

Population means (also called marginal means or LSMEANS), contrasts and estimable functions in the **doBy** package

Søren Højsgaard and Ulrich Halekoh

doBy version 4.5-9 as of 2013-09-21

Contents

1	Introduction	1
2	A simulated dataset	1
3	Linear functions of parameters, contrasts	2
4	Population means	3
4.1	A brute-force calculation	3
4.2	Using <code>esticon()</code>	4
5	Using <code>popMatrix()</code> and <code>popMeans()</code>	5
5.1	Using the <code>at</code> argument	6
5.2	Ambiguous specification when using the <code>effect</code> and <code>at</code> arguments	7
5.3	Using covariates	8
5.4	Using transformed covariates	9
6	The engine argument of <code>popMeans()</code>	9

1 Introduction

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

2 A simulated dataset

Consider these data:

```
> library(doBy)
> dd <- expand.grid(A=factor(1:3),B=factor(1:3),C=factor(1:2))
> dd$y <- rnorm(nrow(dd))
```

```
> dd$x <- rnorm(nrow(dd))^2
> dd$z <- rnorm(nrow(dd))
> head(dd,10)
```

	A	B	C	y	x	z
1	1	1	1	-1.38052509	2.356168125	0.3656083
2	2	1	1	-0.90016562	1.553465476	-0.4076551
3	3	1	1	0.69495708	2.417122053	-0.6567708
4	1	2	1	-0.08726416	0.992583438	0.7579046
5	2	2	1	-0.22240700	0.015879927	-0.2835553
6	3	2	1	0.10224181	4.788656263	-0.5791758
7	1	3	1	-1.25948377	1.081138255	-0.1912286
8	2	3	1	0.50358425	0.435574657	0.7452947
9	3	3	1	0.89295724	0.004447082	-0.2267921
10	1	1	2	0.06730484	0.021745317	1.0301705

Consider the additive model

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

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

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

(Intercept)	A2	A3	B2	B3	C2
-0.54828073	0.10760367	0.91388225	-0.04252357	0.11384476	0.57339090

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

3 Linear functions of parameters, contrasts

For a regression model with parameters $\beta = (\beta^1, \beta^2, \dots, \beta^P)$ we shall refer to a weighted sum of the form

$$\sum_j w_j \beta^j$$

as a contrast. Notice that it is common in the literature to require that $\sum_j w_j = 0$ for the sum $\sum_j w_j \beta^j$ to be called a contrast but we do not follow this tradition here.

The effect of changing the factor A from A2 to A3 can be found as

```
> w <- c(0,-1,1,0,0,0)
> sum(coef(mm)*w)

[1] 0.8062786
```

The `esticon()` function provides this estimate, the standard error etc. as follows:

```
> esticon(mm, w)

      beta0 Estimate Std.Error  t.value DF Pr(>|t|)      Lower      Upper
1      0 0.8062786 0.4001082  2.015151 12 0.0668524 -0.06548227 1.678039
```

4 Population means

Population means (sometimes also called marginal means) are in some sciences much used for reporting marginal effects (to be described below). Population means are known as lsmeans in SAS jargon. Population means is a special kind of contrasts as defined in Section 3.

The model (1) is a model for the conditional mean $\mathbb{E}(y|A, B, C)$. Sometimes one is interested in quantities like $\mathbb{E}(y|A)$. This quantity can not formally be found unless B and C are random variables such that we may find $\mathbb{E}(y|A)$ by integration.

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

4.1 A brute-force calculation

The population mean for $A = 1$ is

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

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

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

This quantity can be estimated as:

```

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

(Intercept)          A2          A3          B2          B3          C2
-0.54828073  0.00000000  0.00000000 -0.01417452  0.03794825  0.28669545

> sum(coef(mm)*w)

[1] -0.2378116

```

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 0.3333333 0.3333333 0.5
[2,]    1    1    0 0.3333333 0.3333333 0.5
[3,]    1    0    1 0.3333333 0.3333333 0.5

> W %*% coef(mm)

      [,1]
[1,] -0.2378116
[2,] -0.1302079
[3,]  0.6760707

```

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.

4.2 Using `esticon()`

Given that one has specified W , the `esticon()` function in the `doBy` package be used for the calculations above and the function also provides standard errors, confidence limits etc:

```

> esticon(mm, W)

  beta0  Estimate Std.Error   t.value DF  Pr(>|t|)    Lower    Upper
1      0 -0.2378116 0.2829192 -0.8405635 12 0.4170269 -0.85423956 0.3786165

```

```

2      0 -0.1302079 0.2829192 -0.4602299 12 0.6535791 -0.74663589 0.4862201
3      0  0.6760707 0.2829192  2.3896245 12 0.0341586  0.05964269 1.2924987

```

5 Using popMatrix() and popMeans()

Writing the matrix W is somewhat tedious and hence error prone. In addition, there is a potential risk of getting the wrong answer if the reference level of a factor has been changed. The `popMatrix()` function provides an automated way of generating such matrices. The above W matrix is constructed by

```

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

      (Intercept) A2 A3      B2      B3 C2
[1,]           1  0  0 0.3333333 0.3333333 0.5
[2,]           1  1  0 0.3333333 0.3333333 0.5
[3,]           1  0  1 0.3333333 0.3333333 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')
> pme

      beta0   Estimate Std.Error    t.value DF Pr(>|t|)      Lower      Upper A
1      0 -0.2378116 0.2829192 -0.8405635 12 0.4170269 -0.85423956 0.3786165 1
2      0 -0.1302079 0.2829192 -0.4602299 12 0.6535791 -0.74663589 0.4862201 2
3      0  0.6760707 0.2829192  2.3896245 12 0.0341586  0.05964269 1.2924987 3

```

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 A
1      0 -0.2378116 0.2829192 -0.8405635 12 0.4170269 -0.85423956 0.3786165 1
2      0 -0.1302079 0.2829192 -0.4602299 12 0.6535791 -0.74663589 0.4862201 2
3      0  0.6760707 0.2829192  2.3896245 12 0.0341586  0.05964269 1.2924987 3
Call:
NULL
Contrast matrix:
Length Class  Mode
      0   NULL  NULL

```

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

Likewise we may do:

```

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

      (Intercept) A2 A3      B2      B3 C2
[1,]           1  0  0 0.3333333 0.3333333  0
[2,]           1  1  0 0.3333333 0.3333333  0
[3,]           1  0  1 0.3333333 0.3333333  0
[4,]           1  0  0 0.3333333 0.3333333  1
[5,]           1  1  0 0.3333333 0.3333333  1
[6,]           1  0  1 0.3333333 0.3333333  1

```

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

```

> popMeans(mm)

      beta0 Estimate Std.Error    t.value DF  Pr(>|t|)      Lower      Upper
1      0 0.1026838 0.1633435 0.628637 12 0.5413679 -0.2532111 0.4585786

```

gives the “total average”.

5.1 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	0.3333333	0.3333333	0		
[2,]	1	1	0	0.3333333	0.3333333	0		
[3,]	1	0	1	0.3333333	0.3333333	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	0.3333333	0.3333333	0		
[2,]	1	1	0	0.3333333	0.3333333	0		
[3,]	1	0	1	0.3333333	0.3333333	0		
[4,]	1	0	0	0.3333333	0.3333333	1		
[5,]	1	1	0	0.3333333	0.3333333	1		
[6,]	1	0	1	0.3333333	0.3333333	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

5.2 Ambiguous specification when using the effect and at arguments

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	0.3333333	0.3333333	0		
[2,]	1	1	0	0.3333333	0.3333333	0		
[3,]	1	0	1	0.3333333	0.3333333	0		

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

This ambiguity is resolved as follows: 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'))
```

5.3 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
-0.23755937	0.05207291	1.14086025	-0.08959278	0.02535410	0.37301091

C1:x	C2:x
-0.21283597	-0.16946515

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	0.3333333	0.3333333	0	1.11891	0		
[2,]	1	1	0	0.3333333	0.3333333	0	1.11891	0		
[3,]	1	0	1	0.3333333	0.3333333	0	1.11891	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 population mean 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	0.3333333	0.3333333	0	12	0		
[2,]	1	1	0	0.3333333	0.3333333	0	12	0		
[3,]	1	0	1	0.3333333	0.3333333	0	12	0		

5.4 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
	-0.45207178	-0.08767834	0.77708906	-0.08418282	-0.10739259	0.83066589
	C1:log(x)	C2:log(x)				
	-0.13983303	0.08640353				

In this case we can not use `popMatrix()` (and hence `popMeans()` directly. 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	0.3333333	0.3333333	0	-1.263012			0
[2,]	1	1	0	0.3333333	0.3333333	0	-1.263012			0
[3,]	1	0	1	0.3333333	0.3333333	0	-1.263012			0

6 The engine argument of popMeans()

The `popMatrix()` is a function to generate a linear transformation matrix of the model parameters with emphasis on constructing such matrices for population means. `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	-0.5245
2 == 0	-0.4169
3 == 0	0.3894

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	-0.5245	0.3267	-1.606	0.134
2 == 0	-0.4169	0.3267	-1.276	0.226
3 == 0	0.3894	0.3267	1.192	0.256

(Univariate p values reported)

```
> confint(g,alpha=univariate_alpha())
```

Simultaneous Confidence Intervals

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

Quantile = 2.1788

95% confidence level

Linear Hypotheses:

	Estimate	lwr	upr
1 == 0	-0.5245	-1.2363	0.1873
2 == 0	-0.4169	-1.1287	0.2949
3 == 0	0.3894	-0.3224	1.1012

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	-0.5245	0.3267	-1.606	0.325
2 == 0	-0.4169	0.3267	-1.276	0.503
3 == 0	0.3894	0.3267	1.192	0.555

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

```
> confint(g)
```

Simultaneous Confidence Intervals

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

Quantile = 2.7299

95% family-wise confidence level

Linear Hypotheses:

	Estimate	lwr	upr
1 == 0	-0.5245	-1.4163	0.3673
2 == 0	-0.4169	-1.3087	0.4749
3 == 0	0.3894	-0.5024	1.2812