*Journal of Statistical Software*

*January 2016, Volume 69, Issue 1.* doi: 10.18637/jss.v069.i01

# Least-Squares Means: The R Package lsmeans

**Russell V. Lenth**
The University of Iowa

**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 (Lenth 2016) provides a simple way of obtaining least-squares means and contrasts thereof. It supports many models fitted by R (R Core Team 2015) 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.

*Keywords*: least-squares means, linear models, experimental design.

## 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 (Oehlert 2000, p. 456, for example). 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 (Milliken and Johnson 1992, Section 11.2).

LS means are not always well understood, in part because the term itself is confusing. 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.

**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
```

## 2.1. 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.lm1, "day")
```

or, equivalently (and more efficiently),

```
R> lsmeans(oranges.rg1, "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 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.

## 2.2. Altering the reference grid

The `at` argument may be used to override defaults in defining 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. To get separate sets of predictions for each `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 = ... )
R> lsmeans(oranges.lm1, c("day", "price2"), at = ... )
R> lsmeans(oranges.lm1, ~ day * price, at = ... )
```

The first line above illustrates the formula method for specifying factors, which is more compact. The `|` character replaces the `by` specification. Lines 2 and 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

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.

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), null = 7.50,
+    level = 0.90, adjust = "bon", by = "day") )

day = 2:
 price2   lsmean      SE df lower.CL upper.CL null t.ratio p.value
     40  6.23623 1.88711 23  2.33245  10.1400  7.5  -0.670  1.0000
     60  9.21317 2.10945 23  4.84944  13.5769  7.5   0.812  0.8501

day = 3:
 price2   lsmean      SE df lower.CL upper.CL null t.ratio p.value
     40 13.40599 2.11938 23  9.02173  17.7903  7.5   2.787  0.0210
     60 16.38293 1.90522 23 12.44169  20.3242  7.5   4.662  0.0002

day = 4:
 price2   lsmean      SE df lower.CL upper.CL null t.ratio p.value
     40  8.48371 1.86651 23  4.62254  12.3449  7.5   0.527  1.0000
     60 11.46065 2.17805 23  6.95500  15.9663  7.5   1.818  0.1641

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 0.95 was overridden; and the default null hypothesis that a mean is zero

is overridden. A Bonferroni adjustment was applied to both the intervals and the $p$ values (which causes some of the latter to hit their maximum of 1.0). Because of the `by` specification, 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 January 24, 2016 exchange rate):

```
R> transform(org.sum, lsrubles = lsmean * 78.47)
```

```
  day price2  lsmean     SE df lower.CL upper.CL null  t.ratio    p.value lsrubles
1   2     40  6.2362 1.8871 23   2.3325   10.140  7.5 -0.66969 1.00000000   489.36
2   3     40 13.4060 2.1194 23   9.0217   17.790  7.5  2.78667 0.02097233  1051.97
3   4     40  8.4837 1.8665 23   4.6225   12.345  7.5  0.52703 1.00000000   665.72
4   2     60  9.2132 2.1094 23   4.8494   13.577  7.5  0.81214 0.85007434   722.96
5   3     60 16.3829 1.9052 23  12.4417   20.324  7.5  4.66243 0.00021593  1285.57
6   4     60 11.4607 2.1781 23   6.9550   15.966  7.5  1.81844 0.16409035   899.32
```

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)`). 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 = 0.99)
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
```

# 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

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> 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.

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" (or optionally, revpairwise):

```
R> pairs(org.lsm, reverse = TRUE)
```

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

price2 = 60:
 contrast estimate      SE df t.ratio p.value
 3 - 2     7.16976 2.47970 23   2.891  0.0216
 4 - 2     2.24748 2.44234 23   0.920  0.6333
 4 - 3    -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, Piepho, Selzer, and Dorai-Raj 2015) to be installed.

```
R> cld(days.lsm, alpha = 0.10)
```

```
 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

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"`.

### 4.3. 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. 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 polynomial contrasts for `day`, averaged over `price2`, for the object `org.lsm`:

```
R> lsmeans(org.lsm, poly ~ day)

$lsmeans
 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: store, price2
Confidence level used: 0.95

$contrasts
 contrast   estimate      SE df t.ratio p.value
 linear      2.24748 2.44234 23   0.920  0.3670
 quadratic -12.09205 4.32824 23  -2.794  0.0103

Results are averaged over the levels of: store, price2
```

## 5. Additional examples

While estimates and pairwise comparisons are the fundamental things we need with LS means, other descriptions and analyses are helpful. This section introduces some more data exam-

ples and discusses **lsmeans**'s capabilities for displaying results graphically, dealing with transformed responses (or link functions in generalized linear models), estimating trends when covariates interact with factors, and dealing with messy data.

**Example: Oat yields.** The `Oats` dataset in the **nlme** package (Pinheiro, Bates, and R Core Team 2015) 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, Mächler, Bolker, and Walker 2015). 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
```

(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.

The results for `Oats.lmer` suggest 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)
R> Oats.lsm2

Variety = Golden Rain:
 nitro  lsmean        SE    df lower.CL upper.CL
   0.0 4.35458 0.0770328 11.77  4.18637  4.52279
   0.2 4.57770 0.0745363 10.34  4.41235  4.74304
   0.4 4.72826 0.0745363 10.34  4.56292  4.89361
   0.6 4.80627 0.0770328 11.77  4.63806  4.97448

Variety = Marvellous:
 nitro  lsmean        SE    df lower.CL upper.CL
   0.0 4.41223 0.0770328 11.77  4.24402  4.58044
   0.2 4.63535 0.0745363 10.34  4.47000  4.80069
   0.4 4.78591 0.0745363 10.34  4.62057  4.95126
   0.6 4.86392 0.0770328 11.77  4.69571  5.03213

Variety = Victory:
 nitro  lsmean        SE    df lower.CL upper.CL
   0.0 4.27515 0.0770328 11.77  4.10694  4.44336
   0.2 4.49827 0.0745363 10.34  4.33292  4.66361
   0.4 4.64883 0.0745363 10.34  4.48349  4.81418
   0.6 4.72684 0.0770328 11.77  4.55863  4.89505

Results are given on the log scale.
Confidence level used: 0.95
```

These LS means follow the same quadratic trend for each variety, but with different intercepts.

Fractional degrees of freedom are displayed in these results. These are obtained from the **pbkrtest** package (Halekoh and Højsgaard 2014), and they use the Kenward-Rogers method. (The degrees of freedom for the polynomial contrasts were also obtained from **pbkrtest**, but the results turn out to be integers.)

Incidentally, the polynomial model above could have been specified using `nitro + I(nitro^2)` instead of `poly(nitro, 2)`. What is *not* recommended is to use `nitro + nitrosq`, where `nitrosq` is a previously calculated variable equal to `nitro^2`. The reason is that with

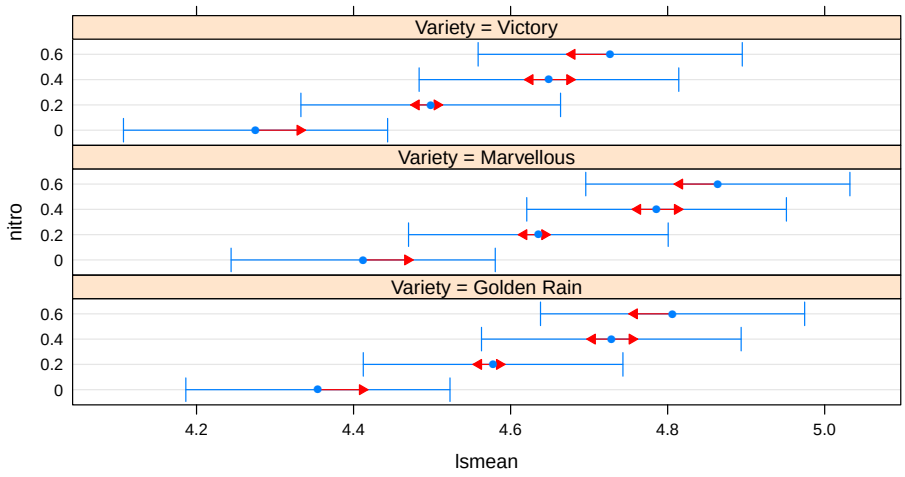


Figure 1: Display of confidence intervals and comparison arrows for the `Oats` example.

`cov.reduce = TRUE`, `lsmeans` will use separate covariate means for `nitro` and `nitrosq`, and the latter will probably not be equal to `mean(nitro)^2` as would be required for consistency.

## 5.1. Displaying LS means graphically

There is a `plot` method for `"ref.grid"` and `"lsmobj"` objects that displays side-by-side confidence intervals, and/or "comparison arrows" that may be used to graphically judge which LS means differ significantly from one another based on whether or not they overlap. In the command below, we request both intervals and comparison arrows. By default, the comparisons will be done using a 0.05 Tukey-adjusted significance level. However, we use `int.adjust` to request that the confidence intervals (defaulting to the 95% level) be unadjusted.

```
R> plot(Oats.lsm2, intervals = TRUE, int.adjust = "none", comparisons = TRUE)
```

The results are shown in Figure 1. Owing to the fact that the model is additive, the comparisons are the same within each `Variety`. The overlapping arrows for `nitro` of 0.4 and 0.6 indicate a nonsignificant difference. The remaining differences are quite significant. Note that it is a mistake to try to use confidence intervals to judge comparisons. In this example, the standard errors of comparisons are much smaller than those of the LS means, because the between-block and between-plot variations cancel out in the comparisons.

In certain other situations where the standard errors of differences vary widely, the comparison-arrow display may not work. In that case, a warning message is displayed.

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
```

The function requires the **lattice** package (Sarkar 2008) to be installed. In this formula, `curve.factors` specifies factor(s) used to delineate one displayed curve from another (i.e.,

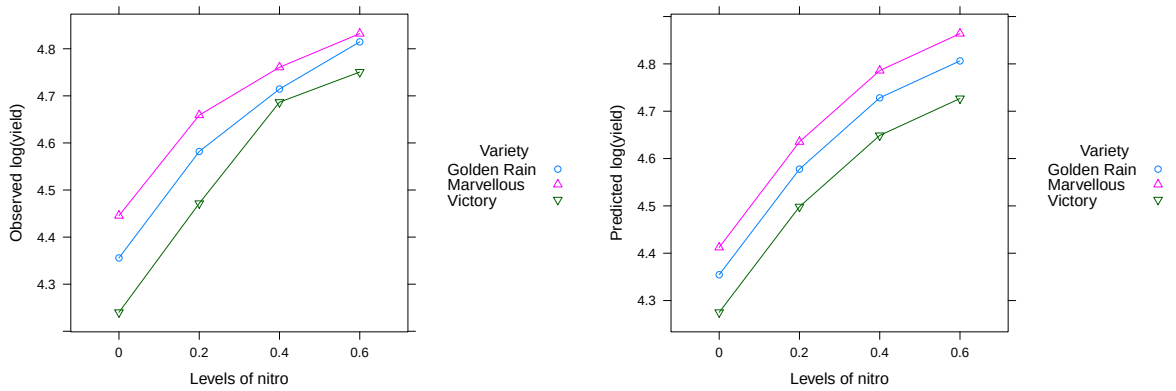


Figure 2: Interaction plots for the cell means and the fitted model, `Oats` example.

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, 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 2. 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.

The help page for `lsmip` gives examples involving several factors, showing the flexibility we have in combining factors or using them to create multi-panel plots.

## 5.2. 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:

```
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 (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

Confidence level used: 0.95
```

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 for transformations is available only for certain standard transformations, namely those supported by the `make.link` function in the **stats** package. Others require more work – see the documentation for `update` for details.

### 5.3. Trends

The `lstrends` function is provided 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 3.

```
R> library(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:

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

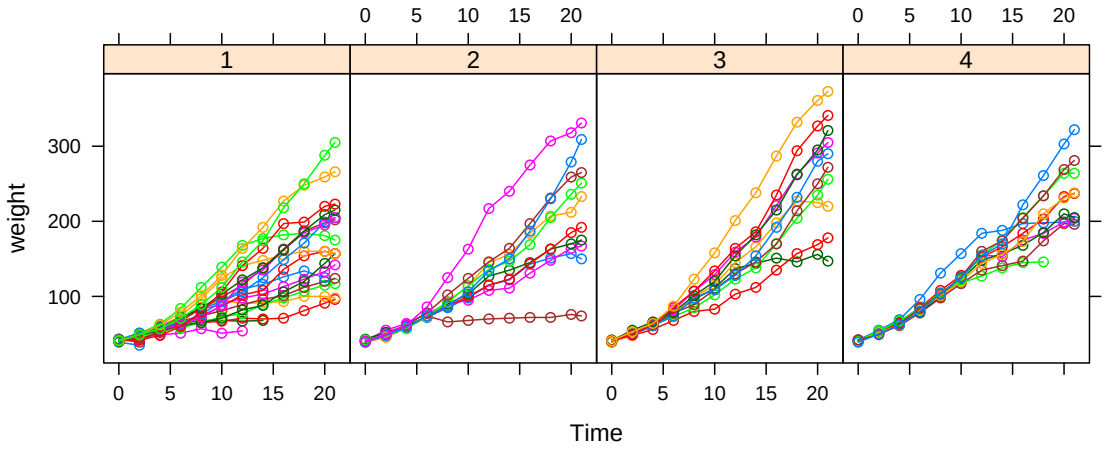


Figure 3: Growth curves of chicks, dataset `ChickWeight`.

We can then call `lstrends` to estimate and compare the average slopes for each diet.

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

 Diet Time.trend       SE    df lower.CL upper.CL
 1       6.33856 0.610488 49.86  5.11227  7.56484
 2       8.60914 0.838003 48.28  6.92447 10.29380
 3      11.42287 0.838003 48.28  9.73821 13.10753
 4       9.55583 0.839245 48.56  7.86892 11.24273

Confidence level used: 0.95
```

Here we obtain estimates and pairwise comparisons of the slopes using a compact letter display.

```
R> cld (Chick.lst)

 Diet Time.trend       SE    df lower.CL upper.CL .group
 1       6.33856 0.610488 49.86  5.11227  7.56484  1
 2       8.60914 0.838003 48.28  6.92447 10.29380  12
 4       9.55583 0.839245 48.56  7.86892 11.24273   2
 3      11.42287 0.838003 48.28  9.73821 13.10753   2

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 0.05), 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.

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

### 5.4. 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 using the **car** package (Fox and Weisberg 2011).

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

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
```

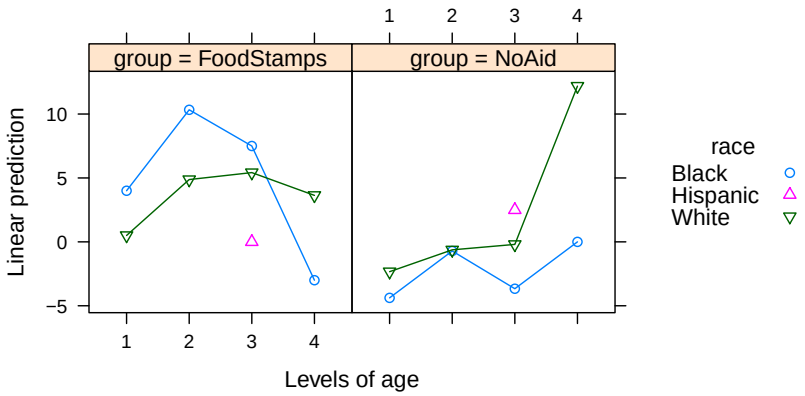


Figure 4: Predictions for the nutrition data.

Figure 4 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. 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")))
```

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.

The `weights` argument is a convenience for certain standard weighting schemes. More general weighting schemes may be implemented by writing a custom function and using `fac.reduce`. For example, to weight by the frequencies in the whole dataset instead of just ages 2 and 3, use (results not shown):

```
R> wtavg <- function(coefs, lev) (23 * coefs[1, ] + 64 * coefs[2, ]) / 87
R> nutr.lsm <- lsmeans(nutr.lm, ~ group * race, fac.reduce = wtavg, ...
```

As a side note, the **effects** package (Fox 2003; Fox and Hong 2009) provides an alternative way to analyze these data:

```
R> library("effects")
R> nutr.eff <- Effect(c("group", "race"), nutr.lm)
R> as.data.frame(nutr.eff)
```

```
       group     race       fit       se     lower     upper
1 FoodStamps    Black  6.604295 2.459631   1.71926 11.489334
2      NoAid    Black -2.633269 1.705714  -6.02096  0.754427
3 FoodStamps Hispanic -0.895705 5.489855 -11.79903 10.007621
4      NoAid Hispanic  3.533398 3.953760  -4.31911 11.385907
5 FoodStamps    White  4.763892 0.770703   3.23321  6.294575
6      NoAid    White  1.073640 1.419825  -1.74625  3.893534
```

These results differ considerably from those we showed above for the first `lsmeans` call for this model. That is because `lsmeans` weights equally while `Effect` weights proportionately, as demonstrated here:

```
R> lsmeans(nutr.lm, ~ group * race, weights = "proportional")
```

```
 group      race       lsmean       SE df lower.CL  upper.CL
 FoodStamps Black     6.60430 2.459631 92  1.71926 11.489334
 NoAid      Black    -2.63327 1.705714 92 -6.02096  0.754427
 FoodStamps Hispanic       NA       NA NA       NA        NA
 NoAid      Hispanic       NA       NA NA       NA        NA
 FoodStamps White     4.76389 0.770703 92  3.23321  6.294575
 NoAid      White     1.07364 1.419825 92 -1.74625  3.893534

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

Note that `Effects` does not do an estimability check, so its results for Hispanics are suspect.

## 6. Interfacing with multcomp

The **multcomp** package (Hothorn, Bretz, and Westfall 2008) supports a greater variety of corrections for simultaneous testing 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 we provide methods such as `as.glht` for working with `glht`. To illustrate, consider testing comparisons of consecutive means in `days.lsm` using the two packages:

```
> ( days.consec <- contrast(days.lsm, "consec", adjust = "mvt") )
```

```
 contrast estimate     SE df t.ratio p.value
 2 - 1     0.93039 2.4668 23   0.377  0.9950
 3 - 2     7.16976 2.4797 23   2.891  0.0359
 4 - 3    -4.92228 2.4901 23  -1.977  0.2277
 5 - 4     6.69951 2.4885 23   2.692  0.0557
 6 - 5    -4.04702 2.5566 23  -1.583  0.4245

Results are averaged over the levels of: store
P value adjustment: mvt method for 5 tests
```

These results provide a single-step adjustment to the family of comparisons. If we want one of the more sophisticated adjustments provided in the **multcomp** package – say, the Westfall (1997) method – just use `as.glht` to construct a `"glht"` object, and summarize it in the desired way:

```
> library(multcomp)
> summary(as.glht(days.consec), test = adjusted("Westfall"))
```

```
     Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:
           Estimate Std. Error t value Pr(>|t|)
```

```
2 - 1 == 0     0.93       2.47    0.38    0.710
3 - 2 == 0     7.17       2.48    2.89    0.036
4 - 3 == 0    -4.92       2.49   -1.98    0.166
5 - 4 == 0     6.70       2.49    2.69    0.047
6 - 5 == 0    -4.05       2.56   -1.58    0.235
(Adjusted p values reported -- Westfall method)
```

If we had specified `test = adjusted("single-step")`, we would have obtained exactly the same results as we did above with the `"mvt"` adjustment. But with the Westfall adjustment, the smallest adjusted $P$ value is the same, but the others are smaller because of the stepwise nature of the adjustment method.

If you are already working primarily with `glht`, the **lsmeans** package provides an `lsm` function that can be used as an alternative to **multcomp**'s `mcp` function, but with the flexibility and context of an `lsmeans` call:

```
> Oats.glht <- glht(Oats.lmer, lsm(~ nitro | Variety))
```

If there is a `by` variable in effect, as in this example, `glht` or `as.glht` returns a `list` of `glht` objects – one for each `by` level.

```
> names(Oats.glht)

[1] "Variety = Golden Rain" "Variety = Marvellous"  "Variety = Victory"

> confint(Oats.glht[[1]])

     Simultaneous Confidence Intervals

Fit: lmer(formula = log(yield) ~ Variety * factor(nitro) + (1 | Block/Variety),
    data = Oats)

Quantile = 2.641
95% family-wise confidence level


Linear Hypotheses:
         Estimate lwr   upr
0 == 0   4.356    4.125 4.586
0.2 == 0 4.582    4.352 4.812
0.4 == 0 4.714    4.484 4.945
0.6 == 0 4.815    4.584 5.045
```

There is a courtesy `summary` method 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.

Finally, we note that `as.glht` may be used as a back-door way to get `glht` to work with a model supported by **lsmeans**, but not **multcomp**.

# 7. Extensions for certain types of models

The **lsmeans** package includes support for a wide and ever-growing variety of fitted-model objects. As this is written, the supported types include `"aov"`, `"aovlist"`, `"betareg"`, `"carbayes"`, `"clm"`, `"clmm"`, `"coxme"`, `"coxph"`, `"gee"`, `"geeglm"`, `"geese"`, `"glm"`, `"glmerMod"`, `"glmmadmb"`, `"gls"`, `"hurdle"`, `"lm"`, `"lme"`, `"lmerMod"`, some `"mcmc"`, `"mcmcglmm"`, `"mer"`, `"merMod"`, `"mlm"`, `"multinom"`, `"nlme"`, `"polr"`, `"rms"`, `"survreg"`, and `"zeroinfl"`; and several others that extend these classes. **lsmeans** support for all these models works similarly to the examples we have presented. Support for some classes is limited.

Extended capabilities for some are available by way of optional arguments passed via `ref.grid` or `lsmeans`. Details may be found using `help("models", package = "lsmeans")`. In this section, we illustrate some examples of these extensions.

## 7.1. 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
```

Suppose that we want to compare the two varieties on each day using this model; and in turn, compare these resulting differences with one another. This is a contrast of contrasts. To start, let's obtain the daily differences:

```
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. It seems odd to have two `by` specifications, 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. This is all made possible by the fact that `lsmeans`, `contrast`, and relatives all produce the same class of object, so we can compound the results as needed to obtain all types of interaction contrasts.

## 7.2. Proportional-odds example

The `housing` data in the **MASS** package (Venables and Ripley 2002) is an example where the response variable is an ordinal variable `Sat` (satisfaction) with a three-point scale of low, medium, high. The 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). Ordinal-data models may be fitted using the `polr` function in **MASS**, or, with more modeling options, `clm` or `clmm` in the **ordinal** package. Here, we fit a second-order model, using the default `logit` link function.

```
R> library("MASS")
R> housing.plr <- polr(Sat ~ (Infl + Type + Cont)^2, data = housing,
+    weights = Freq)
```

One way to view an ordinal-response model is in terms of a latent variable $S$ having a particular type of probability distribution (in this case, logistic, as implied by the logit link) that depends linearly on the predictors. The model posits that the observed ordinal responses comprise a categorization of $S$ into classes, based on fixed thresholds that define the "low," "medium," and "high" levels. By default, `lsmeans` produces estimates of the mean of $S$ for each factor combination. (The location and scale of $S$ are not identifiable from the data, so by default it is scaled to have a marginal mean of zero and scale parameter of 1.) The command below produces the display in Figure 5 of the latent-variable predictions from this model.

```
R> lsmip(housing.plr, Infl ~ Cont | Type, ylab = "Latent mean",
+    layout = c(4, 1))
```

Some interesting interactions are in evidence, e.g., between `Type` and `Cont`, and this warrants looking at various combinations of the factors. But in the interest of compactness, the remaining illustrations ignore these interactions and focus only on the effects of `Infl` on satisfaction.

```
R> cld(lsmeans(housing.plr, ~ Infl))
```

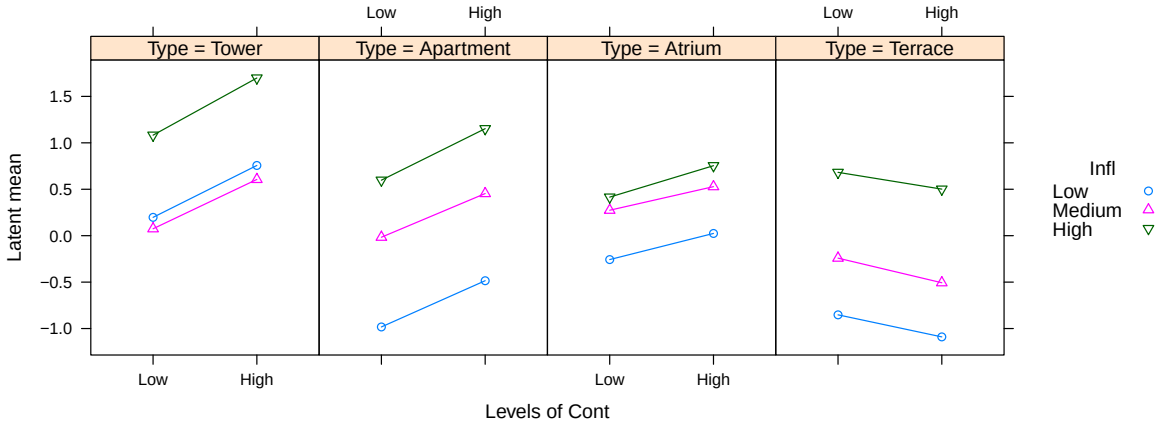


Figure 5: Display of the model for the housing data on the latent-variable scale.

```
 Infl      lsmean        SE df  asymp.LCL asymp.UCL .group
 Low    -0.336147 0.0853213 NA -0.5033735 -0.168920  1
 Medium  0.146357 0.0820731 NA -0.0145029  0.307218   2
 High    0.860442 0.1151551 NA  0.6347421  1.086142    3

Results are averaged over the levels of: Type, Cont
Confidence level used: 0.95
P value adjustment: tukey method for comparing a family of 3 estimates
significance level used: alpha = 0.05
```

From these results, it seems clear that the latent mean satisfaction increases with influence.

Support for ordinal models in **lsmeans** also allows for an optional `mode` argument depending on what you want to summarize. The default, as demonstrated above, is `mode = "latent"`. Using `mode = "linear.predictor"` produces results comparable to those for logistic regression, but for each boundary between ordinal levels. Accordingly, an extra factor named `"cut"` is created to identify which boundary is being referenced. In the code below, we summarize the linear-predictor means and pairwise differences thereof, transformed back to the response scale.

```
R> housing.lsm <- lsmeans(housing.plr, ~ Infl | cut, mode = "lin")
R> summary(housing.lsm, type = "response")

cut = Low|Medium:
 Infl     cumprob        SE df asymp.LCL asymp.UCL
 Low    0.433765 0.0215480 NA  0.392109  0.476379
 Medium 0.321037 0.0189094 NA  0.285154  0.359167
 High   0.187995 0.0184320 NA  0.154498  0.226805

cut = Medium|High:
 Infl     cumprob        SE df asymp.LCL asymp.UCL
```

```
 Low    0.718857 0.0183059 NA  0.681629  0.753307
 Medium 0.612135 0.0202689 NA  0.571745  0.651040
 High   0.435907 0.0283404 NA  0.381382  0.492027

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

```
R> summary(pairs(housing.lsm), type = "response")[1:3, ]
```

```
cut = Low|Medium:
 contrast      odds.ratio       SE df z.ratio p.value
 Low - Medium     1.62013 0.190065 NA 4.11289  0.0001
 Low - High       3.30881 0.472446 NA 8.38040  <.0001
 Medium - High    2.04232 0.286476 NA 5.09078  <.0001

Results are averaged over the levels of: Type, Cont
P value adjustment: tukey method for comparing a family of 3 estimates
Tests are performed on the linear-predictor scale
```

With `type = "response"`, the logits are transformed to cumulative probabilities. Noting that a low cumulative probability at the low cut means a low number of people are *dis*satisfied, we again find that those having more influence tend to be more satisfied. Note also that the pairwise comparisons transform to odds ratios on the `"response"` scale. Only the first three rows of the comparisons table are shown because the comparisons for `cut = Medium|High` will be identical.

Other modes available for ordinal-response models include `"cum.prob"`, `"prob"`, and `"mean.class"`. Results from `"cum.prob"` mode will be similar to those above for the response scale – but not identical because the averaging and statistical tests are performed *after* applying the reverse-logit transformation. Also, the results of `pairs` will be differences rather than odds ratios.

In `"prob"` mode, the grid will include the levels of the response variable itself, and estimation is done of the probabilities of each ordinal level:

```
R> lsmeans(housing.plr, ~ Sat | Infl, mode = "prob")
```

```
Infl = Low:
 Sat        prob        SE df asymp.LCL asymp.UCL
 Low    0.440397 0.0202378 NA  0.400732  0.480063
 Medium 0.264374 0.0119024 NA  0.241046  0.287702
 High   0.295229 0.0175813 NA  0.260770  0.329687

Infl = Medium:
 Sat        prob        SE df asymp.LCL asymp.UCL
 Low    0.325891 0.0187177 NA  0.289205  0.362577
 Medium 0.282483 0.0118429 NA  0.259271  0.305694
 High   0.391627 0.0196727 NA  0.353069  0.430185
```

```
Infl = High:
 Sat        prob        SE df asymp.LCL asymp.UCL
 Low    0.195277 0.0193966 NA  0.157260  0.233294
 Medium 0.244024 0.0130633 NA  0.218420  0.269627
 High   0.560699 0.0275617 NA  0.506679  0.614719

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

The `"mean.class"` mode outputs the means of each of the above probability distributions based on ordinal values of $1, 2, 3$:

```
R> lsmeans(housing.plr, ~ Infl, mode = "mean.class")

 Infl   mean.class        SE df asymp.LCL asymp.UCL
 Low       1.85483 0.0359955 NA   1.78428   1.92538
 Medium    2.06574 0.0365306 NA   1.99414   2.13733
 High      2.36542 0.0458378 NA   2.27558   2.45526

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

These results are comparable to those for `mode = "latent"` (not shown). Both suggest satisfaction scores below center, slightly above center, and considerably above center, respectively.

### 7.3. MCMC samplers

The **lsmeans** package provides special support for Bayesian models that are fitted via Markov chain Monte Carlo (MCMC) methods. A slot exists in the fitted object which holds the sample from the posterior distribution of the fixed-effects coefficients. Since LS means depend on these regression coefficients, we can calculate a posterior sample of LS means (or contrasts thereof) simply by performing the needed calculations with each posterior realization of the coefficients. The `as.mcmc` function returns this posterior sample with the R class `"mcmc"` (defined in the **coda** package), which then may be examined using any of the methods available for that class.

To illustrate, consider once again the `Oats` example, where we instead fit a Bayesian mixed model via the **MCMCglmm** package (Hadfield 2010):

```
R> library("MCMCglmm")
R> Oats.mc <- MCMCglmm(log(yield) ~ Variety + factor(nitro),
+    random = ~ Block + Block:Variety, nitt = 2300, burnin = 300,
+    verbose = FALSE, data = Oats)
R> Oats.mclsm <- lsmeans(Oats.mc, "nitro", data = Oats)
```

Now we may, for exsmple, visualize the posterior densities of the `nitro` LS means as we would for any other `mcmc` object:

```
R> plot(as.mcmc(Oats.mclsm))
```

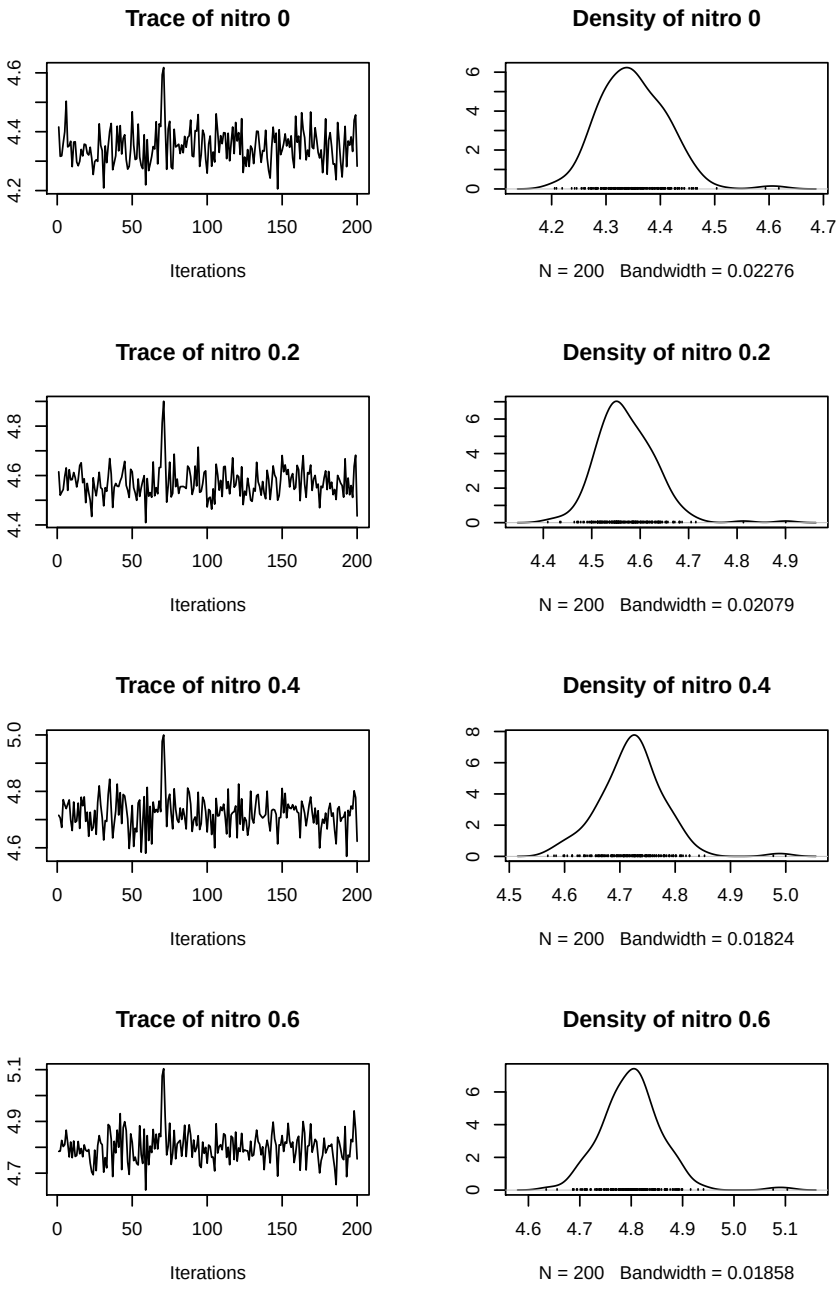


Figure 6: Display of MCMC results for `Oats.mclsm`.

The plot is shown in Figure 6.

The ordinary summary of such results is based on frequentist estimates of the mean and covariance matrix of the sample. For example, here is a summary of consecutive differences of LS means:

```
R> contrast(Oats.mclsm, "consec")
```

```
 contrast   estimate        SE df z.ratio p.value
 0.2 - 0   0.2207535 0.0477878 NA 4.61945  <.0001
 0.4 - 0.2 0.1502174 0.0467240 NA 3.21499  0.0039
 0.6 - 0.4 0.0795495 0.0399272 NA 1.99236  0.1214

Results are averaged over the levels of: Variety
Results are given on the log scale.
P value adjustment: mvt method for 3 tests
```

# 8. Briefly mentioned

## 8.1. Other capabilities

**lsmeans** has several other functions and capabilities not shown in detail in this article, and briefly described here.

The `lsm.options` function may be used to set such things as default confidence and significance levels, disable using **pbkrtest** for obtaining degrees of freedom, and default display options for objects produced by `lsmeans` and `contrast`.

In the `summary` or `test` method, one may provide `null` and `side` specifications to specify null hypotheses or obtain one-sided tests; or `delta` to set a threshold value for a test of equivalence (if two-sided) or noninferiority/nonsuperiority (if one-sided). Also, the `test` method has an optional `joint` argument that, when `TRUE`, will output joint $F$ statistics (or $\chi^2$ statistics when degrees of freedom are unavailable) for the collection of linear functions.

In constructing a reference grid, the user may specify a formula in `cov.reduce`. For example, suppose that `conc` is a covariate and `treat` is a factor, but you believe that `conc` is affected by `treat`. Then ordinary adjusted means are misleading because it is unreasonable to hold `conc` constant while varyig treatment. Instead, one may specify `cov.reduce = conc ~ treat`, which causes a linear model to be fitted to this formula within the original dataset, then to use it to predict `conc` at each point in the reference grid. An example is given in the package vignette `"using-lsmeans"`.

A function `lsmobj` is provided that will construct an `"lsmobj"` object given the factor levels, linear functions, coefficients, covariance matrix, and degrees of freedom. This can be handy for accessing **lsmeans**'s capabilities for a model not supported but these statistics are available, or even using summary statistics in a textbook or article.

There are a few other **lsmeans** features too new to be included in any depth in this article:

- An `rbind` method for `"ref.grid"` objects that makes it easy to combine two or more (conforming) objects; for example,

  ```
  R> rbind(pairs(lsmeans(Oats.lmer, "Variety")),
  +    pairs(lsmeans(Oats.lmer, "nitro")))
  ```

  combines all $3 + 6 = 9$ main-effects comparisons into one family. By default, the summary will apply the `"mvt"` multiplicity adjustment to the family of tests.

  There is also a new `[.ref.grid` method for selecting a subset of the estimates.

- A configurable limit whereby the degrees of freedom and adjusted covariances from **pbkrtest** are bypassed when the dataset is too large, in order to avoid undue computation time.
- For those who prefer the term "predicted marginal means," wrappers such as `pmmeans`, `pmtrends`, and `pmmip` are provided. They call the corresponding `ls...` functions and relabel any results that start with "ls."
- The code for checking for estimability has been moved to a separate **estimability** package (Lenth 2015), which is available from CRAN. This was done to make it easy for other package developers to incorporate estimability checks in their code, e.g., `predict` methods for new data.
- The `contrast` function has a new `interaction` option for constructing interaction contrasts.

## 8.2. 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 functions `nonest.basis` and `is.estble` (recently moved to a separate **estimability** package) are helpful in providing for estimability checking, and `regrid` may be used when it is necessary to reparameterize a reference grid. 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.

## 8.3. Comparisons with other software

A unique feature 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. The use of a reference grid has the effect of slightly extending the concepts as discussed in Searle, Speed, and Milliken (1980), in that covariates can have more than one value in the grid, and predictions may then be averaged over those different covariate values. Often, this would be inappropriate; but it does make sense in our Oats example where we were comparing LS means for the model with `nitro` as a factor with those when `nitro` is a covariate.

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.

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**. We have already mentioned **effects** in Section 5.4. The emphasis of **effects** is on graphical rather than tabular displays. For a model with continuous predictors, **effects** automatically generates a grid of values of those predictors for use in its graphs, whereas one would have to use an `at` specification to do the same in **lsmeans**. Thus, **effects** 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, Halekoh, Robison-Cox, Wright, and Leidi 2014) 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. The **lmerTest** package (Kuznetsova, Brockhoff, and Christensen 2015) offers a competing `lsmeans` function that claims to be based on **doBy**'s `popMatrix` function, but its results don't seem to be consistent with it (as this is written). There is the possibility that one of these `lsmeans` functions will mask the other, so users must be careful if they use both packages.

## References


Bates D, Mächler M, Bolker B, Walker S (2015). "Fitting Linear Mixed-Effects Models Using **lme4**." *Journal of Statistical Software*, **67**(1), 1–48. doi:10.18637/jss.v067.i01.

Fox J (2003). "Effect Displays in R for Generalised Linear Models." *Journal of Statistical Software*, **8**(15), 1–27. doi:10.18637/jss.v008.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. doi:10.18637/jss.v032.i01.

Fox J, Weisberg S (2011). *An R Companion to Applied Regression*. 2nd edition. Sage, Thousand Oaks.

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 (2015). ***multcompView**: Visualizations of Paired Comparisons*. R package version 0.1-7, URL http://CRAN.R-project.org/package=multcompView.

Hadfield JD (2010). "MCMC Methods for Multi-Response Generalized Linear Mixed Models: The **MCMCglmm** R Package." *Journal of Statistical Software*, **33**(2), 1–22. doi:10.18637/jss.v033.i02.

Halekoh U, Højsgaard S (2014). "A Kenward-Roger Approximation and Parametric Bootstrap Methods for Tests in Linear Mixed Models – The R Package **pbkrtest**." *Journal of Statistical Software*, **59**(9), 1–30. doi:10.18637/jss.v059.i09.

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 (2014). ***doBy***: *Groupwise Summary Statistics, LSmeans, General Linear Contrasts, Various Utilities*. R package version 4.5-13, URL http://CRAN.R-project.org/package=doBy.

Hothorn T, Bretz F, Westfall P (2008). "Simultaneous Inference in General Parametric Models." *Biometrical Journal*, **50**(3), 346–363. doi:10.1002/bimj.200810425.

Kuznetsova A, Brockhoff PB, Christensen RHB (2015). ***lmerTest***: *Tests in Linear Mixed Effects Models*. R package version 2.0-29, URL http://CRAN.R-project.org/package=lmerTest.

Lenth RV (2015). ***estimability***: *Tools for Assessing Estimability of Linear Predictions*. R package version 1.1-1, URL http://CRAN.R-project.org/package=estimability.

Lenth RV (2016). ***lsmeans***: *Least-Squares Means*. R package version 2.22, URL https://CRAN.R-project.org/package=lsmeans.

Milliken GA, Johnson DE (1992). *Analysis of Messy Data – Volume I: Designed Experiments*. Chapman & Hall/CRC. ISBN 0-412-99081-4. doi:10.1201/ebk1584883340.

Oehlert G (2000). *A First Course in Design and Analysis of Experiments*. W. H. Freeman. Now a free download (Creative Commons license) at http://users.stat.umn.edu/~gary/Book.html.

Pinheiro J, Bates D, R Core Team (2015). ***nlme***: *Linear and Nonlinear Mixed Effects Models*. R package version 3.1-121, URL http://CRAN.R-project.org/package=nlme.

R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Sarkar D (2008). ***lattice***: *Multivariate Data Visualization with R*. Springer-Verlag, New York.

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

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. doi:10.1080/00031305.1980.10483031.

Venables WN, Ripley BD (2002). *Modern Applied Statistics with S*. 4th edition. Springer-Verlag, New York.

Yates F (1935). "Complex Experiments." *Supplement to the Journal of the Royal Statistical Society*, **2**(2), 181–247. doi:10.2307/2983638.

**Affiliation:**

Russell V. Lenth
Department of Statistics and Actuarial Science
241 Schaeffer Hall
The University of Iowa
Iowa City, IA 52242, United States of America
E-mail: russell-lenth@uiowa.edu
URL: http://www.stat.uiowa.edu/~rlenth/

*Journal of Statistical Software* http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
January 2016, Volume 69, Issue 1 *Submitted:* 2013-12-23
doi:10.18637/jss.v069.i01 *Accepted:* 2015-02-04