

# Changes in `lsmeans`, Version 2.00

Russell V. Lenth

March 2, 2014

## 1 Introduction

Versions of `lsmeans` up through version 1.10 were based on a lot of “spaghetti code” that worked but was increasingly difficult to maintain. So starting with version 2.00, the package underwent a complete overhaul where the code is much more modular and extensible. These changes help make the package better prepared for future use.

Past users of `lsmeans` may use it in much the same ways as in the old version, but not entirely. And of course, that’s the catch—especially when it comes to doing something later with an object created by the `lsmeans` function. The purpose of this document is to explain the changes that are being made before the package is released, so that users may be prepared for it.

### 1.1 Availability of old functionality

For a while, the `lsmeans` package will include a function `.old.lsmeans` which is the old version of `lsmeans` from version 1.10-4. Users should adapt to the new `lsmeans` function as quickly as possible. However, in a clutch, this old one may be used. We use it several times in this document to illustrate the differences.

## 2 Changes that could break existing code that uses `lsmeans`

### 2.1 In a nutshell

If you have existing code that extracts or manipulates the result of `lsmeans` in its old manifestation (i.e. treats it as a list of `data.frames`); or uses the arguments `cov.reduce`, `fac.reduce`, `conf`, `glhargs`, `lf`, or `mlf`, your code will break as-is. Details follow.

### 2.2 Returned objects

Probably the most problematic change for past users is that `lsmeans` used to return a list of `data.frames` (except sometimes a `glht` object was thrown in). But now it returns a single object of a new class `lsmobj`, or a list thereof.

```
> library(lsmeans)
> ### OLD
> warp.lm <- lm(breaks ~ wool * tension, data = warpbreaks)
> warp.oldlsm <- .old.lsmeans(warp.lm, ~ tension | wool)
> class(warp.oldlsm)

[1] "lsm" "list"

> class(warp.oldlsm[[1]])

[1] "data.frame.lsm" "data.frame"
```

```
> ### NEW
> warp.lsmobj <- lsmeans(warp.lm, ~ tension | wool)
> class(warp.lsmobj)
```

```
[1] "lsmobj"
```

Look at the results obtained the new way:

```
> warp.lsmobj

wool = A:
  tension    lsmean      SE df lower.CL upper.CL
L         44.55556  3.646761  48  37.22325  51.88786
M         24.00000  3.646761  48  16.66769  31.33231
H         24.55556  3.646761  48  17.22325  31.88786

wool = B:
  tension    lsmean      SE df lower.CL upper.CL
L         28.22222  3.646761  48  20.88992  35.55453
M         28.77778  3.646761  48  21.44547  36.11008
H         18.77778  3.646761  48  11.44547  26.11008
```

Confidence level used: 0.95

Unlike the old display (not shown), this one pays attention to the | wool part of the specification.

In the old version, users could access/manipulate the results by taking advantage of the fact that they inherited from `data.frame`:

```
> ### OLD
> warp.oldlsm[[1]]$lsmean

[1] 44.55556 24.00000 24.55556 28.22222 28.77778 18.77778

> Try <- function(expr) tryCatch(expr, error = function(e) cat("Oops!\n"))
> ### NEW
> Try(warp.lsmobj$lsmean)
```

Oops!

The `show` method for an `lsmobj` is `summary`, which indeed does produce an object that inherits from `data.frame`. So if you need to access values that you see, call `summary` first:

```
> ### NEW
> summary(warp.lsmobj)$lsmean

[1] 44.55556 24.00000 24.55556 28.22222 28.77778 18.77778
```

In casting to `data.frame`, note that the “by” variable (wool in this case) is included:

```
> as.data.frame(summary(warp.lsmobj))

  tension wool    lsmean      SE df lower.CL upper.CL
1      L   A 44.55556  3.646761  48  37.22325  51.88786
2      M   A 24.00000  3.646761  48  16.66769  31.33231
3      H   A 24.55556  3.646761  48  17.22325  31.88786
4      L   B 28.22222  3.646761  48  20.88992  35.55453
5      M   B 28.77778  3.646761  48  21.44547  36.11008
6      H   B 18.77778  3.646761  48  11.44547  26.11008
```

If there is also a contrast specification, then `lsmeans` does return a list (and not an extension thereof). But each element is of class `lsmobj`, not `data.frame`.

```
> ### NEW
> warp.l2 <- lsmeans(warp.lm, pairwise ~ tension)
> class(warp.l2)

[1] "lsm.list" "list"

> sapply(warp.l2, class)

lsmeans contrasts
"lsmobj" "lsmobj"
```

## 2.3 Changes to `cov.reduce` and `fac.reduce`

The `cov.reduce` and `fac.reduce` arguments to `lsmeans` required a second argument giving the name of the variable. This is awkward and, in the case of `fac.reduce`, doesn't even make sense if you think about it. But if you have existing code that uses these functions, you will have to change it.

In the new version, `cov.reduce` may be a function or a named list of functions of a single numeric variable. The default is `mean`. If it is a named list, then a covariate matching a name on the list is reduced using that function, and any mismatched covariates are reduced using `mean`. As before, `cov.reduce` may also be logical: `TRUE` is equivalent to `mean`, and `FALSE` is equivalent to `function(x) sort(unique(x))`.

`fac.reduce` must now be a function of one matrix argument. Its default is `function(X) apply(X, 2, mean)`. To override it (at least sensibly), you must provide a function that reduces the rows of the matrix into a single vector of the same length.

## 2.4 Arguments no longer provided

The `lsmeans` arguments `conf`, `glhargs`, `lf`, and `mlf` are no longer supported. The needs they serve are supported via `lsmobj` methods or slots.

Continuing with the `warp.lm` example and the returned object `warp.lsmobj`, the `conf` functionality is replaced by the `confint` method:

```
> confint(warp.lsmobj, level = .90)

wool = A:
  tension  lsmean      SE df lower.CL upper.CL
L      44.55556 3.646761 48 38.43912 50.67199
M      24.00000 3.646761 48 17.88356 30.11644
H      24.55556 3.646761 48 18.43912 30.67199

wool = B:
  tension  lsmean      SE df lower.CL upper.CL
L      28.22222 3.646761 48 22.10579 34.33866
M      28.77778 3.646761 48 22.66134 34.89421
H      18.77778 3.646761 48 12.66134 24.89421
```

Confidence level used: 0.9

The `glhargs` capability is replaced by an `as.glht` method to create a `glht` for use with the `multcomp` package:

```
> library(multcomp)
> summary(as.glht(warp.lsmobj))
```

```
$`wool = A`
```

#### Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
L == 0	44.556	3.647	12.218	< 1e-07 ***
M == 0	24.000	3.647	6.581	1.07e-07 ***
H == 0	24.556	3.647	6.734	< 1e-07 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
(Adjusted p values reported -- single-step method)

```
$`wool = B`
```

#### Simultaneous Tests for General Linear Hypotheses

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t )
L == 0	28.222	3.647	7.739	< 1e-05 ***
M == 0	28.778	3.647	7.891	< 1e-05 ***
H == 0	18.778	3.647	5.149	1.38e-05 ***

---

Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
(Adjusted p values reported -- single-step method)

In lieu of `lf`, simply access the `linfct` slot:

```
> warp.lsmobj@linfct
```

	(Intercept)	woolB	tensionM	tensionH	woolB:tensionM	woolB:tensionH
[1,]	1	0	0	0	0	0
[2,]	1	0	1	0	0	0
[3,]	1	0	0	1	0	0
[4,]	1	1	0	0	0	0
[5,]	1	1	1	0	1	0
[6,]	1	1	0	1	0	1

The `mlm` argument was new and gave only rudimentary support for multivariate responses. Now multivariate predictors cause `lsmeans` to create one or more additional factors that can be specified in the `lsmeans` specs. More on this later.

## 3 Corrections

A few bugs turned up in the course of discovering that new results did not match old ones—and the new ones were right! Of course, there could well be undiscovered new bugs.

### 3.1 Degrees of freedom

`lsmeans` uses the `pbkrtest` package to obtain degrees of freedom for models fitted using the `lme4` package. These depend on both the adjusted and unadjusted covariance matrices, but it turns out that the old `lsmeans` supplied the adjusted one for both. This does not always make a difference:

```
> library(lme4)
> data(Oats, package = "nlme")
> Oats.lmer <- lmer(yield ~ factor(nitro) + Variety + (1|Block/Variety),
+   data = Oats, subset = -c(1,2,3,5,8,13,21,34,55))
> ### OLD
> .old.lsmeans(Oats.lmer, pairwise ~ Variety)
```

```
$`Variety lsmeans`
  Variety    lsmean      SE      df lower.CL upper.CL
Golden Rain 105.24082 7.531769 8.458094 88.03491 122.4467
Marvellous 108.46952 7.482684 8.277279 91.31451 125.6245
Victory 96.93448 7.641696 8.793712 79.58575 114.2832
```

```
$`Variety pairwise differences`
              estimate      SE      df t.ratio p.value
Golden Rain - Marvellous -3.228695 6.553864 9.509345 -0.49264 0.87645
Golden Rain - Victory    8.306336 6.707951 9.617594  1.23828 0.46005
Marvellous - Victory    11.535031 6.670503 9.637894  1.72926 0.24384
  p values are adjusted using the tukey method for 3 means
```

```
> ### NEW
> lsmeans(Oats.lmer, pairwise ~ Variety)
```

```
$lsmeans
  Variety    lsmean      SE  df lower.CL upper.CL
Golden Rain 105.24082 7.531769 8.46 88.03692 122.4447
Marvellous 108.46952 7.482684 8.28 91.31558 125.6234
Victory 96.93448 7.641696 8.81 79.59000 114.2790
```

Confidence level used: 0.95

```
$contrasts
  contrast              estimate      SE  df t.ratio p.value
Golden Rain - Marvellous -3.228695 6.553864 9.56 -0.493 0.8764
Golden Rain - Victory    8.306336 6.707951 9.80  1.238 0.4595
Marvellous - Victory    11.535031 6.670503 9.80  1.729 0.2431
```

P value adjustment: tukey method for a family of 3 means

The discrepancies are not huge, but they are there. Without the `subset` that created unbalanced data, the results essentially agree.

## 3.2 Processing at

In models containing `factor` or `ordered` (like `Oats.lmer`), any `at` specification was ignored. The new version handles this correctly, including omitting inappropriate levels.

```
> ### OLD
> .old.lsmeans(Oats.lmer, ~ nitro, at = list(nitro = c(.1,.2,.3)))
```

```
$`nitro lsmeans`
  nitro    lsmean      SE      df lower.CL upper.CL
0.0 78.89208 7.294425 7.775233 61.98609 95.79808
0.2 97.03426 7.136318 7.182087 80.24589 113.82263
0.4 114.19817 7.136234 7.183545 97.41067 130.98567
0.6 124.06857 7.070283 6.953104 107.32712 140.81002
```

```
> ### NEW
> lsmeans(Oats.lmer, ~ nitro, at = list(nitro = c(.1,.2,.3)))

nitro    lsmean      SE    df lower.CL upper.CL
  0.2  97.03426  7.136318  7.19  80.25017 113.8184

Confidence level used: 0.95
```

## 4 New object structure

The more recent vignettes for `lsmeans` have explained least-squares means as predictions on a “reference grid,” or marginal averages thereof. By default, the reference grid consists of all combinations of factor levels, along with the averages of numeric predictors. But this can be changed by `at` or `cov.reduce`. The new design of `lsmeans` uses a reference-grid object explicitly. For example:

```
> (Oats.rg <- ref.grid(Oats.lmer))

'ref.grid' object with variables:
  nitro = 0.0, 0.2, 0.4, 0.6
  Variety = Golden Rain, Marvellous, Victory

> Oats.quad <- update(Oats.lmer, yield ~ Variety + poly(nitro,2) + (1/Block/Variety))
> ref.grid(Oats.quad)

'ref.grid' object with variables:
  Variety = Golden Rain, Marvellous, Victory
  nitro = 0.31429

> ref.grid(Oats.quad, at = list(nitro = c(.1,.2,.3)))

'ref.grid' object with variables:
  Variety = Golden Rain, Marvellous, Victory
  nitro = 0.1, 0.2, 0.3
```

The `ref.grid` function calls two other functions, `recover.data` (to reproduce the dataset) and `lsm.basis` (to get the model matrix, coefficients, etc.), each of which has S3 methods for popular model objects like `lm`, `mlm`, `gls`, `lmer`, etc. This allows `ref.grid`’s capabilities to be easily extended to other model objects not yet supported. `ref.grid` serves as a constructor for an S4 object of class `ref.grid`, which encapsulates all the information needed to compute—and make inferences on—least-squares means, independently of the model object itself.

The `lsmeans` function now consists of S4 methods for a variety of signatures, one of which corresponds to the old version where `object` is a model object and `specs` is a formula. So, for example, we may call `lsmeans` with an existing `ref.grid`, and provide specifications in place of the old formula interface:

```
> (Oats.lsm <- lsmeans(Oats.rg, "nitro", by = "Variety"))

Variety = Golden Rain:
nitro    lsmean      SE    df lower.CL upper.CL
  0.0  80.58463  8.194841 11.70  62.67900  98.49026
  0.2  98.72681  8.020181 10.87  81.04772 116.40590
  0.4 115.89072  8.098937 11.27  98.11661 133.66483
  0.6 125.76112  8.099519 11.21 107.97560 143.54664

Variety = Marvellous:
nitro    lsmean      SE    df lower.CL upper.CL
  0.0  83.81333  8.152141 11.54  65.97316 101.65349
```

0.2	101.95551	8.083595	11.19	84.20096	119.71005
0.4	119.11941	8.005137	10.80	101.45952	136.77931
0.6	128.98982	7.990096	10.71	111.34490	146.63473

Variety = Victory:

nitro	lsmean	SE	df	lower.CL	upper.CL
0.0	72.27830	8.376248	12.49	54.10738	90.44922
0.2	90.42048	8.201327	11.58	72.47869	108.36226
0.4	107.58438	8.200668	11.57	89.64360	125.52517
0.6	117.45478	8.041848	10.91	99.73618	135.17339

Confidence level used: 0.95

Moreover, `lsmobj` is in fact an extension of `ref.grid`, and we can use it as such:

```
> str(Oats.lsm)
```

'ref.grid' object with variables:

```
  nitro = 0.0, 0.2, 0.4, 0.6
  Variety = Golden Rain, Marvellous, Victory
```

```
> (Oats.n <- lsmeans(Oats.lsm, "nitro"))
```

nitro	lsmean	SE	df	lower.CL	upper.CL
0.0	78.89208	7.294425	7.78	61.98918	95.79499
0.2	97.03426	7.136318	7.19	80.25017	113.81836
0.4	114.19817	7.136234	7.19	97.41441	130.98193
0.6	124.06857	7.070283	6.95	107.32782	140.80933

Confidence level used: 0.95

## 4.1 Slots

The classes `ref.grid` and `lsmobj` are essentially identical in structure, with `lsmobj` being a minor extension with the same slots.

```
> slotNames(Oats.lsm)
```

```
[1] "model.info" "roles"      "grid"       "levels"     "matlevs"
[6] "linfct"     "bhat"       "nbasis"     "V"          "dffun"
[11] "dfargs"     "misc"
```

`model.info` has the call and terms. `roles` lists the names of predictors and responses. `grid` is a `data.frame` consisting of all combinations of the variables in the list `levels`. The rows of `grid` go in one-to-one correspondence with those of `linfct`, which contains the linear coefficients associated with each LS mean (or reference-grid combination). `matlevs` has summary information for any matrices in the dataset. `bhat` holds the regression coefficients. `nbasis` holds information for determining non-estimability in rank-deficient situations. `V` is the covariance matrix for `bhat`. `ddfm` is a function to return the degrees of freedom for a linear function of `bhat`. It is passed the contents of the list `misc`, thus allowing for additional parameters. `misc` also is used for bookkeeping tasks such as remembering `by` variables, labels, `adjust` settings, etc.

## 5 New functions and methods

There are numerous methods for `lsmobj` objects. The `summary` method produces what you see in a listing, and is an extension of `data.frame` but it is printed with different formatting and with added messages about adjustments, confidence levels, etc. You can also display the results differently. For example:

```
> summary(Oats.lsm, by = "nitro")
```

(results not shown) will group the `Variety` means for each `nitro` rather than the way it is displayed above. There is also an `infer` argument for flagging whether confidence intervals and/or tests are displayed:

```
> summary(Oats.n, infer = c(TRUE, TRUE))
```

nitro	lsmean	SE	df	lower.CL	upper.CL	t.ratio	p.value
0.0	78.89208	7.294425	7.78	61.98918	95.79499	10.815	<.0001
0.2	97.03426	7.136318	7.19	80.25017	113.81836	13.597	<.0001
0.4	114.19817	7.136234	7.19	97.41441	130.98193	16.003	<.0001
0.6	124.06857	7.070283	6.95	107.32782	140.80933	17.548	<.0001

Confidence level used: 0.95

Most other methods are S3 ones, as those are suitable to our needs and often extend existing S3 methods. The `as.glht` method is illustrated earlier in this document. The `confint` and `test` methods are really courtesy methods for `summary` with argument `infer` set to `c(TRUE, FALSE)` and `c(FALSE, TRUE)` respectively.

An important method is `contrast`:

```
> (warp.con <- contrast(warp.lsmobj, method = "poly"))
```

```
wool = A:
contrast      estimate      SE df t.ratio p.value
linear      -20.000000  5.157299 48  -3.878  0.0003
quadratic    21.111111  8.932705 48   2.363  0.0222
```

```
wool = B:
contrast      estimate      SE df t.ratio p.value
linear      -9.444444  5.157299 48  -1.831  0.0733
quadratic   -10.555556  8.932705 48  -1.182  0.2432
```

These methods all return new objects of class `lsmobj`. Hence they may be further analyzed or reanalyzed. For example, suppose we now want to compare the two linear and the two quadratic contrasts in the above:

```
> contrast(warp.con, "revpairwise", by = "contrast")
```

```
contrast = linear:
contrast1      estimate      SE df t.ratio p.value
B - A         10.55556   7.293523 48   1.447  0.1543
```

```
contrast = quadratic:
contrast1      estimate      SE df t.ratio p.value
B - A        -31.66667  12.632752 48  -2.507  0.0156
```

The `pairs` method is equivalent to `contrast` with `method = "pairwise"`. Closely related is the new `cld` method which produces a compact letter display for which pairwise comparisons are nonsignificant:

```
> cld(Oats.n, sort = FALSE)
```

nitro	lsmean	SE	df	lower.CL	upper.CL	.group
0.0	78.89208	7.294425	7.78	61.98918	95.79499	1
0.2	97.03426	7.136318	7.19	80.25017	113.81836	2
0.4	114.19817	7.136234	7.19	97.41441	130.98193	3
0.6	124.06857	7.070283	6.95	107.32782	140.80933	3

Confidence level used: 0.95

P value adjustment: tukey method for a family of 4 means  
significance level used: alpha = 0.05



Finally, there is the `lstrends` function for estimating fitted trends of a covariate that interacts with a factor. Like the other methods, it returns an `lsmobj` object, subject to further analysis. To illustrate, the R-provided dataset `ChickWeight` has data on growth of chicks given different diets. We will fit a random-slopes model and compare the mean slope for each diet. In addition, we'll chose symbols for the display that mimic the grouping lines that some people use.

```
> chick.lmer <- lmer(weight ~ Time * Diet + (0 + Time | Chick), data = ChickWeight)
> chick.lst <- lstrends(chick.lmer, ~ Diet, var = "Time")
> cld(chick.lst, Letters = "||||")
```

Diet	Time.trend	SE	df	lower.CL	upper.CL	.group
1	6.338552	0.6105022	49.85	5.112234	7.564871	
2	8.609136	0.8380228	48.28	6.924433	10.293840	
4	9.555825	0.8392650	48.56	7.868877	11.242774	
3	11.422871	0.8380228	48.28	9.738167	13.107575	

Confidence level used: 0.95

P value adjustment: tukey method for a family of 4 means

significance level used: alpha = 0.05

Chicks fed with Diet 3 seem to grow faster than chicks with the other diets, and Diet 1 is the worst.

`lstrends` uses a difference quotient to do its work, and there is an optional argument `delta` that can be used to change its increment. It requires a model object—there is no `ref.grid` method for it. The `var` argument may imply a function call, i.e. `var=sqrt(Time)`, in which case the chain rule is applied.

## 6 Support for multivariate models

`lsmeans` now provides for models with multivariate responses, by way of defining factor levels that index the responses. Thus, linear functions of the multivariate response are available for inference. As an example, consider the package-provided dataset `MOats`, which is the same as `Oats` except that each observation is a whole plot with the yields for the four `nitro` levels as responses.

```
> head(MOats)
```

	Variety	Block	yield.1	yield.2	yield.3	yield.4
1	Victory	I	111	130	157	174
2	Golden Rain	I	117	114	161	141
3	Marvellous	I	105	140	118	156
4	Victory	II	61	91	97	100
5	Golden Rain	II	70	108	126	149
6	Marvellous	II	96	124	121	144

Let's fit a model and obtain the reference grid:

```
> MOats.mlm <- lm(yield ~ Block + Variety, data = MOats)
> (MOats.rg <- ref.grid(MOats.mlm, mult.levs = list(nitro = c(0,.2,.4,.6))))
```

```
'ref.grid' object with variables:
  Block = VI, V, III, IV, II, I
  Variety = Golden Rain, Marvellous, Victory
  nitro = multivariate response levels: 0.0, 0.2, 0.4, 0.6
```

(The `mult.levs` argument gives a name and levels for later use; if it had been absent, the multivariate response would have been named `rep.meas`, with levels 1,2,3,4.)

We may now use `nitro` just like we would in the univariate case:

```
> lsmeans(MOats.rg, ~ nitro)
```

nitro	lsmean	SE	df	lower.CL	upper.CL
0.0	79.38889	3.198862	10	72.26138	86.5164
0.2	98.88889	3.811694	10	90.39591	107.3819
0.4	114.22222	5.020268	10	103.03637	125.4081
0.6	123.38889	4.216517	10	113.99390	132.7839

Confidence level used: 0.95

```
> lsmeans(MOats.rg, ~ Variety)
```

Variety	lsmean	SE	df	lower.CL	upper.CL
Golden Rain	104.5000	5.005541	10	93.34696	115.6530
Marvellous	109.7917	5.005541	10	98.63863	120.9447
Victory	97.6250	5.005541	10	86.47196	108.7780

Confidence level used: 0.95

We can verify that the latter is exactly the same as if we had averaged the responses:

```
> MOats <- transform(MOats, avg.yield = apply(yield, 1, mean))
> lsmeans(lm(avg.yield ~ Block + Variety, data = MOats), ~ Variety)
```

Variety	lsmean	SE	df	lower.CL	upper.CL
Golden Rain	104.5000	5.005541	10	93.34696	115.6530
Marvellous	109.7917	5.005541	10	98.63863	120.9447
Victory	97.6250	5.005541	10	86.47196	108.7780

Confidence level used: 0.95

## 7 Support for more models

Several more model types are supported, including `survreg`, `coxph`, `coxme`, and `polr` models. Here's an example for a Cox proportional-hazards model for the `cgd` dataset in the `survival` package:

```
> library(survival)
> cgd.ph <- coxph(Surv(tstart, tstop, status) ~ treat * inherit +
+               sex + age + cluster(id), data = cgd)
> (cgd.lsm <- lsmeans(cgd.ph, ~ treat | inherit))
```

inherit = X-linked:

treat	lsmean	SE	df	asympt.LCL	asympt.UCL
placebo	0.1247488	0.3704204	NA	-0.6012620	0.8507595
rIFN-g	-1.1074031	0.3874101	NA	-1.8667130	-0.3480933

inherit = autosomal:

treat	lsmean	SE	df	asympt.LCL	asympt.UCL
placebo	0.5882164	0.2455766	NA	0.1068951	1.0695377
rIFN-g	-0.2210687	0.3808731	NA	-0.9675661	0.5254288

Confidence level used: 0.95

## 8 Transformations

`lsmeans` tries to discover response transformations and link functions, and provides a `type` argument in `summary`, `lsmip`, and `predict` that allows inverting the transformation. For example, consider the Cox model just fitted.

```

> summary(cgd.lsm, type = "response")

inherit = X-linked:
  treat      hazard      SE df asymp.LCL asymp.UCL
placebo 1.1328638 0.4196359 NA 0.5481195 2.341424
rIFN-g 0.3304159 0.1280065 NA 0.1546311 0.706033

inherit = autosomal:
  treat      hazard      SE df asymp.LCL asymp.UCL
placebo 1.8007736 0.4422279 NA 1.1128175 2.914032
rIFN-g 0.8016616 0.3053313 NA 0.3800068 1.691184

Confidence level used: 0.95

As another example, suppose we transform the response in warp.lm:

> logwarp.rg <- ref.grid(update(warp.lm, log(breaks) ~ .))
> summary(logwarp.rg)

wool tension prediction      SE df
A    L          3.717945 0.1246647 48
B    L          3.282378 0.1246647 48
A    M          3.116750 0.1246647 48
B    M          3.309327 0.1246647 48
A    H          3.117623 0.1246647 48
B    H          2.904152 0.1246647 48

> summary(logwarp.rg, type = "response")

wool tension lsresponse      SE df
A    L          41.17969 5.133656 48
B    L          26.63906 3.320951 48
A    M          22.57289 2.814043 48
B    M          27.36669 3.411661 48
A    H          22.59260 2.816501 48
B    H          18.24975 2.275101 48

```