LSmeans, population means, estimates and contrasts with the `doBy` package

Søren Højsgaard

doBy version 4.5-5 as of feb 05, 2013

Contents

1 Introduction

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

```
library(doBy)
```

2 The `esticon` function

Consider a linear model which explains `Ozone` as a linear function of `Month` and `Wind`:

```
 data(airquality)
 airquality <- transform(airquality, Month=factor(Month))
 m<-lm(Ozone~Month*Wind, data=airquality)
 coefficients(m)

(Intercept)      Month6      Month7      Month8      Month9        Wind
     50.748     -41.793      68.296      82.211      23.439      -2.368
Month6:Wind Month7:Wind Month8:Wind Month9:Wind
      4.051      -4.663      -6.154      -1.874
```

When a parameter vector β of (systematic) effects have been estimated, interest is often in a particular estimable function, i.e. linear combination $\lambda^\top \beta$ and/or testing the hypothesis $H_0 : \lambda^\top \beta = \beta_0$ where λ is a specific vector defined by the user.

Suppose for example we want to calculate the expected difference in ozone between consequtive months at wind speed 10 mph (which is about the average wind speed over the whole period).

The `esticon` function provides a way of doing so. We can specify several λ vectors at the same time. For example

```
 Lambda <- rbind(
    c(0,-1,0,0,0,0,-10,0,0,0),
    c(0,1,-1,0,0,0,10,-10,0,0),
    c(0,0,1,-1,0,0,0,10,-10,0),
    c(0,0,0,1,-1,0,0,0,10,-10)
    )
```

```
 esticon(m, Lambda)

  beta0 Estimate Std.Error t.value  DF Pr(>|t|)   Lower  Upper
1     0   1.2871    10.238  0.1257 106  0.90019 -19.010 21.585
2     0 -22.9503    10.310 -2.2259 106  0.02814 -43.392 -2.509
3     0   0.9954     7.094  0.1403 106  0.88867 -13.069 15.060
4     0  15.9651     6.560  2.4337 106  0.01662   2.959 28.971
```

In other cases, interest is in testing a hypothesis of a contrast $H_0 : \Lambda\beta = \beta_0$ where Λ is a matrix. For example a test of no interaction between `Month` and `Wind` can be made by testing jointly that the last four parameters in `m` are zero (observe that the test is a Wald test):

```
 Lambda <- rbind(
    c(0,0,0,0,0,0,1,0,0,0),
    c(0,0,0,0,0,0,0,1,0,0),
    c(0,0,0,0,0,0,0,0,1,0),
    c(0,0,0,0,0,0,0,0,0,1)
    )
```

```
 esticon(m, Lambda, joint.test=T)

  X2.stat DF Pr(>|X^2|)
1   22.11  4  0.0001906
```

For a linear normal model, one would typically prefer to do a likelihood ratio test instead. However, for generalized estimating equations of glm–type (as dealt with in the packages **geepack** and **gee**) there is no likelihood. In this case `esticon` function provides an operational alternative.

3 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 -1.39918  1.324525  1.2155
2  2 1 1  1.31774  2.325944  0.8665
3  3 1 1  0.02265  2.117689 -0.4598
4  1 2 1 -0.10185  2.453452  0.9988
5  2 2 1  0.89448  0.690510 -0.3352
6  3 2 1 -0.60579  0.115301  1.2075
7  1 3 1 -1.49305  0.479673  0.0138
8  2 3 1  0.68182 11.181538  0.4928
9  3 3 1 -0.72781  0.007692 -0.3969
10 1 1 2 -0.28547  0.226861  1.1120
```

Consider the additive model

$$y_i = \beta_0 + \beta^1_{A(i)} + \beta^2_{B(i)} + \beta^3_{C(i)} + e_i \tag{1}$$

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
   -0.10920     0.64889    -0.31111    -0.03415    -0.44635     0.48645
```

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

4 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 litterature 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] -0.96
```

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    -0.96    0.5963   -1.61 12   0.1334 -2.259 0.3393
```

5 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 4.

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 $\mathbb{E}(y|A)$.

5.1 A brute–force calculation

The population mean for $A = 1$ is

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

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

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

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
   -0.10920     0.00000     0.00000    -0.01138    -0.14878     0.24322

 sum(coef(mm)*w)

[1] -0.02615
```

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,] -0.02615
[2,]  0.62274
[3,] -0.33725
```

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.

5.2 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 -0.02615    0.4217 -0.06201 12   0.9516 -0.9449 0.8926
2     0  0.62274    0.4217  1.47683 12   0.1655 -0.2960 1.5415
3     0 -0.33725    0.4217 -0.79980 12   0.4394 -1.2560 0.5815
```

6 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 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 0.3333 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')
pme

  beta0 Estimate Std.Error  t.value DF Pr(>|t|)   Lower  Upper A
1     0 -0.02615    0.4217 -0.06201 12   0.9516 -0.9449 0.8926 1
2     0  0.62274    0.4217  1.47683 12   0.1655 -0.2960 1.5415 2
3     0 -0.33725    0.4217 -0.79980 12   0.4394 -1.2560 0.5815 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 -0.02615    0.4217 -0.06201 12   0.9516 -0.9449 0.8926 1
2     0  0.62274    0.4217  1.47683 12   0.1655 -0.2960 1.5415 2
3     0 -0.33725    0.4217 -0.79980 12   0.4394 -1.2560 0.5815 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 0.3333 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
```

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  0.08645    0.2435  0.3551 12   0.7287 -0.444 0.6169
```

gives the "total average".

6.1 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
```

6.2 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 0.3333 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 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'))
```

6.3 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
   -0.37992     0.58345    -0.21457     0.11568    -0.47142     0.88353
      C1:x        C2:x
    0.09513    -0.20237
```

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 1.59    0
[2,]           1  1  0 0.3333 0.3333  0 1.59    0
[3,]           1  0  1 0.3333 0.3333  0 1.59    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 0.3333 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
```

6.4 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 
   -0.13970     0.61069    -0.21550     0.01296    -0.37431     0.46806 
  C1:log(x)   C2:log(x) 
    0.08663     0.01046 
```

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 0.3333 0.3333  0    -0.64        0
[2,]           1  1  0 0.3333 0.3333  0    -0.64        0
[3,]           1  0  1 0.3333 0.3333  0    -0.64        0
```

7 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 population means. `popMeans()` invokes 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   -0.269
2 == 0    0.380
3 == 0   -0.580
```

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   -0.269      0.487   -0.55     0.59
2 == 0    0.380      0.487    0.78     0.45
3 == 0   -0.580      0.487   -1.19     0.26
(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 -0.269   -1.330  0.792
2 == 0  0.380   -0.681  1.440
3 == 0 -0.580   -1.641  0.480
```

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   -0.269      0.487   -0.55     0.92
2 == 0    0.380      0.487    0.78     0.81
3 == 0   -0.580      0.487   -1.19     0.55
(Adjusted p values reported -- single-step method)

 confint(g)

         Simultaneous Confidence Intervals

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

Quantile = 2.732
95% family-wise confidence level


Linear Hypotheses:
       Estimate lwr    upr
1 == 0 -0.269   -1.600  1.061
2 == 0  0.380   -0.951  1.710
3 == 0 -0.580   -1.911  0.750
```