Introduction to Linear Models

Mike Love

Matrix Algebra

Matrix Algebra

In this book we try to minimize mathematical notation as much as possible. Furthermore, we avoid using calculus to motivate statistical concepts. However, Matrix Algebra (also referred to as Linear Algebra) and its mathematical notation greatly facilitates the exposition of the advanced data analysis techniques covered in the remainder of this book. We therefore dedicate a chapter of this book to introducing Matrix Algebra. We do this in the context of data analysis and using one of the main applications: Linear Models.

We will describe three examples from the life sciences: one from physics, one related to genetics, and one from a mouse experiment. They are very different, yet we end up using the same statistical technique: fitting linear models. Linear models are typically taught and described in the language of matrix algebra.

Motivating Examples

Falling objects

Imagine you are Galileo in the 16th century trying to describe the velocity of a falling object. An assistant climbs the Tower of Pisa and drops a ball, while several other assistants record the position at different times. Let's simulate some data using the equations we know today and adding some measurement error:

```
set.seed(1)
g <- 9.8 ##meters per second
n <- 25
tt <- seq(0,3.4,len=n) ##time in secs, note:
## we use tt because t is a base function
d <- 56.67  - 0.5*g*tt^2 + rnorm(n,sd=1) ##meters
```

The assistants hand the data to Galileo and this is what he sees:

Falling objects

```
mypar()
plot(tt,d,ylab="Distance in meters",xlab="Time in seconds")
```

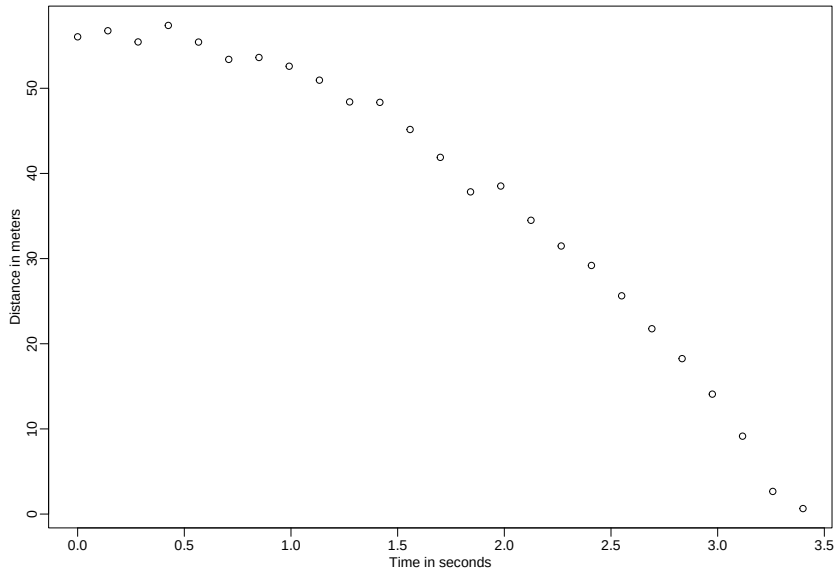


Figure 1: Simulated data for distance travelled versus time of falling object measured with error.

He does not know the exact equation, but by looking at the plot above he deduces that the position should follow a parabola. So he models the data with:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, i = 1, \ldots, n$$

With Y_i representing location, x_i representing the time, and ε_i accounting for measurement error. This is a linear model because it is a linear combination of known quantities (the x's) referred to as predictors or covariates and unknown parameters (the β's).

Father & son heights

Now imagine you are Francis Galton in the 19th century and you collect paired height data from fathers and sons. You suspect that height is inherited. Your data:

```
data(father.son,package="UsingR")
x=father.son$fheight
y=father.son$sheight
```

looks like this:

```
plot(x,y,xlab="Father's height",ylab="Son's height")
```

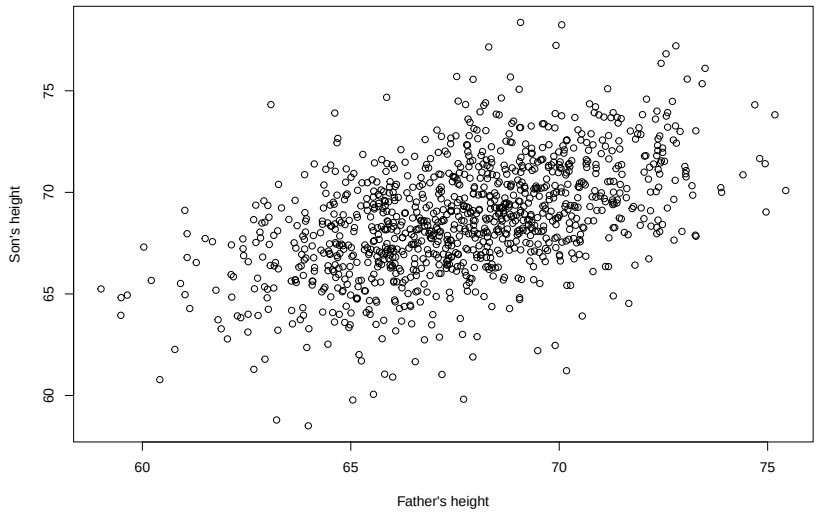


Figure 2: Galton's data. Son heights versus father heights.

Father & son heights

The sons' heights do seem to increase linearly with the fathers' heights. In this case, a model that describes the data is as follows:

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

This is also a linear model with x_i and Y_i, the father and son heights respectively, for the i-th pair and ε_i a term to account for the extra variability. Here we think of the fathers' heights as the predictor and being fixed (not random) so we use lower case. Measurement error alone can't explain all the variability seen in ε_i. This makes sense as there are other variables not in the model, for example, mothers' heights, genetic randomness, and environmental factors.

Random samples from multiple populations

Here we read-in mouse body weight data from mice that were fed two different diets: high fat and control (chow). We have a random sample of 12 mice for each. We are interested in determining if the diet has an effect on weight. Here is the data:

```
dat <- read.csv("femaleMiceWeights.csv")
mypar(1,1)
stripchart(Bodyweight ~ Diet, data=dat, vertical=TRUE,
           method="jitter", pch=1, main="Mice weights")
```

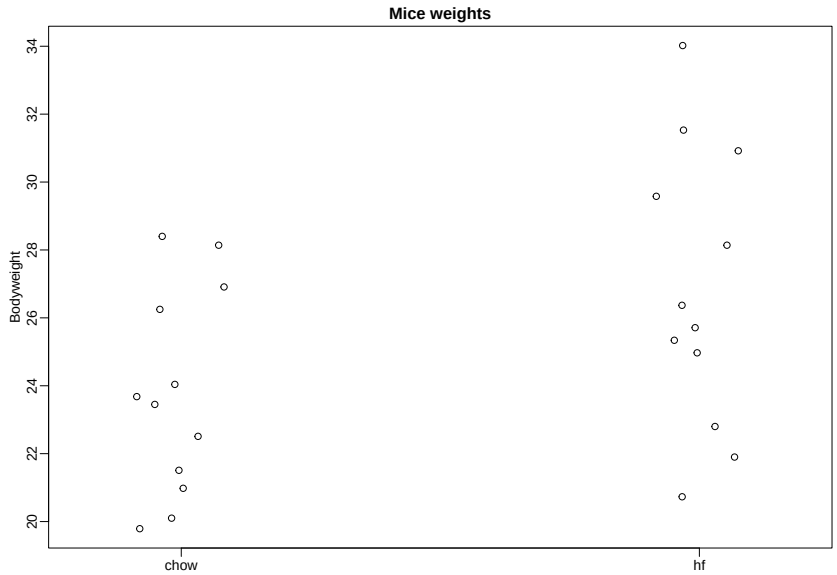


Figure 3: Mouse weights under two diets.

We want to estimate the difference in average weight between populations. We demonstrated how to do this using t-tests and confidence intervals, based on the difference in sample averages. We can obtain the same exact results using a linear model:

Linear models in general

We have seen three very different examples in which linear models can be used. A general model that encompasses all of the above examples is the following:

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

$$Y_i = \beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j} + \varepsilon_i, i = 1, \ldots, n$$

Note that we have a general number of predictors p. Matrix algebra provides a compact language and mathematical framework to compute and make derivations with any linear model that fits into the above framework.

Estimating parameters

For the models above to be useful we have to estimate the unknown β s. In the first example, we want to describe a physical process for which we can't have unknown parameters. In the second example, we better understand inheritance by estimating how much, on average, the father's height affects the son's height. In the final example, we want to determine if there is in fact a difference: if $\beta_1 \neq 0$.

The standard approach in science is to find the values that minimize the distance of the fitted model to the data. The following is called the least squares (LS) equation and we will see it often in this chapter:

$$\sum_{i=1}^{n} \left\{ Y_i - \left(\beta_0 + \sum_{j=1}^{p} \beta_j x_{i,j} \right) \right\}^2$$

Once we find the minimum, we will call the values the least squares estimates (LSE) and denote them with $\hat{\beta}$. The quantity obtained when evaluating the least squares equation at the estimates is called the residual sum of squares (RSS). Since all these quantities depend on Y, *they are random variables*. The $\hat{\beta}$ s are random variables and we will eventually perform inference on them.

Falling object example revisited

Thanks to my high school physics teacher, I know that the equation for the trajectory of a falling object is:

$$d = h_0 + v_0 t - 0.5 \times 9.8 t^2$$

with h_0 and v_0 the starting height and velocity respectively. The data we simulated above followed this equation and added measurement error to simulate `n` observations for dropping the ball ($v_0 = 0$) from the tower of Pisa ($h_0 = 56.67$). This is why we used this code to simulate data:

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

Falling object example revisited

Here is what the data looks like with the solid line representing the true trajectory:

```
plot(tt,y,ylab="Distance in meters",xlab="Time in seconds")
lines(tt,f,col=2)
```

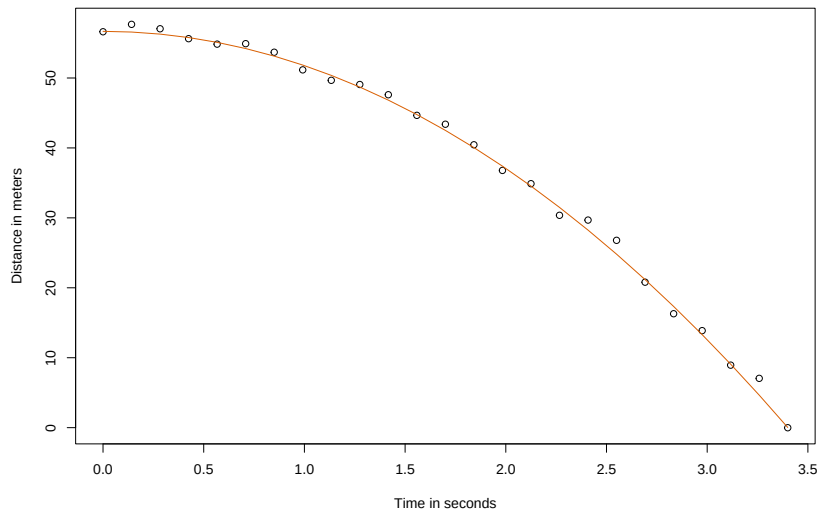


Figure 4: Fitted model for simulated data for distance travelled versus time of falling object measured with error.

But we were pretending to be Galileo and so we don't know the parameters in the model. The data does suggest it is a parabola, so we model it as such:

$$Y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i, i = 1, \ldots, n$$

How do we find the LSE?

The lm function

In R we can fit this model by simply using the lm function. We will describe this function in detail later, but here is a preview:

```
tt2 <-tt^2
fit <- lm(y~tt+tt2)
summary(fit)$coef
```

```
##               Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) 57.1047803  0.4996845 114.281666 5.119823e-32
## tt          -0.4460393  0.6806757  -0.655289 5.190757e-01
## tt2         -4.7471698  0.1933701 -24.549662 1.767229e-17
```

It gives us the LSE, as well as standard errors and p-values.

Part of what we do in this section is to explain the mathematics behind this function.

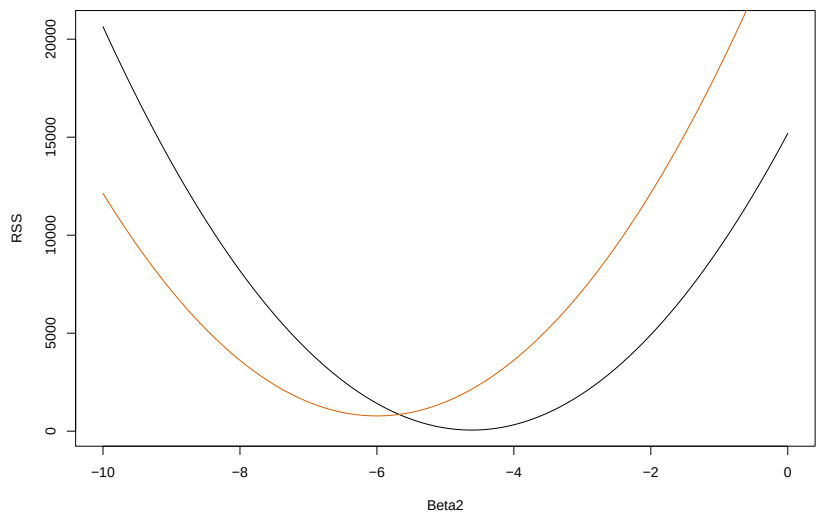
The least squares estimate (LSE)

Let's write a function that computes the RSS for any vector β:

```
rss <- function(Beta0,Beta1,Beta2){
  r <- y - (Beta0+Beta1*tt+Beta2*tt^2)
  return(sum(r^2))
}
```

So for any three dimensional vector we get an RSS. Here is a plot of the RSS as a function of β_2 when we keep the other two fixed:

```
Beta2s<- seq(-10,0,len=100)
plot(Beta2s,sapply(Beta2s,rss,Beta0=55,Beta1=0),
     ylab="RSS",xlab="Beta2",type="l")
##Let's add another curve fixing another pair:
Beta2s<- seq(-10,0,len=100)
lines(Beta2s,sapply(Beta2s,rss,Beta0=65,Beta1=0),col=2)
```



The least squares estimate (LSE)

Trial and error here is not going to work. Instead, we can use calculus: take the partial derivatives, set them to 0 and solve. Of course, if we have many parameters, these equations can get rather complex. Linear algebra provides a compact and general way of solving this problem.