

# Stat 849: Fitting linear models in R

Douglas Bates

2010-09-03

## Outline

## Contents

### 1 lm

#### Fitting linear models in R

The `lm` function in R provides a formula/data interface for fitting linear models

- The first argument, `formula`, is a two-sided formula with the response on the left hand side and model `terms`, separated by `+` signs, on the right hand side.
- The second argument, `data`, is the name of a data frame in which to evaluate the formula. It is optional but recommended.
- Most other arguments have default values that are rarely changed

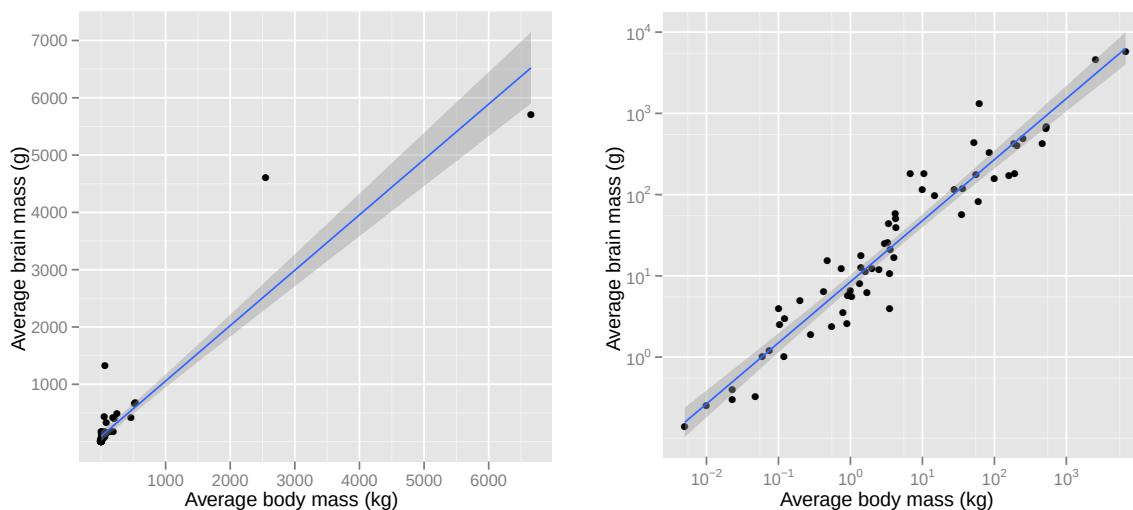
```
1  args(lm)
```

```
function (formula, data, subset, weights, na.action, method = "qr",  
  model = TRUE, x = FALSE, y = FALSE, qr = TRUE, singular.ok = TRUE,  
  contrasts = NULL, offset, ...)  
NULL
```

#### Brain weight data

Recall the structure of the `brains` data in the `alr3` package

```
'data.frame': 62 obs. of 2 variables:  
 $ BrainWt: num 44.5 15.5 8.1 423 119.5 ...  
 $ BodyWt : num 3.38 0.48 1.35 464.98 36.33 ...
```



## Fitting linear models to the brains data

We fit two linear models, corresponding to the lines on the plots:

**lm1** BrainWt as a linear function of BodyWt. From the data plots we expect that this will not fit well

**lm2**  $\log(\text{BrainWt})$  as a linear function of  $\log(\text{BodyWt})$

```
1 lm1 <- lm(BrainWt ~ BodyWt, brains)
2 lm2 <- lm(log(BrainWt) ~ log(BodyWt), brains)
```

Notice that we assign the result of fitting a model to a name. The fitted model is a *classed* object

```
1 class(lm1)
```

```
[1] "lm"
```

to which we can apply several different *extractor* functions, most of which are *methods* for generic functions. It is worthwhile checking the documentation of such methods to see what they do.

## 2 Summaries

### Listing of the methods for class lm

```
1 methods(class="lm")
```

```
[1] add1.lm*      alias.lm*      anova.lm
[4] case.names.lm* confint.lm*    cooks.distance.lm*
[7] deviance.lm*  dfbeta.lm*    dfbetas.lm*
[10] drop1.lm*     dummy.coef.lm* effects.lm*
[13] extractAIC.lm* family.lm*     formula.lm*
[16] fortify.lm    hatvalues.lm  influence.lm*
[19] kappa.lm     labels.lm*    logLik.lm*
```

```

[22] model.frame.lm      model.matrix.lm      plot.lm
[25] predict.lm          print.lm              proj.lm*
[28] residuals.lm        rstandard.lm         rstudent.lm
[31] simulate.lm*        summary.lm            variable.names.lm*
[34] vcov.lm*

```

Non-visible functions are asterisked

## The summary of fitted model lm1

```
1 summary(lm1)
```

Call:

```
lm(formula = BrainWt ~ BodyWt, data = brains)
```

Residuals:

| Min     | 1Q     | Median | 3Q     | Max     |
|---------|--------|--------|--------|---------|
| -810.15 | -88.52 | -79.65 | -13.02 | 2050.52 |

Coefficients:

|             | Estimate | Std. Error | t value | Pr(> t ) |
|-------------|----------|------------|---------|----------|
| (Intercept) | 91.00864 | 43.55574   | 2.089   | 0.0409   |
| BodyWt      | 0.96646  | 0.04767    | 20.276  | <2e-16   |

Residual standard error: 334.7 on 60 degrees of freedom

Multiple R-squared: 0.8726, Adjusted R-squared: 0.8705

F-statistic: 411.1 on 1 and 60 DF, p-value: < 2.2e-16

## The summary of fitted model lm2

```
1 summary(lm2)
```

Call:

```
lm(formula = log(BrainWt) ~ log(BodyWt), data = brains)
```

Residuals:

| Min      | 1Q       | Median   | 3Q      | Max     |
|----------|----------|----------|---------|---------|
| -1.71550 | -0.49228 | -0.06162 | 0.43598 | 1.94833 |

Coefficients:

|             | Estimate | Std. Error | t value | Pr(> t ) |
|-------------|----------|------------|---------|----------|
| (Intercept) | 2.13479  | 0.09604    | 22.23   | <2e-16   |
| log(BodyWt) | 0.75169  | 0.02846    | 26.41   | <2e-16   |

Residual standard error: 0.6943 on 60 degrees of freedom

Multiple R-squared: 0.9208, Adjusted R-squared: 0.9195

F-statistic: 697.4 on 1 and 60 DF, p-value: < 2.2e-16

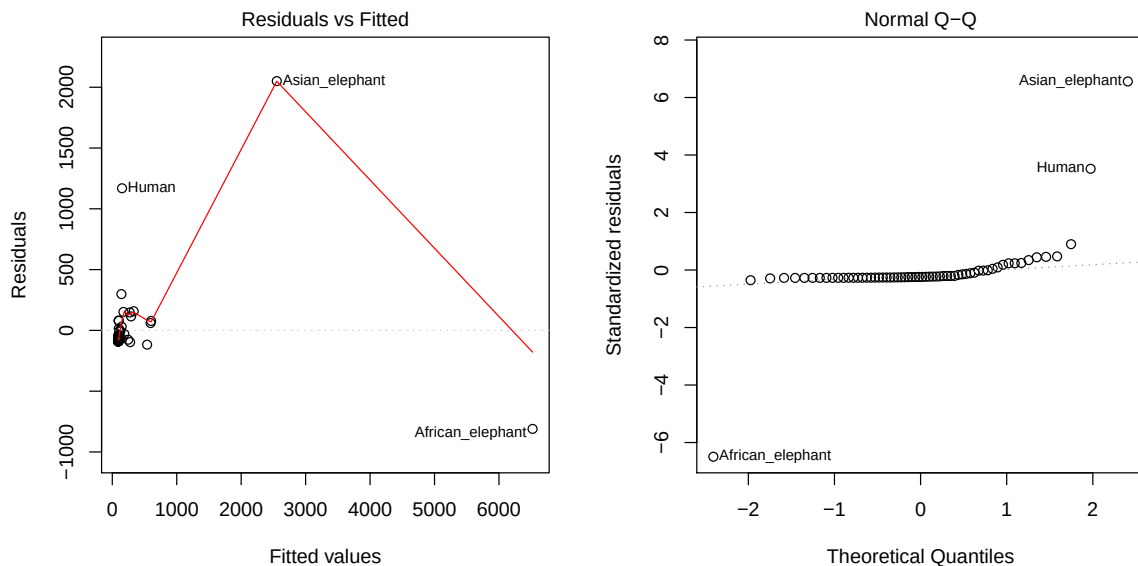
### 3 Residual plots

#### Residual plots

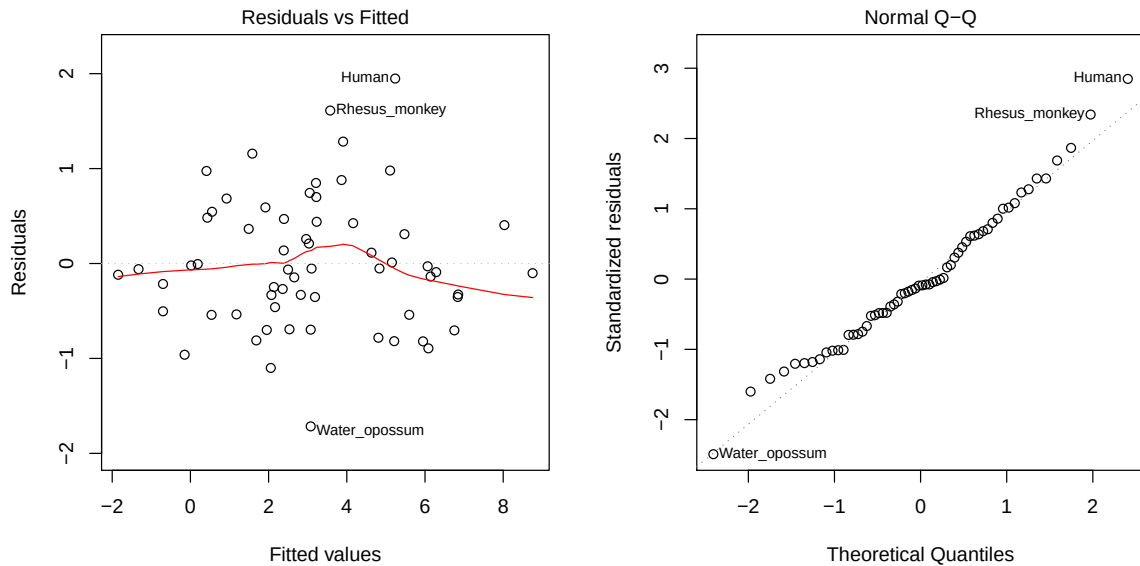
- The base graphics system provides good “canned” residual plots.
- There are 6 possibilities available through the `which` argument to the `plot` method for `lm` objects. See `?plot.lm` for details or experiment for yourself.
- You should always check the first two
- To lay out multiple residual plots in the same figure, set the graphical parameter `mfcol` or `mfrow` using the `par()` function.
- It is a good practice to save the old values of the parameters, which is the value returned by `par()` and restore it after the plot.

#### Residuals vs fitted and normal Q-Q plot, model `lm1`

```
1 opar <- par(mfcol=c(1,2))
2 plot(lm1, which=1:2)
3 par(opar)
```



#### Residuals vs fitted and normal Q-Q plot, model `lm2`



## 4 Anova & coef tables

### Analysis of variance table

For a simple linear regression, the anova table is not too interesting but, for other models, it will be.

```
1 anova(lm2)
```

#### Analysis of Variance Table

```
Response: log(BrainWt)
      Df Sum Sq Mean Sq F value    Pr(>F)
log(BodyWt)  1 336.19   336.19   697.42 < 2.2e-16
Residuals   60  28.92     0.48
```

- This anova table is based on the **sequential** sums of squares. We will discuss later what that means.
- The `anova` generic can take as arguments several fitted models, in which case it provides the analysis of variance for comparing nested models.

### The coefficients table

A little-known extractor

```
1 coef(summary(lm2))
```

```
              Estimate Std. Error t value    Pr(>|t|)
(Intercept) 2.1347883  0.09604321  22.22737 1.183103e-30
log(BodyWt)  0.7516861  0.02846349  26.40878 9.834309e-35
```

provides the coefficients table. To print the table as it is shown in the `summary` output, use

```
1 printCoefmat(coef(summary(lm2)))

              Estimate Std. Error t value Pr(>|t|)
(Intercept) 2.134788    0.096043  22.227 < 2.2e-16
log(BodyWt) 0.751686    0.028463  26.409 < 2.2e-16
```

## Incorporating tables in L<sup>A</sup>T<sub>E</sub>X documents

The `xtable` package provides a L<sup>A</sup>T<sub>E</sub>X table for pretty-printing. Options for captions, labels, special environments, etc. are available.

```
1 library(xtable)
2 xtable(lm2)
```

|             | Estimate | Std. Error | t value | Pr(> t ) |
|-------------|----------|------------|---------|----------|
| (Intercept) | 2.1348   | 0.0960     | 22.23   | 0.0000   |
| log(BodyWt) | 0.7517   | 0.0285     | 26.41   | 0.0000   |

```
1 xtable(anova(lm2))
```

|             | Df | Sum Sq | Mean Sq | F value | Pr(>F) |
|-------------|----|--------|---------|---------|--------|
| log(BodyWt) | 1  | 336.19 | 336.19  | 697.42  | 0.0000 |
| Residuals   | 60 | 28.92  | 0.48    |         |        |

## 5 Other extractors

Extractor methods for  $\hat{\beta}$ , RSS, etc.

```
1 deviance(lm2) # residual sum of squares

[1] 28.92261

1 coef(lm2)      # coefficient estimates

(Intercept) log(BodyWt)
 2.1347883   0.7516861

1 vcov(lm2)      # variance-covariance of coef est.

              (Intercept)    log(BodyWt)
(Intercept)  0.009224299 -0.0010836348
log(BodyWt) -0.001083635  0.0008101703

1 logLik(lm2)    # log-likelihood

'log Lik.' -64.33635 (df=3)
```

## Confidence intervals, model matrix

```
1 confint(lm2) # confidence intervals on coefficients

              2.5 %    97.5 %
(Intercept) 1.9426733 2.3269033
log(BodyWt) 0.6947507 0.8086216

1 kappa(lm2) # condition number of the X matrix

[1] 3.967229

1 str(model.matrix(lm2)) # the X matrix itself

num [1:62, 1:2] 1 1 1 1 1 1 1 1 1 1 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:62] "Arctic_fox" "Owl_monkey" "Beaver" "Cow" ...
..$ : chr [1:2] "(Intercept)" "log(BodyWt)"
- attr(*, "assign")= int [1:2] 0 1
```

## Model frame

- Some of the processing of the formula and of arguments like `subset`, `offset` and `na.action` takes place in the formation of the `model.frame` before the creation of the `model.matrix`.
- The model frame contains many attributes that are useful for extractors like `anova`.
- It contains only the data actually used to fit the model.

```
1 str(model.frame(lm2))

'data.frame': 62 obs. of 2 variables:
 $ log(BrainWt): num 3.8 2.74 2.09 6.05 4.78 ...
 $ log(BodyWt) : num 1.219 -0.734 0.3 6.142 3.593 ...
- attr(*, "terms")=Classes 'terms', 'formula' length 3 log(Bra..
.. ..- attr(*, "variables")= language list(log(BrainWt), log(..
.. ..- attr(*, "factors")= int [1:2, 1] 0 1
.. .. ..- attr(*, "dimnames")=List of 2
.. .. .. ..$ : chr [1:2] "log(BrainWt)" "log(BodyWt)"
.. .. .. ..$ : chr "log(BodyWt)"
.. .. ..- attr(*, "term.labels")= chr "log(BodyWt)"
.. .. ..- attr(*, "order")= int 1
.. .. ..- attr(*, "intercept")= int 1
.. .. ..- attr(*, "response")= int 1
.. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
.. .. ..- attr(*, "predvars")= language list(log(BrainWt), log(B..
.. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "num..
.. .. ..- attr(*, "names")= chr [1:2] "log(BrainWt)" "log(Bod..
```

## 6 Simulation

### Simulation

- The list of methods includes `simulate.lm`, which simulates a matrix of responses from the parameter estimates of the model.
- Furthermore, you can generate all the information on the fitted models for these responses with a single call to `lm` or to `lm.fit`, which does the calculations within `lm`.
- This is orders of magnitude faster than calling `lm` inside a loop.
- To fit model `lm2` to 10000 simulated responses takes a couple of seconds

```
1 system.time({Ymat <- data.matrix(simulate(lm2,10000))
2   lm2Sim <- lm(Ymat ~ log(BodyWt), brains)})

      user system elapsed
3.450    0.060    3.514
```

### Extracting the results of the simulation

All of the extractor functions for an `lm` model apply to `lm2Sim`. Extractors that return scalars now return a vector

```
1 str(deviance(lm2Sim))

Named num [1:10000] 29.3 36.6 25.2 34.2 27.2 ...
- attr(*, "names")= chr [1:10000] "sim_1" "sim_2" "sim_3" "sim_..
```

Extractors that return a vector now return a matrix

```
1 str(coef(lm2Sim))

num [1:2, 1:10000] 1.906 0.726 1.992 0.801 2.09 ...
- attr(*, "dimnames")=List of 2
..$ : chr [1:2] "(Intercept)" "log(BodyWt)"
..$ : chr [1:10000] "sim_1" "sim_2" "sim_3" "sim_4" ...
```

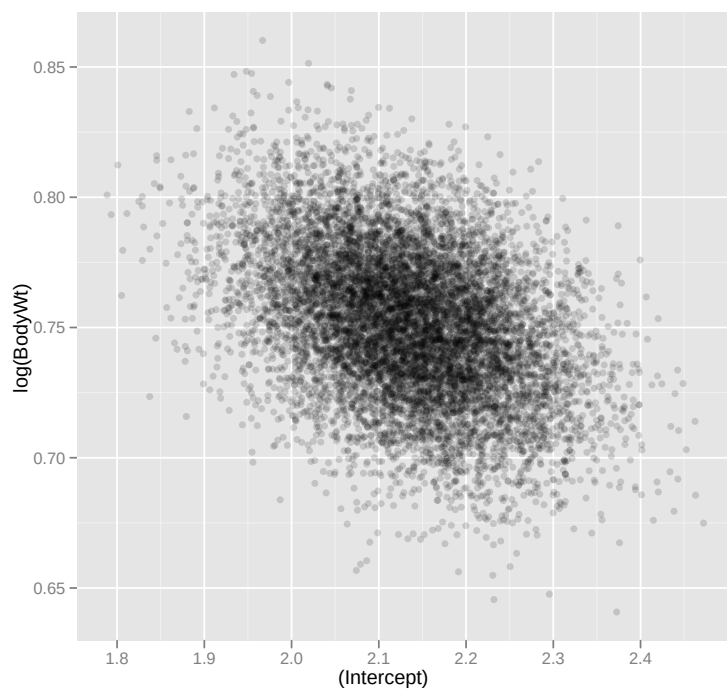
We often want a data frame of the transpose

```
1 coefSim <- data.frame(t(coef(lm2Sim)), check.names=FALSE)
```

### Scatter plot of coefficient estimates

```
1 qqplot('(Intercept)', 'log(BodyWt)', data=coefSim,
2   alpha="0.15")
```

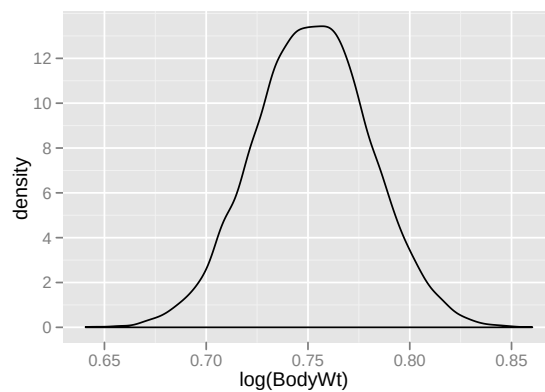
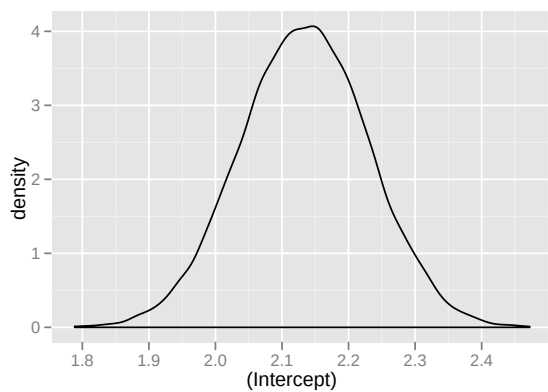




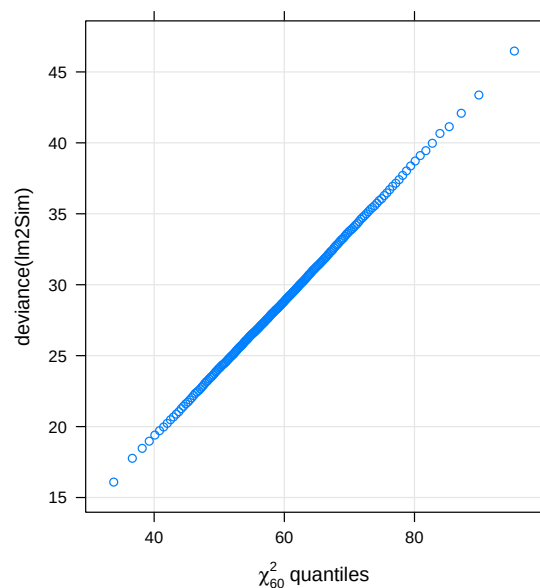
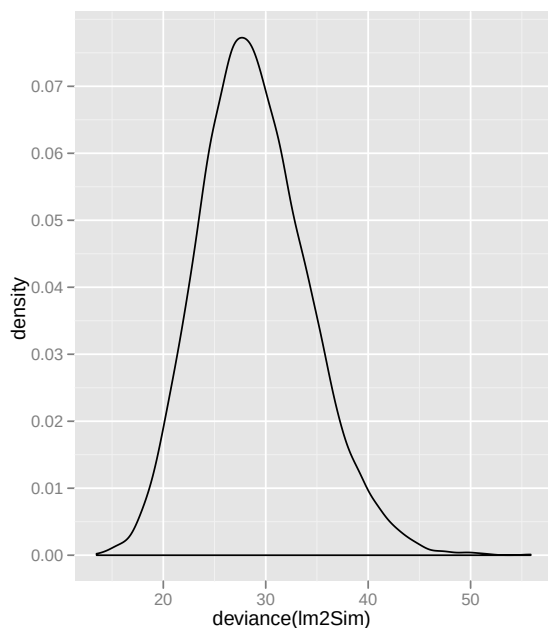
### Distribution of parameter estimates

```
1 cbind(coef(summary(lm2))[,1:2],
2       simMn=sapply(coefSim, mean),
3       simSd=sapply(coefSim, sd))
```

|             | Estimate  | Std. Error | simMn     | simSd      |
|-------------|-----------|------------|-----------|------------|
| (Intercept) | 2.1347883 | 0.09604321 | 2.1337564 | 0.09615131 |
| log(BodyWt) | 0.7516861 | 0.02846349 | 0.7519274 | 0.02852459 |



### Distribution of the residual sum of squares



## 7 Diagnostics

### Diagnostic measures

- Many of the methods for `lm` objects, such as `fitted`, `residuals`, `rstandard`, `rstudent`, `cooks.distance`, `hatvalues`, `dffit`, and `dfbetas` provide information used in diagnostics. We will discuss these later.
- See also the results of the `influence` method and the `lm.influence` function.
- `ggplot2` provides a convenience method `fortify` that extracts the model frame and augments it with diagnostic measures.

```
1 head(fortify(lm2), 3)
```

|            | log(BrainWt) | log(BodyWt) | .hat    | .sigma | .cooks   |
|------------|--------------|-------------|---------|--------|----------|
| Arctic_fox | 3.795        | 1.2194      | 0.01615 | 0.6933 | 0.009584 |
| Owl_monkey | 2.741        | -0.7340     | 0.02334 | 0.6833 | 0.034018 |
| Beaver     | 2.092        | 0.3001      | 0.01794 | 0.6993 | 0.001391 |

|            | .fitted | .resid  | .stdresid |
|------------|---------|---------|-----------|
| Arctic_fox | 3.051   | 0.7441  | 1.0805    |
| Owl_monkey | 1.583   | 1.1577  | 1.6873    |
| Beaver     | 2.360   | -0.2685 | -0.3903   |

## 8 QR and effects

### QR decomposition of the model matrix

- Most formulas you have seen for regression and analysis of variance results (e.g.  $\hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y}$ ) bear no resemblance to the way things are actually calculated.

- Virtually all the numerical work for linear models in R uses the QR decomposition,  $\mathbf{X} = \mathbf{QR} = \mathbf{Q}_1 \mathbf{R}_1$ , where  $\mathbf{Q}$  is orthogonal (i.e.  $\mathbf{Q}'\mathbf{Q} = \mathbf{Q}\mathbf{Q}' = \mathbf{I}_n$ ) and  $\mathbf{R}$  is zero below the main diagonal. Sometimes we write  $\mathbf{R}_1$  for the first  $p$  rows of  $\mathbf{R}$  and  $\mathbf{Q}_1$  for the first  $p$  columns of  $\mathbf{Q}$ .
- From version 2.12.0 (Oct., 2010) R will provide a `qr` extractor for `lm` objects. For now you need to use `$qr`

```
1 str(lm2$qr)

List of 5
 $ qr   : num [1:62, 1:2] -7.874 0.127 0.127 0.127 0.127 ...
 ..- attr(*, "dimnames")=List of 2
 .. ..$ : chr [1:62] "Arctic_fox" "Owl_monkey" "Beaver" "Cow" ..
 .. ..$ : chr [1:2] "(Intercept)" "log(BodyWt)"
 ..- attr(*, "assign")= int [1:2] 0 1
 $ graux: num [1:2] 1.13 1.08
 $ pivot: int [1:2] 1 2
 $ tol   : num 1e-07
 $ rank  : int 2
 - attr(*, "class")= chr "qr"
```

## Relationship to the Cholesky decomposition

```
1 qr.R(lm2$qr)      # R matrix from the QR

              (Intercept) log(BodyWt)
Arctic_fox    -7.874008    -10.53180
Owl_monkey     0.000000     24.39242

1 X <- model.matrix(lm2)
2 chol(XtX <- crossprod(X)) # Cholesky of X'X

              (Intercept) log(BodyWt)
(Intercept)    7.874008    10.53180
log(BodyWt)     0.000000    24.39242

1 XtX

              (Intercept) log(BodyWt)
(Intercept)    62.00000    82.92745
log(BodyWt)     82.92745   705.90906
```

## The effects vector

```
1 head(effects(lm2), 5)

(Intercept) log(BodyWt)
-24.72594592  18.33544667 -0.39394094 -0.59250661 -0.04375128
```

- This curious vector is  $Q'y$ . Not only is it used to calculate  $\hat{\beta}$  as the solution to  $R_1\hat{\beta} = Q_1'y$  but also the `anova` sums of squares are calculated from it.
- Compare sum of squares for `log(BodyWt)` to the `log(BodyWt)` component of

```
1 head(effects(lm2)^2, 5)

(Intercept) log(BodyWt)
6.113724e+02 3.361886e+02 1.551895e-01 3.510641e-01 1.914174e-03
```

### Is that really the analysis of variance calculation?

Yes, Virginia, it is. Consider an example of a two-factor anova with interactions. We simulate the data

```
1 set.seed(1234321)
2 dat <- data.frame(f1=gl(4,6,labels=LETTERS[1:4]),
3                   f2=gl(3,2,labels=letters[1:3]),
4                   y=8+rnorm(24, sd=0.1))
5 xtabs(~ f2 + f1, dat)

      f1
f2  A B C D
a  2 2 2 2
b  2 2 2 2
c  2 2 2 2
```

and fit the model

```
1 lm3 <- lm(y ~ f1 * f2, dat)
```

### Fitting the anova model with interactions

```
1 anova(lm3)
```

#### Analysis of Variance Table

```
Response: y
      Df Sum Sq Mean Sq F value Pr(>F)
f1      3  0.018342  0.0061139   0.4656  0.7116
f2      2  0.036907  0.0184535   1.4053  0.2829
f1:f2    6  0.096525  0.0160876   1.2251  0.3588
Residuals 12  0.157575  0.0131313
```

Compare the sums of squares to the results of summing elements of `effects(am1)^2` according to

```
1 (asgn <- attr(model.matrix(lm3), "assign"))

[1] 0 1 1 1 2 2 3 3 3 3 3 3

1 tapply(effects(lm3)[1:12]^2, asgn, sum)

      0      1      2      3
1.552090e+03 1.834185e-02 3.690692e-02 9.652533e-02
```