1 Linear model formulas

In R linear models are specified using a *model formula*, which is an expression that contains a tilde (the $\sim$ character). The response is on the left-hand side of the tilde, typically as the name of a variable, e.g. `optden`, but it can also be a function of a variable, e.g. `log(BrainWt)`.

The right-hand side of the formula is composed of *model terms* separated by plus signs. In the formulas below we write the response as `y`, continuous covariates as `x`, `z`, `u`, ... and categorical covariates as `f` and `g`. Note that the categorical covariates are assumed to be stored as factors (which includes ordered factors).

Some of the formulas for typical models are:

Simple linear regression The formula

```
## y ~ x
```

denotes the simple linear regression model

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i, \quad i = 1, \ldots, n$$

In the formula shown above, the intercept term is implicit. If you prefer to make it explicit you can write the formula as

```
## y ~ 1 + x
```

Regression through the origin If you do not want the intercept term in the model, you must suppress it using the formula

```
## y ~ 0 + x
```

or, alternatively,

```
## y ~ x - 1
```

The model specified in this way can be written as

$$y_i = \beta x_i + \epsilon_i, \quad i = 1, \ldots, n$$

Notice that you can remove terms, even the implicit intercept, with a negative sign.

Multiple linear regression Multiple covariates can be listed on the right hand side, as in

```
## y ~ 1 + x + z + u
```

corresponding to the model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 z_i + \beta_3 u_i + \epsilon_i, \quad i = 1, \ldots, n$$

Polynomial regression To include polynomial terms in the model you must protect the circumflex operator by surrounding the term with `I()`, which is the identity operator. It implies that the expression inside is to be taken literally in terms of the arithmetic operators, not as the formula language operators. The model

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \epsilon_i, \quad i = 1, \ldots, n$$

is written

```
## y ~ x + I(x^2) + I(x^3)
```

Another specification for a polynomial regression model uses the `poly()` function which generates *orthogonal polynomial* terms. The fitted responses will be the same from the model shown above and from

```
## y ~ poly(x, 3)
```

but the coefficients will be different because they are defined with respect to the orthogonal polynomials. These have some advantages if you are doing the calculations by hand but in practice you don't expect to be doing so.

One categorical covariate The model described as a one-way analysis of variance for the levels of factor, `f`, corresponds to the formula

```
## y ~ f
```

Often we use the function `aov()` instead of `lm()` to fit such models. `aov()` is the same as `lm()` except that it puts an extra tag on the fitted model that designates it as only having categorical covariates. This changes, for example, the `summary()` method, which produces an analysis of variance table instead of a summary of the estimated coefficients. The model that is fit is sometimes written as

$$y_{ij} = \mu + \alpha_i + \epsilon_{ij}, \quad i = 1, \ldots, I \; j = 1, \ldots, n_i$$

although it is not fit in that form.

Two categorical covariates, additive The formula for an additive two-factor analysis of variance model,

$$y_{ijk} = \mu + \alpha_i + \beta_j + \epsilon_{ijk}, \quad i = 1, \ldots, I \; j = 1, \ldots, J \; k = 1, \ldots, n_{ij},$$

is

```
## y ~ f + g
```

This produces a two-factor analysis of variance table. In the balanced case the analysis of variance table for this model is equivalent to that for the model

```
## y ~ g + f
```

in the sense that, although the rows of the table will be in a different order, they are otherwise the same. For unbalanced data the order of the factors is important. The sums of squares in the table are *sequential* sums of squares corresponding to the contribution of the first factor, given the intercept, then the contribution of the second factor, given the first factor and the intercept, and so on. In particular, *blocking factors*, which represent uncontrolable sources of variability, should be listed before experimental factors.

Two categorical covariates, allowing for interactions If the data include replicate observations (more than one observation at the same combination of covariate values) we can fit and analyze a model with interaction terms with a formula like

```
## y ~ f + g + f:g
```

where an expression like `f:g` is a two-factor interaction. Similar expressions are used for higher-order interactions. This model can also be expressed as

```
## y ~ f * g
```

In general the asterisk operator, (`*`), generates the main effects plus interactions. A three-factor model with all the main effects, two-factor interactions and the three-factor interaction can be written as

```
## y ~ f * g * h
```

Combination of continuous and categorical covariates What is sometimes called an *analysis of covariance* model incorporates both categorical and numeric covariates. If there is only one numeric covariate, `x`, then the model can be described in terms of the lines formed by the fitted values on the `y` versus `x` plot. The most common models are the parallel lines (different intercepts, same slope) generated by

```
## y ~ f + x
```

and the model in which slopes and intercepts both vary according to the levels of `f`

```
## y ~ f * x
```

which is equivalent to

```
## y ~ f + x + f:x
```

Occasionally we incorporate an interaction term without a main-effect for `f`.

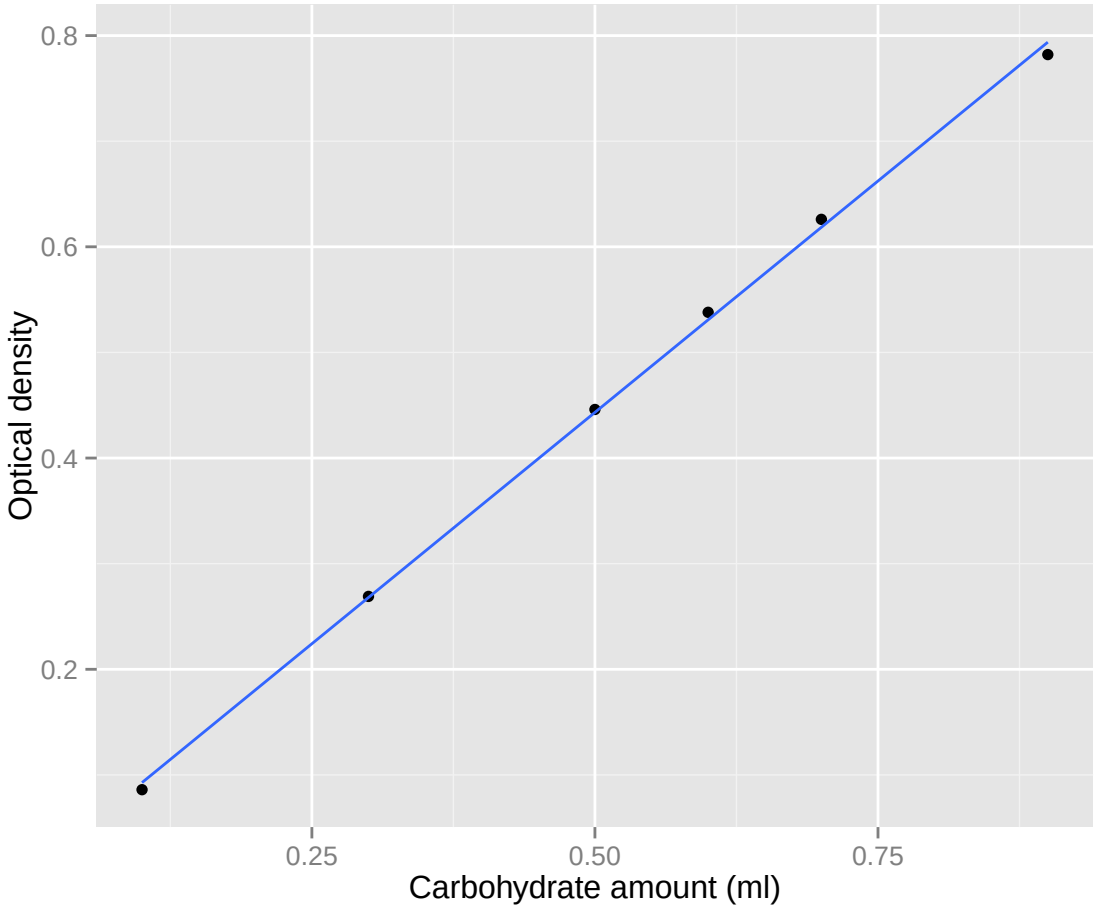

Figure 1: Observations of optical density versus carbohydrate amount from the calibration of a Formaldehyde assay.

```
## y ~ x + f:x
```

I call this the "zero-dose" model because it is used in the case that `x` represents something like a dose and the levels of `f` corresponds to different treatments. We don't have a main effect for `f` in such a model because a zero dose of treatment 1 is the same as a zero dose of treatment 2. Thus the lines for the different levels of the factors should coincide at `x=0`.

2 Examples

The `datasets` package contains several sample datasets that have been used in different texts. By default, this package is attached in an R session.

Simple linear regression The `Formaldehyde` data are a simple example from a calibration study consisting of 6 observations of the carbohydrate content (ml.) (variable `carb`) and the corresponding optical density (variable `optden`). Figure **??** is a data plot with the fitted simple linear regression line. This model is fit as

```
summary(lm1 <- lm(optden ~ 1 + carb, Formaldehyde))

##
## Call:
## lm(formula = optden ~ 1 + carb, data = Formaldehyde)
##
## Residuals:
##         1         2         3         4         5         6
## -0.006714  0.001029  0.002771  0.007143  0.007514 -0.011743
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.005086   0.007834   0.649    0.552
## carb        0.876286   0.013535  64.744 3.41e-07
##
## Residual standard error: 0.008649 on 4 degrees of freedom
## Multiple R-squared:  0.999,Adjusted R-squared:  0.9988
## F-statistic:  4192 on 1 and 4 DF,  p-value: 3.409e-07
```

(In what follows we will often skip the full summary output and concentrate on the coefficients table, produced by `coef(summary())`, or the analysis of variance table, produced by `anova()`.)

Regression through the origin To constrain the line to pass through the origin (that is, to suppress the `(Intercept)` term) we fit the model as

```
coef(summary(lm1a <- lm(optden ~ 0 + carb, Formaldehyde)))

##       Estimate  Std. Error  t value     Pr(>|t|)
## carb 0.8841294 0.005736558 154.1219 2.181654e-10
```

A comparative analysis of variance of these two models

```
anova(lm1a,lm1)

## Analysis of Variance Table
##
## Model 1: optden ~ 0 + carb
## Model 2: optden ~ 1 + carb
##   Res.Df        RSS Df  Sum of Sq      F Pr(>F)
## 1      5 0.00033073
## 2      4 0.00029920  1 3.1526e-05 0.4215 0.5516
```

produces the same p-value as the t-test on the intercept coefficient in model `lm1`, which is as it should be, because these are two versions of the same test.

Polynomial regression Alternatively, we could fit `optden` as a quadratic function of `carb` using

```
coef(summary(lm1b <- lm(optden ~ 1 + carb + I(carb^2), Formaldehyde)))
```

```
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -0.0116827  0.0060015 -1.9466   0.14675
## carb         0.9711234  0.0267825 36.2596 4.613e-05
## I(carb^2)   -0.0962121  0.0263094 -3.6569   0.03532
```

Notice that the quadratic term is significant at the 5% level and generally we would retain it in the model. The reason we don't see much curvature in the data plot (Fig. **??**) is because there is such a strong linear trend that it masks any nonlinear behaviour.

An alternative specification is

```
coef(summary(lm1c <- lm(optden ~ poly(carb, 2), Formaldehyde)))
```

```
##                  Estimate Std. Error  t value  Pr(>|t|)
## (Intercept)     0.4578333  0.0017452 262.3429 1.221e-07
## poly(carb, 2)1  0.5599550  0.0042748 130.9904 9.810e-07
## poly(carb, 2)2 -0.0156326  0.0042748  -3.6569   0.03532
```

Multiple linear regression The `trees` data are measurements of the volume of usable lumber (variable `Volume`) from a sample of 31 black cherry trees. Covariates are a measurement of the girth (`Girth`), which is comparatively easy to measure (you just walk up to the tree and loop a tape measure around it), and the height (`Height`), which is somewhat more difficult to measure. (There is some confusion in the description of the data regarding whether the girth has been converted to an equivalent diameter - we'll assume it is the girth.) If we consider the tree to have the shape of as a cylinder or a cone we would expect that the volume would be related to the square of the girth times the height. In Fig. 2 we show the volume versus the girth on the original scale and on a log-log scale. There is not a tremendous difference in the patterns but careful examination shows better linear behavior in the log-log scale.

Our initial model is

```
coef(summary(lm2 <- lm(log(Volume) ~ log(Girth), trees)))
```

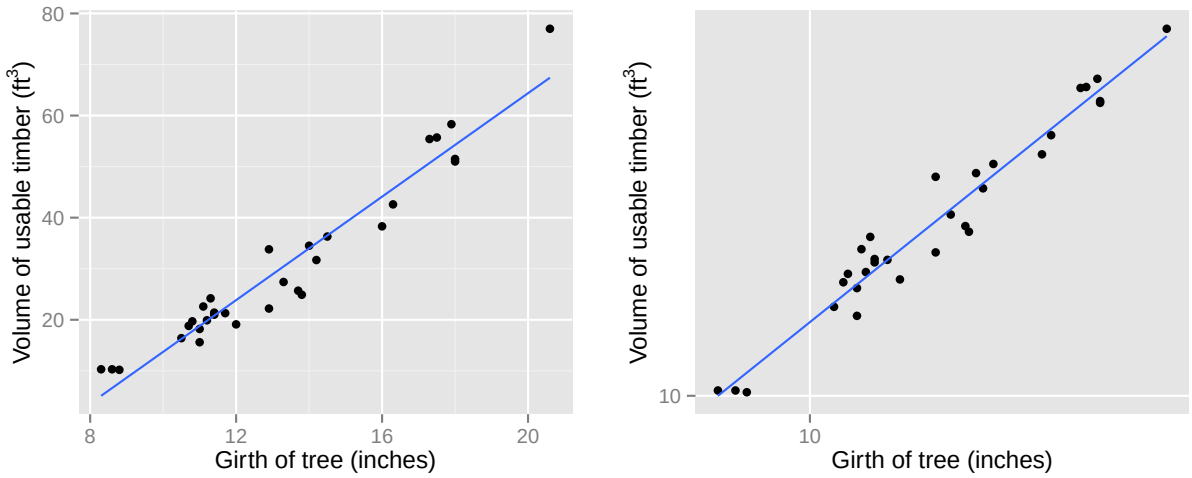

Figure 2: Scatterplot of the volume of usable lumber versus the girth of the tree for 31 black cherry trees. The left panel is on the original scale. The right panel is on the log-log scale.

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -2.353325   0.230663 -10.202  4.18e-11
## log(Girth)   2.199970   0.089835  24.489 < 2.2e-16
```

To fit a model corresponding to a conical or cylindrical shape we add a term in `log(Height)` (recall that we are on the log-log scale)

```
coef(summary(lm2a <- lm(log(Volume) ~ log(Girth) + log(Height), trees)))
```

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -6.631617   0.799790 -8.2917 5.057e-09
## log(Girth)   1.982650   0.075011 26.4316 < 2.2e-16
## log(Height)  1.117123   0.204437  5.4644 7.805e-06
```

Testing specific combinations of parameters At this point we may want to check if a version of the formula for the volume of a cylinder or of a cone, both of which have the form

$$V = k\, d^2 h$$

where k is a constant, d is the diameter (or, equivalently, the girth or circumference at the base) and h is the height. Such an expression would correspond to a value of 2 for the `log(Girth)` coefficient and 1 for the `log(Height)` term. The `log(Height)` term is highly significant and the coefficients of `log(Girth)` and `log(Height)` are reasonably close to 2 and 1. In particular, confidence intervals on these coefficients include 2 and 1

```
confint(lm2a)

##                 2.5 %    97.5 %
## (Intercept) -8.269912 -4.993322
## log(Girth)   1.828998  2.136302
## log(Height)  0.698353  1.535894
```

The confidence intervals do not, by themselves, answer the question of whether a model of the form

$$\log(\text{Volume}_i) = \beta_0 + 2\log(\text{Girth}_i) + \log(\text{Height}_i) + \epsilon_i, \quad i = 1, \ldots, 31$$

is a reasonable fit. To fit this model we use an `offset` expression in the model formula.

```
lm2c <- lm(log(Volume) ~ 1 + offset(2*log(Girth) + log(Height)), trees)
```

and perform a comparative analysis of variance

```
anova(lm2c,lm2a)

## Analysis of Variance Table
##
## Model 1: log(Volume) ~ 1 + offset(2 * log(Girth) + log(Height))
## Model 2: log(Volume) ~ log(Girth) + log(Height)
##   Res.Df     RSS Df Sum of Sq      F Pr(>F)
## 1     30 0.18769
## 2     28 0.18546  2 0.0022224 0.1678 0.8464
```

The large p-value indicates that the more complex model (general values of the coefficients for `log(Girth)` and `log(Height)`) does not fit significantly better than the simpler model (assuming `2*log(Girth) + log(Height)`), thus we prefer the simpler model.

One-way analysis of variance Next consider the `InsectSprays` data with a response, `count`, related to a categorical covariate, `spray`. Comparative boxplots (Fig. 3) show that the square root of the count is a more reasonable scale for the response and that there is considerable differences in the response according to the spray type.

Althought we can fit a model with categorical covariates using the `lm()` function, there is an advantage in using the `aov()` function instead, because it allows us to extract some additional information that applies only to categorical factors. Also, `summary()` applied to an `aov()` model produces the analysis of variance table, which for such models, is more interesting than the coefficients table.

```
summary(av1 <- aov(sqrt(count) ~ spray, InsectSprays))

##             Df Sum Sq Mean Sq F value Pr(>F)
## spray        5  88.44  17.688    44.8 <2e-16
## Residuals   66  26.06   0.395
```

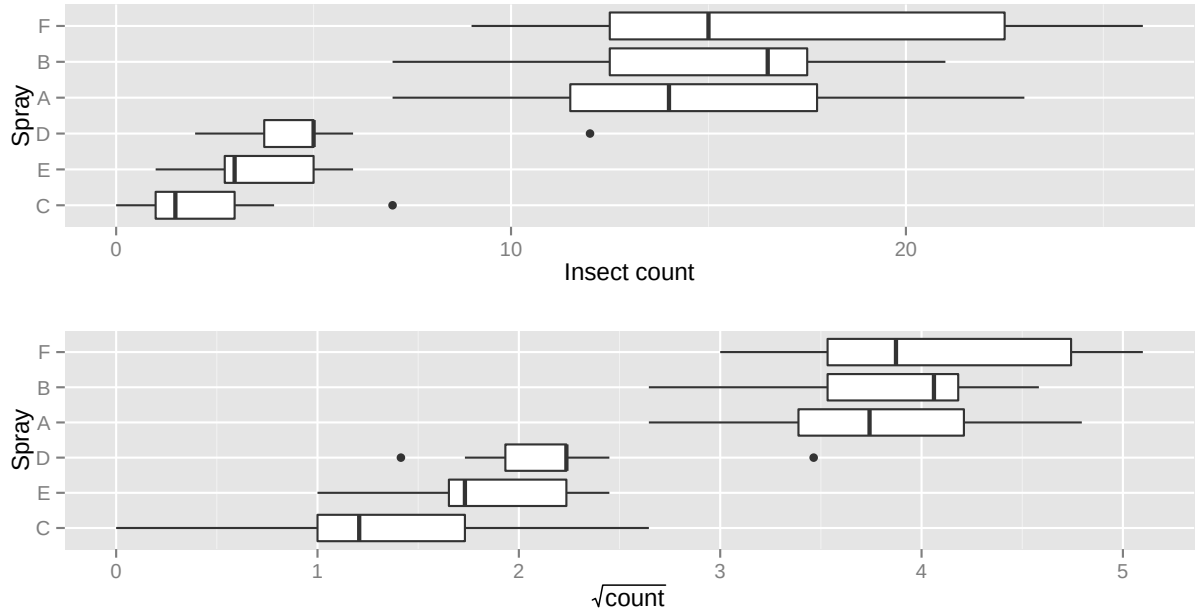

Figure 3: Comparative boxplots of the insect count by spray type in the `InsectSprays` data. The sprays have been reordered according to increasing mean response. In the lower panel the response is the square root of the count.

If we want to express the model in terms of "effects" we can obtain these as

```
model.tables(av1)

## Tables of effects
##
##  spray
## spray
##       A       B       C       D       E       F
##  0.9482  1.0642 -1.5676 -0.6481 -1.0030  1.2062
```

or, if we are interested in the estimates of the means for each group,

```
model.tables(av1, type="means")

## Tables of means
## Grand mean
##
## 2.812433
##
##  spray
## spray
```

```
##     A     B     C     D     E     F
## 3.761 3.877 1.245 2.164 1.809 4.019
```

Various types of "multiple comparisons" methods are also available. We will discuss these later.

Multi-factor analysis of variance When we have more than one categorical covariate, as in the `OrchardSprays` data,

```
str(OrchardSprays)

## 'data.frame': 64 obs. of  4 variables:
##  $ decrease : num  57 95 8 69 92 90 15 2 84 6 ...
##  $ rowpos   : num  1 2 3 4 5 6 7 8 1 2 ...
##  $ colpos   : num  1 1 1 1 1 1 1 1 2 2 ...
##  $ treatment: Factor w/ 8 levels "A","B","C","D",..: 4 5 2 8 7 6 3 1 3 2 ...
```

we simply include them in the model formula. It happens that for this experiment there are two blocking factors, `rowpos` and `colpos`, and one experimental factor, `treatment`, so we put the blocking factors first.

```
summary(av2 <- aov(decrease ~ factor(rowpos) + factor(colpos) + treatment, OrchardSprays))

##                Df Sum Sq Mean Sq F value   Pr(>F)
## factor(rowpos)  7   4767     681   1.788    0.115
## factor(colpos)  7   2807     401   1.053    0.410
## treatment       7  56160    8023  21.067 7.45e-12
## Residuals      42  15995     381
```

These data are arranged in what is called a "Latin square" design, which is a special type of fractional replication. There are 64 observations on three factors, each at 8 levels, so not only are there no replications, we don't even have an observation in each of the possible $8 \times 8 \times 8 = 512$ combinations, and cannot try to fit a model with interaction terms.

Multi-factor anova with replications The `warpbreaks` data, shown in Fig. 4,
are counts of the number of warp breaks per loom (a length of wool) according to the tension setting for the wool and the type of wool. We see that on the original scale of the number of breaks per loom there is increasing variance with an increasing level of the response, whereas on the reciprocal scale (number of looms per break) the variability is much closer to being constant.

Because there are 9 replications at each of the wool/tension combinations

```
xtabs(~ wool + tension, warpbreaks)

##     tension
## wool L M H
##    A 9 9 9
##    B 9 9 9
```

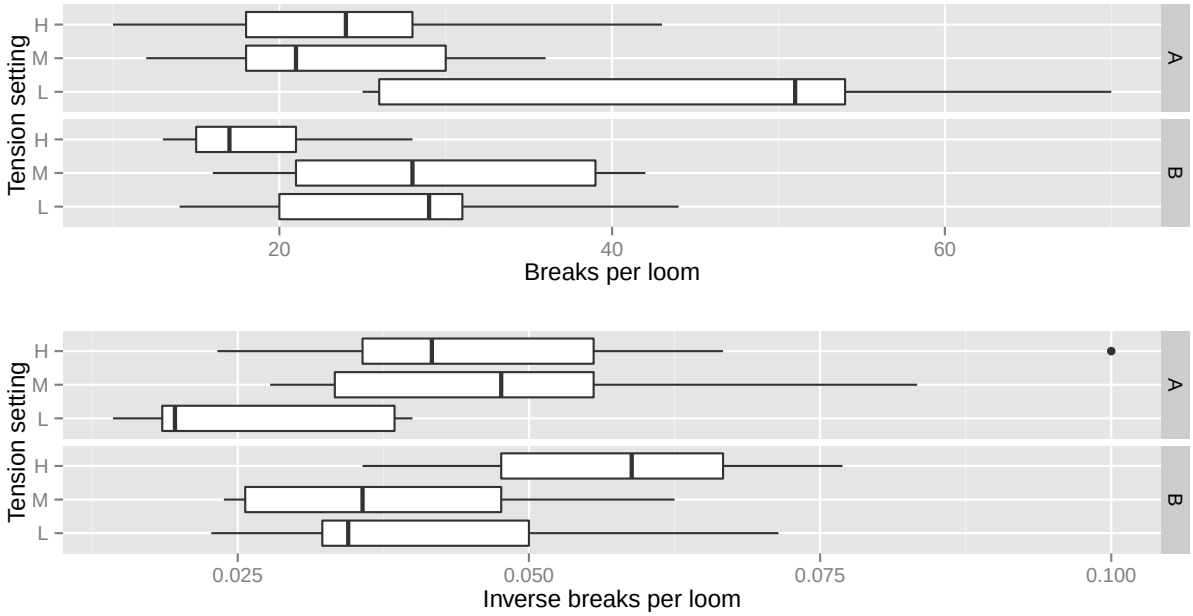

Figure 4: Comparative boxplots of the number of warp breaks per loom by tension setting for the warpbreaks data. Panels are determined by wool type. The upper two panels are on the original scale of number of breaks. The lower two panels are on the reciprocal scale (i.e. number of looms per break).

we can fit a model with main effects for `wool` and for `tension` and the `wool:tension` interaction.

```
summary(av3 <- aov(breaks ~ wool * tension, warpbreaks))

##              Df Sum Sq Mean Sq F value   Pr(>F)
## wool          1    451   450.7   3.765 0.058213
## tension       2   2034  1017.1   8.498 0.000693
## wool:tension  2   1003   501.4   4.189 0.021044
## Residuals    48   5745   119.7
```

In this model the interaction is significant. When an interaction is significant we typically retain both of the main effects in the model.

However, if we fit the model on the reciprocal scale

```
summary(av3a <- aov(1/breaks ~ wool * tension, warpbreaks))

##              Df   Sum Sq   Mean Sq F value  Pr(>F)
## wool          1 0.000240 0.0002403   0.900 0.34751
## tension       2 0.003345 0.0016727   6.264 0.00383
## wool:tension  2 0.001209 0.0006044   2.263 0.11498
## Residuals    48 0.012817 0.0002670
```

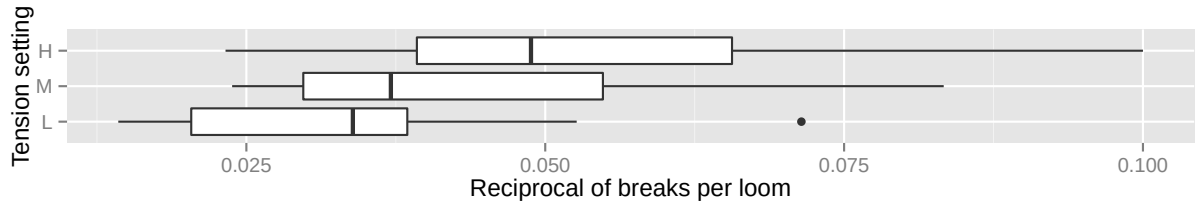

Figure 5: Comparative boxplots of the reciprocal of the number of warp breaks per loom by tension setting for the warpbreaks data.

we no longer have a significant interaction and could reduce the model to the main effects only

```
summary(av3b <- aov(1/breaks ~ tension +  wool, warpbreaks))

##             Df   Sum Sq   Mean Sq F value  Pr(>F)
## tension      2 0.003345 0.0016727   5.963 0.00476
## wool         1 0.000240 0.0002403   0.857 0.35909
## Residuals   50 0.014026 0.0002805
```

Here we have reordered the factors `tension` and `wool` so that `wool` is the last term and thus the second row of the analysis of variance table corresponds to a test of the main effect of the wool given that tension had been taken into account. (If you look closely at the sums of squares, degrees of freedom and mean squares you will see that they are consistent in models `av3b` and `av3a` but that is a consequence of the data being completely balanced with respect to these factors. To be safe, always make the factor you are going to test be the last one in the model formula.) The `wool` factor is not significant and we can reduce the model to a single factor model

```
summary(av3c <- aov(1/breaks ~ tension, warpbreaks))

##             Df   Sum Sq   Mean Sq F value  Pr(>F)
## tension      2 0.003345 0.0016727    5.98 0.00465
## Residuals   51 0.014267 0.0002797
```

corresponding to Fig. **??**
in which we can see a trend across the three ordered levels of tension; low tension gives a low reciprocal number of breaks (corresponding to a higher frequency of breaks), medium tension gives an intermediate reciprocal number and high tension gives the highest reciprocal number.

This is a common situation with a factor like `tension` whose levels are in a natural ordering, `L < M < H`. Details will be given later but, for now, it is enough to see that if we convert the factor to an ordered factor

```
str(warpbreaks <- within(warpbreaks, tension <- ordered(tension)))

## 'data.frame': 54 obs. of  4 variables:
##  $ breaks   : num  26 30 54 25 70 52 51 26 67 18 ...
```

```
##  $ wool     : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
##  $ tension  : Ord.factor w/ 3 levels "L"<"M"<"H": 1 1 1 1 1 1 1 1 1 2 ...
##  $ invbreaks: num  0.0385 0.0333 0.0185 0.04 0.0143 ...
```

and fit the model as before,

```
summary(av3d <- aov(1/breaks ~ tension, warpbreaks))
```

```
##             Df   Sum Sq   Mean Sq F value  Pr(>F)
## tension      2 0.003345 0.0016727    5.98 0.00465
## Residuals   51 0.014267 0.0002797
```

we get the same analysis of variance table but now the two degrees of freedom for `tension` are divided into a linear trend and a quadratic relationship in addition to the linear trend

```
coef(summary.lm(av3d))
```

```
##                Estimate  Std. Error t value  Pr(>|t|)
## (Intercept)  4.2798e-02  2.2760e-03 18.8040 < 2.2e-16
## tension.L    1.3633e-02  3.9422e-03  3.4582  0.001106
## tension.Q   -8.0425e-05  3.9422e-03 -0.0204  0.983803
```

With a p-value of 98.4%, the quadratic term is not at all significant, indicating that we could reduce to only the linear trend.

Analysis of covariance models The data shown in Fig. 6 are from a study (conducted several years ago) described as

> A tax consultant studied the current relation between selling price and assessed valuation of single-family residential dwellings in a large tax district by obtaining data for a random sample of 16 "arm's length" sales transactions of single-family dwellings located on corner lots and for a random sample of 48 recent sales of single-family dwellings not located on corner lots.

Assuming a linear relationship between the selling price and the assessed value, there are three different models we would typically consider:

```
coef(summary(lm3 <- lm(Selling ~ 1 + Assessed, assessed)))
```

```
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -118.79385   15.93796 -7.4535 3.517e-10
## Assessed       2.64738    0.21314 12.4208 < 2.2e-16
```

in which there is no distinction between corner lots and non-corner lots,

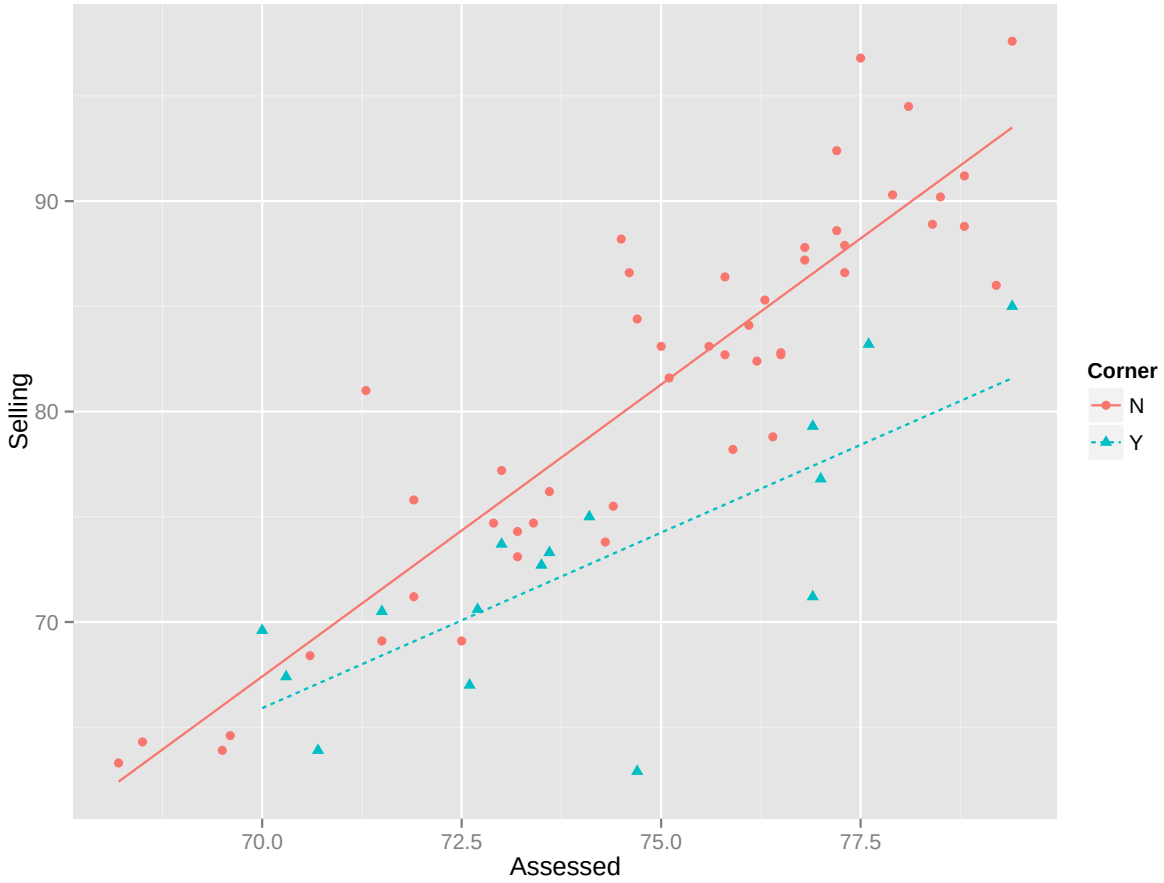

Figure 6: Selling price versus assessed value for a random sample of 16 single-family homes located on corner lots and 48 single-family homes not on corner lots.

```
coef(summary(lm3a <- lm(Selling ~ 1 + Corner + Assessed, assessed)))
```

```
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -107.45972   13.55094 -7.9301 5.803e-11
## CornerY       -6.20568    1.19331 -5.2004 2.447e-06
## Assessed       2.51646    0.18062 13.9321 < 2.2e-16
```

in which there is a common slope but a different intercept for the corner lots, and

```
coef(summary(lm3b <- lm(Selling ~ 1 + Corner * Assessed, assessed)))
```

```
##                    Estimate Std. Error t value  Pr(>|t|)
## (Intercept)      -126.90517   14.72247 -8.6198 4.327e-12
## CornerY            76.02153   30.13136  2.5230  0.014304
## Assessed            2.77590    0.19628 14.1424 < 2.2e-16
## CornerY:Assessed   -1.10748    0.40554 -2.7309  0.008281
```

in which there are different intercepts and slopes for the corner lots.

We could use a comparative analysis of variance to compare a simpler model (H_0) versus a more complex model (H_a).

```
anova(lm3, lm3a)

## Analysis of Variance Table
##
## Model 1: Selling ~ Assessed
## Model 2: Selling ~ 1 + Corner + Assessed
##   Res.Df    RSS Df Sum of Sq      F    Pr(>F)
## 1     62 1475.2
## 2     61 1022.1  1    453.15 27.044 2.447e-06

anova(lm3a, lm3b)

## Analysis of Variance Table
##
## Model 1: Selling ~ 1 + Corner + Assessed
## Model 2: Selling ~ 1 + Corner * Assessed
##   Res.Df    RSS Df Sum of Sq      F   Pr(>F)
## 1     61 1022.1
## 2     60  909.1  1       113 7.4578 0.008281
```

Because these F tests have one numerator degree of freedom there will be a corresponding t-test for a coefficient. This will be the t-test on the coefficient in H_a that distinguishes it from H_0.

3 Classes and methods for linear models

A model fit with `lm()` has class `"lm"`

```
class(lm2)

## [1] "lm"
```

for which there are several methods defined

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

```
##  [1] add1           alias          anova          case.names     coerce
##  [6] confint        cooks.distance deviance       dfbeta         dfbetas
## [11] drop1          dummy.coef     effects        extractAIC     family
## [16] formula        fortify        hatvalues      influence      initialize
## [21] kappa          labels         logLik         model.frame    model.matrix
## [26] nobs           plot           predict        print          proj
## [31] qr             residuals      rstandard      rstudent       show
## [36] simulate       slotsFromS3    summary        variable.names vcov
## see '?methods' for accessing help and source code
```

We have already seen several of these in use:

anova Return the (sequential) analysis of variance table for a single fitted model or a comparative analysis of variance for multiple fitted models.

confint Return confidence intervals on the coefficients

deviance Return the residual sum of squares (RSS) for the model. (This is a misnomer because the RSS is related to but not exactly the same as the deviance.)

formula Return the model formula.

kappa Return the condition number of the model matrix or an upper bound on its condition number.

logLik Return the value of the log-likelihood at the estimated parameter values.

model.frame Return the model frame to which the model was actually fit.

model.matrix Return the model matrix.

plot Produce some common residual plots for evaluating the model fit.

predict Returns evaluations of the fitted model, and optionally their standard errors, at the observed or newly specified values of the covariates.

residuals Returns the residuals from the fit.

rstandard Returns the "standardized residuals" (to be described later).

rstandard Returns the "Studentized residuals" (to be described later).

simulate Return a matrix of simulated response vectors according to the model assuming that the fitted values of the parameters are the true parameter values.

summary Return a summary of the fitted model

vcov Return the (estimated) variance-covariance matrix of $\widehat{\boldsymbol{\beta}}$ (i.e. the matrix that could be expressed as $s^2(\boldsymbol{X}'\boldsymbol{X})^{-1}$).

Other extractor functions such as `coef` and `fitted` do not have specific methods for class `"lm"` but instead apply the default method to objects of this class.

A model fit by `aov` has class

```
class(av3)

## [1] "aov" "lm"
```

`"aov"` and also class `"lm"`. This means that methods for class `"aov"` will be chosen, if they exist, otherwise methods for class `"lm"` and, finally, the default method.

Specific methods for class `"aov"` are

```
methods(class="aov")

##  [1] coef         coerce       extractAIC   initialize   model.tables print
##  [7] proj         se.contrast  show         slotsFromS3  summary      TukeyHSD
## see '?methods' for accessing help and source code
```

from which we can see that specific methods for the `coef`, `extractAIC`, `print`, `proj` and `summary` generics are available for this class and will be chosen in preference to the `"lm"` or default method. Furthermore there are specific methods for the `model.tables`, `se.contrast` and `TukeyHSD` generics.

4 Simulating linear model fits

It is possible to simulate a large number of replications of linear model fits quite quickly, if you do it carefully. The trick is to realize that, if the expression on the left-hand size of the model formula in `lm()` or `aov()` is a matrix with n rows and N columns then the model is fit to all N of the response vectors simultaneously.

As shown above, one of the methods that can be applied to a fitted model is called `simulate` and it generates such a matrix using the parameter estimates (both the coefficients and the variance, σ^2 of the "random noise") for the simulation. Consider again the model `lm1` with coefficient table

```
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.0050857  0.0078337  0.6492    0.5516
## carb        0.8762857  0.0135345 64.7444 3.409e-07
```

To simulate 10,000 response vectors simulated from this model is fast

```
set.seed(1234321)
system.time(Ylst <- simulate(lm1, 10000))

##    user  system elapsed
##   0.028   0.000   0.026

str(Ylst, 0)
```

```
## 'data.frame': 6 obs. of  10000 variables:
##   [list output truncated]
##  - attr(*, "seed")= int  403 624 264578493 -913911462 183580435 973607224 -22559901..
```

but, unfortunately, it is not a matrix, which is what we want. To get a matrix from a data frame we use

```
str(Ymat <- data.matrix(unname(Ylst)))

##  num [1:6, 1:10000] 0.103 0.281 0.459 0.534 0.628 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:6] "1" "2" "3" "4" ...
##   ..$ : NULL
```

(the reason for using `unname()` is to get rid of the 10,000 column names).

Finally, we fit the model to all 10,000 response vectors simultaneously.

```
system.time(lm1sim <- lm(Ymat ~ 1 + carb, Formaldehyde))

##    user  system elapsed
##   0.016   0.000   0.016
```

which is much faster than you could ever hope to do with a loop.

Now the coefficients are in the form of a 2 by 10,000 matrix

```
str(simcoef <- coef(lm1sim))

##  num [1:2, 1:10000] 0.024168 0.85297 0.013392 0.868404 0.000351 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:2] "(Intercept)" "carb"
##   ..$ : NULL
```

for which we could produce density plots, etc.

To obtain a density plot of the estimated intercepts we use

```
intercepts <- simcoef[1,]
slopes <- simcoef[2,]
```

from which we produce the plots in Fig. 7

We can also check that the mean of the estimated intercept is close to the value used in the simulation (i.e. the estimated intercept in model `lm1`) and standard deviation of the simulated intercepts is close to the standard error of the that estimate.

```
c(mean=mean(intercepts), sd=sd(intercepts))
```

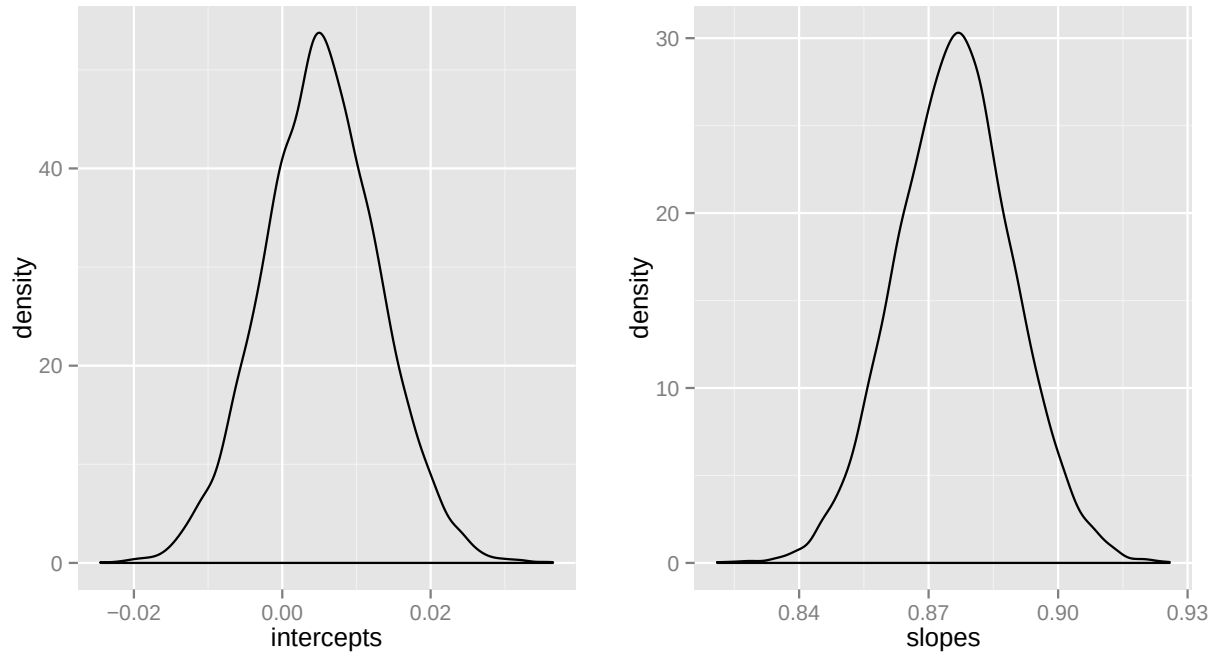

Figure 7: Estimated intercepts and slopes from data simulated according to model `lm1`

```
##        mean          sd
## 0.005210419 0.007801148

c(mean=mean(slopes), sd=sd(slopes))

##       mean         sd
## 0.87606392 0.01350473

printCoefmat(coef(summary(lm1)))

##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) 0.0050857  0.0078337  0.6492    0.5516
## carb        0.8762857  0.0135345 64.7444 3.409e-07
```