Week2

Neeraj Vashistha

Thursday, April 30, 2015

Testing Linear Models

Inference Review Assessment

The standard error of an estimate is the standard deviation of the sampling distribution of an estimate. In PH525.1x, we saw that our estimate of the mean of a population changed depending on the sample that we took from the population. If we repeatedly sampled from the population, and each time estimated the mean, the collection of mean estimates formed the sampling distribution of the estimate. When we took the standard deviation of those estimates, that was the standard error of our mean estimate.

Here, we aren't sampling individuals from a population, but we do have random noise in our observations Y. The estimate for the linear model terms (beta-hat) will not be the same if we were to re-run the experiment, because the random noise would be different. If we were to re-run the experiment many times, and estimate linear model terms (beta-hat) each time, this is called the sampling distribution of the estimates. If we take the standard deviation of all of these estimates from repetitions of the experiment, this is called the standard error of the estimate. While we are not sampling individuals, you can think about the repetition of the experiment that we are "sampling" new errors in our observation of Y.

QUESTION 2.1.1

We have shown how to find the least squares estimates with matrix algebra. These estimates are random variables as they are linear combinations of the data. For these estimates to be useful we also need to compute the standard errors.

Here we review standard errors in the context of linear models.

It is useful to think about where randomness comes from. In our falling object example, randomness was introduced through measurement errors. Every time we rerun the experiment a new set of measurement errors will be made which implies our data will be random. This implies that our estimate of, for example, the gravitational constant will change. The constant is fixed, but our estimates are not. To see this we can run a Monte Carlo simulation. Specifically we will generate the data repeatedly and compute the estimate for the quadratic term each time.

```r
g = 9.8 ## meters per second
h0 = 56.67
v0 = 0
n = 25
tt = seq(0,3.4,len=n) ##time in secs, t is a base function
y = h0 + v0 *tt - 0.5* g*tt^2 + rnorm(n,sd=1)
```

Now we act as if we didn't know h0, v0 and -0.5*g and use regression to estimate these. We can rewrite the model as y = b0 + b1 t + b2 t^2 + e and obtain the LSE we have used in this class. Note that g = -2 b2.

To obtain the LSE in R we could write:

```r
X = cbind(1,tt,tt^2)
A = solve(crossprod(X))%*%t(X)
```

Given how we have defined A, which of the following is the LSE of g, the acceleration due to gravity (suggestion: try the code in R)?

```r
-2 * (A %*% y) [3]
```

```
## [1] 10.02888
```

QUESTION 2.1.2

In the lines of code above, there was a call to a random function rnorm(). This means that each time the lines of code above are repeated, the estimate of g will be different.

Use the code above in conjunction with the function replicate() to generate 100,000 Monte Carlo simulated datasets. For each dataset compute an estimate of g (remember to multiply by -2)

What is the standard error of this estimate?:

```r
betahat = replicate(100000,{
  g = 9.8 ## meters per second
h0 = 56.67
v0 = 0
n = 25
tt = seq(0,3.4,len=n) ##time in secs, t is a base function
y = h0 + v0 *tt - 0.5* g*tt^2 + rnorm(n,sd=1)
X = cbind(1,tt,tt^2)
A = solve(crossprod(X))%*%t(X)
-2 * (A %*% y) [3]
})
sqrt(mean((betahat-mean(betahat))^2))
```

```
## [1] 0.4287723
```

QUESTION 2.1.3

In the father and son height examples we have randomness because we have a random sample of father and son pairs. For the sake of illustration let’s assume that this is the entire population:

```r
library(UsingR)
```

```
## Loading required package: MASS
## Loading required package: HistData
## Loading required package: Hmisc
## Loading required package: grid
## Loading required package: lattice
## Loading required package: survival
## Loading required package: Formula
## Loading required package: ggplot2
##
```

```
## Attaching package: 'Hmisc'
##
## The following objects are masked from 'package:base':
##
##     format.pval, round.POSIXt, trunc.POSIXt, units
##
##
## Attaching package: 'UsingR'
##
## The following object is masked from 'package:ggplot2':
##
##     movies
##
## The following object is masked from 'package:survival':
##
##     cancer
```

```
x = father.son$fheight

y = father.son$sheight

n = length(y)
```

Now let's run a Monte Carlo simulation in which we take a sample of size 50 over and over again. Here is how we obtain one sample:

```
N = 50

index = sample(n,N)

sampledat = father.son[index,]

x = sampledat$fheight

y = sampledat$sheight

betahat = lm(y~x)$coef
```

Use the function replicate to take 10,000 samples.

What is the standard error of the slope estimate? That is, calculate the standard deviation of the estimate from many random samples.

```
N=50
B=10000
betahat = replicate(B,{
  index=sample(n,N)
  sampledat=father.son[index,]
  x=sampledat$fheight
  y=sampledat$sheight
  lm(y~x)$coef[2]
})
sqrt(mean((betahat-mean(betahat))^2))
```

```
## [1] 0.1230063
```

QUESTION 2.1.4

We are defining a new concept: covariance. The covariance of two lists of numbers X=X1,...,Xn and Y=Y1,...,Yn is mean((Y - mean(Y))*(X-mean(X))).

Which of the following is closest to the covariance between father heights and son heights

```
X<-c(x)
Y<-c(y)
mean( (Y - mean(Y))*(X-mean(X) ) )
```

```
## [1] 1.821083
```

Explanation Note we know it can't be 0 because only independent variables have covariance 0. And we know it can't be negative because these variables are positively correlated thus (Y - mean(Y)) and (X-mean(X)) tend to have the same sign.

Technical Note on Variance

The standard approach to writing linear models either assume the X are fixed or that we are conditioning on them. Thus X*beta has no variance as the X is considered fixed. This is why we write var(Y_i) = var(e_i)=sigma^2. This can cause confusion in practice because if you, for example, compute the following:

```
x =  father.son$fheight

beta =  c(34,0.5)

var(beta[1]+beta[2]*x)
```

```
## [1] 1.883576
```

it is nowhere near 0. This is an example in which we have to be careful in distinguishing code from math. The function *var* is simply computing the variance of the list we feed it, while the mathematical use of var is considering only quantities that are random variables. In the R code above, x is not fixed at all: we are letting it vary but when we write var(Y_i) = sigma^2 we are imposing, mathematically, x to be fixed. Similarly if we use R to compute the variance of Y in our object dropping example we obtain something very different than sigma^2=1 (the known variance):

```
y = h0 + v0*tt  - 0.5*g*tt^2 + rnorm(n,sd=1)
```

```
## Warning in h0 + v0 * tt - 0.5 * g * tt^2 + rnorm(n, sd = 1): longer object
## length is not a multiple of shorter object length
```

```
var(y)
```

```
## [1] 311.9826
```

Again, this is because we are not fixing tt.

Standard Error Assessment

In the previous assessment, we used a Monte Carlo technique to see that the linear model coefficients are random variables when the data is a random sample. Now we will use the matrix algebra from the previous video to try to estimate the standard error of the linear model coefficients. Again, take a random sample of the father.son heights data:

```
library(UsingR)
x = father.son$fheight
y = father.son$sheight
n = length(y)
N = 50
set.seed(1)
index = sample(n,N)
sampledat = father.son[index,]
x = sampledat$fheight
y = sampledat$sheight
betahat = lm(y~x)$coef
```

The formula for the standard error in the previous video was:

SE(betahat) = sqrt(var(betahat))

var(betahat) = sigma^2 (X^T X)^-1

This is also listed in the standard error book page.

We will estimate or calculate each part of this equation and then combine them.

First, we want to estimate sigma^2, the variance of Y. As we have seen in the previous unit, the random part of Y is only coming from epsilon, because we assume X*beta is fixed. So we can try to estimate the variance of the epsilons from the residuals, the Y_i minus the fitted values from the linear model.

QUESTION 2.2.1

Note that the fitted values (Y-hat) from a linear model can be obtained with:

```
fit = lm(y ~ x)

head(fit$fitted.values)
```

```
##        1        2        3        4        5        6
## 70.62707 70.36129 70.86093 68.73019 65.59181 70.55285
```

What is the sum of the squared residuals (where residuals are given by r_i = Y_i - Y-hat_i)?

```
fit=lm(y~x)
sum((y-fit$fitted.values)^2)
```

```
## [1] 256.2152
```

Our estimate of sigma^2 will be the sum of squared residuals divided by (N - p), the sample size minus the number of terms in the model. Since we have a sample of 50 and 2 terms in the model (an intercept and a slope), our estimate of sigma^2 will be the sum of squared residuals divided by 48. Save this to a variable 'sigma2':

```
SSR<-sum((y-fit$fitted.values)^2)
sigma2 = SSR / 48
```

where SSR is the answer to the previous question.

QUESTION 2.2.2

Form the design matrix X (note: use a capital X!). This can be done by combining a column of 1's with a column of 'x' the father's heights.

```
X = cbind(rep(1,N), x)
```

Now calculate (X^T X)^-1, the inverse of X transpose times X. Use the solve() function for the inverse and t() for the transpose. What is the element in the first row, first column?

```
solve(t(X)%*%X)
```

```
##                         x
##   11.302749 -0.166027040
## x -0.166027  0.002443108
```

QUESTION 2.2.3

Now we are one step away from the standard error of beta-hat. Take the diagonals from the (X^T X)^-1 matrix above, using the diag() function. Now multiply our estimate of sigma^2 and the diagonals of this matrix. This is the estimated variance of beta-hat, so take the square root of this. You should end up with two numbers, the standard error for the intercept and the standard error for the slope.

What is the standard error for the slope?

```
sqrt(diag(solve(t(X)%*%X))*sigma2)
```

```
##                     x
## 7.7673671 0.1141966
```

Inference for LSE

We have shown how we can obtain standard errors for our estimates. But, as we learned in PH525.1x to perform inference we need to know the distribution of these random variables. The reason we went through the effort of computing the standard errors is because the CLT applies in linear models. If N is large enough, then the LSE will be normally distributed with mean beta and standard errors as described in our videos. For small samples, if the error term is normally distributed then the betahat-beta follow a t-distribution. Proving this mathematically is rather advanced, but the results are extremely useful as it is how we construct p-values and confidence intervals in the context of linear models.

Design Assessment

Suppose we have an experiment with the following design: on three different days, we perform an experiment with two treated and two control samples. We then measure some outcome Y_i, and we want to test the effect of condition, while controlling for whatever differences might have occured due to the the different day (maybe the temperature in the lab affects the measuring device). Assume that the true condition effect is the same for each day (no interaction between condition and day). We then define factors in R for 'day' and for 'condition'.

```
    day: | A | B | C

condition:----------

  treated | 2 | 2 | 2

  control | 2 | 2 | 2
```

QUESTION 2.3.1

Given the factors we have defined above, and not defining any new ones, which of the following R formula will produce a design matrix (model matrix) that let's us analyze the effect of condition, controlling for the different days:

```
~ day + condition
```

Linear Model in Practice Assessment

In the last videos we saw how to use lm() to run a simple two group linear model, and then compared the t-value from the linear model with the t-value from a t-test with the equal variance assumption. Though the linear model in this case is equivalent to a t-test, we will soon explore more complicated designs, where the linear model is a useful extension (confounging variables, testing contrasts of terms, testing interactions, testing many terms at once, etc.)

Here we will review the mathematics on why these produce the same t-value and therefore p-value.

We already know that the numerator of the t-value in both cases is the difference between the average of the groups, so we only have to see that the denominator is the same. Of course, it makes sense that the denominator should be the same, since we are calculating the standard error of the same quanity (the difference) under the same assumptions (equal variance), but here we will show equivalence of the formula.

In the linear model, we saw how to calculate this standard error using the design matrix X and the estimate of sigmaˆ2 from the residuals. The estimate of sigmaˆ2 was the sum of squared residuals divided by (N - p), where N is the total number of samples and p is the number of terms (an intercept and a group indicator, so here p=2).

In the t-test, the denominator of the t-value is the standard error of the difference. The t-test formula for the standard error of the difference, if we assume equal variance in the two groups is:

SE = sqrt(var(diff))

var(diff) = (1/nx + 1/ny) (sum { (x_i - mu_x)ˆ2 } + sum { (y_i - mu_y)ˆ2 }) / (nx + ny - 2)

Where nx is the number of samples in the first group and ny is the number of samples in the second group.

If we look carefully, the second part of this equation is the sum of squared residuals, divided by (N - 2).

So all that is left to show is that

((X^T X)^-1)[2,2] = (1/nx + 1/ny)

. . . where [2,2] indicates the 2nd row, 2nd column, with X as the design matrix of a linear model of two groups.

QUESTION 2.4.1

You can make a design matrix X for a two group comparison either using model.matrix or simply with:

```
nx<-5
ny<-7
X = cbind(rep(1,nx + ny),rep(c(0,1),c(nx, ny)))
```

For a comparison of two groups, where the first group has nx=5 samples, and the second group has ny=7 samples, what is the element in the 1st row and 1st column of X^T X?

```
XtX<-t(X)%*%X
XtX[1,1]
```

```
## [1] 12
```

QUESTION 2.4.2

What are all the other elements of (X^t X)?

```
t(X)%*%X
```

```
##      [,1] [,2]
## [1,]   12    7
## [2,]    7    7
```

Complex Designs

Contrast Assessment

Remember, you can check the book page for contrasts here.

Suppose we have an experiment with two species A and B, and two conditions: control and treated.

```
species <- factor(c("A","A","B","B"))
condition <- factor(c("control","treated","control","treated"))
```

And we will use a formula of '~ species + condition'.

The model matrix is then:

```
model.matrix(~ species + condition)
```

```
##   (Intercept) speciesB conditiontreated
## 1           1        0                0
## 2           1        0                1
## 3           1        1                0
## 4           1        1                1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$species
## [1] "contr.treatment"
##
## attr(,"contrasts")$condition
## [1] "contr.treatment"
```

QUESTION 2.5.1

Suppose we want to build a contrast of coefficients for the above experimental design.

You can either figure this question out through logic, by looking at the design matrix, or using the contrast() function from the contrast library. The contrast vector is returned as contrast(...)$X.

What should the contrast vector be, for the contrast of (species=B and condition=control) vs (species=A and condition=treatment)? Assume that the beta vector from the model fit by R is: Intercept, speciesB, conditiontreated.

```r
library(contrast)
```

```
## Warning: package 'contrast' was built under R version 3.1.3

## Loading required package: rms

## Warning: package 'rms' was built under R version 3.1.3

## Loading required package: gridExtra

## Warning: package 'gridExtra' was built under R version 3.1.3

## Loading required package: SparseM

## Warning: package 'SparseM' was built under R version 3.1.3

##
## Attaching package: 'SparseM'
##
## The following object is masked from 'package:base':
##
##     backsolve
```

```r
y = rnorm(4)

fit = lm(y ~ species + condition)

contrast(fit, list(species="B",condition="control"), list(species="A",condition="treated"))$X
```

```
## Warning: package 'sandwich' was built under R version 3.1.3

##   (Intercept) speciesB conditiontreated
## 1           0        1               -1
## attr(,"assign")
## [1] 0 1 2
## attr(,"contrasts")
## attr(,"contrasts")$species
## [1] "contr.treatment"
##
## attr(,"contrasts")$condition
## [1] "contr.treatment"
```

QUESTION 2.5.2

Use the Rmd script of the spider dataset. Suppose we build a model using two variables: ~ type + leg.

What is the t-value for the contrast of leg pair L4 vs leg pair L2?

```
library(contrast)
filename <- "spider_wolff_gorb_2013.csv"
spider <- read.csv(filename, skip=1)
fitTL <- lm(friction ~ type+leg, data=spider)
(L3vsL2 <- contrast(fitTL,list(leg="L4",type="pull"),list(leg="L2",type="pull")))
```

```
## lm model parameter contrast
##
##   Contrast       S.E.      Lower     Upper    t  df Pr(>|t|)
##  0.1094167 0.04462392 0.02157158 0.1972618 2.45 277   0.0148
```

In the book page, we computed Sigma using:

```
X <- model.matrix(~ type + leg, data=spider)
(Sigma <- sum(fitTL$residuals^2)/(nrow(X) - ncol(X)) * solve(t(X) %*% X))
```

```
##               (Intercept)      typepush         legL2         legL3
## (Intercept)  0.0007929832 -3.081306e-04 -0.0006389179 -0.0006389179
## typepush    -0.0003081306  6.162612e-04  0.0000000000  0.0000000000
## legL2       -0.0006389179 -6.439411e-20  0.0020871318  0.0006389179
## legL3       -0.0006389179 -6.439411e-20  0.0006389179  0.0010566719
## legL4       -0.0006389179 -1.191291e-19  0.0006389179  0.0006389179
##                     legL4
## (Intercept) -0.0006389179
## typepush     0.0000000000
## legL2        0.0006389179
## legL3        0.0006389179
## legL4        0.0011819981
```

Our contrast matrix is:

```
(C <- matrix(c(0,0,-1,0,1),1,5))
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0   -1    0    1
```

QUESTION 2.5.3

Using Sigma, what is Cov(beta-hat_L4, beta-hat_L2)?

```
Sigma[3,5]
```

```
## [1] 0.0006389179
```

Confirm that

`sqrt(Var(beta-hat_L4 - beta-hat_L2)) #= sqrt(Var(beta-hat_L4) + Var(beta-hat_L2) - 2 Cov(beta-hat_L4, beta-hat_L2))`

is equal to

```
sqrt(C %*% Sigma %*% t(C))
```

```
##            [,1]
## [1,] 0.04462392
```

is equal to the standard error from the contrast() for the leg L4 vs L2 difference.

Interaction Assessment

Remember, you can check the book page for contrasts here.

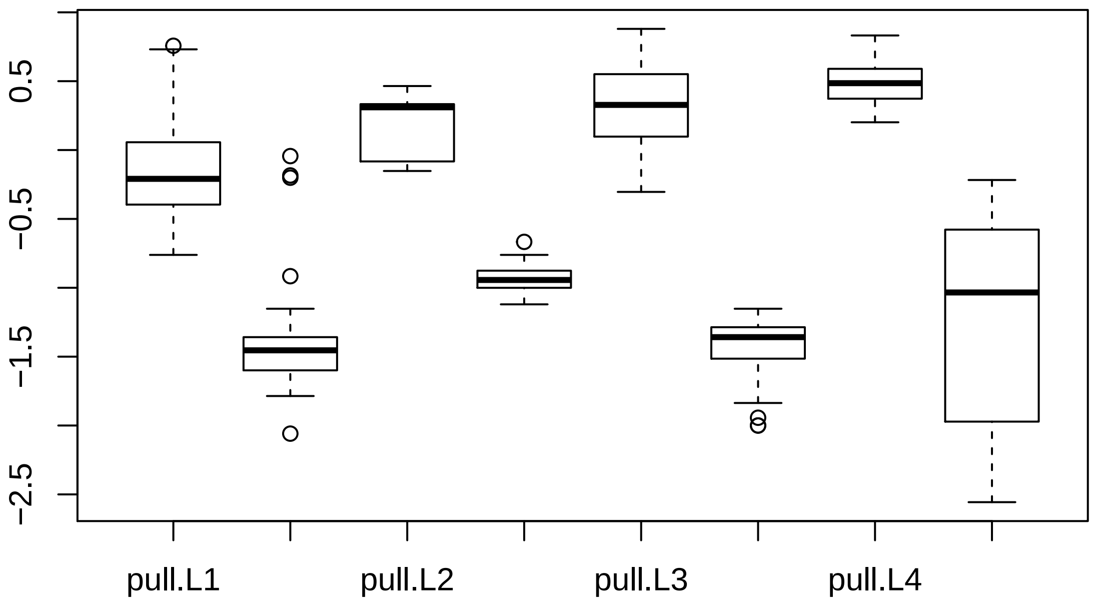
Suppose that we notice that the within-group variances for the groups with smaller frictional coefficients are generally smaller, and so we try to apply a transformation to the frictional coefficients to make the within-group variances more constant.

Add a new variable log2friction to the spider dataframe:

```
spider$log2friction <- log2(spider$friction)
```

The 'Y' values now look like:

```
boxplot(log2friction ~ type*leg, data=spider)
```



Run a linear model of log2friction with type, leg and interactions between type and leg.

```
fitTLLog2 <- lm(log2friction ~ type + leg +type:leg, data=spider)
summary(fitTLLog2)
```

```
##
## Call:
## lm(formula = log2friction ~ type + leg + type:leg, data = spider)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.35902 -0.19193  0.00596  0.16315  1.33090
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -0.16828    0.06613  -2.545 0.011487 *
## typepush       -1.20656    0.09352 -12.901  < 2e-16 ***
## legL2           0.34681    0.11952   2.902 0.004014 **
## legL3           0.48999    0.08505   5.762 2.24e-08 ***
## legL4           0.64668    0.08995   7.189 6.20e-12 ***
## typepush:legL2  0.09967    0.16903   0.590 0.555906
## typepush:legL3 -0.54075    0.12027  -4.496 1.02e-05 ***
## typepush:legL4 -0.46920    0.12721  -3.689 0.000272 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.3856 on 274 degrees of freedom
## Multiple R-squared:  0.8125, Adjusted R-squared:  0.8077
## F-statistic: 169.6 on 7 and 274 DF,  p-value: < 2.2e-16
```

QUESTION 2.6.1

What is the t-value for the interaction of type push and leg L4? If this t-value is sufficiently large, we would reject the null hypothesis that the push vs pull effect on log2(friction) is the same in L4 as in L1.

Ans: Read the t-value for typepush:leg4

QUESTION 2.6.2

What is the F-value for all of the type:leg interaction terms, in an analysis of variance? If this value is sufficiently large, we would reject the null hypothesis that the push vs pull effect on log2(friction) is the same for all leg pairs.

```r
anova(fitTLLog2)
```

```
## Analysis of Variance Table
##
## Response: log2friction
##            Df  Sum Sq Mean Sq  F value    Pr(>F)
## type        1 164.709 164.709 1107.714 < 2.2e-16 ***
## leg         3   7.065   2.355   15.838 1.589e-09 ***
## type:leg    3   4.774   1.591   10.701 1.130e-06 ***
## Residuals 274  40.742   0.149
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

QUESTION 2.6.3

What is the L2 vs L1 estimate in log2friction for the pull samples?

```r
contrast(fitTLLog2,list(type="pull",leg="L2"),list(type="pull",leg="L1"))
```

```
## lm model parameter contrast
##
##   Contrast      S.E.     Lower     Upper   t  df Pr(>|t|)
##  0.3468125 0.1195246 0.1115092 0.5821157 2.9 274    0.004
```

QUESTION 2.6.4

What is the L2 vs L1 estimate in log2friction for the push samples? Remember, because of the interaction terms, this is not the same as the L2 vs L1 difference for the pull samples. If you're not sure use the contrast() function. Another hint: consider the arrows plot for the model with interactions.

```r
contrast(fitTLLog2,list(type="push",leg="L2"),list(type="push",leg="L1"))
```

```
## lm model parameter contrast
##
##   Contrast      S.E.     Lower     Upper    t  df Pr(>|t|)
##  0.4464843 0.1195246 0.211181 0.6817875 3.74 274    2e-04
```

Note that taking the log2 of a Y value and then performing a linear model has a meaningful effect on the coefficients. If we have,

log2(Y_1) = beta_0

and

log2(Y_2) = beta_0 + beta_1

Then Y_2/Y_1 = 2^(beta_0 + beta_1) / 2^(beta_0)

= 2^beta_1

So beta_1 represents a log2 fold change of Y_2 over Y_1. If beta_1 = 1, then Y_2 is 2 times Y_1. If beta_1 = -1, then Y_2 is half of Y_1, etc.

In the video we briefly mentioned the analysis of variance (or ANOVA, performed in R using the anova() function), which allows us to test whether a number of coefficients are equal to zero, by comparing a linear model including these terms to a linear model where these terms are set to 0.

The book page for this section has a section, "Testing all differences of differences", which explains the ANOVA concept and the F-test in some more detail. You can read over that section before or after the following question.

In this last question, we will use Monte Carlo techniques to observe the distribution of the ANOVA's "F-value" under the null hypothesis, that there are no differences between groups.

Suppose we have 4 groups, and 10 samples per group, so 40 samples overall:

```
N <- 40
p <- 4
group <- factor(rep(1:p,each=N/p))
X <- model.matrix(~ group)
```

We will show here how to calculate the "F-value", and then we will use random number to observe the distribution of the F-value under the null hypothesis.

The F-value is the mean sum of squares explained by the terms of interest (in our case, the 'group' terms) divided by the mean sum of squares of the residuals of a model including the terms of interest. So it is the explanatory power of the terms divided by the leftover variance.

Intuitively, if this number is large, it means that the group variable explains a lot of the variance in the data, compared to the amount of variance left in the data after using group information. We will calculate these values exactly here:

First generate some random, null data, where the mean is the same for all groups:

```
Y <- rnorm(N,mean=42,7)
```

The base model we wil compare against is simply Y-hat = mean(Y), which we will call mu0, and the initial sum of squares is the Y values minus mu0:

```
mu0 <- mean(Y)
initial.ss <- sum((Y - mu0)^2)
```

We then need to calculate the fitted values for each group, which is simply the mean of each group, and the residuals from this model, which we will call "after.group.ss" for the sum of squares after using the group information:

```r
s <- split(Y, group)
after.group.ss <- sum(sapply(s, function(x) sum((x - mean(x))^2)))
```

Then the explanatory power of the group variable is the initial sum of squares minus the residual sum of squares:

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

```
## [1] 8.557431
```

We calculate the mean of these values, but we divide by terms which remove the number of fitted parameters. For the group sum of squares, this is number of parameters used to fit the groups (3, because the intercept is in the initial model). For the after group sum of squares, this is the number of samples minus the number of parameters total (So N - 4, including the intercept).

```r
group.ms <- group.ss / (p - 1)
after.group.ms <- after.group.ss / (N - p)
```

The F-value is simply the ratio of these mean sum of squares.

```r
f.value <- group.ms / after.group.ms
```

What's the point of all these calculations? The point is that, after following these steps, the exact distribution of the F-value has a nice mathematical formula under the null hypothesis. We will see this below.

QUESTION 2.6.5

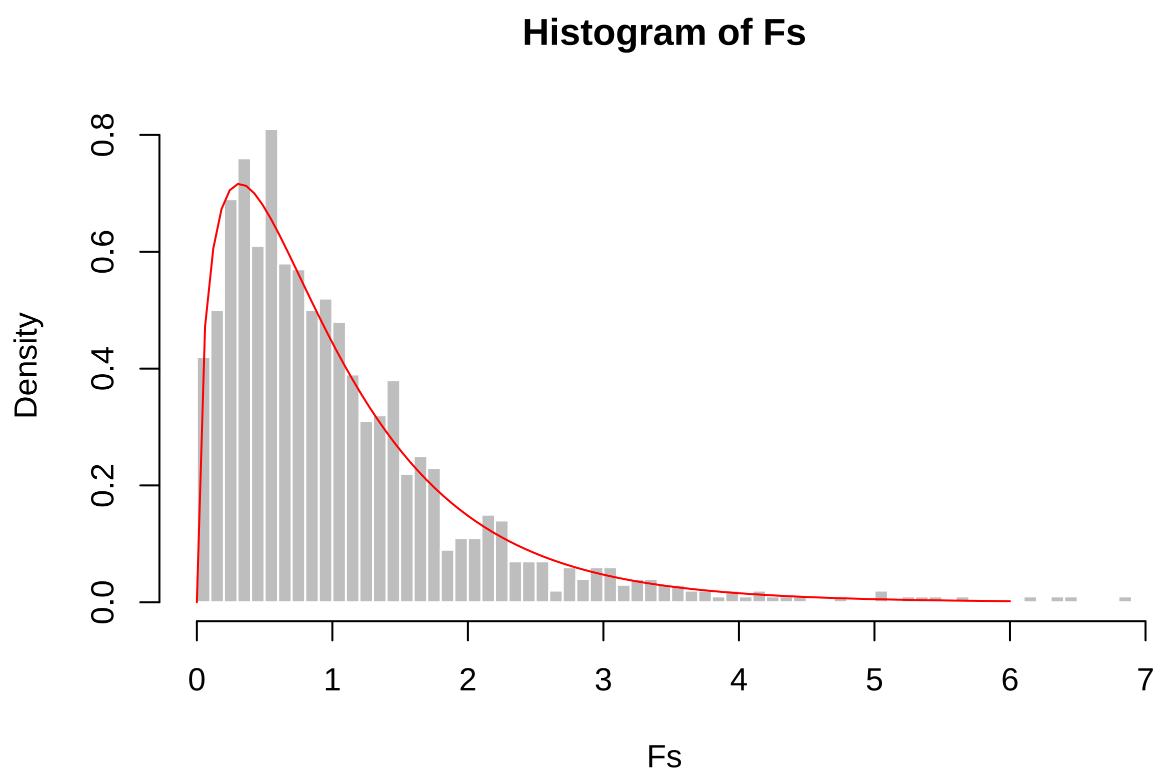
Calculate the F-value for 1000 random versions of Y. What is the mean of these F-values?

```r
Fs=replicate(1000,{
  Y <- rnorm(N,mean=42,7)
  mu0 <- mean(Y)
  initial.ss <- sum((Y - mu0)^2)
  s <- split(Y, group)
  after.group.ss <- sum(sapply(s, function(x) sum((x - mean(x))^2)))
  group.ss <- initial.ss - after.group.ss
  group.ms <- group.ss / (p - 1)
  after.group.ms <- after.group.ss / (N - p)
  f.value <- group.ms / after.group.ms
  return(f.value)
})
mean(Fs)
```

```
## [1] 1.060446
```

Plot the distribution of the 1000 F-values:

```r
hist(Fs, col="grey", border="white", breaks=50, freq=FALSE)
xs <- seq(from=0,to=6,length=100)
lines(xs, df(xs, df1 = p - 1, df2 = N - p), col="red")
```



Overlay the theoretical F-distribution, with parameters df1=p - 1, df2=N - p.

This is the distribution which is used to calculate the p-values for the ANOVA table produced by anova().

Calculation of Linear Model

Collinearnity Assessment

QUESTION 2.7.1

Which of the above design matrices does NOT have the problem of collinearity?

Ans: E.

Explanation You can check in R, the rank of the E matrix is equal to the number of columns, so all of the columns are independent.

```r
m = matrix(c(1,1,1,1,0,0,1,1,0,1,0,1,0,0,0,1),4,4)

qr(m)$rank
```

```
## [1] 4
```

Let's use the example from the lecture to visualize how there is not a single best beta-hat, when the design matrix has collinearity of columns.

An example can be made with:

```
sex <- factor(rep(c("female","male"),each=4))
trt <- factor(c("A","A","B","B","C","C","D","D"))
```

The model matrix can then be formed with:

```
X <- model.matrix( ~ sex + trt)
```

And we can see that the number of independent columns is less than the number of columns of X:

```
qr(X)$rank
```

```
## [1] 4
```

Suppose we observe some outcome, Y. For simplicity we will use synthetic data:

```
Y <- 1:8
```

Now, we will fix the value for two beta's and optimize the remaining betas. We will fix beta_male and beta_D. And then we will find the optimal value for the remaining betas, in terms of minimizing sum((Y - X beta)^2).

The optimal value for the other betas is the one that minimizes:

`sum( ( (Y - X* beta_male - X** beta_D) - X*** beta*** )^2 )`

Where X* is the male column of the design matrix, X** is the D column, and X*** has the remaining columns.

So all we need to do is redefine Y as Y* = Y - X* beta_male - X** beta_D and fit a linear model. The following line of code creates this variable Y*, after fixing beta_male to a value 'a', and beta_D to a value, 'b':

```
makeYstar <- function(a,b) Y - X[,2] * a - X[,5] * b
```

Now we'll construct a function which, for a given value a and b, gives us back the the sum of squared residuals after fitting the other terms.

```
fitTheRest <- function(a,b) {
  Ystar <- makeYstar(a,b)
  Xrest <- X[,-c(2,5)]
  betarest <- solve(t(Xrest) %*% Xrest) %*% t(Xrest) %*% Ystar
  residuals <- Ystar - Xrest %*% betarest
  sum(residuals^2)
}
```

QUESTION 2.7.2

What is the sum of squared residuals when the male coefficient is 1 and the D coefficient is 2, and the other coefficients are fit using the linear model solution?

```
fitTheRest(1,2)
```

```
## [1] 11
```

We can apply our function fitTheRest to a grid of values for beta_male and beta_D, using the outer() function in R. outer() takes three arguments, a grid of values for the first argument, a grid of values for the second argument, and finally a function which takes two arguments.

Try it out:

```
outer(1:3,1:3,`*`)
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    2    4    6
## [3,]    3    6    9
```

We can run fitTheRest on a grid of values, using the following code (the Vectorize() is necessary as outer() requires only vectorized functions):

```
outer(-2:8,-2:8,Vectorize(fitTheRest))
```

```
##       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]
##  [1,]  102   83   66   51   38   27   18   11    6     3     2
##  [2,]   83   66   51   38   27   18   11    6    3     2     3
##  [3,]   66   51   38   27   18   11    6    3    2     3     6
##  [4,]   51   38   27   18   11    6    3    2    3     6    11
##  [5,]   38   27   18   11    6    3    2    3    6    11    18
##  [6,]   27   18   11    6    3    2    3    6   11    18    27
##  [7,]   18   11    6    3    2    3    6   11   18    27    38
##  [8,]   11    6    3    2    3    6   11   18   27    38    51
##  [9,]    6    3    2    3    6   11   18   27   38    51    66
## [10,]    3    2    3    6   11   18   27   38   51    66    83
## [11,]    2    3    6   11   18   27   38   51   66    83   102
```

QUESTION 2.7.3

In the grid of values, what is the smallest sum of squared residuals? Ans: 2

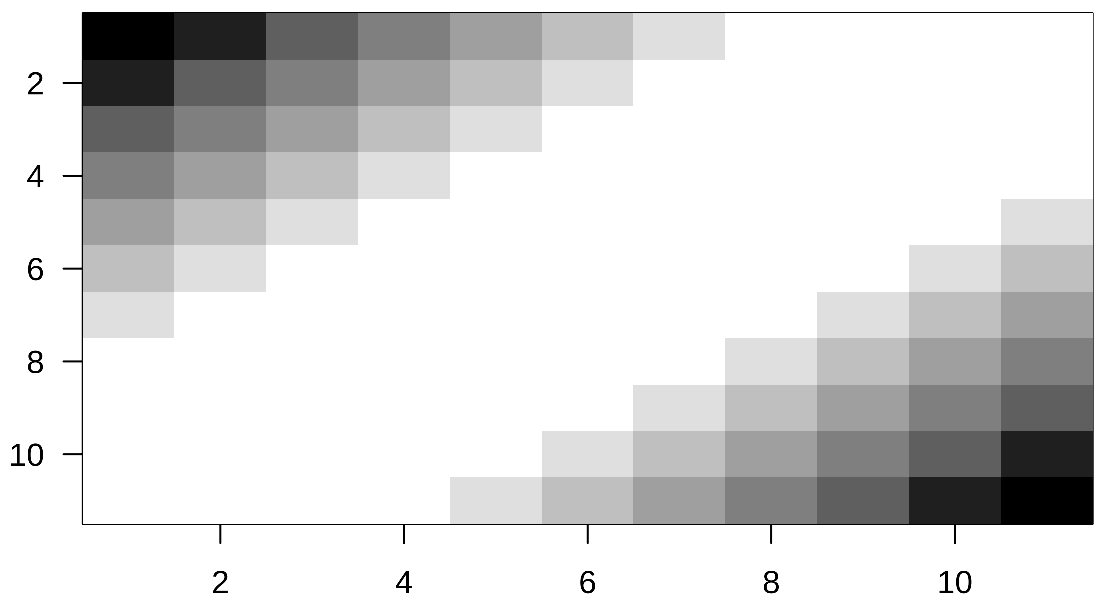
It's fairly clear from just looking at the numbers, but we can also visualize the sum of squared residuals over our grid with the imagemat() function from rafalib:

```
# install_github("ririzarr/rafalib")
library(rafalib)
```

```
## Loading required package: RColorBrewer
```

```
## Warning: package 'RColorBrewer' was built under R version 3.1.3
```

```r
imagemat(outer(-2:8,-2:8,Vectorize(fitTheRest)))
```



There is clearly not a single beta which optimizes the least squares equation, due to collinearity, but an infinite line of solutions which produce an identical sum of squares values.

QR Assessment

We will use the spider dataset to try out the QR decomposition as a solution to linear models. Load the full spider dataset, by using the code in the Interactions and Contrasts book page. Run a linear model of the friction coefficient with two variable (no interactions):

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

The solution we are interested in solving is:

```r
betahat <- coef(fit)
```

So for our matrix work,

```r
Y <- matrix(spider$friction, ncol=1)
X <- model.matrix(~ type + leg, data=spider)
```

In the material on QR decomposition, we saw that the solution for beta is:

```
R beta = Q^T Y
```

QUESTION 2.8.1

What is the first row, first column element in the Q matrix for this linear model?

```
QR <- qr(X)
(Q <- qr.Q( QR ))[1,1]
```

```
## [1] -0.05954913
```

QUESTION 2.8.2

What is the first row, first column element in the R matrix for this linear model?

```
(R <- qr.R( QR ))[1,1]
```

```
## [1] -16.79286
```

QUESTION 2.8.3

What is the first row, first column element of Q^T Y (use crossprod() as in the book page)

```
(crossprod(Q,Y))[1,1]
```

```
## [1] -13.79873
```

Finally convince yourself that the QR gives the least squares solution by putting all the pieces together:R^-1 (Q^T Y)

```
(betahat <- backsolve(R, crossprod(Q,Y) ) )
```

```
##             [,1]
## [1,]  1.0539153
## [2,] -0.7790071
## [3,]  0.1719216
## [4,]  0.1604921
## [5,]  0.2813382
```

compared to betahat