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

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

May 6, 2011

Contents

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  0.57073286 0.1210406  0.3473319
2  2 1 1 -1.84035705 1.3014560  0.6839562
3  3 1 1 -0.56373556 0.1636201 -0.7699442
4  1 2 1 -0.57060310 2.7351006 -0.5522332
5  2 2 1 -2.47289384 0.1755856  0.2461491
6  3 2 1  1.10373530 0.7580677 -0.6050502
7  1 3 1 -0.83044198 2.2886760 -1.4677773
8  2 3 1  0.09124191 0.2928978  0.4785219
9  3 3 1 -0.35329523 0.3299424  1.8181484
10 1 1 2 -1.72907365 3.1067122 -0.3055015
```

Consider the additive model:

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

(Intercept)          A2          A3          B2          B3          C2
-0.80284004 -0.59160283  0.65378393  0.08931764  0.63514915  0.21930883
```

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
-0.80284004  0.00000000  0.00000000  0.02977255  0.21171638  0.10965442

> sum(coef(mm)*w)

[1] -0.4516967
```

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 0.3333333 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,] -0.4516967
[2,] -1.0432995
[3,]  0.2020872
```

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 -0.4516967 0.4649318 -0.9715333 12 0.3504591 -1.464696  0.56130258
2     0 -1.0432995 0.4649318 -2.2439842 12 0.0444767 -2.056299 -0.03030024
3     0  0.2020872 0.4649318  0.4346600 12 0.6715228 -0.810912  1.21508652
```

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 0.3333333 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')
```

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 -0.4516967 0.4649318 -0.9715333 12 0.3504591 -1.464696  0.56130258
2     0 -1.0432995 0.4649318 -2.2439842 12 0.0444767 -2.056299 -0.03030024
3     0  0.2020872 0.4649318  0.4346600 12 0.6715228 -0.810912  1.21508652
Call:
popMeans.lm(object = mm, effect = "A")
Contrast matrix:
     (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 `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 0.3333333 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
```

Consequently

```
> popMeans(mm)

  beta0   Estimate Std.Error   t.value DF Pr(>|t|)     Lower     Upper
1     0 -0.4309697 0.2684285 -1.605529 12 0.134356 -1.015825 0.1538857
```

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 0.3333333 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
```

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 0.3333333 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 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
-0.64279227 -0.69617956  0.56989514  0.18333439  0.69429565  0.01587321
       C1:x        C2:x
-0.16341659  0.04336812
```

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 1.089585    0
[2,]           1  1  0 0.3333333 0.3333333  0 1.089585    0
[3,]           1  0  1 0.3333333 0.3333333  0 1.089585    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 0.3333333 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
```

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
 -0.9403438  -0.6368012   0.6582143   0.2260927   0.7129556   0.3478980
  C1:log(x)   C2:log(x)
 -0.1159179   0.1288051
```

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 0.3333333 0.3333333  0 -0.5334997        0
[2,]           1  1  0 0.3333333 0.3333333  0 -0.5334997        0
[3,]           1  0  1 0.3333333 0.3333333  0 -0.5334997        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` envokes by

default the `esticon` function on this linear transformation matrix for calculating parameter estimates and confidecne 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 -0.56135
2 == 0 -1.15295
3 == 0  0.09243
```

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 -0.56135    0.53686  -1.046   0.3163
2 == 0 -1.15295    0.53686  -2.148   0.0529 .
3 == 0  0.09243    0.53686   0.172   0.8662
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Univariate p values reported)

> confint(g,calpha=univariate_calpha())

         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 -0.56135 -1.73106  0.60836
2 == 0 -1.15295 -2.32266  0.01676
3 == 0  0.09243 -1.07728  1.26214
```

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 -0.56135    0.53686  -1.046    0.648
2 == 0 -1.15295    0.53686  -2.148    0.139
3 == 0  0.09243    0.53686   0.172    0.997
(Adjusted p values reported -- single-step method)

> confint(g)

         Simultaneous Confidence Intervals

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

Quantile = 2.7319
95% family-wise confidence level


Linear Hypotheses:
       Estimate lwr      upr
```

```
1 == 0 -0.56135 -2.02802  0.90532
2 == 0 -1.15295 -2.61962  0.31371
3 == 0  0.09243 -1.37423  1.55910
```