Statistics for Omics with R

Material from PH525x series - Biomedical Data Science

Contents

Introduction 2

Pooling experiment data 2

Multiple comparisons with genewise t-tests: 7
- Introduction 7

T-tests 7

Multiple testing 8

Moderated t-test 8
- First, simple t-tests 8
- limma steps 11
- Footnotes 14

Linear Models 14
- The Design Matrix 14
    - Choice of design 15
    - More groups 17
    - More variables 18
    - Releveling 19
    - Where does model.matrix look for the data? 20
    - Continuous variables 20

Linear models in practice 21
- The mouse diet example 21
- The Mathematics Behind lm() 22
    - Examining the coefficients 23
- More complex designs 24
    - Initial visual inspection of the data 24
    - A linear model with one variable 26
    - Examining the estimated coefficients 28
    - A linear model with two variables 28
    - Mathematical representation 30
    - Examining the estimated coefficients 30
    - Contrasting coefficients 32
- Linear Model with Interactions 34
    - Examining the estimated coefficients 37
    - Contrasts 38
    - Differences of differences 38
- Analysis of Variance 39
    - A different specification of the same model 41
    - Examining the estimated coefficients 43
    - Simple contrasts using the contrast package 44
    - Differences of differences when there is no intercept 44

```
## Warning: package 'knitr' was built under R version 3.5.2
```

Introduction

In the following sections we will cover inference in the context of genomics experiments. We apply some of the concepts we have covered in previous sections including t-tests and multiple comparisons. Here we introduce a concept that is particularly important in the analysis of genomics data: the distinction between biological and technical variability.

In general, the variability we observe across biological units, such as individuals, within a population is referred to as *biological*. We refer to the variability we observe across measurements of the same biological unit, such a aliquots from the same biological sample, as *technical*. Because newly developed measurement technologies are common in genomics, technical replicates are used many times to assess experimental data. By generating measurements from samples that are designed to be the same, we are able to measure and assess technical variability. We also use the terminology *biological replicates* and *technical replicates* to refer to samples from which we can measure biological and technical variability respectively.

It is important not to confuse biological and technical variability when performing statistical inference as the interpretation is quite different. For example, when analyzing data from technical replicates, the population is just the one sample from which these come from as opposed to more general population such as healthy humans or control mice. Here we explore this concept with a experiment that was designed to include both technical and biological replicates.

Pooling experiment data

The dataset we will study includes data from gene expression arrays. In this experiment, RNA was extracted from 12 randomly selected mice from two strains. All 24 samples were hybridized to microarrays but we also formed pools, including two pools from with the RNA from all twelve mice from each of the two strains. Other pools were also created, as we will see below, but we will ignore these here.

We will need the following library which you need to install if you have not done so already:

```r
library(devtools)
install_github("genomicsclass/maPooling")
```

We can see the experimental design using the `pData` function. Each row represents a sample and the column are the mice. A 1 in cell i, j indicates that RNA from mouse j was included in sample i. The strain can be identified from the row names (this is not a recommended approach, you can add additional variables to the phenoData to make strain information explicit.)

```r
library(Biobase)
library(maPooling)
data(maPooling)
head(pData(maPooling))
```

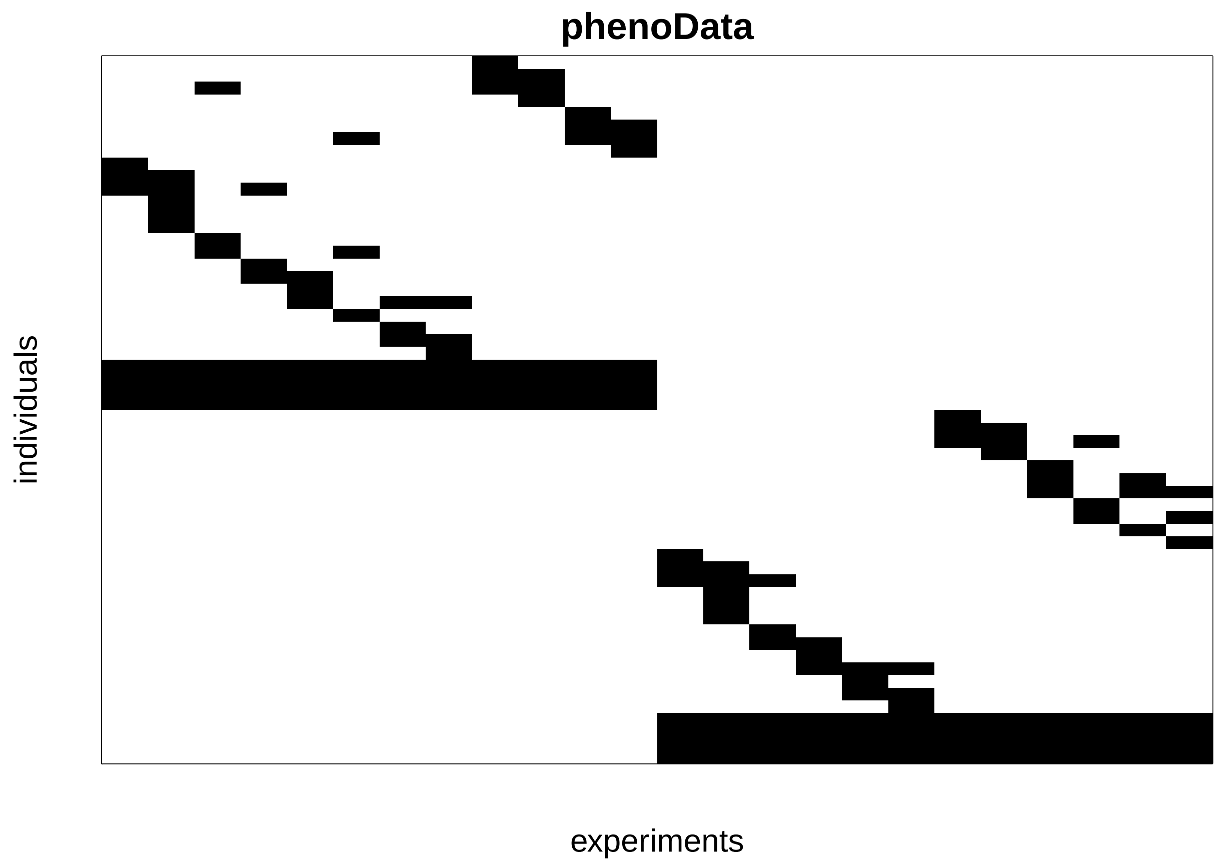
```
##          a2 a3 a4 a5 a6 a7 a8 a9 a10 a11 a12 a14 b2 b3 b5 b6 b8 b9 b10 b11
## a10       0  0  0  0  0  0  0  0   1   0   0   0  0  0  0  0  0  0   0   0
## a10a11    0  0  0  0  0  0  0  0   1   1   0   0  0  0  0  0  0  0   0   0
## a10a11a4  0  0  1  0  0  0  0  0   1   1   0   0  0  0  0  0  0  0   0   0
## a11       0  0  0  0  0  0  0  0   0   1   0   0  0  0  0  0  0  0   0   0
## a12       0  0  0  0  0  0  0  0   0   0   1   0  0  0  0  0  0  0   0   0
## a12a14    0  0  0  0  0  0  0  0   0   0   1   1  0  0  0  0  0  0   0   0
##          b12 b13 b14 b15
## a10        0   0   0   0
## a10a11     0   0   0   0
## a10a11a4   0   0   0   0
## a11        0   0   0   0
## a12        0   0   0   0
## a12a14     0   0   0   0
```

Below we create an image to illustrate which mice were included in which samples:

```r
library(rafalib)
```

```
## Warning: package 'rafalib' was built under R version 3.5.2
```

```r
mypar()
flipt <- function(m) t(m[nrow(m):1,])
myimage <- function(m,...) {
  image(flipt(m),xaxt="n",yaxt="n",...)
}
myimage(as.matrix(pData(maPooling)),col=c("white","black"),
        xlab="experiments",
        ylab="individuals",
        main="phenoData")
```



Note that ultimately we are interested in detecting genes that are differentially expressed between the two strains of mice which we will refer to as strain 0 and 1. We can apply tests to the technical replicates of pooled samples or the data from 12 individual mice. We can identify these pooled samples because all mice from each strain were represented in these samples and thus the sum of the rows of experimental design matrix add up to 12:

```r
data(maPooling)
pd=pData(maPooling)
pooled=which(rowSums(pd)==12)
```

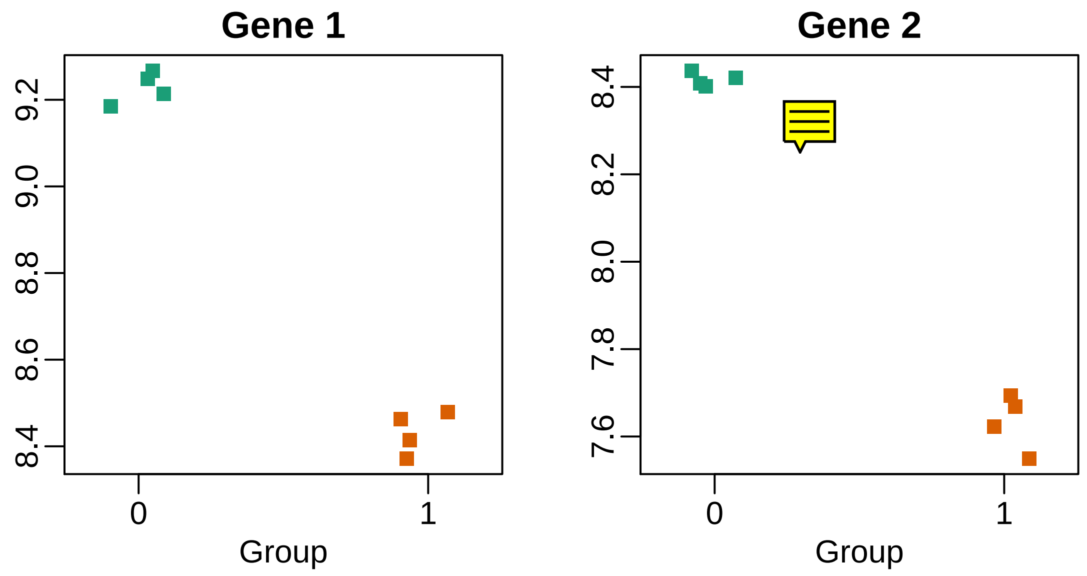
We can determine the strain from the column names:

```
factor(as.numeric(grepl("b",names(pooled))))
```

```
## [1] 0 0 0 0 1 1 1 1
## Levels: 0 1
```

If we compare the mean expression between groups for each gene we find several showing consistent differences. Here are two examples:

```
###look at 2 pre-selected genes for illustration
i=11425;j=11878
pooled_y=exprs(maPooling[,pooled])
pooled_g=factor(as.numeric(grepl("b",names(pooled))))
mypar(1,2)
stripchart(split(pooled_y[i,],pooled_g),vertical=TRUE,method="jitter",col=c(1,2),
           main="Gene 1",xlab="Group",pch=15)
stripchart(split(pooled_y[j,],pooled_g),vertical=TRUE,method="jitter",col=c(1,2),
           main="Gene 2",xlab="Group",pch=15)
```



Note that if we compute a t-test from these values we obtain highly significant results

```
library(genefilter)
```

```
## Warning: replacing previous import 'BiocGenerics::dims' by 'Biobase::dims'
## when loading 'AnnotationDbi'
```

```
pooled_tt=rowttests(pooled_y,pooled_g)
pooled_tt$p.value[i]
```

```
## [1] 2.075617e-07
```

```
pooled_tt$p.value[j]
```

```
## [1] 3.400476e-07
```

But would these results hold up if we selected another 24 mice? Note that the definition for the t-test includes the standard deviations of the populations being compared. Are these quantities measured here?

Observe that what is being replicated here is the experimental protocol. We have created four *technical replicates* for each pooled sample. Gene 1 may be a highly variable gene within strain of mice while Gene 2 a stable one, but we have no way of seeing this, because mouse-to-mouse variability is submerged in the act of pooling.

We also have microarray data for each individual mouse. For each strain we have 12 *biological replicates*. We can find them by looking for rows with just one 1.

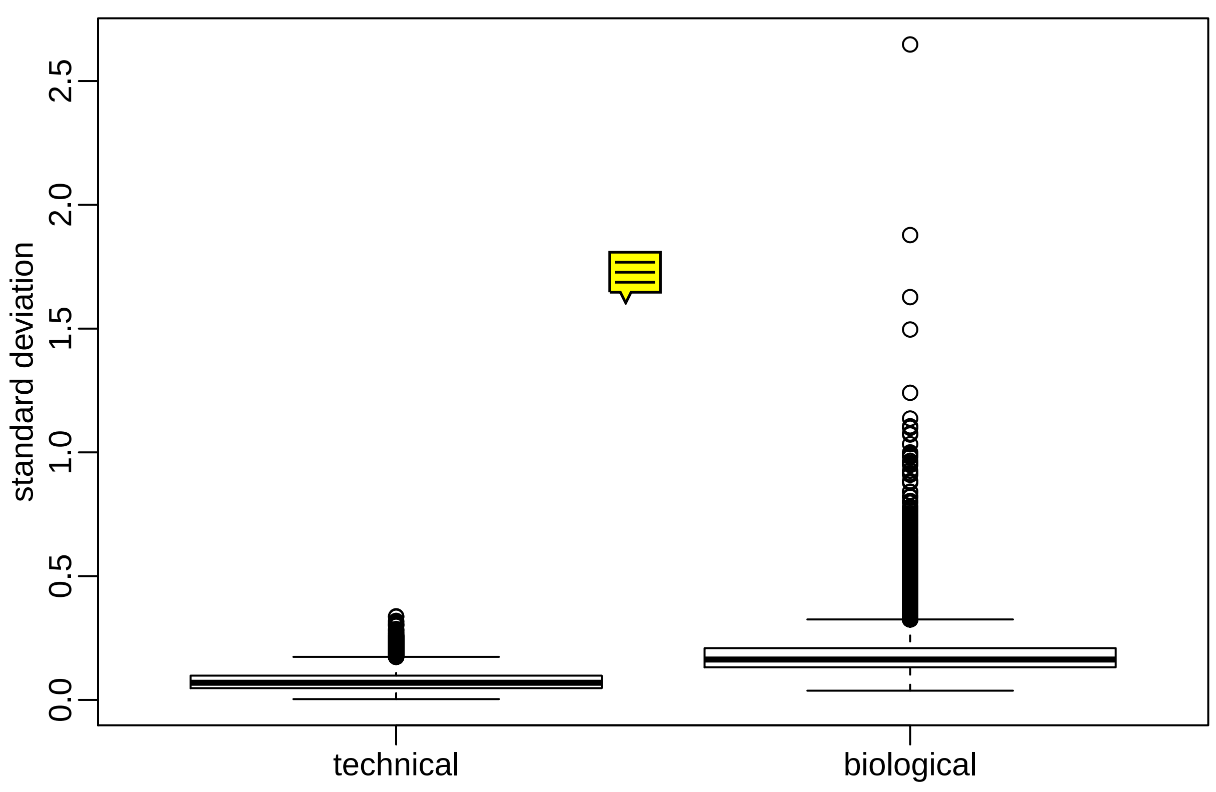
```r
individuals=which(rowSums(pd)==1)
```

It turns out that some technical replicates were included for some individual mice so we remove them to illustrate an analysis with only biological replicates:

```
r   ##remove replicates   individuals=individuals[-grep("tr",names(individuals))]   y=exprs(maPooling)[
g=factor(as.numeric(grepl("b",names(individuals))))
```

We can compute the sample variance for each gene and compare to the standard deviation obtained with the technical replicates.

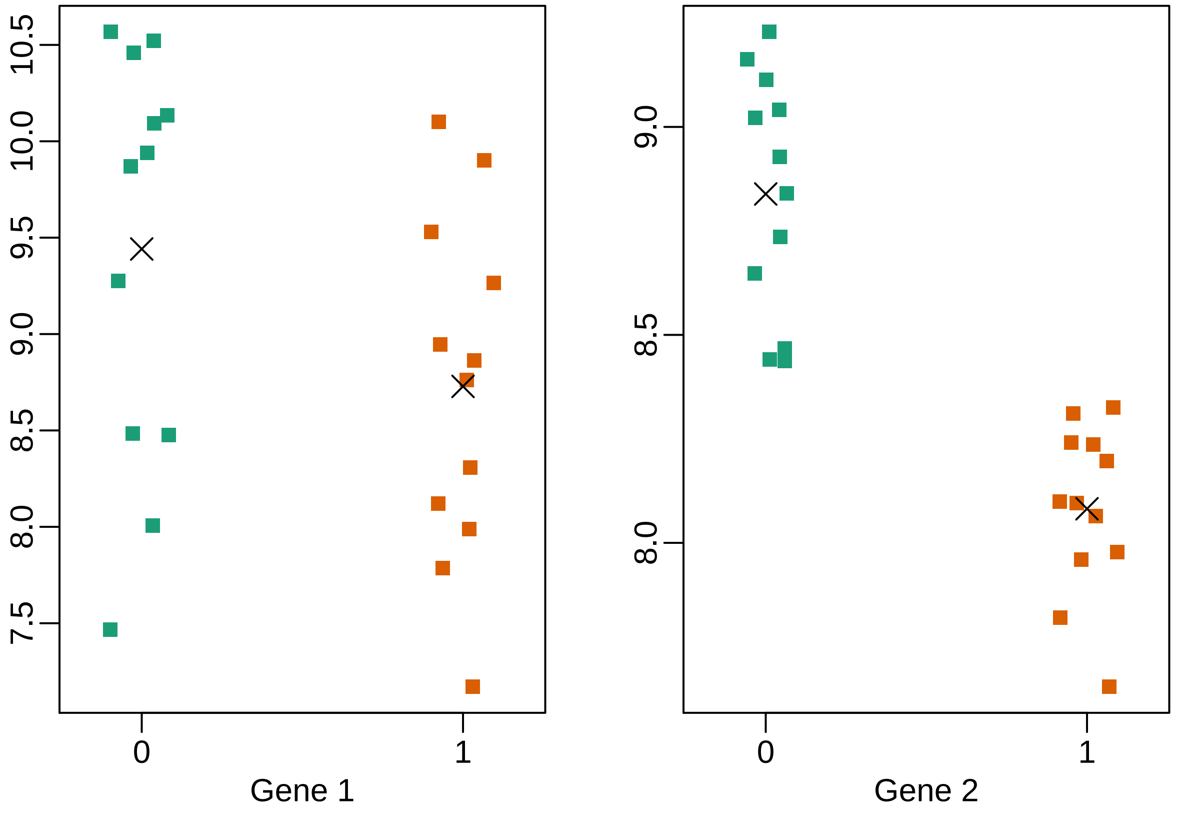
```r
technicalsd <- rowSds(pooled_y[,pooled_g==0])
biologicalsd <- rowSds(y[,g==0])
LIM=range(c(technicalsd,biologicalsd))
mypar(1,1)
boxplot(technicalsd,biologicalsd,names=c("technical","biological"),ylab="standard deviation")
```



Note the biological variance is much larger than the technical variance. And also that the variability of variances is also larger for biological variance. Here are the two genes we showed above but now we show expression values measured on each individual mouse

```r
mypar(1,2)
stripchart(split(y[i,],g),vertical=TRUE,method="jitter",col=c(1,2),xlab="Gene 1",pch=15)
points(c(1,2),tapply(y[i,],g,mean),pch=4,cex=1.5)
```

```
stripchart(split(y[j,],g),vertical=TRUE,method="jitter",col=c(1,2),xlab="Gene 2",pch=15)
points(c(1,2),tapply(y[j,],g,mean),pch=4,cex=1.5)
```



Now the p-values tell a different story

```
library(genefilter)
tt=rowttests(y,g)
tt$p.value[i]
```

```
## [1] 0.0898726
```

```
tt$p.value[j]
```

```
## [1] 1.979172e-07
```

Which of these two genes do we feel more confident reporting as being differentially expressed between strains? If another investigator takes another random sample of mice and tries the same experiment, which gene do you think will be identified? Measuring biological variability is essential if we want our conclusions to be about the strain of mice in general as opposed to the specific mice we have.

An analysis with biological replicates has as a population these two strains of mice. An analysis with technical replicates has as a population the twelve mice we selected and the variability is related to the measurement technology. In science we typically are concerned with populations. As a very practical example, note that if another lab performs this experiment they will have another set of twelve mice and thus inferences about *populations* are more likely to be reproducible.

Multiple comparisons with genewise t-tests:

Introduction

In the previous section, we focused on a pair of genes to illustrate two aspects of variation. One of the genes appeared to have high between-mouse variation that was hidden in the act of pooling samples within strain. When strains were compared on the basis of the pooled data, there was an appearance of a significant strain effect for this gene ($p < 10^{-6}$), but when individual-level data were used to perform the comparison, the strain effect was found to be very weak at best ($p = 0.089$). The lesson is to recognize that the most scientifically compelling questions concern biological variation, which can only be directly measured with good experimental design. Accurate interpretation of origin and size of biological variation requires appropriate statistical analysis.

In this section we will cover inference in the context of genome-scale experiments. There are several serious conceptual problems:

- there are many tests, often at least one test for each one of tens of thousands of features
- each feature (typically a gene) exhibits its own technical and biological variability
- there may be unmeasured or unreported sources of biological variation (such as time of day)
- many features are inherently interrelated, so the tests are not independent

We will apply some of the concepts we have covered in previous sections including t-tests and multiple comparisons;

We start by loading the pooling experiment data

```r
library(Biobase)
library(maPooling)
data(maPooling)
pd=pData(maPooling)
individuals=which(rowSums(pd)==1)
```

And extracting the individual mice as well as their strain

```r
individuals=which(rowSums(pd)==1)
individuals=individuals[-grep("tr",names(individuals))]
y=exprs(maPooling)[,individuals]
g=factor(as.numeric(grepl("b",names(individuals))))
```

T-tests

We can now apply a t-test to each gene using the `rowttest` function in the `genefilter` package

```r
library(genefilter)
tt=rowttests(y,g)
```

Now which genes do we report as statistically significant? For somewhat arbitrary reasons, in science p-values of 0.01 and 0.05 are used as cutoff. In this particular example we get

```r
NsigAt01 = sum(tt$p.value<0.01)
NsigAt01
```

```
## [1] 1578
```

```r
NsigAt05 = sum(tt$p.value<0.05)
NsigAt05
```

```
## [1] 2908
```

Multiple testing

We described multiple testing in detail in course 3. Here we provide a quick summary.

Do we report all the nominally significant genes identified above? Let's explore what happens if we split the first group into two, forcing the null hypothesis to be true

```
set.seed(0)
shuffledIndex <- factor(sample(c(0,1),sum(g==0),replace=TRUE ))
nulltt <- rowttests(y[,g==0],shuffledIndex)
NfalselySigAt01 = sum(nulltt$p.value<0.01)
NfalselySigAt01
```

```
## [1] 79
```

```
NfalselySigAt05 = sum(nulltt$p.value<0.05)
NfalselySigAt05
```

```
## [1] 840
```

If we use the 0.05 cutoff we will be reporting 840 false positives. We have described several ways to adjust for this including the `qvalue` method . After this adjustment we acquire a smaller list of genes.

```
source("http://bioconductor.org/biocLite.R")
```

```
## Bioconductor version 3.7 (BiocInstaller 1.30.0), ?biocLite for help
```

```
## A newer version of Bioconductor is available for this version of R,
##   ?BiocUpgrade for help
```

```
library(qvalue)
qvals = qvalue(tt$p.value)$qvalue
sum(qvals<0.05)
```

```
## [1] 1179
```

```
sum(qvals<0.01)
```

```
## [1] 538
```

And now the null case generates no false positives:

```
library(qvalue)
nullqvals = qvalue(nulltt$p.value)$qvalue
sum(nullqvals<0.05)
```

```
## [1] 0
```

```
sum(nullqvals<0.01)
```

```
## [1] 0
```

This addresses in a fairly general way the problem of inflating significance claims when performing many hypothesis tests at a fixed nominal level of significance.

Moderated t-test

First, simple t-tests

We will now show the difference between using the simple t-test and doing differential expression with the `limma` hierarchical model. The reference is Smyth 2004, listed in the footnotes.

Here we also show the basic steps for performing a `limma` analysis. Note that the `limma` package is very powerful, and has hundreds of pages of documentation which we cannot cover in this course, however we recommend that users wanting to explore further should check out this guide.

We start by loading the spike-in data which was introduced in lecture, which has already been normalized.

```
# biocLite("SpikeInSubset")
library(SpikeInSubset)
```

```
## Loading required package: affy
```

```
data(rma95)
fac <- factor(rep(1:2,each=3))
```

We can now perform simple t-tests using the `rowttests` function in the `genefilter` package:

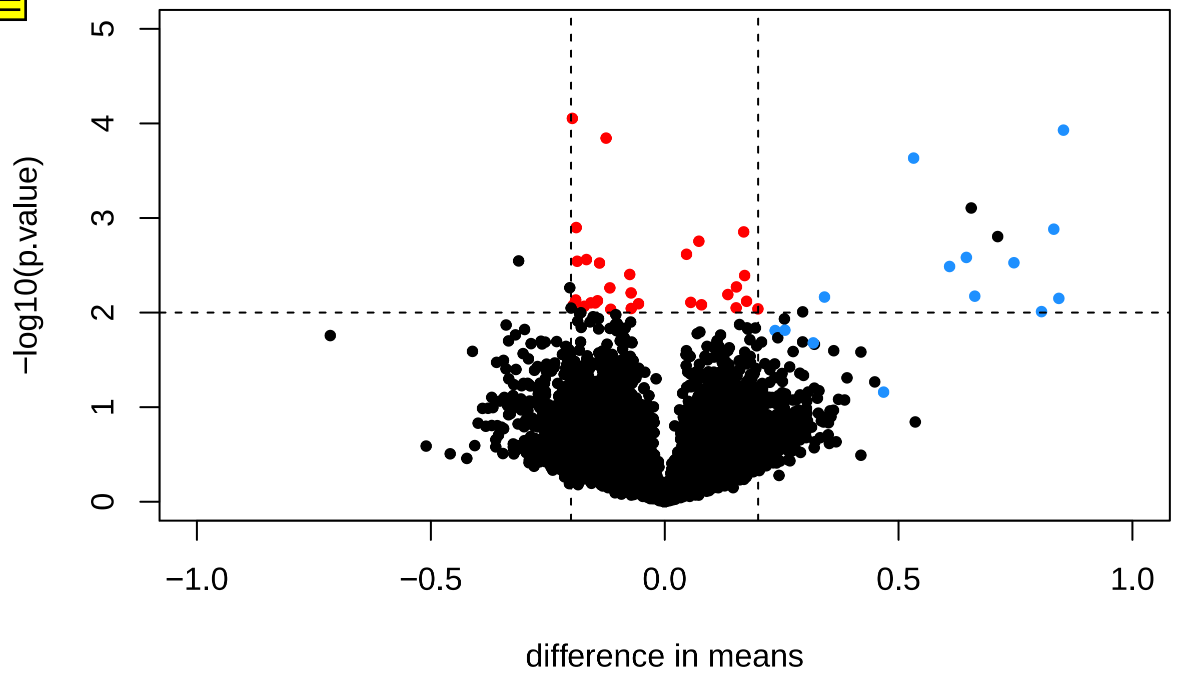
```
library(genefilter)
rtt <- rowttests(exprs(rma95),fac)
```

We will define colors depending on whether the p-value is small, the absolute difference in means is large, and whether the feature is a spike-in value.

```
mask <- with(rtt, abs(dm) < .2 & p.value < .01)
spike <- rownames(rma95) %in% colnames(pData(rma95))
cols <- ifelse(mask,"red",ifelse(spike,"dodgerblue","black"))
```

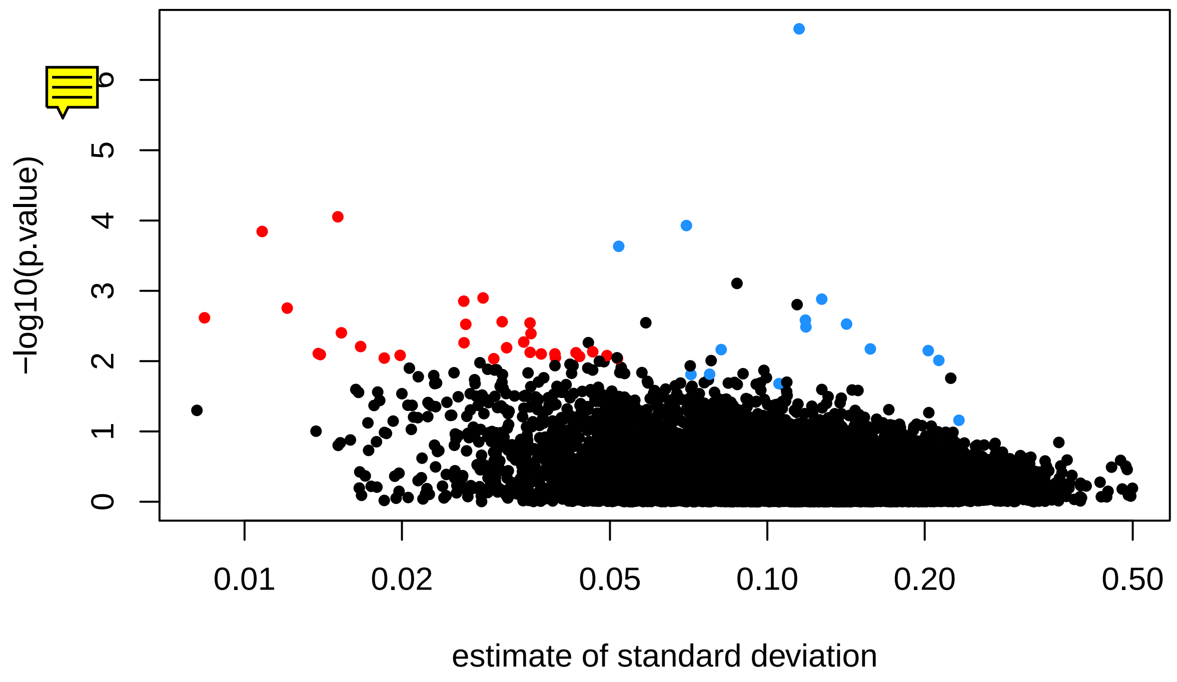
We now plot the results, using the colors defined above. We multiply the `dm` by `-1`, because we are interested in the difference from the second group to the first (this is the difference used by `lm` and the `limma` package by default). The spike-in genes are in blue, which have mostly small p-value and large difference in means. The red points indicate genes which have small p-values but also small differences in means. We will see how these points change after using `limma`.

```
with(rtt, plot(-dm, -log10(p.value), cex=.8, pch=16,
     xlim=c(-1,1), ylim=c(0,5),
     xlab="difference in means",
     col=cols))
abline(h=2,v=c(-.2,.2), lty=2)
```



Note that the red genes have mostly low estimates of standard deviation.

```r
rtt$s <- apply(exprs(rma95), 1, function(row) sqrt(.5 * (var(row[1:3]) + var(row[4:6]))))
with(rtt, plot(s, -log10(p.value), cex=.8, pch=16,
              log="x",xlab="estimate of standard deviation",
              col=cols))
```



limma steps

The following three steps perform the basic `limma` analysis. We specify `coef=2` because we are interested in the difference between groups, not the intercept.

```
library(limma)
```

```
##
## Attaching package: 'limma'

## The following object is masked from 'package:BiocGenerics':
##
##     plotMA
```

```
fit <- lmFit(rma95, design=model.matrix(~ fac))
colnames(coef(fit))
```

```
## [1] "(Intercept)" "fac2"
```

```
fit <- eBayes(fit)
tt <- topTable(fit, coef=2)
tt
```

```
##                 logFC   AveExpr          t      P.Value    adj.P.Val
## 1708_at    -7.0610613  7.945276 -73.529269 7.816370e-17 9.868948e-13
## 36202_at    0.8525527  9.373033   9.975114 4.935683e-07 3.115897e-03
## 36311_at    0.8318298  8.564315   8.363252 3.017008e-06 1.269758e-02
## 33264_at    0.7118997  4.918953   7.434888 9.666328e-06 2.706595e-02
## 32660_at    0.6554022  8.680132   7.356180 1.071834e-05 2.706595e-02
## 38734_at  0.7467142  6.255772   7.185131 1.345115e-05 2.830571e-02
## 1024_at   0.8426550  9.697281   6.730664 2.503461e-05 4.400123e-02
## 36085_at  0.6449402 12.193130   6.653830 2.787976e-05 4.400123e-02
## 33818_at  0.5321749 12.285643   6.454504 3.699480e-05 5.189960e-02
## 39058_at  0.6090625  7.534532   6.278815 4.767986e-05 5.687699e-02
##                 B
## 1708_at  8.646866
## 36202_at 4.587736
## 36311_at 3.567790
## 33264_at 2.835849
## 32660_at 2.768151
## 38734_at 2.617789
## 1024_at  2.195944
## 36085_at 2.121308
## 33818_at 1.923063
## 39058_at 1.742696
```

`topTable` will return the top genes ranked by whichever value you define. You can also ask topTable to return all the values, sorted by `"none"`. Note that a column automatically is included which gives the *adjusted p-values* for each gene. By default the method of Benjamini-Hochberg is used, by calling the `p.adjust` function.

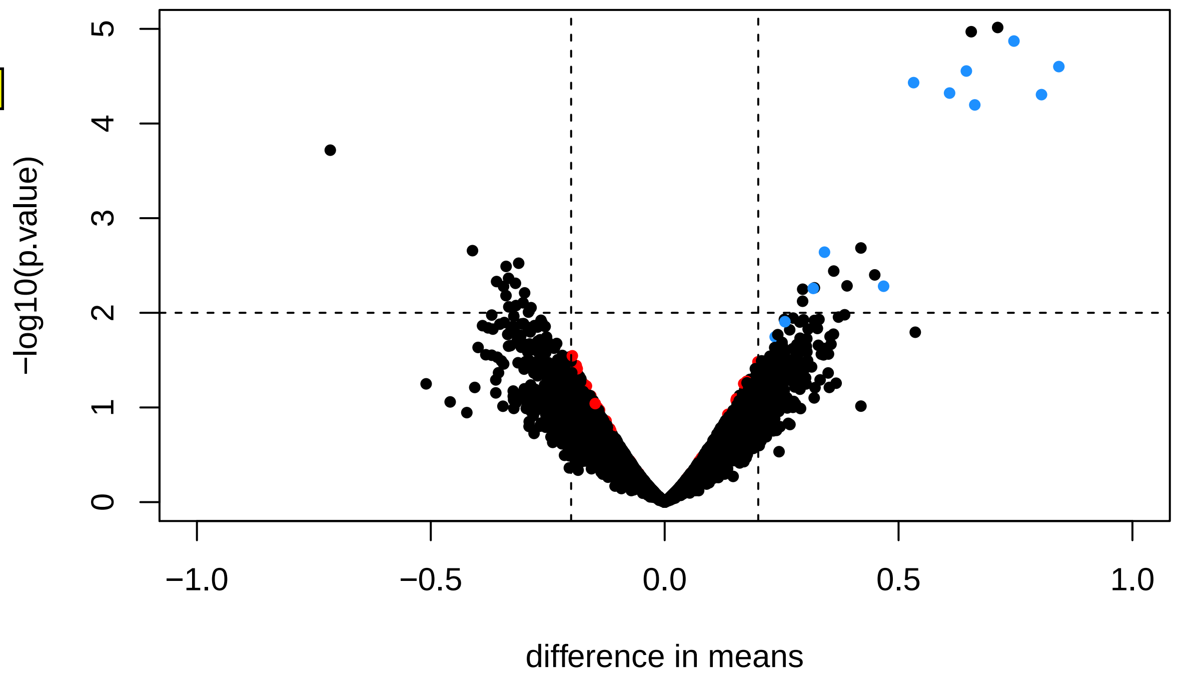
```
# ?topTable
dim(topTable(fit, coef=2, number=Inf, sort.by="none"))
```

```
## [1] 12626     6
```

```
# ?p.adjust
```

Here we will compare the previous volcano plot with the `limma` results. Note that the red points are now all under the line where `-log10(p.value)` is equal to 2. Also, the blue points which represent real differences have p-values which are even higher than before.

```
limmares <- data.frame(dm=coef(fit)[,"fac2"], p.value=fit$p.value[,"fac2"])
with(limmares, plot(dm, -log10(p.value),cex=.8, pch=16,
     col=cols,xlab="difference in means",
     xlim=c(-1,1), ylim=c(0,5)))
abline(h=2,v=c(-.2,.2), lty=2)
```



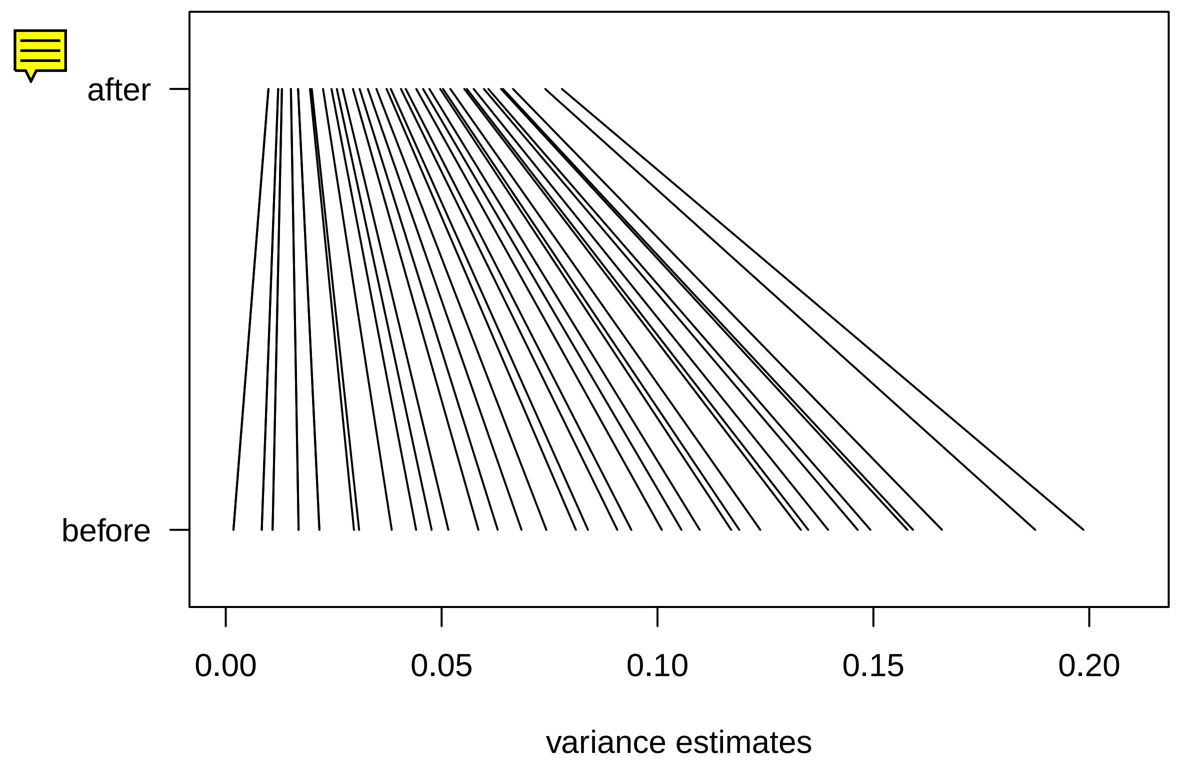
Finally, we will construct a plot which shows how `limma` shrinks the variance estimates towards a common value, eliminating false positives which might arise from too-low estimates of variance.

Here we pick, for each of 40 bins of different variance estimates, a single gene which falls in that bin. We remove bins which do not have any such genes.

```
n <- 40
qs <- seq(from=0,to=.2,length=n)
idx <- sapply(seq_len(n),function(i) which(as.integer(cut(rtt$s^2,qs)) == i)[1])
idx <- idx[!is.na(idx)]
```

Now we will plot a line, from the initial estimate of variance for these genes to the estimate after running `limma`.

```
par(mar=c(5,5,2,2))
plot(1,1,xlim=c(0,.21),ylim=c(0,1),type="n",
     xlab="variance estimates",ylab="",yaxt="n")
axis(2,at=c(.1,.9),c("before","after"),las=2)
segments((rtt$s^2)[idx],rep(.1,n),
         fit$s2.post[idx],rep(.9,n))
```



Footnotes

Smyth GK, "Linear models and empirical bayes methods for assessing differential expression in microarray experiments". Stat Appl Genet Mol Biol. 2004 http://www.ncbi.nlm.nih.gov/pubmed/16646809

Linear Models

Many of the models we use in data analysis can be presented using matrix algebra. We refer to these types of models as *linear models*. "Linear" here does not refer to lines, but rather to linear combinations. The representations we describe are convenient because we can write models more succinctly and we have the matrix algebra mathematical machinery to facilitate computation. In this chapter, we will describe in some detail how we use matrix algebra to represent and fit.

In this book, we focus on linear models that represent dichotomous groups: treatment versus control, for example. The effect of diet on mice weights is an example of this type of linear model. Here we describe slightly more complicated models, but continue to focus on dichotomous variables.

As we learn about linear models, we need to remember that we are still working with random variables. This means that the estimates we obtain using linear models are also random variables. Although the mathematics is more complex, the concepts we learned in previous chapters apply here. We begin with some exercises to review the concept of random variables in the context of linear models.

The Design Matrix

Here we will show how to use the two R functions, `formula` and `model.matrix`, in order to produce *design matrices* (also known as *model matrices*) for a variety of linear models. For example, in the mouse diet examples we wrote the model as

$$Y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, i = 1, \dots, N$$

with Y_i the weights and x_i equal to 1 only when mouse i receives the high fat diet. We use the term *experimental unit* to N different entities from which we obtain a measurement. In this case, the mice are the experimental units.

This is the type of variable we will focus on in this chapter. We call them *indicator variables* since they simply indicate if the experimental unit had a certain characteristic or not. We can use linear algebra to represent this model:

$$\mathbf{Y} = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix}, \mathbf{X} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_N \end{pmatrix}, \boldsymbol{\beta} = \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} \text{ and } \boldsymbol{\varepsilon} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

as:

$$\begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_N \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \\ 1 & x_N \end{pmatrix} \begin{pmatrix} \beta_0 \\ \beta_1 \end{pmatrix} + \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_N \end{pmatrix}$$

or simply:

$$\mathbf{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$$

The design matrix is the matrix $\mathbf{X}$.

Once we define a design matrix, we are ready to find the least squares estimates. We refer to this as *fitting the model.* For fitting linear models in R, we will directly provide a *formula* to the `lm` function. In this script, we will use the `model.matrix` function, which is used internally by the `lm` function. This will help us to connect the R `formula` with the matrix $\mathbf{X}$. It will therefore help us interpret the results from `lm`.

Choice of design

The choice of design matrix is a critical step in linear modeling since it encodes which coefficients will be fit in the model, as well as the inter-relationship between the samples. A common misunderstanding is that the choice of design follows straightforward from a description of which samples were included in the experiment. This is not the case. The basic information about each sample (whether control or treatment group, experimental batch, etc.) does not imply a single 'correct' design matrix. The design matrix additionally encodes various assumptions about how the variables in $\mathbf{X}$ explain the observed values in $\mathbf{Y}$, on which the investigator must decide.

For the examples we cover here, we use linear models to make comparisons between different groups. Hence, the design matrices that we ultimately work with will have at least two columns: an *intercept* column, which consists of a column of 1's, and a second column, which specifies which samples are in a second group. In this case, two coefficients are fit in the linear model: the intercept, which represents the population average of the first group, and a second coefficient, which represents the difference between the population averages of the second group and the first group. The latter is typically the coefficient we are interested in when we are performing statistical tests: we want to know if there is a difference between the two groups.

We encode this experimental design in R with two pieces. We start with a formula with the tilde symbol `~`. This means that we want to model the observations using the variables to the right of the tilde. Then we put the name of a variable, which tells us which samples are in which group.

Let's try an example. Suppose we have two groups, control and high fat diet, with two samples each. For illustrative purposes, we will code these with 1 and 2 respectively. We should first tell R that these values should not be interpreted numerically, but as different levels of a *factor*. We can then use the paradigm `~ group` to, say, model on the variable `group`.

```
group <- factor( c(1,1,2,2) )
model.matrix(~ group)
```

```
##   (Intercept) group2
## 1           1      0
## 2           1      0
## 3           1      1
## 4           1      1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

(Don't worry about the `attr` lines printed beneath the matrix. We won't be using this information.)

What about the `formula` function? We don't have to include this. By starting an expression with `~`, it is equivalent to telling R that the expression is a formula:

```
model.matrix(formula(~ group))
```

```
##   (Intercept) group2
## 1           1      0
## 2           1      0
## 3           1      1
## 4           1      1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

What happens if we don't tell R that `group` should be interpreted as a factor?

```
group <- c(1,1,2,2)
model.matrix(~ group)
```

```
##   (Intercept) group
## 1           1     1
## 2           1     1
## 3           1     2
## 4           1     2
## attr(,"assign")
## [1] 0 1
```

This is **not** the design matrix we wanted, and the reason is that we provided a numeric variable as opposed to an *indicator* to the `formula` and `model.matrix` functions, without saying that these numbers actually referred to different groups. We want the second column to have only 0 and 1, indicating group membership.

A note about factors: the names of the levels are irrelevant to `model.matrix` and `lm`. All that matters is the order. For example:

```r
group <- factor(c("control","control","highfat","highfat"))
model.matrix(~ group)
```

```
##   (Intercept) grouphighfat
## 1           1            0
## 2           1            0
## 3           1            1
## 4           1            1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

produces the same design matrix as our first code chunk.

More groups

Using the same formula, we can accommodate modeling more groups. Suppose we have a third diet:

```r
group <- factor(c(1,1,2,2,3,3))
model.matrix(~ group)
```

```
##   (Intercept) group2 group3
## 1           1      0      0
## 2           1      0      0
## 3           1      1      0
## 4           1      1      0
## 5           1      0      1
## 6           1      0      1
## attr(,"assign")
## [1] 0 1 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

Now we have a third column which specifies which samples belong to the third group.

An alternate formulation of design matrix is possible by specifying `+ 0` in the formula:

```r
group <- factor(c(1,1,2,2,3,3))
model.matrix(~ group + 0)
```

```
##   group1 group2 group3
## 1      1      0      0
## 2      1      0      0
## 3      0      1      0
## 4      0      1      0
## 5      0      0      1
## 6      0      0      1
## attr(,"assign")
## [1] 1 1 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

This group now fits a separate coefficient for each group. We will explore this design in more depth later on.

More variables

We have been using a simple case with just one variable (diet) as an example. In the life sciences, it is quite common to perform experiments with more than one variable. For example, we may be interested in the effect of diet and the difference in sexes. In this case, we have four possible groups:

```
diet <- factor(c(1,1,1,1,2,2,2,2))
sex <- factor(c("f","f","m","m","f","f","m","m"))
table(diet,sex)
```

```
##     sex
## diet f m
##    1 2 2
##    2 2 2
```

If we assume that the diet effect is the same for males and females (this is an assumption), then our linear model is:

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \varepsilon_i$$

To fit this model in R, we can simply add the additional variable with a + sign in order to build a design matrix which fits based on the information in additional variables:

```
diet <- factor(c(1,1,1,1,2,2,2,2))
sex <- factor(c("f","f","m","m","f","f","m","m"))
model.matrix(~ diet + sex)
```

```
##   (Intercept) diet2 sexm
## 1           1     0    0
## 2           1     0    0
## 3           1     0    1
## 4           1     0    1
## 5           1     1    0
## 6           1     1    0
## 7           1     1    1
## 8           1     1    1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$diet
## [1] "contr.treatment"
##
## attr(,"contrasts")$sex
## [1] "contr.treatment"
```

The design matrix includes an intercept, a term for `diet` and a term for `sex`. We would say that this linear model accounts for differences in both the group and condition variables. However, as mentioned above, the model assumes that the diet effect is the same for both males and females. We say these are an *additive* effect. For each variable, we add an effect regardless of what the other is. Another model is possible here, which fits an additional term and which encodes the potential interaction of group and condition variables. We will cover interaction terms in depth in a later script.

The interaction model can be written in either of the following two formulas:

```
model.matrix(~ diet + sex + diet:sex)
```

or

```
model.matrix(~ diet*sex)
```

```
##   (Intercept) diet2 sexm diet2:sexm
## 1           1     0    0          0
## 2           1     0    0          0
## 3           1     0    1          0
## 4           1     0    1          0
## 5           1     1    0          0
## 6           1     1    0          0
## 7           1     1    1          1
## 8           1     1    1          1
## attr(,"assign")
## [1] 0 1 2 3
## attr(,"contrasts")
## attr(,"contrasts")$diet
## [1] "contr.treatment"
##
## attr(,"contrasts")$sex
## [1] "contr.treatment"
```

Releveling

The level which is chosen for the *reference level* is the level which is contrasted against. By default, this is simply the first level alphabetically. We can specify that we want group 2 to be the reference level by either using the `relevel` function:

```
group <- factor(c(1,1,2,2))
group <- relevel(group, "2")
model.matrix(~ group)
```

```
##   (Intercept) group1
## 1           1      1
## 2           1      1
## 3           1      0
## 4           1      0
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

or by providing the levels explicitly in the `factor` call:

```
group <- factor(group, levels=c("1","2"))
model.matrix(~ group)
```

```
##   (Intercept) group2
## 1           1      0
## 2           1      0
## 3           1      1
## 4           1      1
## attr(,"assign")
## [1] 0 1
## attr(,"contrasts")
## attr(,"contrasts")$group
## [1] "contr.treatment"
```

Where does model.matrix look for the data?

The `model.matrix` function will grab the variable from the R global environment, unless the data is explicitly provided as a data frame to the `data` argument:

```
group <- 1:4
model.matrix(~ group, data=data.frame(group=5:8))
```

```
##   (Intercept) group
## 1           1     5
## 2           1     6
## 3           1     7
## 4           1     8
## attr(,"assign")
## [1] 0 1
```

Note how the R global environment variable `group` is ignored.

Continuous variables

In this chapter, we focus on models based on indicator values. In certain designs, however, we will be interested in using numeric variables in the design formula, as opposed to converting them to factors first. For example, in the falling object example, time was a continuous variable in the model and time squared was also included:

```
tt <- seq(0,3.4,len=4)
model.matrix(~ tt + I(tt^2))
```

```
##   (Intercept)       tt   I(tt^2)
## 1           1 0.000000  0.000000
## 2           1 1.133333  1.284444
## 3           1 2.266667  5.137778
## 4           1 3.400000 11.560000
## attr(,"assign")
## [1] 0 1 2
```

The `I` function above is necessary to specify a mathematical transformation of a variable. For more details, see the manual page for the `I` function by typing `?I`.

In the life sciences, we could be interested in testing various dosages of a treatment, where we expect a specific relationship between a measured quantity and the dosage, e.g. 0 mg, 10 mg, 20 mg.

The assumptions imposed by including continuous data as variables are typically hard to defend and motivate than the indicator function variables. Whereas the indicator variables simply assume a different mean between two groups, continuous variables assume a very specific relationship between the outcome and predictor variables.

In cases like the falling object, we have the theory of gravitation supporting the model. In the father-son height example, because the data is bivariate normal, it follows that there is a linear relationship if we condition. However, we find that continuous variables are included in linear models without justification to "adjust" for variables such as age. We highly discourage this practice unless the data support the model being used.

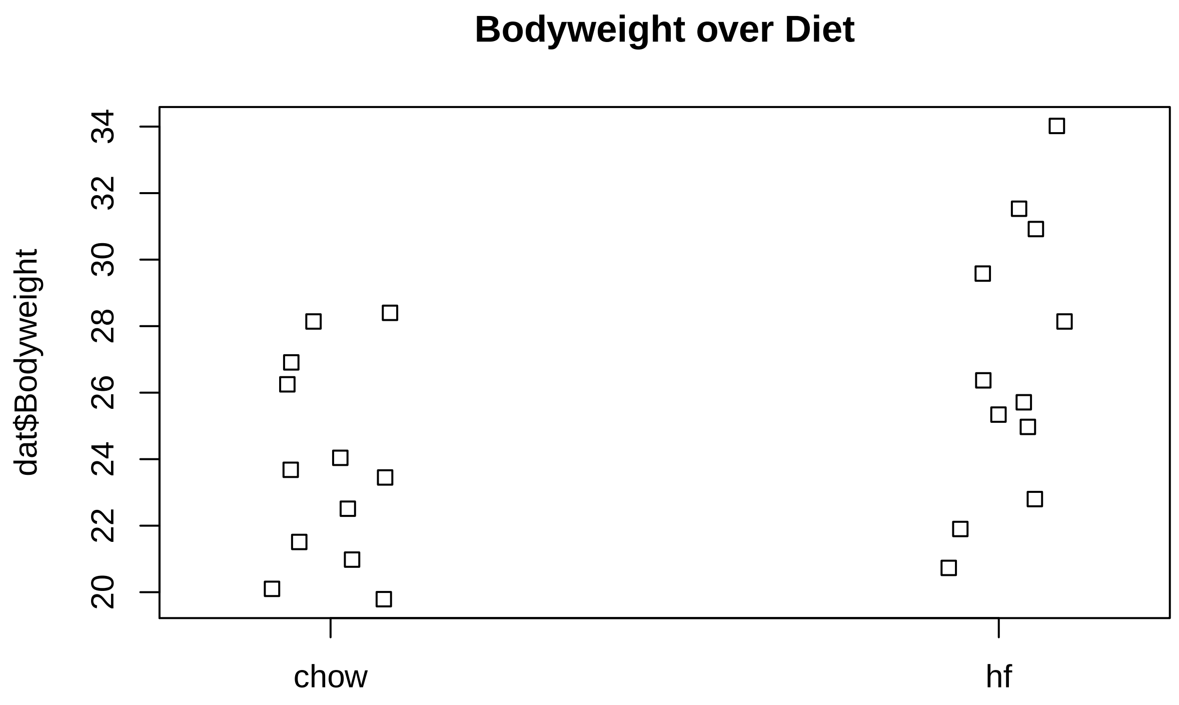


Figure 1: Mice bodyweights stratified by diet.

Linear models in practice

The mouse diet example

We will demonstrate how to analyze the high fat diet data using linear models instead of directly applying a t-test. We will demonstrate how ultimately these two approaches are equivalent.

We start by reading in the data and creating a quick stripchart:

```
## Warning: package 'downloader' was built under R version 3.5.2
```

```
dat <- read.csv("femaleMiceWeights.csv") ##previously downloaded
stripchart(dat$Bodyweight ~ dat$Diet, vertical=TRUE, method="jitter",
           main="Bodyweight over Diet")
```

We can see that the high fat diet group appears to have higher weights on average, although there is overlap between the two samples.

For demonstration purposes, we will build the design matrix $\mathbf{X}$ using the formula `~ Diet`. The group with the 1's in the second column is determined by the level of `Diet` which comes second; that is, the non-reference level.

```
levels(dat$Diet)
```

```
## [1] "chow" "hf"
```

```
X <- model.matrix(~ Diet, data=dat)
head(X)
```

```
##   (Intercept) Diethf
## 1           1      0
## 2           1      0
## 3           1      0
## 4           1      0
## 5           1      0
## 6           1      0
```

The Mathematics Behind lm()

Before we use our shortcut for running linear models, `lm`, we want to review what will happen internally. Inside of `lm`, we will form the design matrix $\mathbf{X}$ and calculate the $\boldsymbol{\beta}$, which minimizes the sum of squares using the previously described formula. The formula for this solution is:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

We can calculate this in R using our matrix multiplication operator `%*%`, the inverse function `solve`, and the transpose function `t`.

```
Y <- dat$Bodyweight
X <- model.matrix(~ Diet, data=dat)
solve(t(X) %*% X) %*% t(X) %*% Y
```

```
##                  [,1]
## (Intercept) 23.813333
## Diethf       3.020833
```

These coefficients are the average of the control group and the difference of the averages:

```
s <- split(dat$Bodyweight, dat$Diet)
mean(s[["chow"]])
```

```
## [1] 23.81333
```

```
mean(s[["hf"]]) - mean(s[["chow"]])
```

```
## [1] 3.020833
```

Finally, we use our shortcut, `lm`, to run the linear model:

```
fit <- lm(Bodyweight ~ Diet, data=dat)
summary(fit)
```

```
##
## Call:
## lm(formula = Bodyweight ~ Diet, data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1042 -2.4358 -0.4138  2.8335  7.1858
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   23.813      1.039  22.912   <2e-16 ***
## Diethf         3.021      1.470   2.055   0.0519 .
```

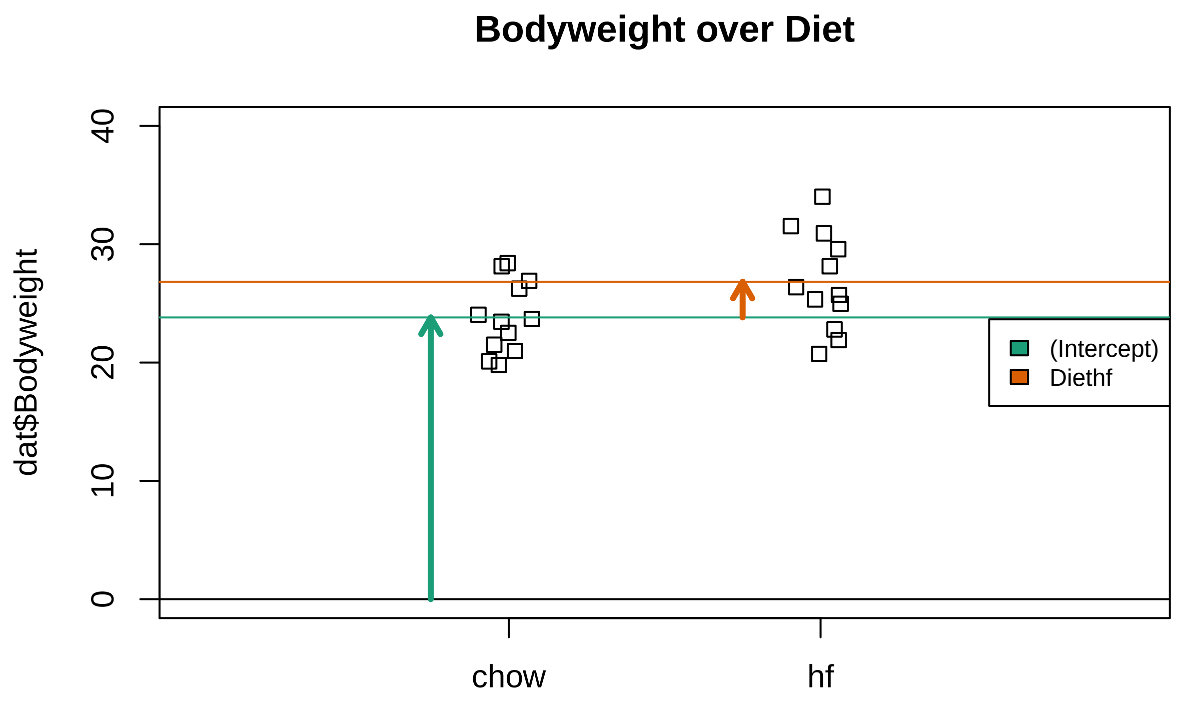


Figure 2: Estimated linear model coefficients for bodyweight data illustrated with arrows.

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.6 on 22 degrees of freedom
## Multiple R-squared:  0.1611, Adjusted R-squared:  0.1229
## F-statistic: 4.224 on 1 and 22 DF,  p-value: 0.05192
```

```
(coefs <- coef(fit))
```

```
## (Intercept)      Diethf
##   23.813333    3.020833
```

Examining the coefficients

The following plot provides a visualization of the meaning of the coefficients with colored arrows (code not shown):

```
## Warning: package 'RColorBrewer' was built under R version 3.5.2
```

To make a connection with material presented earlier, this simple linear model is actually giving us the same result (the t-statistic and p-value) for the difference as a specific kind of t-test. This is the t-test between two groups with the assumption that the population standard deviation is the same for both groups. This was encoded into our linear model when we assumed that the errors ε were all equally distributed.

Although in this case the linear model is equivalent to a t-test, we will soon explore more complicated designs, where the linear model is a useful extension. Below we demonstrate that one does in fact get the exact same results:

Our `lm` estimates were:

```r
summary(fit)$coefficients
```

```
##              Estimate Std. Error   t value     Pr(>|t|)
## (Intercept) 23.813333   1.039353 22.911684 7.642256e-17
## Diethf       3.020833   1.469867  2.055174 5.192480e-02
```

And the t-statistic is the same:

```r
ttest <- t.test(s[["hf"]], s[["chow"]], var.equal=TRUE)
summary(fit)$coefficients[2,3]
```

```
## [1] 2.055174
```

```r
ttest$statistic
```

```
##        t
## 2.055174
```

More complex designs

As a running example to learn about more complex linear models, we will be using a dataset which compares the different frictional coefficients on the different legs of a spider. Specifically, we will be determining whether more friction comes from a pushing or pulling motion of the leg. The original paper from which the data was provided is:

Jonas O. Wolff & Stanislav N. Gorb, Radial arrangement of Janus-like setae permits friction control in spiders, Scientific Reports, 22 January 2013.

The abstract of the paper says,

> The hunting spider Cupiennius salei (Arachnida, Ctenidae) possesses hairy attachment pads (claw tufts) at its distal legs, consisting of directional branched setae… Friction of claw tufts on smooth glass was measured to reveal the functional effect of seta arrangement within the pad. Figure 1 includes some pretty cool electron microscope images of the tufts. We are interested in the comparisons in Figure 4, where the pulling and pushing motions are compared for different leg pairs (for a diagram of pushing and pulling see the top of Figure 3).

We include the data in our dagdata package and can download it from here.

```r
spider <- read.csv("spider_wolff_gorb_2013.csv", skip=1)
```

Initial visual inspection of the data

Each measurement comes from one of our legs while it is either pushing or pulling. So we have two variables:

```r
table(spider$leg,spider$type)
```

```
##
##      pull push
##   L1   34   34
##   L2   15   15
##   L3   52   52
##   L4   40   40
```

We can make a boxplot summarizing the measurements for each of the eight pairs. This is similar to Figure 4 of the original paper:

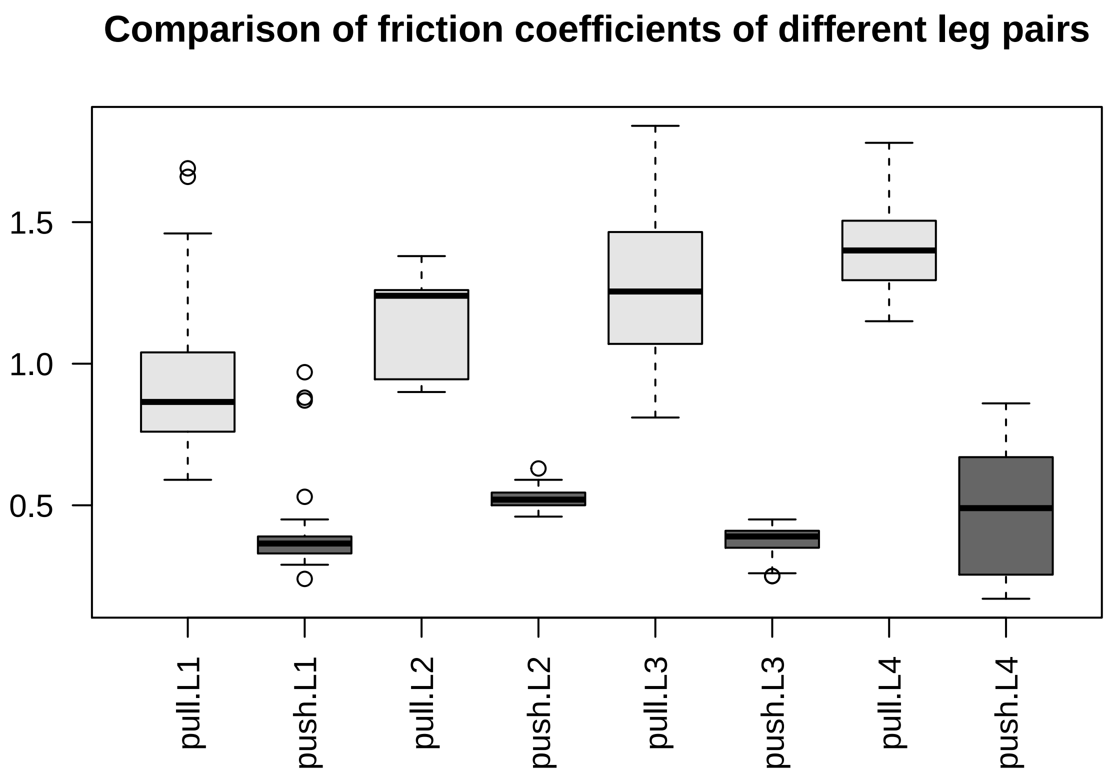


Figure 3: Comparison of friction coefficients of spiders' different leg pairs. The friction coefficient is calculated as the ratio of two forces (see paper Methods) so it is unitless.

```
boxplot(spider$friction ~ spider$type * spider$leg,
        col=c("grey90","grey40"), las=2,
        main="Comparison of friction coefficients of different leg pairs")
```

What we can immediately see are two trends:

- The pulling motion has higher friction than the pushing motion.
- The leg pairs to the back of the spider (L4 being the last) have higher pulling friction.

Another thing to notice is that the groups have different spread around their average, what we call *within-group variance*. This is somewhat of a problem for the kinds of linear models we will explore below, since we will be assuming that around the population average values, the errors ε_i are distributed identically, meaning the same variance within each group. The consequence of ignoring the different variances for the different groups is that comparisons between those groups with small variances will be overly "conservative" (because the overall estimate of variance is larger than an estimate for just these groups), and comparisons between those groups with large variances will be overly confident. If the spread is related to the range of friction, such that groups with large friction values also have larger spread, a possibility is to transform the data with a function such as the `log` or `sqrt`. This looks like it could be useful here, since three of the four push groups (L1, L2, L3) have the smallest friction values and also the smallest spread.

Some alternative tests for comparing groups without transforming the values first include: t-tests without the equal variance assumption using a "Welch" or "Satterthwaite approximation", or the Wilcoxon rank sum test mentioned previously. However here, for simplicity of illustration, we will fit a model that assumes equal variance and shows the different kinds of linear model designs using this dataset, setting aside the issue of different within-group variances.

A linear model with one variable

To remind ourselves how the simple two-group linear model looks, we will subset the data to include only the L1 leg pair, and run `lm`:

```
spider.sub <- spider[spider$leg == "L1",]
fit <- lm(friction ~ type, data=spider.sub)
summary(fit)
```

```
##
## Call:
## lm(formula = friction ~ type, data = spider.sub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.33147 -0.10735 -0.04941 -0.00147  0.76853
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.92147    0.03827  24.078  < 2e-16 ***
## typepush    -0.51412    0.05412  -9.499  5.7e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2232 on 66 degrees of freedom
## Multiple R-squared:  0.5776, Adjusted R-squared:  0.5711
## F-statistic: 90.23 on 1 and 66 DF,  p-value: 5.698e-14
```

```
(coefs <- coef(fit))
```

```
## (Intercept)    typepush
##   0.9214706  -0.5141176
```

These two estimated coefficients are the mean of the pull observations (the first estimated coefficient) and the difference between the means of the two groups (the second coefficient). We can show this with R code:

```
s <- split(spider.sub$friction, spider.sub$type)
mean(s[["pull"]])
```

```
## [1] 0.9214706
```

```
mean(s[["push"]]) - mean(s[["pull"]])
```

```
## [1] -0.5141176
```

We can form the design matrix, which was used inside `lm`:

```
X <- model.matrix(~ type, data=spider.sub)
colnames(X)
```

```
## [1] "(Intercept)" "typepush"
```

```
head(X)
```

```
##   (Intercept) typepush
## 1           1        0
## 2           1        0
```

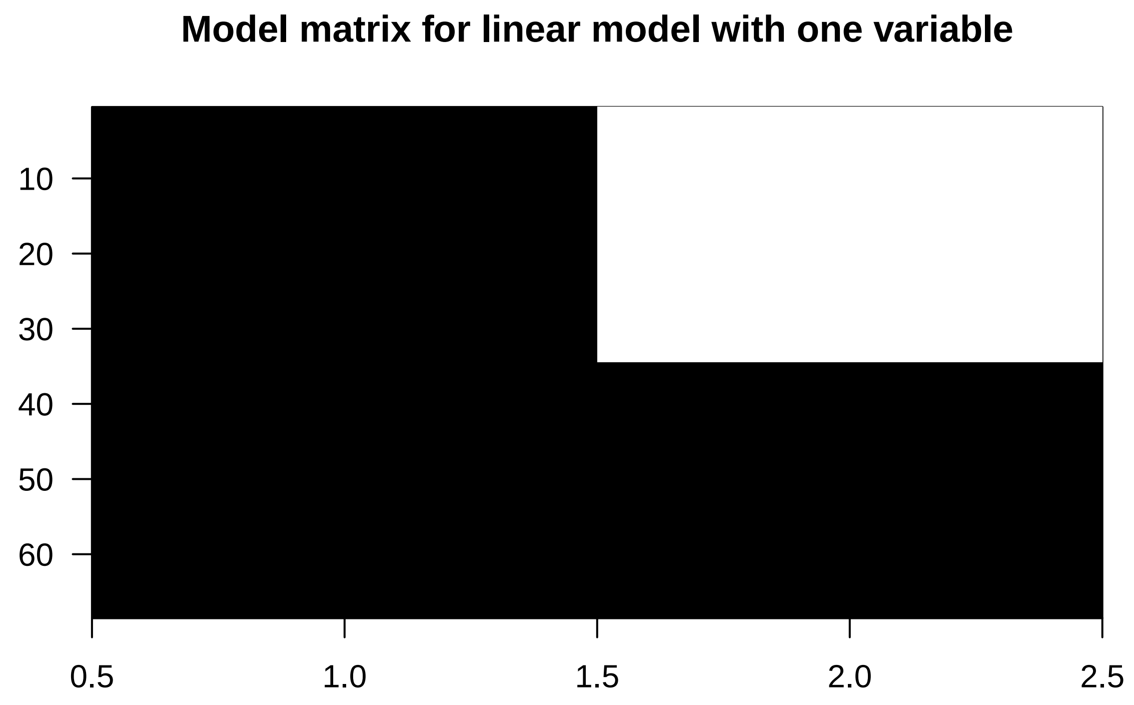


Figure 4: Model matrix for linear model with one variable.

```
## 3            1        0
## 4            1        0
## 5            1        0
## 6            1        0
```

```
tail(X)
```

```
##    (Intercept) typepush
## 63           1        1
## 64           1        1
## 65           1        1
## 66           1        1
## 67           1        1
## 68           1        1
```

Now we'll make a plot of the **X** matrix by putting a black block for the 1's and a white block for the 0's. This plot will be more interesting for the linear models later on in this script. Along the y-axis is the sample number (the row number of the `data`) and along the x-axis is the column of the design matrix **X**. If you have installed the *rafalib* library, you can make this plot with the `imagemat` function:

```
library(rafalib)
imagemat(X, main="Model matrix for linear model with one variable")
```

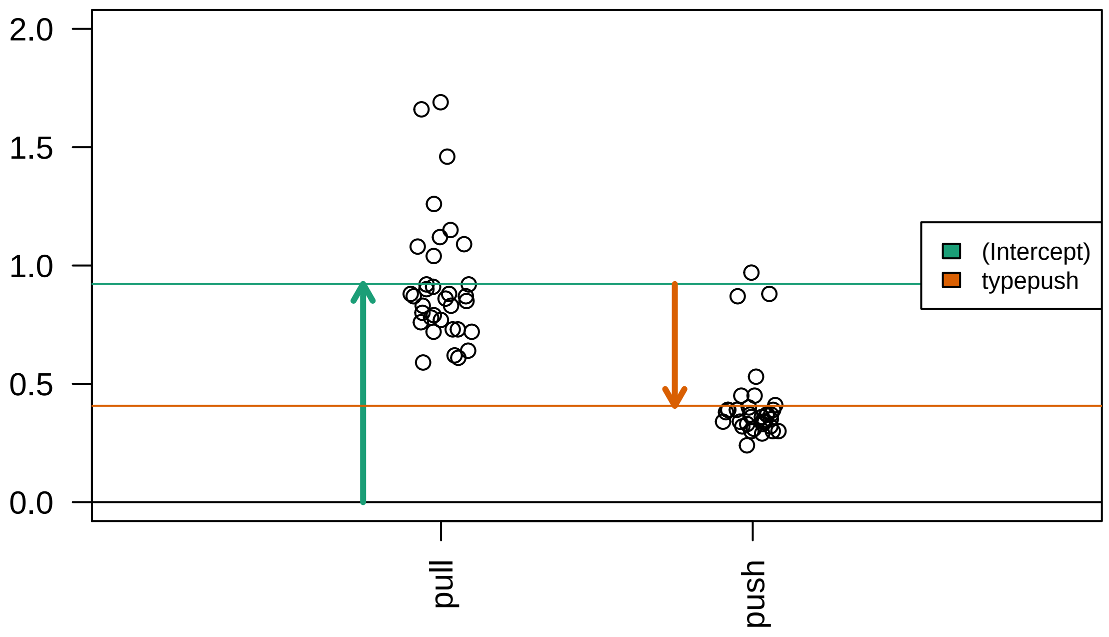


Figure 5: Diagram of the estimated coefficients in the linear model. The green arrow indicates the Intercept term, which goes from zero to the mean of the reference group (here the 'pull' samples). The orange arrow indicates the difference between the push group and the pull group, which is negative in this example. The circles show the individual samples, jittered horizontally to avoid overplotting.

Examining the estimated coefficients

Now we show the coefficient estimates from the linear model in a diagram with arrows (code not shown).

A linear model with two variables

Now we'll continue and examine the full dataset, including the observations from all leg pairs. In order to model both the leg pair differences (L1, L2, L3, L4) and the push vs. pull difference, we need to include both terms in the R formula. Let's see what kind of design matrix will be formed with two variables in the formula:

```
X <- model.matrix(~ type + leg, data=spider)
colnames(X)
```

```
## [1] "(Intercept)" "typepush"    "legL2"       "legL3"       "legL4"
```

```
head(X)
```

```
##   (Intercept) typepush legL2 legL3 legL4
## 1           1        0     0     0     0
## 2           1        0     0     0     0
## 3           1        0     0     0     0
## 4           1        0     0     0     0
## 5           1        0     0     0     0
```

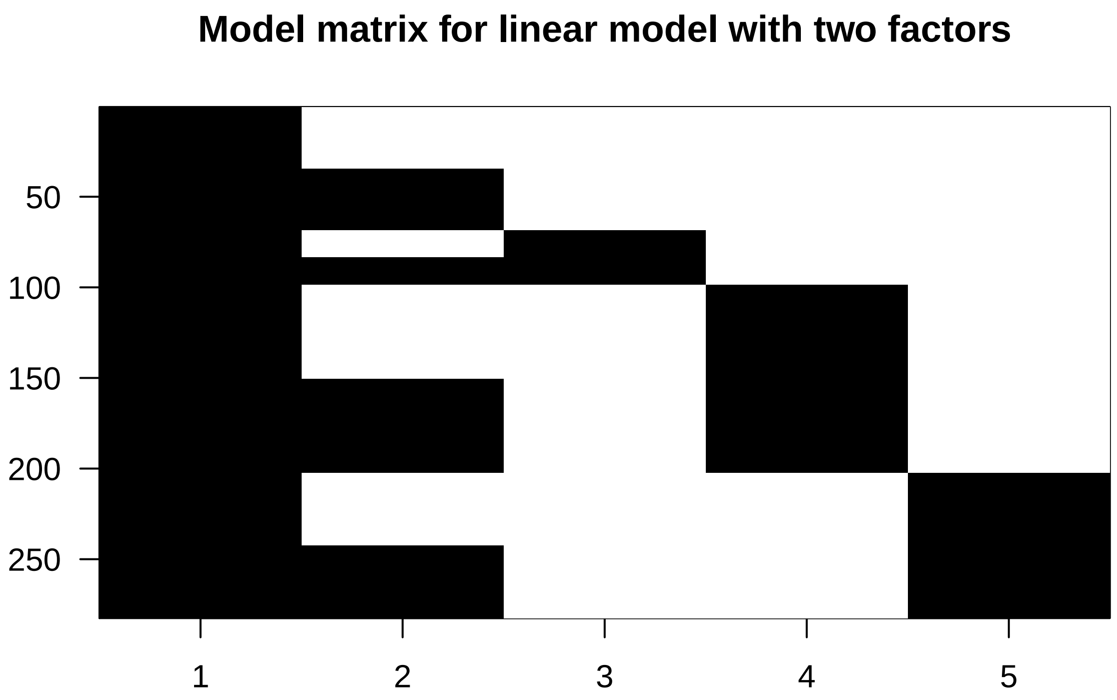


Figure 6: Image of the model matrix for a formula with type + leg

```
## 6           1         0      0      0      0
```

```r
imagemat(X, main="Model matrix for linear model with two factors")
```

The first column is the intercept, and so it has 1's for all samples. The second column has 1's for the push samples, and we can see that there are four groups of them. Finally, the third, fourth and fifth columns have 1's for the L2, L3 and L4 samples. The L1 samples do not have a column, because *L1* is the reference level for `leg`. Similarly, there is no *pull* column, because *pull* is the reference level for the `type` variable.

To estimate coefficients for this model, we use `lm` with the formula `~ type + leg`. We'll save the linear model to `fitTL` standing for a *fit* with *Type* and *Leg*.

```r
fitTL <- lm(friction ~ type + leg, data=spider)
summary(fitTL)
```

```
##
## Call:
## lm(formula = friction ~ type + leg, data = spider)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46392 -0.13441 -0.00525  0.10547  0.69509
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.05392    0.02816  37.426  < 2e-16 ***
## typepush    -0.77901    0.02482 -31.380  < 2e-16 ***
## legL2        0.17192    0.04569   3.763 0.000205 ***
## legL3        0.16049    0.03251   4.937 1.37e-06 ***
## legL4        0.28134    0.03438   8.183 1.01e-14 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2084 on 277 degrees of freedom
## Multiple R-squared:  0.7916, Adjusted R-squared:  0.7886
## F-statistic:   263 on 4 and 277 DF,  p-value: < 2.2e-16
```

```
(coefs <- coef(fitTL))
```

```
## (Intercept)    typepush       legL2       legL3       legL4
##   1.0539153  -0.7790071   0.1719216   0.1604921   0.2813382
```

R uses the name `coefficient` to denote the component containing the least squares **estimates**. It is important to remember that the coefficients are parameters that we do not observe, but only estimate.

Mathematical representation

The model we are fitting above can be written as

$$Y_i = \beta_0 + \beta_1 x_{i,1} + \beta_2 x_{i,2} + \beta_3 x_{i,3} + \beta_4 x_{i,4} + \varepsilon_i, i = 1, \dots, N$$

with the x all indicator variables denoting push or pull and which leg. For example, a push on leg 3 will have $x_{i,1}$ and $x_{i,3}$ equal to 1 and the rest would be 0. Throughout this section we will refer to the β s with the effects they represent. For example we call β_0 the intercept, β_1 the pull effect, β_2 the L2 effect, etc. We do not observe the coefficients, e.g. β_1, directly, but estimate them with, e.g. $\hat{\beta}_4$.

We can now form the matrix $\mathbf{X}$ depicted above and obtain the least square estimates with:

$$\hat{\boldsymbol{\beta}} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{Y}$$

```
Y <- spider$friction
X <- model.matrix(~ type + leg, data=spider)
beta.hat <- solve(t(X) %*% X) %*% t(X) %*% Y
t(beta.hat)
```

```
##      (Intercept)   typepush     legL2     legL3     legL4
## [1,]    1.053915 -0.7790071 0.1719216 0.1604921 0.2813382
```

```
coefs
```

```
## (Intercept)    typepush       legL2       legL3       legL4
##   1.0539153  -0.7790071   0.1719216   0.1604921   0.2813382
```

We can see that these values agree with the output of `lm`.

Examining the estimated coefficients

We can make the same plot as before, with arrows for each of the estimated coefficients in the model (code not shown).

In this case, the fitted means for each group, derived from the fitted coefficients, do not line up with those we obtain from simply taking the average from each of the eight possible groups. The reason is that our model uses five coefficients, instead of eight. We are **assuming** that the effects are additive. However, as

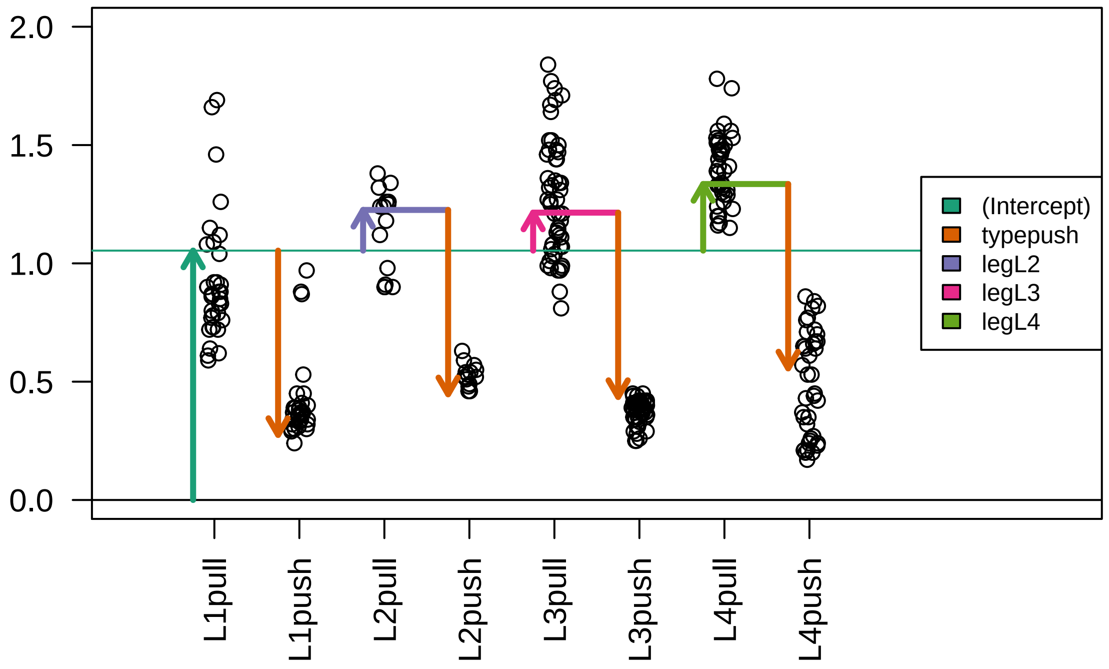


Figure 7: Diagram of the estimated coefficients in the linear model. As before, the teal-green arrow represents the Intercept, which fits the mean of the reference group (here, the pull samples for leg L1). The purple, pink, and yellow-green arrows represent differences between the three other leg groups and L1. The orange arrow represents the difference between the push and pull samples for all groups.

we demonstrate in more detail below, this particular dataset is better described with a model including interactions.

```
s <- split(spider$friction, spider$group)
mean(s[["L1pull"]])
```

```
## [1] 0.9214706
```

```
coefs[1]
```

```
## (Intercept)
##    1.053915
```

```
mean(s[["L1push"]])
```

```
## [1] 0.4073529
```

```
coefs[1] + coefs[2]
```

```
## (Intercept)
##   0.2749082
```

Here we can demonstrate that the push vs. pull estimated coefficient, `coefs[2]`, is a weighted average of the difference of the means for each group. Furthermore, the weighting is determined by the sample size of each group. The math works out simply here because the sample size is equal for the push and pull subgroups within each leg pair. If the sample sizes were not equal for push and pull within each leg pair, the weighting is more complicated but uniquely determined by a formula involving the sample size of each subgroup, the total sample size, and the number of coefficients. This can be worked out from $(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top$.

```
means <- sapply(s, mean)
##the sample size of push or pull groups for each leg pair
ns <- sapply(s, length)[c(1,3,5,7)]
(w <- ns/sum(ns))
```

```
##    L1pull    L2pull    L3pull    L4pull
## 0.2411348 0.1063830 0.3687943 0.2836879
```

```
sum(w * (means[c(2,4,6,8)] - means[c(1,3,5,7)]))
```

```
## [1] -0.7790071
```

```
coefs[2]
```

```
##   typepush
## -0.7790071
```

Contrasting coefficients

Sometimes, the comparison we are interested in is represented directly by a single coefficient in the model, such as the push vs. pull difference, which was `coefs[2]` above. However, sometimes, we want to make a comparison which is not a single coefficient, but a combination of coefficients, which is called a *contrast*. To introduce the concept of *contrasts*, first consider the comparisons which we can read off from the linear model summary:

```
coefs
```

```
## (Intercept)    typepush       legL2       legL3       legL4
##   1.0539153  -0.7790071   0.1719216   0.1604921   0.2813382
```

Here we have the intercept estimate, the push vs. pull estimated effect across all leg pairs, and the estimates for the L2 vs. L1 effect, the L3 vs. L1 effect, and the L4 vs. L1 effect. What if we want to compare two groups and one of those groups is not L1? The solution to this question is to use *contrasts*.

A *contrast* is a combination of estimated coefficient: $\mathbf{c}^\top \hat{\boldsymbol{\beta}}$, where $\mathbf{c}$ is a column vector with as many rows as the number of coefficients in the linear model. If $\mathbf{c}$ has a 0 for one or more of its rows, then the corresponding estimated coefficients in $\hat{\boldsymbol{\beta}}$ are not involved in the contrast.

If we want to compare leg pairs L3 and L2, this is equivalent to contrasting two coefficients from the linear model because, in this contrast, the comparison to the reference level *L1* cancels out:

$$(\text{L3} - \text{L1}) - (\text{L2} - \text{L1}) = \text{L3} - \text{L2}$$

An easy way to make these contrasts of two groups is to use the `contrast` function from the *contrast* package. We just need to specify which groups we want to compare. We have to pick one of *pull* or *push* types, although the answer will not differ, as we will see below.

```
library(contrast) #Available from CRAN
L3vsL2 <- contrast(fitTL,list(leg="L3",type="pull"),list(leg="L2",type="pull"))
L3vsL2
```

```
## lm model parameter contrast
##
##     Contrast       S.E.      Lower      Upper     t  df Pr(>|t|)
##  -0.01142949 0.04319685 -0.0964653 0.07360632 -0.26 277   0.7915
```

The first column `Contrast` gives the L3 vs. L2 estimate from the model we fit above.

We can show that the least squares estimates of a linear combination of coefficients is the same linear combination of the estimates. Therefore, the effect size estimate is just the difference between two estimated coefficients. The contrast vector used by `contrast` is stored as a variable called `X` within the resulting object (not to be confused with our original $\mathbf{X}$, the design matrix).

```
coefs[4] - coefs[3]
```

```
##       legL3
## -0.01142949
```

```
(cT <- L3vsL2$X)
```

```
##   (Intercept) typepush legL2 legL3 legL4
## 1           0        0    -1     1     0
## attr(,"assign")
## [1] 0 1 2 2 2
## attr(,"contrasts")
## attr(,"contrasts")$type
## [1] "contr.treatment"
##
## attr(,"contrasts")$leg
## [1] "contr.treatment"
```

```
cT %*% coefs
```

```
##          [,1]
## 1 -0.01142949
```

What about the standard error and t-statistic? As before, the t-statistic is the estimate divided by the standard error. The standard error of the contrast estimate is formed by multiplying the contrast vector $\mathbf{c}$ on either side of the estimated covariance matrix, $\hat{\Sigma}$, our estimate for $\text{var}(\hat{\boldsymbol{\beta}})$:

$$\sqrt{\mathbf{c}^\top \hat{\boldsymbol{\Sigma}} \mathbf{c}}$$

where we saw the covariance of the coefficients earlier:

$$\mathbf{\Sigma} = \sigma^2(\mathbf{X}^\top\mathbf{X})^{-1}$$

We estimate σ^2 with the sample estimate $\hat{\sigma}^2$ described above and obtain:

```r
Sigma.hat <- sum(fitTL$residuals^2)/(nrow(X) - ncol(X)) * solve(t(X) %*% X)
signif(Sigma.hat, 2)
```

```
##             (Intercept) typepush    legL2    legL3    legL4
## (Intercept)     0.00079 -3.1e-04 -0.00064 -0.00064 -0.00064
## typepush       -0.00031  6.2e-04  0.00000  0.00000  0.00000
## legL2          -0.00064 -6.4e-20  0.00210  0.00064  0.00064
## legL3          -0.00064 -6.4e-20  0.00064  0.00110  0.00064
## legL4          -0.00064 -1.2e-19  0.00064  0.00064  0.00120
```

```r
sqrt(cT %*% Sigma.hat %*% t(cT))
```

```
##             1
## 1 0.04319685
```

```r
L3vsL2$SE
```

```
## [1] 0.04319685
```

We would have obtained the same result for a contrast of L3 and L2 had we picked `type="push"`. The reason it does not change the contrast is because it leads to addition of the `typepush` effect on both sides of the difference, which cancels out:

```r
L3vsL2.equiv <- contrast(fitTL,list(leg="L3",type="push"),list(leg="L2",type="push"))
L3vsL2.equiv$X
```

```
##   (Intercept) typepush legL2 legL3 legL4
## 1           0        0    -1     1     0
## attr(,"assign")
## [1] 0 1 2 2 2
## attr(,"contrasts")
## attr(,"contrasts")$type
## [1] "contr.treatment"
##
## attr(,"contrasts")$leg
## [1] "contr.treatment"
```

Linear Model with Interactions

In the previous linear model, we assumed that the push vs. pull effect was the same for all of the leg pairs (the same orange arrow). You can easily see that this does not capture the trends in the data that well. That is, the tips of the arrows did not line up perfectly with the group averages. For the L1 leg pair, the push vs. pull estimated coefficient was too large, and for the L3 leg pair, the push vs. pull coefficient was somewhat too small.

Interaction terms will help us overcome this problem by introducing additional coefficients to compensate for differences in the push vs. pull effect across the 4 groups. As we already have a push vs. pull term in the model, we only need to add three more terms to have the freedom to find leg-pair-specific push vs. pull differences. As we will see, interaction terms are added to the design matrix by multiplying the columns of the design matrix representing existing terms.

We can rebuild our linear model with an interaction between `type` and `leg`, by including an extra term in the formula `type:leg`. The `:` symbol adds an interaction between the two variables surrounding it. An equivalent way to specify this model is `~ type*leg`, which will expand to the formula `~ type + leg + type:leg`, with main effects for `type`, `leg` and an interaction term `type:leg`.

```
X <- model.matrix(~ type + leg + type:leg, data=spider)
colnames(X)
```

```
## [1] "(Intercept)"    "typepush"       "legL2"          "legL3"
## [5] "legL4"          "typepush:legL2" "typepush:legL3" "typepush:legL4"
```

```
head(X)
```

```
##   (Intercept) typepush legL2 legL3 legL4 typepush:legL2 typepush:legL3
## 1           1        0     0     0     0              0              0
## 2           1        0     0     0     0              0              0
## 3           1        0     0     0     0              0              0
## 4           1        0     0     0     0              0              0
## 5           1        0     0     0     0              0              0
## 6           1        0     0     0     0              0              0
##   typepush:legL4
## 1              0
## 2              0
## 3              0
## 4              0
## 5              0
## 6              0
```

```
imagemat(X, main="Model matrix for linear model with interactions")
```

Columns 6-8 (`typepush:legL2`, `typepush:legL3`, and `typepush:legL4`) are the product of the 2nd column (`typepush`) and columns 3-5 (the three `leg` columns). Looking at the last column, for example, the `typepush:legL4` column is adding an extra coefficient $\beta_{\text{push,L4}}$ to those samples which are both push samples and leg pair L4 samples. This accounts for a possible difference when the mean of samples in the L4-push group are not at the location which would be predicted by adding the estimated intercept, the estimated push coefficient `typepush`, and the estimated L4 coefficient `legL4`.

We can run the linear model using the same code as before:

```
fitX <- lm(friction ~ type + leg + type:leg, data=spider)
summary(fitX)
```

```
##
## Call:
## lm(formula = friction ~ type + leg + type:leg, data = spider)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46385 -0.10735 -0.01111  0.07848  0.76853
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)     0.92147    0.03266  28.215  < 2e-16 ***
## typepush       -0.51412    0.04619 -11.131  < 2e-16 ***
## legL2           0.22386    0.05903   3.792 0.000184 ***
## legL3           0.35238    0.04200   8.390 2.62e-15 ***
## legL4           0.47928    0.04442  10.789  < 2e-16 ***
## typepush:legL2 -0.10388    0.08348  -1.244 0.214409
## typepush:legL3 -0.38377    0.05940  -6.461 4.73e-10 ***
## typepush:legL4 -0.39588    0.06282  -6.302 1.17e-09 ***
```

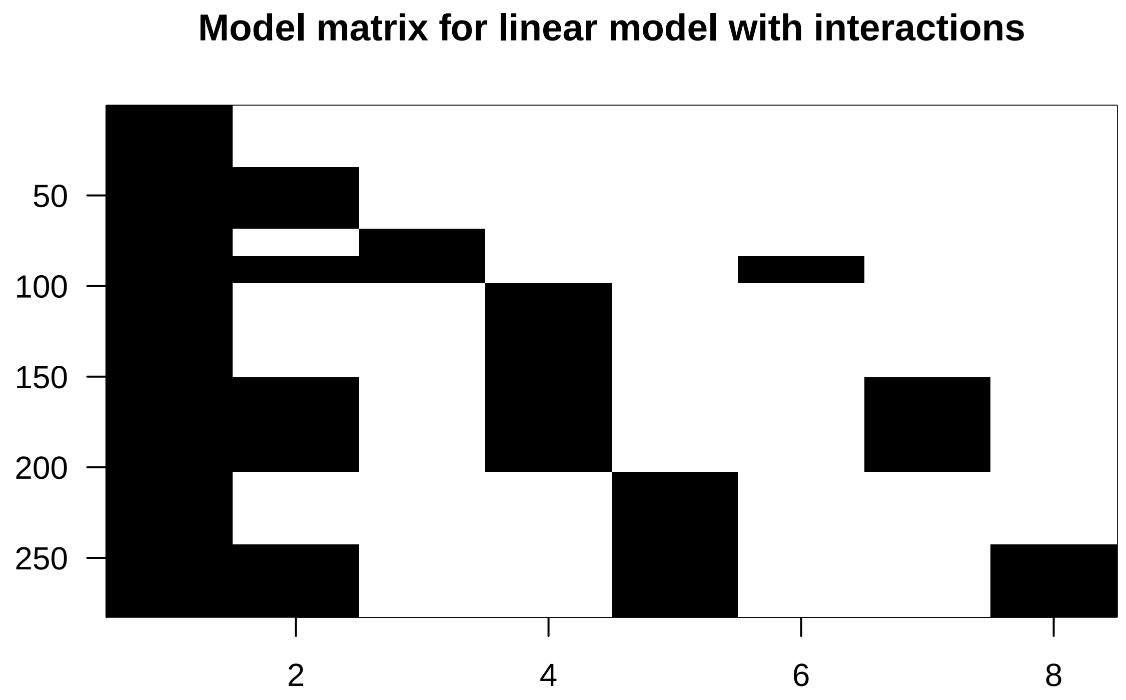


Figure 8: Image of model matrix with interactions.

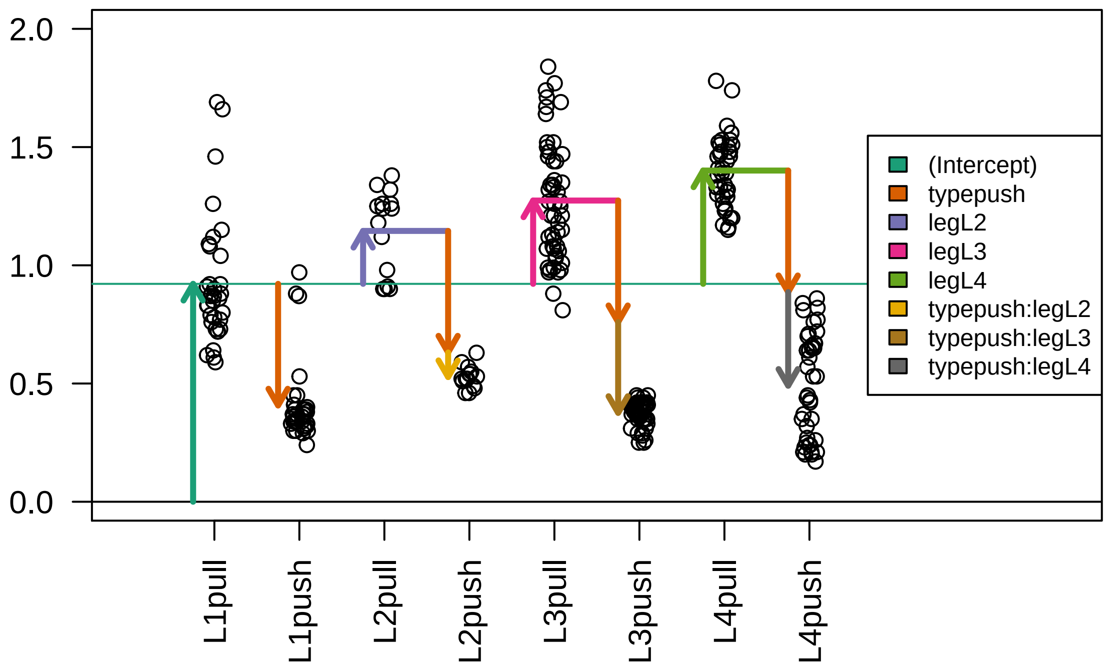


Figure 9: Diagram of the estimated coefficients in the linear model. In the design with interaction terms, the orange arrow now indicates the push vs. pull difference only for the reference group (L1), while three new arrows (yellow, brown and grey) indicate the additional push vs. pull differences in the non-reference groups (L2, L3 and L4) with respect to the reference group.

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1904 on 274 degrees of freedom
## Multiple R-squared:  0.8279, Adjusted R-squared:  0.8235
## F-statistic: 188.3 on 7 and 274 DF,  p-value: < 2.2e-16
```

```
coefs <- coef(fitX)
```

Examining the estimated coefficients

Here is where the plot with arrows really helps us interpret the coefficients. The estimated interaction coefficients (the yellow, brown and silver arrows) allow leg-pair-specific differences in the push vs. pull difference. The orange arrow now represents the estimated push vs. pull difference only for the reference leg pair, which is L1. If an estimated interaction coefficient is large, this means that the push vs. pull difference for that leg pair is very different than the push vs. pull difference in the reference leg pair.

Now, as we have eight terms in the model and eight parameters, you can check that the tips of the arrowheads are exactly equal to the group means (code not shown).

Contrasts

Again we will show how to combine estimated coefficients from the model using contrasts. For some simple cases, we can use the contrast package. Suppose we want to know the push vs. pull effect for the L2 leg pair samples. We can see from the arrow plot that this is the orange arrow plus the yellow arrow. We can also specify this comparison with the `contrast` function:

```
library(contrast) ##Available from CRAN
L2push.vs.pull <- contrast(fitX,
                   list(leg="L2", type = "push"),
                   list(leg="L2", type = "pull"))
L2push.vs.pull
```

```
## lm model parameter contrast
##
##  Contrast      S.E.      Lower      Upper     t  df Pr(>|t|)
##    -0.618 0.0695372 -0.7548951 -0.4811049 -8.89 274        0
```

```
coefs[2] + coefs[6] ##we know this is also orange + yellow arrow
```

```
## typepush
##   -0.618
```

Differences of differences

The question of whether the push vs. pull difference is *different* in L2 compared to L1, is answered by a single term in the model: the `typepush:legL2` estimated coefficient corresponding to the yellow arrow in the plot. A p-value for whether this coefficient is actually equal to zero can be read off from the table printed with `summary(fitX)` above. Similarly, we can read off the p-values for the differences of differences for L3 vs. L1 and for L4 vs. L1.

Suppose we want to know if the push vs. pull difference is *different* in L3 compared to L2. By examining the arrows in the diagram above, we can see that the push vs. pull effect for a leg pair other than L1 is the `typepush` arrow plus the interaction term for that group.

If we work out the math for comparing across two non-reference leg pairs, this is:

$$(\text{typepush} + \text{typepush:legL3}) - (\text{typepush} + \text{typepush:legL2})$$

…which simplifies to:

$$= \text{typepush:legL3} - \text{typepush:legL2}$$

We can't make this contrast using the `contrast` function shown before, but we can make this comparison using the `glht` (for "general linear hypothesis test") function from the *multcomp* package. We need to form a 1-row matrix which has a -1 for the `typepush:legL2` coefficient and a +1 for the `typepush:legL3` coefficient. We provide this matrix to the `linfct` (linear function) argument, and obtain a summary table for this contrast of estimated interaction coefficients.

Note that there are other ways to perform contrasts using base R, and this is just our preferred way.

```
library(multcomp) ##Available from CRAN
```

```
## Warning: package 'MASS' was built under R version 3.5.2
```

```
C <- matrix(c(0,0,0,0,0,-1,1,0), 1)
L3vsL2interaction <- glht(fitX, linfct=C)
summary(L3vsL2interaction)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: lm(formula = friction ~ type + leg + type:leg, data = spider)
##
## Linear Hypotheses:
##        Estimate Std. Error t value Pr(>|t|)
## 1 == 0 -0.27988    0.07893  -3.546  0.00046 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
coefs[7] - coefs[6] ##we know this is also brown - yellow
```

```
## typepush:legL3
##     -0.2798846
```

Analysis of Variance

Suppose that we want to know if the push vs. pull difference is different across leg pairs in general. We do not want to compare any two leg pairs in particular, but rather we want to know if the three interaction terms which represent differences in the push vs. pull difference across leg pairs are larger than we would expect them to be if the push vs. pull difference was in fact equal across all leg pairs.

Such a question can be answered by an *analysis of variance*, which is often abbreviated as ANOVA. ANOVA compares the reduction in the sum of squares of the residuals for models of different complexity. The model with eight coefficients is more complex than the model with five coefficients where we assumed the push vs. pull difference was equal across leg pairs. The least complex model would only use a single coefficient, an intercept. Under certain assumptions we can also perform inference that determines the probability of improvements as large as what we observed. Let's first print the result of an ANOVA in R and then examine the results in detail:

```
anova(fitX)
```

```
## Analysis of Variance Table
##
## Response: friction
##            Df Sum Sq Mean Sq  F value    Pr(>F)
## type        1 42.783  42.783 1179.713 < 2.2e-16 ***
## leg         3  2.921   0.974   26.847 2.972e-15 ***
## type:leg    3  2.098   0.699   19.282 2.256e-11 ***
## Residuals 274  9.937   0.036
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first line tells us that adding a variable `type` (push or pull) to the design is very useful (reduces the sum of squared residuals) compared to a model with only an intercept. We can see that it is useful, because this single coefficient reduces the sum of squares by 42.783. The original sum of squares of the model with just an intercept is:

```
mu0 <- mean(spider$friction)
(initial.ss <- sum((spider$friction - mu0)^2))
```

```
## [1] 57.73858
```

Note that this initial sum of squares is just a scaled version of the sample variance:

```r
N <- nrow(spider)
(N - 1) * var(spider$friction)
```

```
## [1] 57.73858
```

Let's see exactly how we get this 42.783. We need to calculate the sum of squared residuals for the model with only the type information. We can do this by calculating the residuals, squaring these, summing these within groups and then summing across the groups.

```r
s <- split(spider$friction, spider$type)
after.type.ss <- sum( sapply(s, function(x) {
  residual <- x - mean(x)
  sum(residual^2)
  }) )
```

The reduction in sum of squared residuals from introducing the `type` coefficient is therefore:

```r
(type.ss <- initial.ss - after.type.ss)
```

```
## [1] 42.78307
```

Through simple arithmetic, this reduction can be shown to be equivalent to the sum of squared differences between the fitted values for the models with formula `~type` and `~1`:

```r
sum(sapply(s, length) * (sapply(s, mean) - mu0)^2)
```

```
## [1] 42.78307
```

Keep in mind that the order of terms in the formula, and therefore rows in the ANOVA table, is important: each row considers the reduction in the sum of squared residuals after adding coefficients *compared to the model in the previous row.*

The other columns in the ANOVA table show the "degrees of freedom" with each row. As the `type` variable introduced only one term in the model, the `Df` column has a 1. Because the `leg` variable introduced three terms in the model (`legL2`, `legL3` and `legL4`), the `Df` column has a 3.

Finally, there is a column which lists the *F value*. The F value is the *mean of squares* for the inclusion of the terms of interest (the sum of squares divided by the degrees of freedom) divided by the mean squared residuals (from the bottom row):

$$r_i = Y_i - \hat{Y}_i$$

$$\text{Mean Sq Residuals} = \frac{1}{N-p} \sum_{i=1}^{N} r_i^2$$

where p is the number of coefficients in the model (here eight, including the intercept term).

Under the null hypothesis (the true value of the additional coefficient(s) is 0), we have a theoretical result for what the distribution of the F value will be for each row. The assumptions needed for this approximation to hold are similar to those of the t-distribution approximation we described in earlier chapters. We either need a large sample size so that CLT applies or we need the population data to follow a normal approximation.

As an example of how one interprets these p-values, let's take the last row `type:leg` which specifies the three interaction coefficients. Under the null hypothesis that the true value for these three additional terms is actually 0, e.g. $\beta_{\text{push,L2}} = 0, \beta_{\text{push,L3}} = 0, \beta_{\text{push,L4}} = 0$, then we can calculate the chance of seeing such a large F-value for this row of the ANOVA table. Remember that we are only concerned with large values here, because we have a ratio of sum of squares, the F-value can only be positive. The p-value in the last column for the `type:leg` row can be interpreted as: under the null hypothesis that there are no differences in the push vs. pull difference across leg pair, this is the probability of an estimated interaction coefficient explaining so much of the observed variance. If this p-value is small, we would consider rejecting the null hypothesis that the push vs. pull difference is the same across leg pairs.

The F distribution has two parameters: one for the degrees of freedom of the numerator (the terms of interest) and one for the denominator (the residuals). In the case of the interaction coefficients row, this is 3, the number of interaction coefficients divided by 274, the number of samples minus the total number of coefficients.

A different specification of the same model

Now we show an alternate specification of the same model, wherein we assume that each combination of type and leg has its own mean value (and so that the push vs. pull effect is not the same for each leg pair). This specification is in some ways simpler, as we will see, but it does not allow us to build the ANOVA table as above, because it does not split interaction coefficients out in the same way.

We start by constructing a factor variable with a level for each unique combination of `type` and `leg`. We include a `0 +` in the formula because we do not want to include an intercept in the model matrix.

```
##earlier, we defined the 'group' column:
spider$group <- factor(paste0(spider$leg, spider$type))
X <- model.matrix(~ 0 + group, data=spider)
colnames(X)
```

```
## [1] "groupL1pull" "groupL1push" "groupL2pull" "groupL2push" "groupL3pull"
## [6] "groupL3push" "groupL4pull" "groupL4push"
```

```
head(X)
```

```
##   groupL1pull groupL1push groupL2pull groupL2push groupL3pull groupL3push
## 1           1           0           0           0           0           0
## 2           1           0           0           0           0           0
## 3           1           0           0           0           0           0
## 4           1           0           0           0           0           0
## 5           1           0           0           0           0           0
## 6           1           0           0           0           0           0
##   groupL4pull groupL4push
## 1           0           0
## 2           0           0
## 3           0           0
## 4           0           0
## 5           0           0
## 6           0           0
```

```
imagemat(X, main="Model matrix for linear model with group variable")
```

We can run the linear model with the familiar call:

```
fitG <- lm(friction ~ 0 + group, data=spider)
summary(fitG)
```

```
##
## Call:
## lm(formula = friction ~ 0 + group, data = spider)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.46385 -0.10735 -0.01111  0.07848  0.76853
```

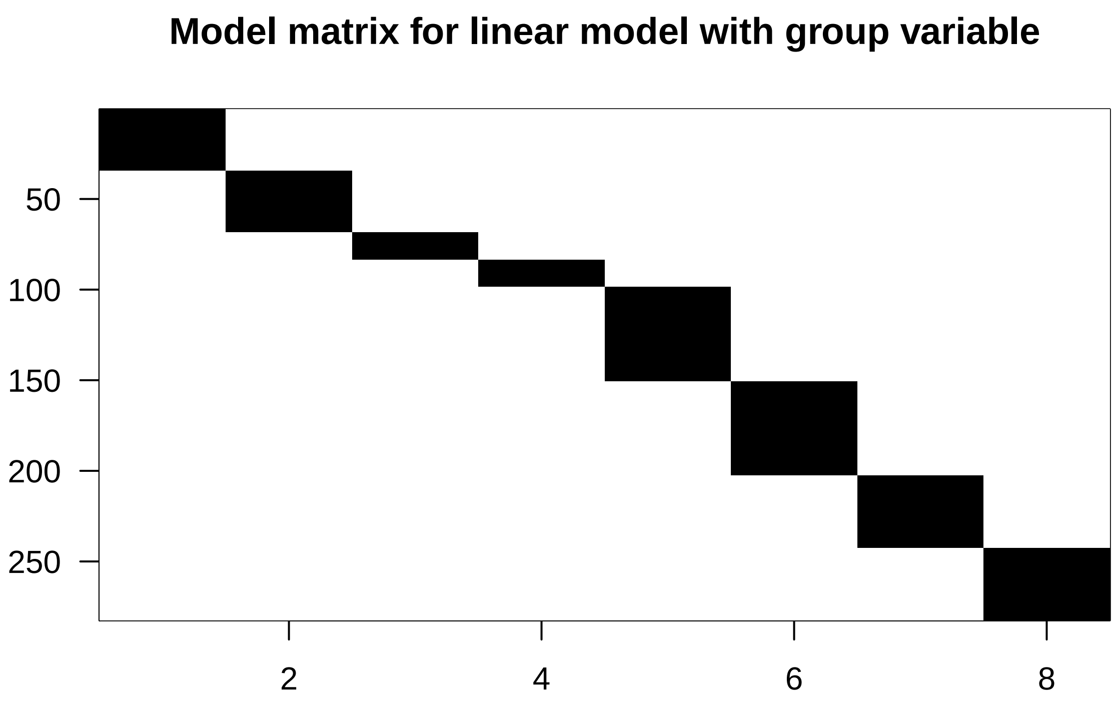


Figure 10: Image of model matrix for linear model with group variable. This model, also with eight terms, gives a unique fitted value for each combination of type and leg.

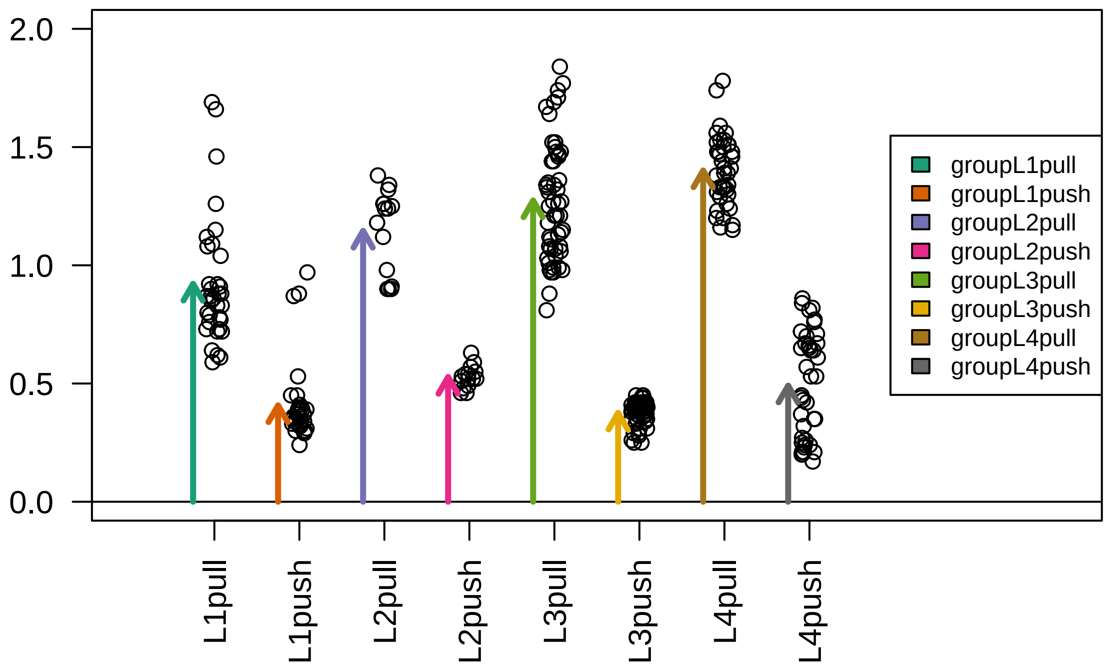


Figure 11: Diagram of the estimated coefficients in the linear model, with each term representing the mean of a combination of type and leg.

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## groupL1pull  0.92147    0.03266   28.21   <2e-16 ***
## groupL1push  0.40735    0.03266   12.47   <2e-16 ***
## groupL2pull  1.14533    0.04917   23.29   <2e-16 ***
## groupL2push  0.52733    0.04917   10.72   <2e-16 ***
## groupL3pull  1.27385    0.02641   48.24   <2e-16 ***
## groupL3push  0.37596    0.02641   14.24   <2e-16 ***
## groupL4pull  1.40075    0.03011   46.52   <2e-16 ***
## groupL4push  0.49075    0.03011   16.30   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1904 on 274 degrees of freedom
## Multiple R-squared:   0.96,  Adjusted R-squared:  0.9588
## F-statistic:   821 on 8 and 274 DF,  p-value: < 2.2e-16
```

```r
coefs <- coef(fitG)
```

Examining the estimated coefficients

Now we have eight arrows, one for each group. The arrow tips align directly with the mean of each group:

Simple contrasts using the contrast package

While we cannot perform an ANOVA with this formulation, we can easily contrast the estimated coefficients for individual groups using the `contrast` function:

```
groupL2push.vs.pull <- contrast(fitG,
                                list(group = "L2push"),
                                list(group = "L2pull"))
groupL2push.vs.pull
```

```
## lm model parameter contrast
##
##   Contrast      S.E.      Lower      Upper     t  df Pr(>|t|)
## 1   -0.618 0.0695372 -0.7548951 -0.4811049 -8.89 274        0
```

```
coefs[4] - coefs[3]
```

```
## groupL2push
##      -0.618
```

Differences of differences when there is no intercept

We can also make pair-wise comparisons of the estimated push vs. pull difference across leg pair. For example, if we want to compare the push vs. pull difference in leg pair L3 vs. leg pair L2:

$$(\text{L3push} - \text{L3pull}) - (\text{L2push} - \text{L2pull})$$

$$= \text{L3 push} + \text{L2pull} - \text{L3pull} - \text{L2push}$$

```
C <- matrix(c(0,0,1,-1,-1,1,0,0), 1)
groupL3vsL2interaction <- glht(fitG, linfct=C)
summary(groupL3vsL2interaction)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Fit: lm(formula = friction ~ 0 + group, data = spider)
##
## Linear Hypotheses:
##        Estimate Std. Error t value Pr(>|t|)
## 1 == 0 -0.27988    0.07893  -3.546  0.00046 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
names(coefs)
```

```
## [1] "groupL1pull" "groupL1push" "groupL2pull" "groupL2push" "groupL3pull"
## [6] "groupL3push" "groupL4pull" "groupL4push"
```

```
(coefs[6] - coefs[5]) - (coefs[4] - coefs[3])
```

```
## groupL3push
##  -0.2798846
```