 <- update(ref.grid(Oats.log1),
                        tran = make.tran("genlog", 5)) )

'ref.grid' object with variables:
    Variety = Golden Rain, Marvellous, Victory
    nitro = 0.0, 0.2, 0.4, 0.6
Transformation: "log(mu + 5)"
```

Here, we created a reference grid for the model, then updated it with its `tran` component set to the result of `make.tran` for a generalized log transformation with parameter 5. This updated reference grid has all the information needed to back-transform the results to the original `yield` scale:

```
R> round(predict(Oats.rg1, type = "response"), 1)

 [1]  77.9  82.6  72.0  97.4 103.2  90.0 113.1 119.7 104.6 122.3 129.5 113.1
```

Using `make.tran` prospectively makes use of the fact that the transformation itself is included in the returned list as a function named `linkfun` (somewhat oddly named due to the fact that `make.tran` mimics the functionality of `make.link`). When a model is fitted with `linkfun` as the transformation, its enclosing environment is automatically used to obtain the transformation definitions. For illustration, consider a rather far-fetched transformation:

```
R> my.tran <- make.tran("boxcox", c(.567, 10))
R> my.tran$linkfun(10:15)

[1] -1.76367  0.00000  0.84910  1.52443  2.10699  2.62905
```

This specifies a Box-Cox transformation with the origin shifted to 10:[3]

$$h(y) = \frac{(y-10)^{.567} - 1}{1 - .567}$$

If we use `my.tran` as an enclosing environment for fitting the model, the transformation is saved automatically:

```
R> Oats.bc <- with(my.tran, lmer(linkfun(yield) ~ Variety + factor(nitro)
                                 + (1|Block/Variety), data = Oats))
R> ( rg.bc <- ref.grid(Oats.bc) )

'ref.grid' object with variables:
    Variety = Golden Rain, Marvellous, Victory
    nitro = 0.0, 0.2, 0.4, 0.6
Transformation: "Box-Cox (lambda = 0.567) with origin at 10"

R> round(predict(rg.bc, type = "response"), 1)

 [1]  78.9  83.8  72.5  98.4 103.9  91.2 113.9 119.9 106.2 123.1 129.3 115.1
```

10.3 Using `regrid`

The `regrid` function may be used to, in essence, give a new beginning to an existing reference grid (or `lsmobj`), most redefined on the response scale (i.e., back-transformed). Consider the preceding Box-Cox example, after applying `regrid`:

```
R> rg.bc.regrid <- regrid(rg.bc)
```

By default, the estimates are back-transformed to the response scale. In a `regrid` result, the `linfct` slot (linear functions) become the identity matrix, and the `bhat` slot (regression coefficients) become the predictions at each grid point:

```
R> round(rg.bc.regrid@bhat, 1)

 [1]  78.9  83.8  72.5  98.4 103.9  91.2 113.9 119.9 106.2 123.1 129.3 115.1
```

which matches the predictions shown previously.

The interesting thing is what happens if we subsequently obtain LS means. Compare these results:

```
R> summary(lsmeans(rg.bc, "Variety"), type = "response")
```

[3] To obtain an ordinary Box-Cox transformation, provide just one parameter: `make.tran("boxcox", .567)`.

```
 Variety     response      SE   df lower.CL upper.CL
 Golden Rain 102.8965 7.59415 9.07  86.4283  120.738
 Marvellous  108.5580 7.79119 9.07  91.6389  126.839
 Victory      95.5741 7.32891 9.07  79.7132  112.823

Results are averaged over the levels of: nitro
Degrees-of-freedom method: satterthwaite
Confidence level used: 0.95
Intervals are back-transformed from the Box-Cox (lambda = 0.567) with origin at 10 scale
```

```
R> lsmeans(rg.bc.regrid, "Variety")
```

```
 Variety     response      SE    df lower.CL upper.CL
 Golden Rain 103.5782 7.58751 11.34  86.9391  120.217
 Marvellous  109.2342 7.78483 11.34  92.1623  126.306
 Victory      96.2634 7.32187 11.34  80.2068  112.320

Results are averaged over the levels of: nitro
Degrees-of-freedom method: satterthwaite
Confidence level used: 0.95
```

Why are the answers somewhat different? Recall that LS means are obtained via equally-weighted averages of predictions. In the first `lsmeans` call, the predictions, on the Box-Cox scale, are averaged together and then back-transformed to the response scale; whereas in the second `lsmeans` call, the predictions being averaged were already on the response scale. (Hence, the results are the usual arithmetic means of the predictions on the grid.) Since the Box-Cox transformation is nonlinear, averaging then back-transforming is not the same as back-transforming then averaging.

Even the degrees of freedom (d.f.) differ in the two results, because degrees-of-freedom calculations take place on the linear-predictor scale. Once results are back-transformed, `regrid` "freezes" the calculated degrees of freedom for each prediction. Subsequently, a containment method is used whereby the returned d.f. is the minimum d.f. of predictions involved in each LS mean.

Some users prefer averaging the predictions on the response scale as they are then the arithmentic means; and now you see that the way to make that happen is through the `regrid` function.

10.4 Reverse-engineering a log transformation

When a response has been log-transformed, then there are useful special properties of back-transformed summaries:

- LS means, when back-transformed to the response scale, are actually the *geometric* means of the response-scale predictions.
- A difference of two LS means on the log scale, after back-transforming, becomes an estimate of the *ratio* of the two geometric means. Such comparisons via ratios can be quite useful for positive responses.

The `regrid` function provides a `"log"` option that recomputes the reference grid *as if* the response transformation had been the natural logarithm. We can then take advantage of the above special properties of log-transformed responses. The only proviso is that, on the response scale, all of the reference-grid predictions must be positive.

To illustrate, we revisit the above Box-Cox model once again, and regrid it on the log scale:

```
R> rg.log <- regrid(rg.bc, "log")
R> lsm.log <- lsmeans(rg.log, "Variety")
R> summary(lsm.log, type = "response")

 Variety     response      SE    df lower.CL upper.CL
 Golden Rain 102.1431 7.59228 11.34  86.7796  120.226
 Marvellous  107.8026 7.79008 11.34  92.0040  126.314
 Victory      94.8242 7.32582 11.34  80.0462  112.331

Results are averaged over the levels of: nitro
Degrees-of-freedom method: satterthwaite
Confidence level used: 0.95
Intervals are back-transformed from the log scale

R> summary(pairs(lsm.log), type = "response")

 contrast                 response.ratio        SE    df t.ratio p.value
 Golden Rain - Marvellous       0.947502 0.0642564 11.34  -0.795  0.7133
 Golden Rain - Victory          1.077183 0.0755633 11.34   1.060  0.5563
 Marvellous - Victory           1.136867 0.0787699 11.34   1.851  0.1974

Results are averaged over the levels of: nitro
P value adjustment: tukey method for comparing a family of 3 estimates
Tests are performed on the log scale
```

The LS means shown are the geometric means of the predictions, as opposed to the arithmetic means obtained above from `rg.bc.regrid`. And the pairwise comparisons come out as ratios of these.

10.5 The `transform` argument

For convenience, the user may use a `transform` argument to re-grid as part of a `ref.grid` or `lsmeans` call. For example, `lsmeans(Oats.bc, "Variety", transform = "response")` is equivalent to `lsmeans(rg.bc.regrid, "Variety")` but without needing the two code steps previously used to produce `rg.bc` and `rg.bc.regrid`.

11 More on tests

The default settings for `test` yield traditional two-tailed t (or z) tests of significance against zero. So if $\theta^{(j)}$ is the jth parameter (e.g., LS mean or contrast) being estimated, we are testing the null hypothesis $H_0 : \theta^{(j)} = 0$ versus the alternative $H_1 : \theta^{(j)} \neq 0$. We can, however, specify different types of tests in the `test` or `summary` functions.

11.1 Nonzero null values

If we wish to use nonzero null values, i,e., test $H_0 : \theta^{(j)} = \theta_0^{(j)}$, use `test` or `summary` with the `null` argument set to the desired $\theta_0^{(j)}$ values. For example, in the Oat-yield example, suppose we want to test each of the `Variety` yields against 100 (actually $\log 100$ since the response was transformed):

```
R> Oats.Vlsm = lsmeans(Oats.lmer2, "Variety")
R> test(Oats.Vlsm, null = log(100), type = "response")

 Variety     response      SE    df null t.ratio p.value
 Golden Rain 105.8528 7.95052 10.65  100   0.757  0.4653
 Marvellous  112.1345 8.42233 10.65  100   1.525  0.1564
 Victory      97.7701 7.34343 10.65  100  -0.300  0.7698

Degrees-of-freedom method: satterthwaite
Tests are performed on the log scale
```

Note that `null` should always be given on the linear-predictor scale (in this case log yield), even when specifying `type="response"`. We could have specified different null values for each hypothesis by providing a vector of three numbers.

11.2 Equivalence tests

The preceding results say that none of the variety means differs significantly from 100, after transforming. But this is not the same as saying that we have evidence that the means are close to 100 (that is, absence of proof is not proof of absence). To make a strong statement that an effect is small, we should use an equivalence test, which more-or-less turns the hypotheses around:

$$H_0 : |\theta^{(j)} - \theta_0^{(j)}| \geq \delta \qquad \text{versus} \qquad H_1 : |\theta^{(j)} - \theta_0^{(j)}| < \delta$$

where δ is a specified threshold of equivalence. A common test procedure is the two one-sided test (TOST) method (Schuirmann, 1987), whereby we obtain equivalence only if we can establish both that $\theta^{(j)} - \theta_0^{(j)} > -\delta$ and that $\theta^{(j)} - \theta_0^{(j)} < \delta$. In `lsmeans`, we do this by pre-identifying the less significant of these two tests:

$$t = \frac{|\hat{\theta}^{(j)} - \theta_0^{(j)}| - \delta}{SE(\hat{\theta}^{(j)})}$$

and the P value is the *left*-tail probability of this quantity from the central t distribution.

In `test` or `summary`, an equivalence test is requested by specifying a nonzero `delta` argument, which in turn is used as the threshold δ. In the Oat-yield example, the following results are obtained using a threshold of $\delta = 0.20$:

```
R> test(Oats.Vlsm, null = log(100), delta = 0.20, type = "r")

 Variety     response      SE    df null t.ratio p.value
 Golden Rain 105.8528 7.95052 10.65  100  -1.905  0.0420
 Marvellous  112.1345 8.42233 10.65  100  -1.138  0.1400
 Victory      97.7701 7.34343 10.65  100  -2.363  0.0192

Degrees-of-freedom method: satterthwaite
Statistics are tests of equivalence with a threshold of 0.2
P values are left-tailed
Tests are performed on the log scale
```

So two of the three Variety means are established as being within .20 of $\log 100$. The natural log scale has the special property that small increments on the log scale translate to approximate percentage differences of the same size. That is, a threshold of .20 corresponds to about a 20% difference: $\log 80 - \log 100 = \log .8 \approx -.223$, and $\log 120 - \log 100 = \log 1.2 \approx .182$.

11.3 One-sided tests, noninferiority, nonsuperiority

The `side` argument is also available to specify one-sided tests. A right-tailed alternative may be requested using `side` partially matching one of `"+"`, `"right"`, `">"`, `+1`, `1`, `"superiority"`, or (see later) `"noninferiority"`. Similarly, a left-tailed alternative may be specified using `side` equal to `"-"`, `"left"`, `"<"`, `-1`, `"inferiority"`, or `"nonsuperiority"`. (And for completeness, a two-sided alternative is specified using `0`, `2`, `"!="`, `"both"`, `"two-sided"`, `"equivalence"`, or `"="`.) In the following example, we test to see if either Golden Rain or Marvellous has better yield than Victory:

```
R> test(contrast(Oats.Vlsm, "trt.vs.ctrlk"), side = ">")

 contrast               estimate        SE df t.ratio p.value
 Golden Rain - Victory 0.0794312 0.0686844 10   1.156  0.2556
 Marvellous - Victory  0.1370802 0.0686844 10   1.996  0.0725

Results are given on the log (not the response) scale.
P value adjustment: sidak method for 2 tests
P values are right-tailed
```

The one-sided version of an equivalence test is called a noninferiority or nonsuperiority test. It is obtained by specifying both `side` and a nonzero `delta`. For example, to test whether Victory is as good as the other two within a 25% threshold, use

```
R> test(contrast(Oats.Vlsm, "trt.vs.ctrlk"), side = "nonsup", delta = .25)

 contrast               estimate        SE df t.ratio p.value
 Golden Rain - Victory 0.0794312 0.0686844 10  -2.483  0.0321
 Marvellous - Victory  0.1370802 0.0686844 10  -1.644  0.1269

Results are given on the log (not the response) scale.
P value adjustment: sidak method for 2 tests
Statistics are tests of nonsuperiority with a threshold of 0.25
P values are left-tailed
```

We find strong evidence that, with the stated threshold of .25, Golden Rain is nonsuperior to Victory (so that Victory is noninferior to Golden Rain); but not strong evidence that Victory is noninferior to Marvellous.

12 Trends

The **lsmeans** package provides a function `lstrends` for estimating and comparing the slopes of fitted lines (or curves). To illustrate, consider the built-in R dataset `ChickWeight` which has data on the growths of newly hatched chicks under four different diets. The following code produces the display in Figure 4.

```
R> require("lattice")
R> xyplot(weight ~ Time | Diet, groups = ~ Chick, data = ChickWeight,
       type = "o", layout=c(4, 1))
```

Let us fit a model to these data using random slopes for each chick and allowing for a different average slope for each diet (a square-root transformation straightens-out the curves somewhat):

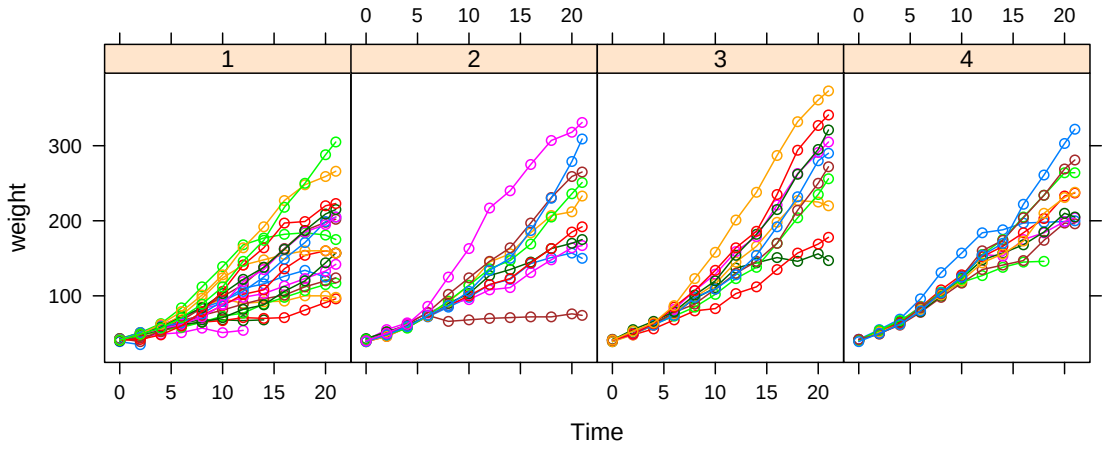

Figure 4: Growth curves of chicks, dataset `ChickWeight`.

```
R> Chick.lmer <- lmer(sqrt(weight) ~ Diet * Time + (0 + Time | Chick),
       data = ChickWeight)
```

We can then call `lstrends` (or, its anti-SAS alias, `pmtrends`) to estimate and compare the average slopes for each diet.

```
R> Chick.lst <- lstrends (Chick.lmer, ~ Diet, var = "Time")
```

Now, let's summarize the estimated trends and pairwise comparisons of these slopes using a compact letter display.

```
R> cld (Chick.lst)
```

```
 Diet Time.trend        SE    df lower.CL upper.CL .group
 1      0.309277 0.0239921 47.63 0.261028 0.357526  1
 2      0.392980 0.0330126 46.20 0.326537 0.459423  12
 4      0.431702 0.0330498 46.41 0.365192 0.498213   2
 3      0.490856 0.0330126 46.20 0.424413 0.557299   2

Trends are based on the sqrt (transformed) scale
Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 4 estimates
significance level used: alpha = 0.05
```

According to the Tukey HSD comparisons (with default significance level of .05), there are two groupings of slopes: Diet 1's mean slope is significantly less than 3 or 4's, Diet 2's slope is not distinguished from any other.

Because of the response transformation, the slopes we just computed are for trends on the square-root-weight scale. If you want the trends on the actual weight scale after back-transforming, that is possible via the `transform` argument:

```
R> lstrends(Chick.lmer, ~ Diet | Time, var = "Time",
       transform = "response", at = list(Time = c(5, 15)))
```

```
Time =  5:
 Diet Time.trend       SE    df lower.CL upper.CL
 1       4.89475 0.379708 47.63  4.13114  5.65835
 2       6.52867 0.548445 46.20  5.42484  7.63251
 3       8.35752 0.562086 46.20  7.22623  9.48881
 4       7.52962 0.576443 46.41  6.36957  8.68966

Time = 15:
 Diet Time.trend       SE    df lower.CL upper.CL
 1       6.80779 0.528112 47.63  5.74574  7.86984
 2       9.61734 0.807911 46.20  7.99129 11.24339
 3      13.17630 0.886174 46.20 11.39273 14.95987
 4      11.25696 0.861797 46.41  9.52266 12.99126

Trends are obtained after back-transforming from the sqrt scale
Confidence level used: 0.95
```

We specified two different `Time` values to emphasize that after back-transforming, the slopes are different at each `Time`, whereas (by the model specification) the slopes don't depend on `Time` when we leave it on the square-root scale.

Note: `lstrends` computes a difference quotient based on two slightly different reference grids. Thus, if it it must be called with a model object, not a `ref.grid` object.

13 User preferences

lsmeans sets certain defaults for displaying results—for example, using .95 for the confidence coefficient, and showing intervals for `lsmeans` output and test statistics for `contrast` results. As discussed before, one may use arguments in `summary` to change what is displayed, or `update` to change the defaults for a given object. But suppose you want different defaults to begin with. These can be set using the `lsm.options` statement. For example:

```
R> lsm.options(ref.grid = list(level = .90),
               lsmeans = list(),
               contrast = list(infer = c(TRUE, TRUE)))
```

This requests that any object created by `ref.grid` be set to have confidence levels default to 90%, and that `contrast` results are displayed with both intervals and tests. No new options are set for `lsmeans` results, and the `lsmeans` part could have been omitted. These options are stored with objects created by `ref.grid`, `lsmeans`, and `contrast`. For example, even though no new defaults are set for `lsmeans`, future calls to `lsmeans` *on a model object* will be displayed with 90% confidence intervals, because `lsmeans` calls `ref.grid`. However, calling `lsmeans` on an existing `"ref.grid"` object will inherit that object's setting.

Certain other options are available; for example, the `"estble.tol"` option sets the tolerance for determining estimability of linear contrasts. To see its current value:

```
R> get.lsm.option("estble.tol")
```

```
[1] 1e-08
```

Defaults for this and some other parameters are saved in `defaults.lsm`.

14 Two-sided formulas

In its original design, the only way to obtain contrasts and comparisons in **lsmeans** was to specify a two-sided formula, e.g., `pairwise ~ treatment`, in the `lsmeans` call. The result is then a list of `lsmobj` objects (class `"lsm.list"`). In its newer versions, **lsmeans** offers a richer family of objects that can be re-used, and dealing with a list of objects can be awkward or confusing, so its continued use is not encouraged. Nonetheless, it is still available for backward compatibility.

Here is an example where, with one command, we obtain both the LS means and pairwise comparisons for `Variety` in the model `Oats.lmer2`:

```
R> lsmeans(Oats.lmer2, pairwise ~ Variety)

$lsmeans
 Variety      lsmean        SE    df lower.CL upper.CL
 Golden Rain 4.66205 0.0751092 10.65  4.52676  4.79734
 Marvellous  4.71970 0.0751092 10.65  4.58441  4.85499
 Victory     4.58262 0.0751092 10.65  4.44733  4.71791

Degrees-of-freedom method: satterthwaite
Results are given on the log (not the response) scale.
Confidence level used: 0.9

$contrasts
 contrast                   estimate        SE df   lower.CL upper.CL t.ratio p.value
 Golden Rain - Marvellous -0.0576490 0.0686844 10 -0.2164788 0.101181  -0.839  0.6883
 Golden Rain - Victory     0.0794312 0.0686844 10 -0.0793986 0.238261   1.156  0.5036
 Marvellous - Victory      0.1370802 0.0686844 10 -0.0217496 0.295910   1.996  0.1636

Results are given on the log (not the response) scale.
Confidence level used: 0.9
Conf-level adjustment: tukey method for comparing a family of 3 estimates
P value adjustment: tukey method for comparing a family of 3 estimates
```

This example also illustrates the effect of the preceding `lsm.options` settings. Let us now return to the default display for contrast results.

```
R> lsm.options(ref.grid = NULL, contrast = NULL)
```

15 Messy data

To illustrate some more `lsmeans` capabilities, consider the dataset named `nutrition` that is provided with the **lsmeans** package. These data come from Milliken and Johnson (1992), and contain the results of an observational study on nutrition education. Low-income mothers are classified by race, age category, and whether or not they received food stamps (the `group` factor); and the response variable is a gain score (post minus pre scores) after completing a nutrition training program.

Consider the model that includes all main effects and two-way interactions. A Type-II (hierarchical) analysis-of-variance table is also shown.

```
R> nutr.lm <- lm(gain ~ (age + group + race)^2, data = nutrition)
R> library("car")
R> Anova(nutr.lm)
```

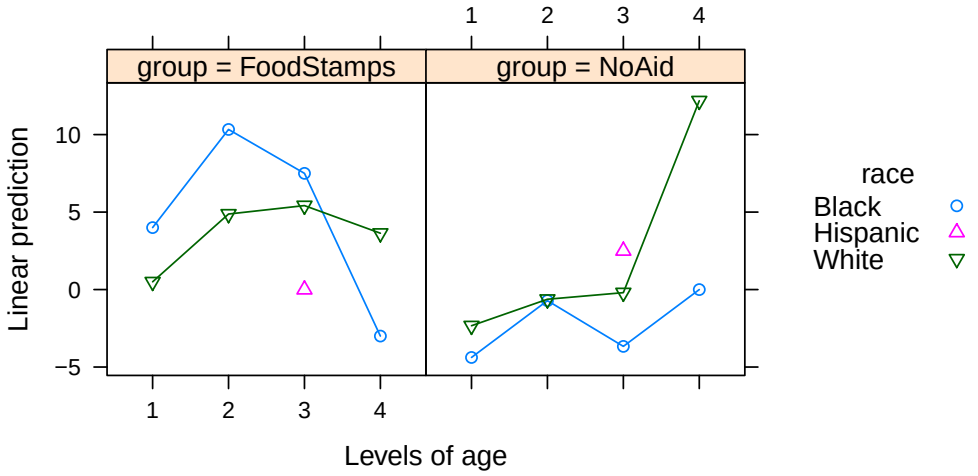

Figure 5: Predictions for the nutrition data

```
Anova Table (Type II tests)

Response: gain
           Sum Sq Df F value  Pr(>F)
age          82.4  3   0.961   0.414
group       658.1  1  23.044 6.1e-06
race         11.2  2   0.196   0.823
age:group    91.6  3   1.069   0.366
age:race     87.3  3   1.019   0.388
group:race  113.7  2   1.991   0.142
Residuals  2627.5 92
```

One main effect (`group`) is quite significant, and there is possibly an interaction with `race`. Let us look at the `group` by `race` LS means:

```
R> lsmip(nutr.lm, race ~ age | group)
R> lsmeans(nutr.lm, ~ group*race)

 group      race       lsmean      SE df    lower.CL upper.CL
 FoodStamps Black     4.70826 2.36812 92  0.00497136  9.41154
 NoAid      Black    -2.19040 2.49058 92 -7.13689810  2.75610
 FoodStamps Hispanic       NA      NA NA          NA       NA
 NoAid      Hispanic       NA      NA NA          NA       NA
 FoodStamps White     3.60768 1.15562 92  1.31252147  5.90284
 NoAid      White     2.25634 2.38927 92 -2.48896668  7.00164

Results are averaged over the levels of: age
Confidence level used: 0.95
```

Figure 5 shows the predictions from this model. One thing the output illustrates is that `lsmeans` incorporates an estimability check, and returns a missing value when a prediction cannot be made

uniquely. In this example, we have very few Hispanic mothers in the dataset, resulting in empty cells. This creates a rank deficiency in the fitted model, and some predictors are thrown out.

We can avoid non-estimable cases by using `at` to restrict the reference levels to a smaller set. A useful summary of the results might be obtained by narrowing the scope of the reference levels to two races and the two middle age groups, where most of the data lie. However, always keep in mind that whenever we change the reference grid, we also change the definition of the LS means. Moreover, it may be more appropriate to average the two ages using weights proportional to their frequencies in the data set. The simplest way to do this is to add a `weights` argument.[4] With those ideas in mind, here are the LS means and comparisons within rows and columns:

```
R> nutr.lsm <- lsmeans(nutr.lm, ~ group * race, weights = "proportional",
       at = list(age = c("2","3"), race = c("Black","White")))
```

So here are the results

```
R> nutr.lsm

 group      race     lsmean       SE df lower.CL  upper.CL
 FoodStamps Black  8.275710 2.917880 92  2.48055 14.070871
 NoAid      Black -2.858277 1.678104 92 -6.19114  0.474582
 FoodStamps White  5.270305 0.868032 92  3.54632  6.994292
 NoAid      White -0.316369 1.010292 92 -2.32290  1.690158

Results are averaged over the levels of: age
Confidence level used: 0.95

R> summary(pairs(nutr.lsm, by = "race"), by = NULL)

 contrast           race  estimate      SE df t.ratio p.value
 FoodStamps - NoAid Black 11.13399 3.55123 92   3.135  0.0023
 FoodStamps - NoAid White  5.58667 1.33198 92   4.194  0.0001

Results are averaged over the levels of: age

R> summary(pairs(nutr.lsm, by = "group"), by = NULL)

 contrast      group      estimate      SE df t.ratio p.value
 Black - White FoodStamps  3.00540 3.01639 92   0.996  0.3217
 Black - White NoAid      -2.54191 1.95876 92  -1.298  0.1976

Results are averaged over the levels of: age
```

The general conclusion from these analyses is that for age groups 2 and 3, the expected gains from the training are higher among families receiving food stamps. Note that this analysis is somewhat different than the results we would obtain by subsetting the data before analysis, as we are borrowing information from the other observations in estimating and testing these LS means.

[4] It may also be done by specifying a custom function in the `fac.reduce` argument, but for simple weighting, `weights` is simpler.

```
R> lsmeans(nutr.lm, "race", weights = "equal")

 race      lsmean      SE df  lower.CL upper.CL
 Black    1.25893 1.64600 92 -2.010175  4.52803
 Hispanic      NA      NA NA        NA       NA
 White    2.93201 1.34661 92  0.257519  5.60650

Results are averaged over the levels of: age, group
Confidence level used: 0.95

R> lsmeans(nutr.lm, "race", weights = "prop")

 race      lsmean      SE df  lower.CL upper.CL
 Black    1.92655 1.39403 92 -0.842112  4.69522
 Hispanic      NA      NA NA        NA       NA
 White    2.52282 0.60446 92  1.322310  3.72333

Results are averaged over the levels of: age, group
Confidence level used: 0.95
```

```
R> lsmeans(nutr.lm, "race", weights = "outer")

 race      lsmean       SE df  lower.CL upper.CL
 Black    2.54667 1.431362 92 -0.296136  5.38948
 Hispanic      NA       NA NA        NA       NA
 White    3.14294 0.749415 92  1.654536  4.63134

Results are averaged over the levels of: age, group
Confidence level used: 0.95

R> lsmeans(nutr.lm, "race", weights = "cells")

 race       lsmean       SE df lower.CL upper.CL
 Black    0.380952 1.166180 92 -1.93518  2.69709
 Hispanic 1.666667 3.085422 92 -4.46125  7.79458
 White    2.795181 0.586592 92  1.63016  3.96020

Results are averaged over the levels of: age, group
Confidence level used: 0.95
```

Figure 6: Comparison of four different weighting methods

15.1 More on weighting

The `weights` argument can be a vector of numerical weights (it has to be of the right length), or one of four text values: `"equal"` (weight the predictions equally when averaging them, the default), `"proportional"` (weight them proportionally to the observed frequencies of the factor combinations being averaged over), `"outer"` (weight according to the outer products of the one-factor marginal counts), or `"cells"` (weight each mean differently, according to the frequencies of the predictions being averaged). Figure 6 shows the LS means for `race` using the four different weighting schemes. (Note: If the model itself has weights, then the total weights are used instead of counts.)

Note there are four different sets of answers. The `"equal"` weighting is self-explanatory. But what's the distinction between `"proportional"` and `"outer"`? To clarify, consider:

```
R> temp = lsmeans(nutr.lm, c("group","race"), weights = "prop")
R> lsmeans(temp, "race", weights = "prop")

 race      lsmean       SE df  lower.CL upper.CL
 Black    2.54667 1.431362 92 -0.296136  5.38948
 Hispanic      NA       NA NA        NA       NA
 White    3.14294 0.749415 92  1.654536  4.63134

Results are averaged over the levels of: age, group
Confidence level used: 0.95
```

The previous results using `"outer"` weights are the same as those using `"proportional"` weights on one factor at a time. Thus, if only one factor is being averaged over, `"outer"` and `"proportional"` are the same. Another way to look at it is that outer weights are like the expected counts in a chi-square test; each factor is weighted independently of the others.

The results for `"cells"` weights stand out because everything is estimable—that's because the empty cells in the data were given weight zero. These results are the same as the unadjusted means:

```
R> with(nutrition, tapply(gain, race, mean))
```

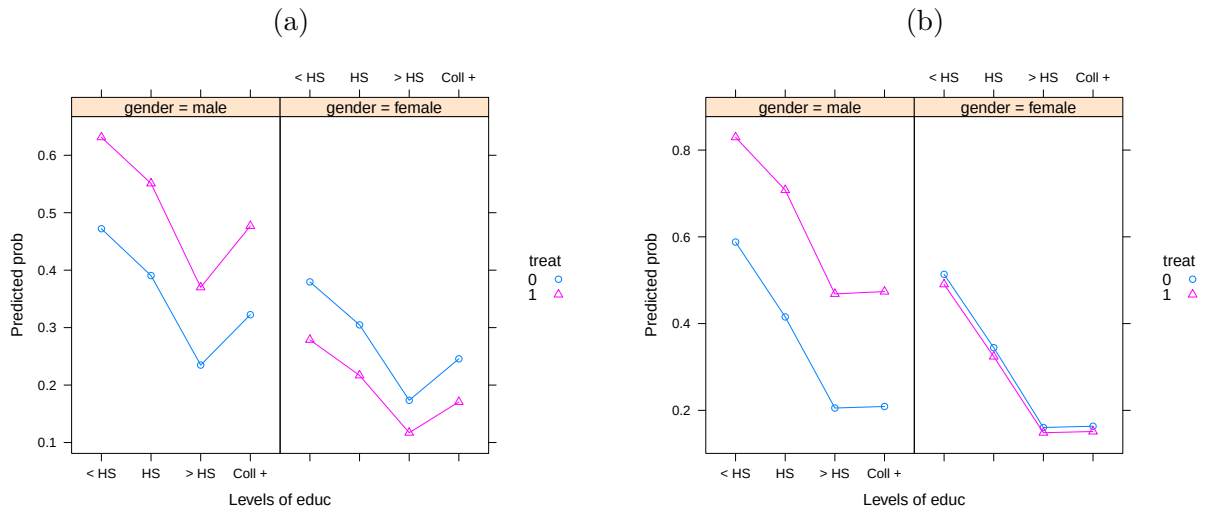

Figure 7: Estimated responses for the `framing` data. (a) Holding `emo` constant at its mean; (b) Using predictions of `emo` for each `treat`.

```
   Black Hispanic    White
0.380952 1.666667 2.795181
```

15.2 Alternative covariate adjustments

The `framing` data in the **mediation** package has the results of an experiment conducted by Brader *et al.* (2008) where subjects were given the opportunity to send a message to Congress regarding immigration. However, before being offered this, some subjects (`treat=1`) were first shown a news story that portrays Latinos in a negative way. Besides the binary response (whether or not they elected to send a message), we also measured `emo`, the subjects' emotional state after the treatment was applied. There are various demographic variables as well.

Before fitting a logistic regression model, I will change the labels for `educ` to shorter strings.

```
R> library("mediation")
R> levels(framing$educ) = c("NA","Ref","< HS", "HS", "> HS","Coll +")
R> framing.glm = glm(cong_mesg ~ age + income + educ + emo + gender * factor(treat),
                     family = binomial, data = framing)
```

The left-hand plot in Figure 7 displays the conventional adjusted means, where predictions are made with the covariates `age`, `income`, and `emo` set to their mean values:

```
R> lsmip(framing.glm, treat ~ educ | gender, type = "response")
```

This plot is rather implausible because the displayed treatment effects are the opposite for females as for males, and the effect of education isn't monotone as one might expect.

However, `emo` is a post-treatment measurement. This means that the treatment could have affected it (it is a *mediating* covariate). If it is indeed affected by `treat`, then Figure 7(a) would be misleading because `emo` is held constant. Instead, consider making the predictions where `emo` is set to its predicted value at each combination of `treat` and the demographic variables. This is easily done by setting `cov.reduce` to a formula for how to predict `emo`:

```
R> lsmip(framing.glm, treat ~ educ | gender, type = "response",
         cov.reduce = emo ~ treat*gender + age + educ + income)
```

This plot is shown in Figure 7(b). It is quite different, suggesting that `emo` does indeed play a strong mediating role. (The **mediation** package has functions for estimating the strength of these effects.) The predictions suggest that, taking emotional response into account, male subjects exposed to the negative news story are more likely to send the message than are females or those not seeing the negative news story. Also, the effect of `educ` is (almost) monotone. You can see what values of `emo` are used in these predictions by looking at the `grid` slot in the reference grid:

```
R> ref.grid(framing.glm,
       cov.reduce = emo ~ treat*gender + age + educ + income)@grid
```

```
      age  income   educ     emo gender treat .wgt.
1  47.766 10.7962   < HS 8.32779   male     0     6
2  47.766 10.7962     HS 7.27257   male     0    32
3  47.766 10.7962   > HS 6.47527   male     0    24
4  47.766 10.7962 Coll + 5.26561   male     0    34
5  47.766 10.7962   < HS 8.55422 female     0     8
6  47.766 10.7962     HS 7.49899 female     0    37
7  47.766 10.7962   > HS 6.70169 female     0    23
8  47.766 10.7962 Coll + 5.49203 female     0    33
9  47.766 10.7962   < HS 9.99963   male     1     2
10 47.766 10.7962     HS 8.94440   male     1     9
11 47.766 10.7962   > HS 8.14711   male     1    13
12 47.766 10.7962 Coll + 6.93745   male     1     6
13 47.766 10.7962   < HS 9.61805 female     1     4
14 47.766 10.7962     HS 8.56282 female     1    14
15 47.766 10.7962   > HS 7.76552 female     1    10
16 47.766 10.7962 Coll + 6.55587 female     1    10
```

whereas the overall mean of 6.974 is used as the value of `emo` in Figure 7(a).

Another use of formulas in `cov.reduce` is to create representative values of some covariates when others are specified in `at`. For example, suppose there are three covariates x_1, x_2, x_3 in a model, and we want to see predictions at a few different values of x_1. We might use

```
R> rg <- ref.grid(my.model, at = list(x1 = c(5,10,15)),
                  cov.reduce = list(x2 ~ x1,  x3 ~ x1 + x2))
```

(When more than one formula is given, they are processed in the order given.) The values used for x_2 and x_3 will depend on x_1 and should in some sense be more realistic values of those covariates as x_1 varies than would be the overall means of x_2 and x_3. Of course, it would be important to display the values used—available as `rg@grid`—when reporting the analysis.

16 Other types of models

16.1 Models supported by lsmeans

The **lsmeans** package comes with built-in support for quite a number of packages and model classes, including `"lm"`, `"mlm"`, `"aov"`, `"aovlist"`, and `"glm"` in the **stats** package, mixed models

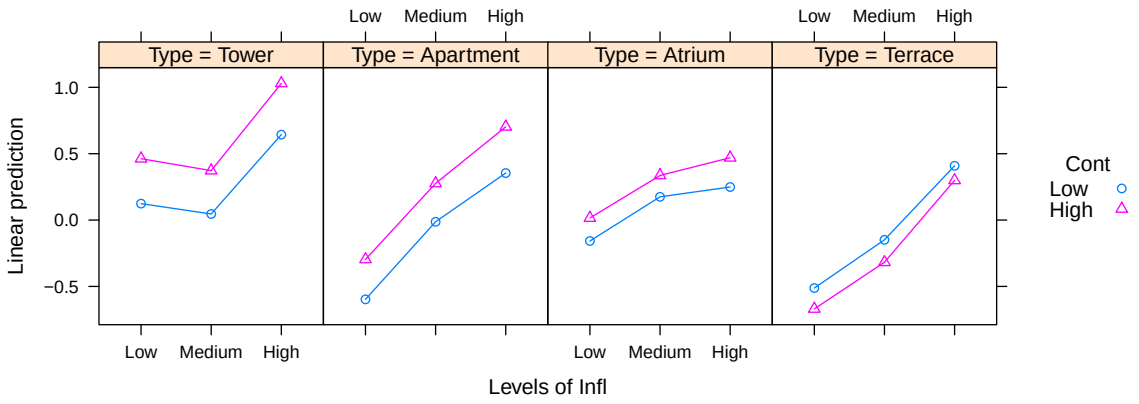

Figure 8: Interaction plot for the latent variable in the `housing` example.

such as `"lme"`, `"lmerMod"`, and `"glmerMod"`, several survival models, GEE-type models, models having responses that are ordinal, multinomial, counts, and interval-(0,1), and Bayesian models. For a complete list, use `help("models")`.

lsmeans support for all these models works similarly to the examples we have presented. Note that generalized linear or mixed models, and several others such as survival models, typically employ link functions such as `log` or `logit`. In most cases, the LS means displayed are on the scale of the linear predictor, and any averaging over the reference grid is performed on the linear-predictor scale; but there are exceptions. Some objects have optional arguments that can be specified in the `ref.grid` or `lsmeans` call: see `?models` for details.

16.2 Ordinal-data example

The `clm` and `clmm` functions in **ordinal**, as well as the `polr` function in **MASS**, fit polytomous regression models to Likert-scale data. They do this by modeling the ordinal response as a categorization of a continuous latent variable S, then estimating thresholds for this categorization and fitting a generalized linear model to the cumulative probabilities for each threshold. By default, `lsmeans` produces predictions of the latent variable.

The example shown here is based on the `housing` data in the **MASS** package, where the response variable is satisfaction (`Sat`) on a three-point scale of low, medium, high; and predictors include `Type` (type of rental unit, four levels), `Infl` (influence on management of the unit, three levels), and `Cont` (contact with other residents, two levels). We will assume that the latent variable is normally distributed (by specifying a probit link).

```
R> library("ordinal")
R> data(housing, package = "MASS")
R> housing.clm <- clm(Sat ~ (Infl + Type + Cont)^2,
                      data = housing, weights = Freq, link = "probit")
R> lsmip(housing.clm, Cont ~ Infl | Type, layout = c(4,1))
```

The plot is shown in Figure 8. Generally, the higher the influence, the higher the satisfaction. Overall F tests of the `Infl` effect suggest that it is strong for all four housing types:

```
R> test(pairs(lsmeans(housing.clm, ~ Infl | Type)), joint = TRUE)
```

```
 Type      df1 df2      F p.value
 Tower       2  NA  7.938  0.0004
 Apartment   2  NA 40.041  <.0001
 Atrium      2  NA  3.135  0.0435
 Terrace     2  NA 10.823  <.0001
```

The tests are asymptotic (signaled by `df2 = NA`), so they are actually chi-square tests for the statistics $X^2 = df_1 \cdot F$ with df_1 degrees of freedom. Higher contact also seems to be associated with higher satisfaction, but terrace apartments may be an exception. Let's see:

```
R> test(pairs(lsmeans(housing.clm, ~ Cont | Type)), joint = TRUE)

 Type      df1 df2      F p.value
 Tower       1  NA  8.466  0.0036
 Apartment   1  NA 13.098  0.0003
 Atrium      1  NA  1.421  0.2332
 Terrace     1  NA  0.931  0.3346
```

So the effect is inconclusive for both atria and terraces.

The `mode` argument may be used to choose what to examine. Modes `"linear.predictor"` and `"cum.prob"` create an additional pseudo-factor named `cut` for the thresholds at which the predictions are made.

```
R> ref.grid(housing.clm, mode = "cum.prob")

'ref.grid' object with variables:
    Infl = Low, Medium, High
    Type = Tower, Apartment, Atrium, Terrace
    Cont = Low, High
    cut = multivariate response levels: Low|Medium, Medium|High
```

So here are our estimated marginal probabilities for `Infl` of being less than highly satisfied:

```
R> lsmeans(housing.clm, ~ Infl, at = list(cut = "Medium|High"),
           mode = "cum.prob")

 Infl    cumprob        SE df asymp.LCL asymp.UCL
 Low    0.704649 0.0178354 NA  0.669692  0.739605
 Medium 0.606165 0.0197268 NA  0.567501  0.644828
 High   0.442192 0.0269420 NA  0.389386  0.494997

Results are averaged over the levels of: Type, Cont
Confidence level used: 0.95
```

Compare these results with those for the back-transformed linear predictor:

```
R> summary(lsmeans(housing.clm, ~ Infl, at = list(cut = "Medium|High"),
                   mode = "linear.predictor"), type = "response")
```

```
 Infl    cumprob        SE df asymp.LCL asymp.UCL
 Low    0.716271 0.0185478 NA  0.678853  0.751465
 Medium 0.609159 0.0202367 NA  0.568995  0.648191
 High   0.439691 0.0275347 NA  0.386446  0.494052

Results are averaged over the levels of: Type, Cont
Confidence level used: 0.95
Intervals are back-transformed from the probit scale
```

The results are similar, but somewhat different because of the back-transformation coming before (first case) or after (second case) averaging or computing confidence limits.

16.3 Chick weights, revisited

Previously, we used the `ChickWeight` data to illustrate the use of `lstrends`. That example made the simplifying assumption that the growth trends are linear, which is clearly questionable. To do a better job of fitting the data, consider instead the idea of fitting a logistic curve to each chick's data. The **stats** package provides the `SSlogis` function for this purpose: it is an S-shaped curve (scaled from the cdf of a logistic distribution) having three parameters `asym` (the asymptotic value at which it levels off), `xmid` (the x coordinate of its inflection point), and `scal` (roughly the difference between the median and the .73rd quantile). Also, the **nlme** package's `nlme` function can fit a set of nonlinear curves such that the parameters of those curves may be modeled using a mixed-effects linear model.

Accordingly, let us fit a model where each chick has a logistic curve for which the `asym` parameter varies randomly for each chick, and for which both `asym` and `xmid` depend on the chick's diet. We chose starting values by examining the curves and making a rough judgment of the typical asymptotic value, midpoint, and scale for each diet. We need to keep firmly in mind how factors are coded; so we explicitly show that we intend to use `"contr.treatment"` coding, by which the first mean is estimated directly, and the remaining estimates are offsets from that. We need a set of four starting values for `asym` and `xmid`, and one for `scal`.

```
R> require("nlme")
R> options(contrasts = c("contr.treatment", "contr.poly"))
R> Chick.nlme = nlme(weight ~ SSlogis(Time, asym, xmid, scal),
       data = ChickWeight,
       fixed = list(asym + xmid ~ Diet, scal ~ 1),
       random = asym ~ 1 | Chick,
       start = c(200, 100, 200, 100,   10, 0, 0, 0,   7))
R> Chick.nlme

Nonlinear mixed-effects model fit by maximum likelihood
  Model: weight ~ SSlogis(Time, asym, xmid, scal)
  Data: ChickWeight
  Log-likelihood: -2439.4
  Fixed: list(asym + xmid ~ Diet, scal ~ 1)
asym.(Intercept)       asym.Diet2       asym.Diet3       asym.Diet4 xmid.(Intercept)
      221.207493        75.235197       198.767610        90.692332        12.216286
      xmid.Diet2       xmid.Diet3       xmid.Diet4             scal
        1.870092         4.464178         0.929361         7.059857
```

```
Random effects:
 Formula: asym ~ 1 | Chick
        asym.(Intercept) Residual
StdDev:          66.9958  13.8516

Number of Observations: 578
Number of Groups: 50
```

Now we can use `lsmeans` to compare the parameters based on `Diet`:

```
R> cld(lsmeans(Chick.nlme, ~ Diet, param = "asym"))

 Diet  lsmean      SE  df lower.CL upper.CL .group
 1    221.207 16.8641 520  188.077  254.338  1
 2    296.443 24.3132 520  248.679  344.207   2
 4    311.900 24.0879 520  264.578  359.221   2
 3    419.975 29.4935 520  362.034  477.916    3

Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 4 estimates
significance level used: alpha = 0.05

R> cld(lsmeans(Chick.nlme, ~ Diet, param = "xmid"))

 Diet  lsmean       SE  df lower.CL upper.CL .group
 1    12.2163 0.552564 520  11.1308  13.3018  1
 4    13.1456 0.598941 520  11.9690  14.3223  12
 2    14.0864 0.642599 520  12.8240  15.3488   2
 3    16.6805 0.726923 520  15.2524  18.1085    3

Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 4 estimates
significance level used: alpha = 0.05
```

The result is that diet 3 has both a higher mean `asym` an a higher mean `xmid` than the other diets. This is compatible with the results of the earlier `lstrends` analysis, but grounded in terms of the parameters of the logistic curve.

16.4 Extending to more models

The functions `ref.grid` and `lsmeans` work by first reconstructing the dataset (so that the reference grid can be identified) and extracting needed information about the model, such as the regression coefficients, covariance matrix, and the linear functions associated with each point in the reference grid. For a fitted model of class, say, `"modelobj"`, these tasks are accomplished by defining S3 methods `recover.data.modelobj` and `lsm.basis.modelobj`. The help page `"extending-lsmeans"` and the vignette by the same name provide details and examples.

Developers of packages that fit models are encouraged to include support for **lsmeans** by incorporating (and exporting) `recover.data` and `lsm.basis` methods for their model classes.

16.5 Bayesian models

Certain Bayesian models are now supported by **lsmeans**. For illustration, consider a two-factor Poisson regression example given in the **MCMCpack** package:

```
R> library("MCMCpack")
R> counts <- c(18, 17, 15,   20, 10, 20,   25, 13, 12)
R> outcome <- gl(3, 1, 9)
R> treatment <- gl(3, 3)
R> posterior <- MCMCpoisson(counts ~ outcome + treatment, mcmc = 1000)
```

The result is an `mcmc` object (defined in the **coda** package), but it has an added `"call"` attribute that enables `lsmeans` to do its work. Here are results for treatments, averaged over outcomes:

```
R> ( post.lsm <- lsmeans(posterior, "treatment") )
```

```
 treatment  lsmean       SE df asymp.LCL asymp.UCL
 1         2.78820 0.158523 NA   2.47750   3.09890
 2         2.80098 0.129532 NA   2.54711   3.05486
 3         2.80179 0.138729 NA   2.52988   3.07369

Results are averaged over the levels of: outcome
Confidence level used: 0.95
```

This is a frequentist summary, based on the mean and covariance of the regression parameters in the `posterior` sample. But **lsmeans** provides an `as.mcmc` method to obtain a sample from the posterior distribution of the LS means (that is, the original posterior sample of regression coefficients, transformed by the appropriate linear functions.)

```
R> library("coda")
R> summary(as.mcmc(post.lsm))
```

```
Iterations = 1:1000
Thinning interval = 1
Number of chains = 1
Sample size per chain = 1000

1. Empirical mean and standard deviation for each variable,
   plus standard error of the mean:

            Mean    SD Naive SE Time-series SE
treatment 1 2.79 0.159  0.00501         0.0233
treatment 2 2.80 0.130  0.00410         0.0160
treatment 3 2.80 0.139  0.00439         0.0167

2. Quantiles for each variable:

            2.5%  25%  50%  75% 97.5%
treatment 1 2.45 2.69 2.81 2.91  3.07
treatment 2 2.53 2.71 2.80 2.89  3.04
treatment 3 2.52 2.72 2.81 2.90  3.04
```

Since `as.mcmc` produces an `mcmc` object, any of the other available methods may be used with it.

17 Discussion

The design goal of **lsmeans** is primarily to provide the functionality of the `LSMEANS` statement in various SAS procedures. Thus its emphasis is on tabular results which, of course, may also be used as data for further analysis or graphics. By design, it can be extended with relative ease to additional model classes. A unique capability of **lsmeans** is its explicit reliance on the concept of a reference grid, which I feel is a useful approach for understanding what is being computed.

Some **lsmeans** capabilities exceed those of SAS, including the `lstrends` capability, more flexibility in organizing the output, and more built-in contrast families. In addition, SAS does not allow LS means for factor combinations when the model does not include the interaction of those factors; or creating a grid of covariate values using `at`.

There are a few other R packages that provide capabilities that overlap with those of **lsmeans**. The **effects** package (Fox, 2003; Fox and Hong, 2009) can compute LS means. However, for an unbalanced dataset, it does not use equal weights, but rather it appears to use "outer" weights, as described in Section 15.1. Also, it does not check estimability, so some results could be questionable. The emphasis of **effects** is on graphical rather than tabular displays. It has special strengths for curve-fitting models such as splines. In contrast, **lsmeans**'s strengths are more in the area of factorial models where one wants traditional summaries in the form of estimates, contrasts, and interaction plots.

The **doBy** package (Højsgaard *et al.*, 2013) provides an `LSmeans` function that has some of the capabilities of `lsmeans`, but it produces a data frame rather than a reusable object. In earlier versions of the package, this function was named `popMeans`. The package also has an `LSmatrix` function to obtain the linear functions needed to obtain LS means. Also, the **lmerTest** package also offers an `lsmeans` function, as well as `difflsmeans` for differences of LS means. These are designed particularly for `lmerMod` objects.

References

Bates D, Maechler M, Bolker B, Walker S (2013). ***lme4****: Linear Mixed-effects Models Using* ***Eigen*** *and S4*. R package version 1.1-0, URL http://lme4.r-forge.r-project.org/.

Brader T, Valentino N, Suhay E (2008). "What triggers public opposition to immigration? Anxiety, group cues, and immigration threat." *American Journal of Political Science*, **52**(4), 959–978.

Dahl DB (2016). ***xtable****: Export Tables to LaTeX or HTML*. R package version 1.8-2/r110, URL https://R-Forge.R-project.org/projects/xtable/.

Fox J (2003). "Effect Displays in R for Generalised Linear Models." *Journal of Statistical Software*, **8**(15), 1–27. URL http://www.jstatsoft.org/v08/i15/.

Fox J, Hong J (2009). "Effect Displays in R for Multinomial and Proportional-Odds Logit Models: Extensions to the **effects** Package." *Journal of Statistical Software*, **32**(1), 1–24. URL http://www.jstatsoft.org/v32/i01/.

Goodnight JH, Harvey WR (1997). "Least squares means in the fixed effects general model." *Technical Report SAS Technical Report R-103*, SAS Institute Inc.

Graves S, Piepho HP, Selzer L, Dorai-Raj S (2012). ***multcompView****: Visualizations of Paired Comparisons*. R package version 0.1-5, URL http://CRAN.R-project.org/package=multcompView.

Halekoh U, Højsgaard S (2013). ***pbkrtest**: Parametric Bootstrap and Kenward Roger Based Methods for Mixed Model Comparison*. R package version 0.3-8, URL http://CRAN.R-project.org/package=pbkrtest.

Harvey W (1960). "Least-squares analysis of data with unequal subclass numbers." *Technical Report ARS-20-8*, USDA National Agricultural Library.

Harvey W (1976). "Use of the HARVEY Procedure." In *SUGI Proceedings*. URL http://www.sascommunity.org/sugi/SUGI76/Sugi-76-20%20Harvey.pdf.

Harvey WR (1977). *User's guide for LSML 76. Mixed model least-squares and maximum likelihood computer program*. Ohio State University.

Højsgaard S, Halekoh U, Robison-Cox J, Wright K, Leidi AA (2013). ***doBy**: Groupwise summary statistics, LSmeans, general linear contrasts, various utilities*. R package version 4.5-10, URL http://CRAN.R-project.org/package=doBy.

Hothorn T, Bretz F, Westfall P (2013). ***multcomp**: Simultaneous Inference in General Parametric Models*. R package version 1.3-1, URL http://CRAN.R-project.org/package=multcomp.

Kuznetsova A, Bruun Brockhoff P, Haubo Bojesen Christensen R (2016). *lmerTest: Tests in Linear Mixed Effects Models*. R package version 2.0-32, URL https://CRAN.R-project.org/package=lmerTest.

Milliken GA, Johnson DE (1992). *Analysis of Messy Data – Volume I: Designed Experiments*. Chapman & Hall/CRC. ISBN 0-412-99081-4.

Pinheiro J, Bates D, R-core (2013). ***nlme**: Linear and Nonlinear Mixed Effects Models*. R package version 3.1-113, URL http://CRAN.R-project.org/package=nlme.

SAS Institute Inc (2012). "`LSMEANS` Statement." In *SAS/STAT(R) 9.3 User's Guide*. URL http://support.sas.com/documentation/cdl/en/statug/63962/HTML/default/viewer.htm#statug_introcom_a0000003362.htm.

Schuirmann D (1987). "A Comparison of the Two One-Sided Tests Procedure and the Power Approach for Assessing the Equivalence of Average Bioavailability." *Journal of Pharmacokinetics and Biopharmaceutics*, **15**(6), 657–680.

Searle SR, Speed FM, Milliken GA (1980). "Population marginal means in the linear model: An alternative to least squares means." *The American Statistician*, **34**(4), 216–221.

Yates F (1935). "Complex Experiments." *Journal of the Royal Statistical Society (Supplement)*, **2**, 181–247.

Index