A study of the analysis of variance helper functions within R

by Marek Rychlik

March 3, 2009

ANOVA model for completely randomized designs

Following [?], we can write the model down as follows:

$$y_{ij} = \mu_i + \epsilon_{ij}, \ 1 \leq i \leq t, 1 \leq j \leq r_i \, .$$

The model can be rewritten as a General Linear Model:

$$\begin{pmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1r_1} \\ y_{21} \\ \vdots \\ y_{t1} \\ y_{t2} \\ \vdots \\ y_{tr_t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 0 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 1 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & \vdots & \ldots & \vdots \\ 0 & 0 & 0 & \ldots & 1 \\ 0 & 0 & 0 & \ldots & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_t \end{pmatrix} + \begin{pmatrix} \epsilon_{11} \\ \epsilon_{12} \\ \vdots \\ \epsilon_{1r_1} \\ \epsilon_{21} \\ \vdots \\ \epsilon_{t1} \\ \epsilon_{t2} \\ \vdots \\ \epsilon_{tr_t} \end{pmatrix}.$$

- This has the generic form $y = A\mu + \epsilon$ where A is the design matrix.

- The i-th group of rows of A repeats exactly r_i-times.

A transformation to the intercept format

R does not use the above model literally but first rewrites it in the form:

$$\begin{aligned} y_{1\,j} &= \mu_1 + \epsilon_{1\,j} \qquad \text{for } 1 \le j \le r_1, \\ y_{i\,j} &= \mu_1 + \tau_i + \epsilon_{i\,j} \text{ for } i \ge 2 \text{ and } 1 \le j \le r_i \end{aligned}$$

This model is valid when the default contrasts are used. Thus, the correspondence with the first model is established by defining:

$$\tau_i = \mu_i - \mu_1 \quad \text{for } i = 2, 3, ..., t.$$

Hence, μ_1 plays the role of the intercept and is recorded as such by various R functions.

The final form of the linear model

The linear model of a completely randomized design is as follows:

$$\begin{pmatrix} y_{11} \\ y_{12} \\ \vdots \\ y_{1r_1} \\ y_{21} \\ \vdots \\ y_{t1} \\ y_{t2} \\ \vdots \\ y_{tr_t} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & 0 & 0 & \dots & 1 \\ 1 & 0 & 0 & \dots & 1 \end{pmatrix} \begin{pmatrix} \mu_1 \\ \tau_2 \\ \vdots \\ \tau_t \end{pmatrix} + \begin{pmatrix} \epsilon_{11} \\ \epsilon_{12} \\ \vdots \\ \epsilon_{1r_1} \\ \epsilon_{21} \\ \vdots \\ \epsilon_{t1} \\ \epsilon_{t2} \\ \vdots \\ \epsilon_{tr_t} \end{pmatrix}.$$

That is, the first column of the model consists of ones, and the remaining columns are unchanged. Alternatively, we may interpret this form as R using the contrasts $\tau_i = \mu_i - \mu_1$ and thus supporting the pairwise comparison of the successive means to the first mean.

A review of the QR method for least squares

We start with the linear system:

$$y = A\mu + \epsilon.$$

The problem of minimizing the norm $\|y - A\mu\|$ is solved by formally multiplying the above by A^T:

$$A^T y = A^T A \mu.$$

We solve for μ, so the solution is:

$$\mu = (A^T A)^{-1} A y.$$

This is indeed true when the matrix $A^T A$ is invertible, which is quite common.

The QR-algorithm, which is a practical implementation of the Gram-Shmidt orthogonalization process, is applied to the matrix A to facilitate the solution of the system. We recall that the QR-algorithm produces the following factorization:

$$A = Q\,R$$

where Q is a matrix with orthogonal columns, and R is a non-singular (and thus square) upper-triangular matrix. Thus, $Q^T Q = I$. Hence,

$$A^T A = (Q\,R)^T (Q\,R) = R^T Q^T Q\,R = R^T I\,R = R^T R.$$

and $A^T y = R^T Q^T y$.

Thus, the system $A^T y = A^T A\mu$ can be rewritten as

$$R^T Q^T y = R^T R\mu.$$

Multiplying both sides by $(R^T)^{-1}$ we obtain the simplified equation:

$$R\mu = Q^T y.$$

Hence, the solution may be expressed as:

$$\mu = R^{-1} Q^T y.$$

In practice, because R is upper triangular, R^{-1} would never be computed. The system would be solved by back-substitution.

The goal of what follows

We will do some "reverse engineering" of the R function `aov`. We will not call `aov` directly, but simulate its functionality by calling small bits of code that can be found in `aov` and `lm` that it calls. Both functions perform least squares fitting of a model to data. The implementations of these functions can be examined by typing in their names at the R prompt. However, any production code addresses many concerns that have nothing to do with the mathematical idea behind it,

such as error checking, parsing etc. This makes the code long and hard to understand unless you are a regular R developer. Therefore we will write a mock `aov` in a form of a minimalistic script that produces equivalent result for a narrow class of problems. There are several benefits:

- We understand the algorithms better
- We gain the knowledge of the R primitives that go into implementing ANOVA with R. We can reuse these primitives in other situations. For instance, we may implement non-standard versions of ANOVA to solve specific problems for which the general framework does not work well.

The data

We define a very simple data set:

```
> N <- 12; t <- 4; repl <- N / t;
> Response <- 1:N
> Treatment <- as.factor(c(rep("T1", repl), rep("T2", repl),
  rep("T3", repl), rep("T4", repl)))
> fake.data <- data.frame(Treatment, Response)
> fake.data
```

```
   Treatment Response
1         T1        1
2         T1        2
3         T1        3
4         T2        4
5         T2        5
6         T2        6
7         T3        7
8         T3        8
9         T3        9
10        T4       10
11        T4       11
12        T4       12
```

```
>
```

Analysis of variance steps

Define contrasts

Contrasts between means can be defined by assigning an attribute `contrasts` to a factor.

```
> contrasts(Treatment) <- contr.treatment(t)
>
```

After the contrasts are assigned to a factor, they will be used in any calculation that involves this factor, unless they are overridden, for instance, by giving an argument `contrasts` to the function `aov`.

Define the full model design matrix

There is an R function for creating design matrices: `model.matrix`.

```
> full.dm <- model.matrix(Response ~ Treatment, fake.data)
> full.dm
```

	(Intercept)	TreatmentT2	TreatmentT3	TreatmentT4
1	1	0	0	0
2	1	0	0	0
3	1	0	0	0
4	1	1	0	0
5	1	1	0	0
6	1	1	0	0
7	1	0	1	0

```
8            1            0            1            0
9            1            0            1            0
10           1            0            0            1
11           1            0            0            1
12           1            0            0            1
attr(,"assign")
[1] 0 1 1 1
attr(,"contrasts")
attr(,"contrasts")$Treatment
[1] "contr.treatment"
```

```
>
```

Define the reduced model design matrix

```
> reduced.dm <- model.matrix(Response ~ Treatment - Treatment,
  fake.data)
> reduced.dm
```

```
   (Intercept)
1            1
2            1
```

```
3             1
4             1
5             1
6             1
7             1
8             1
9             1
10            1
11            1
12            1
attr(,"assign")
[1] 0
attr(,"contrasts")
attr(,"contrasts")$Treatment
[1] "contr.treatment"
```

```
>
```

We used a somewhat artificial construction: the right-hand side of the model is `Treatment-Treatment`, which means that the factor is `Treatment`, but we take it away. So, the only component left is the intercept.

Perform a least squares fit using QR-decomposition

We use the standard approach of solving least squares problems using the QR-decomposition of matrices. Since we have to perform it twice (once for the full model and once for the reduced model) we define a function that encapsulated our calculation.

```
> design.fit <- function(design.matrix, y) {
      qr <- qr(design.matrix) # Find QR-decomposition
      Q  <- qr.Q(qr)  # Extract Q
      R  <- qr.R(qr)  # Extract R
      Ri <- solve(R) # Find the inverse of R
      ## We create an environment to put variables in
      env <- new.env()
      env$coef <- Ri %*% crossprod(Q, y) # coefficients
      env$err  <- qr.resid(qr, y) # residues
      env$resid <- sum(env$err^2)
      return(as.list(env)) # convert environment to a list
  }
>
```

Solve least squares problems

```
> full <- design.fit(full.dm, Response)
> reduced <- design.fit(reduced.dm, Response)
> full
$resid
[1] 8

$err
 [1] -1.000000e+00  8.049117e-16  1.000000e+00 -1.000000e+00 -
5.551115e-17
 [6]  1.000000e+00 -1.000000e+00  2.220446e-16  1.000000e+00 -
1.000000e+00
[11]  1.665335e-16  1.000000e+00

$coef

             [,1]
(Intercept)    2
TreatmentT2    3
```

```
   TreatmentT3    6
   TreatmentT4    9
> reduced
   $resid
   [1] 143

   $err
    [1] -5.5 -4.5 -3.5 -2.5 -1.5 -0.5  0.5  1.5  2.5  3.5  4.5  5.5

   $coef
                [,1]
   (Intercept)  6.5
>
```

Find various sums of squares

```
> SSTotal <- sum((Response - mean(Response))^2)
> SSMean  <- SSTotal / (N - 1)
> SSE     <- full$resid
```

```
> MSE     <- SSE / (N - t)
> SST     <- reduced$resid - full$resid
> MST     <- SST / (t - 1)
> c(SSTotal, SSMean, SSE, MSE, SST, MST)
   [1] 143  13   8   1 135  45
>
```

Find the F-statistic

```
> F       <- MST / MSE
> F
   [1] 45
>
```

Comparing the results with `aov`

In the code below, we compute the `aov` object and print the summary. Subsequently, we calculate standard error for the difference of the means of treatment groups `T1` and `T2`.

```
> fake.aov <- aov(Response ~ Treatment, fake.data)
> summary(fake.aov)
             Df Sum Sq Mean Sq F value    Pr(>F)
Treatment     3    135      45      45 2.356e-05 ***
Residuals     8      8       1
---
Signif. codes:  0 âĂŸ***âĂŹ 0.001 âĂŸ**âĂŹ 0.01 âĂŸ*âĂŹ 0.05
âĂŸ.âĂŹ 0.1 âĂŸ âĂŹ 1
> se.contrast(fake.aov,list(Treatment == "T1", Treatment == "T2"))
[1] 0.8164966
>
```