

Population means (LSMEANS), contrasts and estimable functions in the **doBy** package

Søren Højsgaard and Ulrich Halekoh
Department of Genetics and Biotechnology
Aarhus University

June 5, 2011

Contents

1	Introduction	1
2	Population means (LSMEANS)	1
2.1	A brute-force calculation	2
2.2	Using <code>esticon()</code>	4
3	Using <code>popMatrix()</code> and <code>popMeans()</code>	4
3.1	Using the <code>at</code> argument	6
3.2	Ambiguous specification	7
3.3	Using covariates	8
3.4	Using transformed covariates	9
4	The engine argument of <code>popMeans</code>	9

1 Introduction

This is a working document; please feel free to suggest improvements.

2 Population means (LSMEANS)

Population means (also known as LSMEANS in SAS jargon) are much used in some sciences. Consider these data:

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

	A	B	C	y	x	z
1	1	1	1	-0.86038	1.31964	-0.86682
2	2	1	1	-0.05959	0.57000	-1.01970
3	3	1	1	-0.79735	0.04337	0.04559
4	1	2	1	0.16534	0.23580	0.39837
5	2	2	1	0.61985	0.68288	-0.38010
6	3	2	1	1.80208	1.16678	1.66317
7	1	3	1	2.06347	0.01345	0.03358
8	2	3	1	0.19531	0.36981	0.63341
9	3	3	1	-0.30760	0.60901	-0.58603
10	1	1	2	-0.61780	0.12456	-0.48690

Consider the additive model:

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

(Intercept)	A2	A3	B2	B3	C2
-0.2955	0.0922	0.2399	1.0162	0.4784	-0.3210

This is a model for the conditional mean $\mathbb{E}(y|A, B, C)$. Sometimes one is interested in quantities like $\mathbb{E}(y|A)$. This quantity can not formally be found unless B and C are random variables such that we may find $\mathbb{E}(y|A)$ by integration.

However, suppose that A is a treatment of main interest, B is a blocking factor and C is a day. Then it is tempting to average $\mathbb{E}(y|A, B, C)$ over B and C (average over block and day) and think of this average as $\mathbb{E}(y|A)$.

2.1 A brute-force calculation

The population mean for $A = 1$ can be found as:

```

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

(Intercept)      A2      A3      B2      B3      C2
    -0.2955    0.0000    0.0000    0.3387    0.1595   -0.1605

sum(coef(mm)*w)

[1] 0.04223

```

Notice that although B has 3 levels we only get two terms of $1/3$ because the parameter for $B = 1$ is set to zero to obtain identifiability. Similarly for C which has 2 levels and therefore we only get one term of $1/2$.

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

```

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

      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    0    0 0.3333 0.3333 0.5
[2,]    1    1    0 0.3333 0.3333 0.5
[3,]    1    0    1 0.3333 0.3333 0.5

W %*% coef(mm)

      [,1]
[1,] 0.04223
[2,] 0.13443
[3,] 0.28217

```

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

2.2 Using `esticon()`

The `esticon()` function in the `doBy` package be used for calculating such quantities along with standard errors, confidence limits etc.

```
esticon(mm, W)
```

	beta0	Estimate	Std.Error	t.value	DF	Pr(> t)	Lower	Upper
1	0	0.04223	0.3541	0.1193	12	0.9070	-0.7292	0.8137
2	0	0.13443	0.3541	0.3797	12	0.7108	-0.6370	0.9059
3	0	0.28217	0.3541	0.7969	12	0.4410	-0.4893	1.0536

3 Using `popMatrix()` and `popMeans()`

Writing such matrices by hand is somewhat tedious. In addition, there is a potential risk of getting the wrong answer if the the reference level has been changed.

The `popMatrix()` function provides some help. The above `W` matrix is constructed by

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

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

```
summary(pma)
```

```
(Intercept) A2 A3      B2      B3 C2
[1,]          1  0  0 0.3333 0.3333 0.5
[2,]          1  1  0 0.3333 0.3333 0.5
[3,]          1  0  1 0.3333 0.3333 0.5
grid:
'data.frame':      3 obs. of  1 variable:
 $ A: chr  "1" "2" "3"
at:
NULL
```

The `popMeans()` function is simply a wrapper around first a call to `popMatrix()` followed by a call to (by default) `esticon()`:

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

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

```
summary(pme)

      beta0 Estimate Std.Error t.value DF Pr(>|t|)  Lower Upper
1      0  0.04223    0.3541  0.1193 12  0.9070 -0.7292 0.8137
2      0  0.13443    0.3541  0.3797 12  0.7108 -0.6370 0.9059
3      0  0.28217    0.3541  0.7969 12  0.4410 -0.4893 1.0536
Call:
popMeans.lm(object = mm, effect = "A")
Contrast matrix:
      (Intercept) A2 A3      B2      B3 C2
[1,]             1  0  0 0.3333 0.3333 0.5
[2,]             1  1  0 0.3333 0.3333 0.5
[3,]             1  0  1 0.3333 0.3333 0.5
grid:
'data.frame':      3 obs. of  1 variable:
 $ A: chr  "1" "2" "3"
at:
NULL
```

The `effect` argument requires to calculate the LSMEANS at *all* levels of *A* aggregating across the levels of the other variables in the data.

Likewise we may do:

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

      (Intercept) A2 A3      B2      B3 C2
[1,]             1  0  0 0.3333 0.3333 0
[2,]             1  1  0 0.3333 0.3333 0
```

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

Consequently

```
popMeans(mm)
```

```
beta0 Estimate Std.Error t.value DF Pr(>|t|) Lower Upper
1      0   0.1529   0.2044  0.7482 12  0.4688 -0.2925 0.5984
```

gives the “total average”.

3.1 Using the at argument

We may be interested in finding the population means at all levels of A but only at $C = 1$. This is obtained by using the `at` argument:

```
popMatrix(mm,effect='A', at=list(C='1'))
```

```
(Intercept) A2 A3      B2      B3 C2
[1,]          1  0  0 0.3333 0.3333  0
[2,]          1  1  0 0.3333 0.3333  0
[3,]          1  0  1 0.3333 0.3333  0
```

Notice here that average is only taken over B . Another way of creating the population means at all levels of (A, C) is therefore

```
popMatrix(mm,effect='A', at=list(C=c('1','2')))
```

	(Intercept)	A2	A3	B2	B3	C2
[1,]	1	0	0	0.3333	0.3333	0
[2,]	1	1	0	0.3333	0.3333	0
[3,]	1	0	1	0.3333	0.3333	0
[4,]	1	0	0	0.3333	0.3333	1
[5,]	1	1	0	0.3333	0.3333	1
[6,]	1	0	1	0.3333	0.3333	1

We may have several variables in the `at` argument:

```
popMatrix(mm,effect='A', at=list(C=c('1','2'), B='1'))
```

	(Intercept)	A2	A3	B2	B3	C2
[1,]	1	0	0	0	0	0
[2,]	1	1	0	0	0	0
[3,]	1	0	1	0	0	0
[4,]	1	0	0	0	0	1
[5,]	1	1	0	0	0	1
[6,]	1	0	1	0	0	1

3.2 Ambiguous specification

There is room for an ambiguous specification if a variable appears in both the `effect` and the `at` argument, such as

```
popMatrix(mm,effect=c('A','C'), at=list(C='1'))
```

	(Intercept)	A2	A3	B2	B3	C2
[1,]	1	0	0	0.3333	0.3333	0
[2,]	1	1	0	0.3333	0.3333	0
[3,]	1	0	1	0.3333	0.3333	0

This ambiguity is due to the fact that the `effect` argument asks for the LSMEANS at all levels of the variables but the `at` chooses only specific levels.

In this case of ambiguity any variable in the `at` argument is removed from the `effect` argument such as the statement above is equivalent to

```
popMatrix(mm,effect='A', at=list(C='1'))
```

3.3 Using covariates

Next consider the model where a covariate is included:

```
mm2 <- lm(y~A+B+C+C:x, data=dd)
coef(mm2)
```

(Intercept)	A2	A3	B2	B3	C2
-0.1333	0.1235	0.2994	1.0286	0.3884	-0.3712
C1:x	C2:x				
-0.2992	-0.1583				

In this case we get

```
popMatrix(mm2,effect='A', at=list(C='1'))
```

	(Intercept)	A2	A3	B2	B3	C2	C1:x	C2:x
[1,]	1	0	0	0.3333	0.3333	0	0.6459	0
[2,]	1	1	0	0.3333	0.3333	0	0.6459	0
[3,]	1	0	1	0.3333	0.3333	0	0.6459	0

Above, x has been replaced by its average and that is the general rule for models including covariates. However we may use the `at` argument to ask for calculation of the LSMEANS at some user-specified value of x , say 12:

```
popMatrix(mm2,effect='A', at=list(C='1',x=12))
```


	(Intercept)	A2	A3	B2	B3	C2	C1:x	C2:x
[1,]	1	0	0	0.3333	0.3333	0	12	0
[2,]	1	1	0	0.3333	0.3333	0	12	0
[3,]	1	0	1	0.3333	0.3333	0	12	0

3.4 Using transformed covariates

Next consider the model where a transformation of a covariate is included:

```
mm3 <- lm(y~A+B+C+C:log(x), data=dd)
coef(mm3)
```

(Intercept)	A2	A3	B2	B3	C2
-0.66721	0.19656	0.36574	1.07954	0.29755	-0.08275
C1:log(x)	C2:log(x)				
-0.27610	-0.04090				

In this case we can not use `popMatrix`. Instead we have first to generate a new variable, say `log.x`, with `log.x = log(x)`, in the data and then proceed as

```
dd <- transform(dd, log.x = log(x))
mm3 <- lm(y~A+B+C+C:log.x, data=dd)
popMatrix(mm3, effect='A', at=list(C='1'))
```

	(Intercept)	A2	A3	B2	B3	C2	C1:log.x	C2:log.x
[1,]	1	0	0	0.3333	0.3333	0	-1.778	0
[2,]	1	1	0	0.3333	0.3333	0	-1.778	0
[3,]	1	0	1	0.3333	0.3333	0	-1.778	0

4 The engine argument of popMeans

The `popMatrix` is a function to generate a linear tranformation matrix of the model parameters with emphasis on constructing such matrices for LSMEANS. `popMeans` invokes by

default the `esticon` function on this linear transformation matrix for calculating parameter estimates and confidence intervals. A similar function to `esticon` is the `glht` function of the `multcomp` package.

The `glht()` function can be chosen via the `engine` argument of `popMeans`.

```
library(multcomp)
g<-popMeans(mm,effect='A', at=list(C='1'),engine="glht")
g
```

General Linear Hypotheses

Linear Hypotheses:

	Estimate
1 == 0	0.203
2 == 0	0.295
3 == 0	0.443

This allows to apply the methods available on the `glht` object like

```
summary(g,test=univariate())
```

Simultaneous Tests for General Linear Hypotheses

Fit: `lm(formula = y ~ A + B + C, data = dd)`

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t)
1 == 0	0.203	0.409	0.50	0.63
2 == 0	0.295	0.409	0.72	0.48
3 == 0	0.443	0.409	1.08	0.30

(Univariate p values reported)

```
confint(g,calpha=univariate_calpha())
```

Simultaneous Confidence Intervals

Fit: `lm(formula = y ~ A + B + C, data = dd)`

```
Quantile = 2.179
95% confidence level
```

```
Linear Hypotheses:
```

	Estimate	lwr	upr
1 == 0	0.203	-0.688	1.094
2 == 0	0.295	-0.596	1.186
3 == 0	0.443	-0.448	1.334

which yield the same results as the `esticon` function.

By default the functions will adjust the tests and confidence intervals for multiplicity

```
summary(g)
```

```
Simultaneous Tests for General Linear Hypotheses
```

```
Fit: lm(formula = y ~ A + B + C, data = dd)
```

```
Linear Hypotheses:
```

	Estimate	Std. Error	t value	Pr(> t)
1 == 0	0.203	0.409	0.50	0.94
2 == 0	0.295	0.409	0.72	0.84
3 == 0	0.443	0.409	1.08	0.62

(Adjusted p values reported -- single-step method)

```
confint(g)
```

```
Simultaneous Confidence Intervals
```

```
Fit: lm(formula = y ~ A + B + C, data = dd)
```

```
Quantile = 2.731
```

```
95% family-wise confidence level
```

```
Linear Hypotheses:
```

	Estimate	lwr	upr
1 == 0	0.203	-0.914	1.319
2 == 0	0.295	-0.821	1.411
3 == 0	0.443	-0.674	1.559