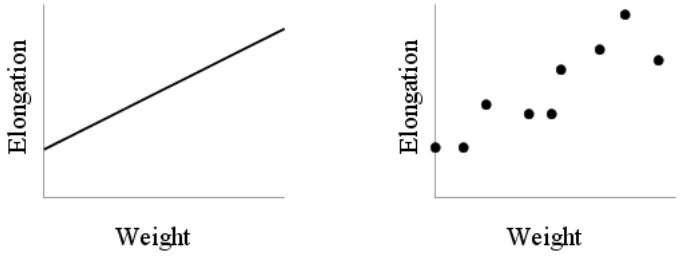
Regression

Contents

1 Introduction

In every walk of science we come across variables related to one another. Mathematical formulations of such relations occupy an important place in scientific research. When only two variables are involved we often plot a curve to show functional relationships as in the left hand figure below which depicts the relation between the elongation of a steel wire and the weight suspended from it.



However, what the experimenter sees in the laboratory is not this neat line. He sees a bunch of points as shown in the right hand figure. One idealizes these points to get the straight line. This process of extracting an ideal (and, desirably, simple) relation between variables based on observed data is called **regression analysis**.

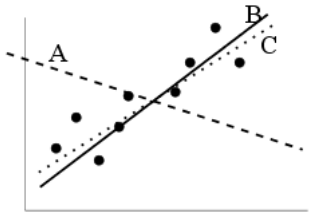
Often the relationship between two variables has a causal flavor. For example, in our example, we like to think of the weight as *causing* the elongation. Thus, weight is the variable that is under direct control of the experimenter. So we call it the **explanatory** variable. The other variable (elongation) is called the **response**. If we have a single explanatory variable and a single response (as here), then we have a **bivariate** regression problem. A **multivariate** regression problem has more than one explanatory variable, but (usually) a single response. We shall talk about bivariate regression first.

2 Bivariate regression

A typical regression analysis proceeds in four steps:

1. First we postulate a *form* of the relation, like a straight line or a parabola or an exponential curve. The form involves unknown numbers to be determined. For instance, a straight line has equation $y = a + bx$, where a, b are unknown numbers to be determined. Typically the form of the relation is obtained by theoretical considerations and/or looking at the scatterplot.

2. Next we have to decide upon an objective criterion of goodness of fit. For instance, in the plot below, we can see that line A is a bad fit. But both B and C appear equally good to the *naked eye*. An objective criterion for goodness of fit is needed to choose one over the other.



3. Once we have chosen the form as well as the criterion, it is a matter of routine computation to find a relation of our chosen form that fits the data best. Statistics textbooks spend many pages elaborating on this step. However, for us it is just a matter of invoking R.

4. Last but not the least, we have to check whether the "best" relation is indeed what we expected. (Common sense above routine computation!)

3 Ordinary least squares

We start with ordinary least squares.

```
# Linear regression with heteroskedastic measurement errors
# Construct and plot data set of SDSS quasars with i between
# 18 and 22
qso = read.table(
'http://astrostatistics.psu.edu/MSMA/datasets/SDSS_17K.dat',
header=T,fill=T)
qso = na.omit(qso)
dim(qso)
summary(qso)
```

In this example we shall work with only a subset of the data where `i_mag` lies in the range (18,22).

```
qso1 = with(qso,qso[(i_mag<22) & (i_mag>18),])
dim(qso1)
summary(qso1)
attach(qso1)
plot(i_mag, z_mag, col=grey(0.6),pch='.',
     xlab="SDSS i (mag)", ylab="SDSS z (mag)")
```

Recall that we had written a function for making a scatterplot with error bars.

```
errplot = function(x,y,
                   xerrlo,
                   yerrlo,
                   xerrhi=xerrlo,
                   yerrhi=yerrlo,...) {
```

```
   plot(x,y,xlim=range(x-xerrlo,x+xerrhi),
            ylim=range(y-yerrlo,y+yerrhi),...)
   segments(x,y-yerrlo,x,y+yerrhi)
   segments(x-xerrlo,y,x+xerrhi,y)
}
```

We shall now use it.

```
dev.new() #Open a new graphics window.
          #Note the numbers in the titlebars
          #of the two graphics windows.

errplot(i_mag, z_mag, Err_i, Err_z,
        pch=20, cex=0.1, col=grey(0.5),
        xlab="SDSS i (mag)", ylab="SDSS z (mag)")
```

If we denote the `z_mag` and `i_mag` values by y_i's and x_i's, respectively, then the linear regression equation is

$$y_i = \alpha + \beta x_i + \epsilon_i, \qquad (*)$$

where the errors ϵ_i's are assumed to be a random sample from $N(0, \sigma^2)$ distribution for some unknown σ^2.

```
fit_ols = lm(z_mag~i_mag)
```

Notice how the equation (*) is represented by the formula

```
z_mag ~ i_mag
```

The '$\sim$' denotes the '=' in (*). To its left we write the name of the dependent variable (y). All the explanatory variables are listed to the right. The constant term in included by default, unless we use

```
z_mag ~ i_mag - 1
```

where the `-1` specifically forbids the inclusion of the intercept term.

```
fit_ols #Prints the estimated coefficients
dev.set(??) #Go back to the simple scatterplot.
            #Replace ?? by its number
abline(fit_ols$coef)
summary(fit_ols) #Prints estimated sigma square and
                 #the results of tests
```

Here is part of the output.

```
Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.715328   0.142582  -5.017 5.33e-07 ***
i_mag        1.023566   0.006834 149.776  < 2e-16 ***
```

To make sense of the last two columns recall how statistical tests are done. First the data are summarized into a single number (the test statistic) and it is compared against a theoretical distribution using p-value. In this case we are performing two such tests, one for each coefficients. For the intercept term we are testing if that term is 0 or not. The value of the test statistic (whose formula need not concern us here) turns out to be 0.843. When compared to the expected distribution (expected if the intercept were really 0) the p-value is 0.4, which is much larger than the rule-of-thumb threshold of 0.05. So we conclude that α could really be 0. However, the p-value corresponding to the `i_mag` term is very small, and so we conclude that β must be nonzero. So let us drop the intercept term from our model:

```
fit_ols2 = lm(z_mag~i_mag-1)
summary(fit_ols2)
abline(a=0,b=fit_ols2$coef,lty=1,lwd=2,col='red')
```

Let us extract various information.

```
confint(fit_ols2, level=0.997)
```

It is always a good idea to look at some diagnostic plots.

```
dev.new()
plot(fit_ols2, which=1, caption='', sub.caption='',
     pch=20, cex=0.3, cex.lab=1.3, cex.axis=1.3)
```

Let us understand this step carefully. First we check the `class` of the object `fit_ols2`:

```
class(fit_ols2)
```

We see that it is an object of class `lm`. Thus the generic `plot` command here actually means the `plot.lm` function. If we look up its help we see that it

is capable of producing six different diagnostic plots. Which plot we want is determined by the `which` parameter. Here we have used `which=1` to produce a plot of the residuals against the fitted values.

Exercise 1: How can you produce all the 6 plots on the same page? ■

The very first plot shows something that draws our attention. The residuals seem to fan out. In other words, the homoskedasticity assumption (assumption of equal variances) seems to be violated. Let us make another residual plot, but now with the regressor `i_val` along the horizontal axis. By the way, the wavy red line is an attempt to show the general pattern of the residuals.

```
with(qso1,plot(i_mag,fit_ols2$res,pch='.'))
```

Clearly, the spread of the residuals increase with increasing `i_mag`. There does not seem any physical reason for this. However, there is a simple explanation possible, as we show next.

4 Mixed effects model

In such a situation it is wiser to fit a random effects model, where the slope coefficient is allowed to be random. Consider the following example to understand the interpretation. We know that for a more or less homogeneous group of adults there is an approximate linear relation between weight and height. However, these may differ slightly from one group to another (e.g., men and women, or young and old). If, however, we merely a draw a random sample from the entire population, then our sample will consist of representatives of different groups. A random effects model captures this scenario. A random effects model owes its name to the fact that the coefficient in it is random. A mixed effects model is more general: it allows some coefficients to be random, keeping the rest fixed. The R package to handle (non-linear) mixed effects models is `nlme`.

```
library(nlme)
```

There is a little annoying feature of the `lme` function that we are about to use. It wants the data to be *grouped*. For instance, if we have height-weight data for people of different countries, then this data set may be considered as grouped by nationality. Such a grouping variable is naturally present in most social science studies. However, we do not have any grouping variable. So we create one artificially here.

```
qso1.grp = groupedData(z_mag ~ i_mag | grp,
```

```
                        data=data.frame(qso1,grp=1))
```

Notice the syntax carefully. The dependent variable `z_mag` comes first, then comes '∼', followed by the independent variable `i_mag`, followed by '—', and the grouping variable `grp` (which is just a dummy variable taking value 1) comes last.

```
fit_lme = lme(z_mag ~ i_mag-1,data=qso1.grp)
fit_lme
```

5 Weighted least squares

Next let us take the error bars of `z_mag` into account. A case with large amount of error deserves less importance. Accordingly we shall use the reciprocal of `Err_z` as a weight variable. We shall illustrate the procedure first with OLS. As we are going to compute the reciprocal of `Err_z`, we should better guard against values that are too close to 0.

```
Err_z[Err_z<=0.03] = 0.03

fit_ols.wt = lm(z_mag~i_mag-1, x=T, weights=1/Err_z)
summary(fit_ols.wt)
dev.set(??) #Replace ?? with the no. in the titlebar of
            #the window showing the original scatterplot.
abline(a=0,fit_ols.wt$coef,lty=2,lwd=2)
```

A very similar syntax works for the random effects model as well:

```
qso2 = with(qso1, qso1[Err_z>0, ])
qso2.grp = groupedData(z_mag ~ i_mag | grp,
                       data=data.frame(qso2,grp=1))

fit_lme.wt = lme(z_mag ~ i_mag-1,
                 weights=~I(1/Err_z),data=qso2.grp)

fit_lme
```

The specification of the weight variable is rather irritating. First, the weight must be given as a formula (which means you have to precede the variable by a

$\sim$). Second, if you are not using a variable directly, but rather some function of the variable (as in our case, we are using the reciprocal of `Err_z`) then we must "protect" the expression using the `I` function.

6 Major axis regression

In both the approaches so far, we are minimizing the total squared *vertical* errors. This is a natural thing to do when the variable represented by the horizontal axis is known to be free of errors (e.g., controlled by the experimenter). However, our situation is not like this. We are really not interested in predicting `z_mag` based on `i_mag`. We are merely interested in exploring their linear relation. So it makes more sense to choose the line which minimizes some criterion symmetric w.r.t. the two variables. One such criterion is the total of squared *perpendicular* errors. The resulting regression is called **major axis regression**. A related approach called **standard major axis regression** minimizes the total areas of the all the right-angled triangles with the fitted line as the base and the data points as the vertices.

```
install.packages('lmodel2')
library(lmodel2)
lmodel2(z_mag~i_mag)
```

7 Nonlinear Least Squares

7.1 Simple fitting

So far we have worked with only linear regression models. By linear we mean linear *in the parameters*, and not necessarily linear in the regressors. Now we shall learn how R can fit a nonlinear regression model. The main work horse function here is `nls`.
As an example we consider the univariate radial brightness distribution of elliptical galaxies. These distribution are monotonic, bright in the center and fading with radial distance, but have proved difficult to parametrize and understand. Sérsic (1968) proposed a generalized de Vaucouleurs law that is still used today,

$$\log_{10} I(r) = \log_{10} I_e - b_n[(r/r_e)^{1/n} - 1],$$

where $b_n \approx 0.868n - 0.142$, $0.5 < n < 16.5$, and the observed surface brightness profile is measured in units of

$$\mu(r) \propto -2.5 \log_{10} I(r) \text{ mag/arcsec}^2.$$

First we read the data set.

```
# Fit Sersic function to NGC 4472 elliptical galaxy
#surface brightness profile
NGC4472 =
read.table(
"http://astrostatistics.psu.edu/MSMA/datasets/NGC4472_profile.dat",
header=T)
dim(NGC4472)
names(NGC4472)
```

Here $I(r)$ is `surf_mag` and r is `radius`. Then we fit the model. Notice how in the following line we have considered $\log_{10} I_e$ as a single parameter `log10I.e`.

```
NGC4472.fit = nls(surf_mag ~
  -2.5*log10I.e+2.5*(0.868*n-0.142)*((radius/r.e)^(1/n)-1),
  data=NGC4472,
  start=list(log10I.e=log10(20.),r.e=120.,n=4.),
  trace=T)
```

Here is a typical output

```
36142.99 :    1.30103 120.00000   4.00000
3.613789 :   -9.120594 179.091209   5.211069
0.6414811 :   -9.300810 239.956548   5.833813
0.08432762 :   -9.340969 265.407791   5.948181
0.07817004 :   -9.342811 267.630197   5.952091
0.07816977 :   -9.342816 267.642605   5.952098
```

Let us understand the command first. We have passed 4 arguments to `nls`: the formula, the data set, the starting values of the parameters, and the request to print the iteration steps. The formula, as we can guess, may be just anything. Any name that is used in the rhs but does not occur as a variable name, is treated as a parameter. A complicated numerical optimization step is done to estimate the parameters. The nonlinear optimization procedure is not very robust, and depends very much on the choice of the starting values. Unfortunately, it may be numerically unstable also. To see this let us try to fit the same model in a slightly different form:

```
NGC4472.fit2 = nls(surf_mag ~
  -2.5*log10(I.e)+2.5*(0.868*n-0.142)*((radius/r.e)^(1/n)-1),
  data=NGC4472,
  start=list(I.e=20,r.e=120.,n=4.),
  trace=T)
```

Now the output is

```
36142.99 :   20 120   4
Error in numericDeriv(form[[3L]], names(ind), env) :
  Missing value or an infinity produced when evaluating the model
In addition: Warning message:
In log(-459.933543897112, 10) : NaNs produced
```

The optimistic moral is that even if `nls` fails for fit a model, do not give up, it is still worth making minor tweaks!
Let us now take a look inside the fitted model.

```
summary(NGC4472.fit)
```

Here is part of the output, which is very similar to that for linear regression.

```
Parameters:
          Estimate Std. Error t value Pr(>|t|)
log10I.e  -9.34282    0.02191 -426.34   <2e-16 ***
r.e      267.64261    7.05769   37.92   <2e-16 ***
n          5.95210    0.09718   61.25   <2e-16 ***
```

Let's next plot the output.

```
# Plot NGC 4472 data and best-fit model
attach(NGC4472)
plot(radius,surf_mag, pch=20,
     xlab="r (arcsec)", ylab=expression(mu ~~ (mag/sq.arcsec)),
     ylim=c(16,28), cex.lab=1.5, cex.axis=1.5)
lines(radius,fitted(NGC4472.fit))
```

Exercise 2: Repeat the same procedure with the data from

`http://astrostatistics.psu.edu/MSMA/datasets/NGC4406_profile.dat`

■

7.2 Diagnostics

Behind the scene, `nls` assumed that the differences between the observed $I(r)$'s and that predicted by the complicated formula, are actually due to some random errors, which form a random sample from $N(0, \sigma^2)$. The `logLik` function shows the log of the likelihood value which measures the chance of the data arising from the estimated model. Indeed, it is this quantity that `nls` maximizes numerically.

```
logLik(NGC4472.fit) #larger the better
```

But this one-number summary has little diagnostic value. We should better look at the residuals.

```
# Residual plot
plot(radius,residuals(NGC4472.fit),
     xlab="r (arcsec)", ylab="Residuals",
     pch=20, cex.lab=1,   cex.axis=1.5)
lines(supsmu(radius, residuals(NGC4472.fit), span=0.05), lwd=2)
abline(h=0)
```

Here the function supsmu denotes "super smoother", and is an *ad hoc* way to draw a curve through a cloud of points. Now let us check if the normality assumption was OK.

```
# Test for normality of residuals
stan.res = residuals(NGC4472.fit) / summary(NGC4472.fit)$sigma
qqnorm(stan.res)
abline(a=0,b=1)
shapiro.test(stan.res)
```

Well, both the plot and the Shapiro-Wilk's test votes in favor of normality. Even though the normality assumption is not a suspect here, let us illustrate a bootstrap procedure that works even without this assumption. The technique is to draw samples from the residuals, and add them back to the fitted values to get new simulated values of `surf_mag`. Then we perform the entire analysis with the new value to get a new set of estimates. We repeat this process many times (each time sampling afresh from the original residuals). This gives us many values of the estimates, based on which we can compute quantities like standard error and confidence intervals.

```
# Bootstrap parameter estimates
install.packages('nlstools')
library(nlstools)
NGC4472.boot = nlsBoot(NGC4472.fit)
summary(NGC4472.boot)
hist(NGC4472.boot$coefboot[,3], breaks=50, prob=T)
curve(dnorm(x,m=5.95, sd=0.10), ylim=c(0,5),add=T)
```

Note that with the `add=T` option, curve automatically knows the horizontal plot range. The normal distribution definitely looks like a good fit!

8 LOESS

LOESS (which is derived from LOWESS, standing for LOcally WEighted Scatterplot Smoother) is an ad hoc smoother, somewhat akin to `supsmu` in spirit. You may like to consier this as an objective way to draw a freehand curve through a data cloud. Basically it is made up of tiny pieces of simple polynomials, each adjusting to the pattern of the points lying nearby. The method works nicely only when we have a pretty dense data cloud.
We shall try it out on the SDSS data set. The scatterplot here has lots of points. In fact, the points are so very dense that we need to plot a smoothed version of them. We shall employ our familiar `ash` package for the smoothing.

```
qso = na.omit(read.table(
"http://astrostatistics.psu.edu/datasets/SDSS_QSO.dat",
head=T,fill=T))
q1 = qso[qso[,10] < 0.3,] ; q1 = q1[q1[,12]<0.3,]
dim(q1) ; names(q1) ; summary(q1)
r_i = with(q1, r_mag - i_mag)
z = q1[,4]
install.packages('ash')
library(ash)
nbin = c(500, 500)
ab = matrix(c(0.0,-0.5,5.5,2.), 2,2)
bins = bin2(cbind(z,r_i), ab, nbin)
f = ash2(bins, c(5,5))
names(f)
logfz = log10(f$z)
image(f$x, f$y, logfz,
      col=gray(seq(0.2,0.5,by=0.05)), zlim=c(-1.8,3.0),
      main="SDSS quasars", xlab="Redshift", ylab="r-i")
contour(f$x, f$y, logfz, zlim=c(0.1,5) ,nlevels=5, add=T)
```

Then we add the LOESS curve.

```
z1 = sort(z)
r_i1 = r_i[order(z)]
loct1 = loess(r_i1~z1, span=0.1)
lines(z1, predict(loct1), lwd=2, col=2)
```

We can also fit LOESS on part of the data.

```
z2 = z1[z1>2.5]
```

```
r_i2 = r_i1[z1>2.5]
loct2 = loess(r_i2~z2, span=0.5, data.frame(x=z2,y=r_i2))
lines(z2, predict(loct2), lwd=2, col=3)
```

A LOESS fit is based on an ad hoc piecewise polynomial equation, so there isn't much sense in preserving the coefficients of the polynomials. It is only the fit that matters. Here is how we can save the fitted values.

```
x1 = seq(0.0, 2.5, by=0.02)
x2 = seq(2.52, 5.0, by=0.02)
loctdat1 = predict(loct1, data.frame(z1=x1))
loctdat2=predict(loct2, data.frame(z2=x2))
write(rbind(x1,loctdat1), sep=' ', ncol=2, file='qso.txt')
write(rbind(x2,loctdat2), sep=' ', ncol=2, file='qso.txt', append=T)
```

9 Quantile regression

All the regression techniques encountered so far try to fit a curve through the centre of a data cloud. Quantile regression extends this idea: it tries to fit curves such that a given fraction of the data cloud is below it, the rest being above. If the fraction is 0.5, then we get yet another way to fit a curve through the center of the data cloud. But if we fit two curves through say 0.1 and 0.9 then the two curves will hold between them the central 80% of the data cloud.

9.1 Linear

If we are interested in curves that are linear (in the parameters; the curves may bend), the `rq` function of the package `quantreg` does the trick.

```
# Linear quantile regression
install.packages('quantreg')
library(quantreg)
fit_rq = rq(z_mag~i_mag, tau=c(0.10,0.50,0.90))
fit_rq
plot(i_mag, z_mag,
     pch=20, cex=0.1, col=grey(0.5),
     xlab="SDSS i (mag)", ylab="SDSS z (mag)")
abline(coef(fit_rq)[,1])
abline(coef(fit_rq)[,2])
abline(coef(fit_rq)[,3])
```

9.2 Nonlinear

If we want to use formulas nonlinear in the parameters, then the function `rqss` from the package `MatrixModels` is what we need.

```
# Nonlinear quantile regression
install.packages('MatrixModels')
library(MatrixModels)
fit_rqss.1 = rqss(z_mag~qss(i_mag), data=qso1, tau=0.10)
fit_rqss.5 = rqss(z_mag~qss(i_mag), data=qso1, tau=0.50)
fit_rqss.9 = rqss(z_mag~qss(i_mag), data=qso1, tau=0.90)
plot.rqss(fit_rqss.1, rug=F, ylim=c(17,23), titles='')
points(i_mag, z_mag, cex=0.1, pch=20, col=grey(0.5))
plot.rqss(fit_rqss.1, bands='pt',coverage=0.99, shade=F,
          rug=F, add=T, titles='', lwd=2)
plot.rqss(fit_rqss.5, bands='pt',coverage=0.99, shade=F,
          rug=F, add=T, titles='', lwd=2)
plot.rqss(fit_rqss.9, bands='pt',coverage=0.99, shade=F,
          rug=F, add=T, titles='', lwd=2)
par(mfrow=c(1,1))
```

10 Robust regression

Least squares regression is *non-robust* in the sense that it is easily affected by an *outier*. The reason is that for an outlier the error is large, and its square is even larger. So even a few outliers can cause an otherwise good fit to appear bad. We shall now discuss *robust* techniques that are less affected by outliers.

10.1 Least Trimmed Square (LTS) regression

In this method we do not sum the squares of *all* the residuals, but only a subset consisting of the smaller residuals. This technique is implemented in the `ltsreg` function of the `MASS` package.

```
library(MASS)
fit_lts = ltsreg(z_mag~i_mag,data=qso1)
fit_lts
```

10.2 M-estimation

In many standard regression techniques we minimize some badness-of-fit criterion (e.g., sum of the squared errors in OLS). If the criterion happens to be

differentiable w.r.t. the parameters, then we can equate the derivative to 0. M-estimation attacks the problem at this level: it suggests tweaking this equation to make the regression robust.
Typically to fit a straight line $y = \alpha + \beta x$ to a data set (x_i, y_i) by M-estimation we solve an equation of the form

$$\sum \psi(y_i - \alpha - \beta x_i) = 0.$$

The theory of M-estimation suggests various ψ functions. One popular choice (the default in R) is Huber's ψ-function:

$$\psi(u) = \min\{1, \tfrac{k}{|u|}\},$$

where k is a suitable constant.

```
library(MASS)
fit_M = rlm(z_mag~i_mag, data=qso2,method='M', weights=1/Err_z)
summary(fit_M)
plot(i_mag,z_mag,pch='.')
abline(fit_M$coef,lty=3, lwd=3)
```

You may try the following to get a sequence of diagnostic plots.

```
plot(fit_M)
```

10.3 Theil-Sen regression

This is an entirely different approach to robust regression. Here we take pairs of points at a time, and find the slope of the line joining them. Then we take the median of these slopes. We choose the intercept so that the line passes through the mean.

```
# Theil-Sen regression line
install.packages('zyp')
library(zyp)
fit_ts = zyp.sen(z_mag~i_mag-1, qso1[1:8000,]) #be patient!
confint.zyp(fit_ts) #It still provides a CI for intercept!
plot(i_mag,z_mag)
abline(a=0,b=fit_ts$coef[2], lty=4, lwd=2)
```

Not a particularly nice job!

11 Hints for selected exercises

1.

```
oldpar = par(mfrow=c(2,3))
plot(fit_ols2, which=1:6)
par(oldpar)
```