STAT 270 March 26, 2007

The plan for today is to introduce the idea of "regression", review "simulation", and illustrate how R makes these things easy to do.

Although it is not compulsory in this course, I want to encourage you to try R. I will use R to illustrate regression and simulation, but you don't have to know R to understand the points I am making.

So to start, here is how to get your own free copy of this amazing program — you do have to have your own computer or install privileges on one you can use.

Go to http://cran.r-project.org/
and click on the operating system that your computer uses - then follow instructions to download R.

After R has been installed (usually automatic with very little intervention), just click on the R icon and in 5-10 seconds you should see the R prompt (looks like ">").

Then you can issue commands at the ">" prompt.

The first thing I want to show you is how you can predict one variable from another, assuming you have data available that estimates the relationship (i.e. pairs of data values (x,y) so that you can use x to predict y for values outside the data set.)

Lets start by setting up an x and a y for which we know how the prediction of y from x should be, and we will see if the data alone can be used to make a good guess of this predictive relationship.

Lets use the digits 1,2,...,10 for x and for y lets use x + noise where the noise is generated as independent values from a N(0,1) population.
Then we will plot the result:

```
x=1:10  # let x be a vector (1,2,...,10)
y=x+rnorm(10) # let y be x with some N(0,1) noise added
plot(x,y)  # the simplest plot
```

Note that R replaces the thing on the left of the = sign with what is on the right — the above defines x and y. Also, note that we plot(x,y) not plot(y,x) since the convention is to have the predicted thing as the "ordinate" and the predicting thing as the "abcissa". Finally, the # sign tells R to ignore what follows on the line — this is useful to make comments in a program.

In Chapter 5, we introduced the correlation coefficient as a measure of "linear association" between two variables. Calculating it is easy in R:

```
cor(x,y) # the correlation between vector x and vector y
pause() #this just stops execution until you press return
```

The positive correlation confirms what we see in the plot, that knowing x does give you a pretty good idea of the associated y, although it would be nice to have a formula to do this. This is what "regression" does. But before getting into that, note that a more informative plot can be produced by adding a little more info:

```
plot(x,y, xlab="this is x",ylab="this is y",main="Title of Plot", col="red")
```

Now lets see what R gives us for estimating a "linear model" for predicting y from x. (We talk of "regressing y on x" in order to find a predictive relationship for y based on x.)

```
lm(y~x)  # simple linear regression
```

This tells us the intercept and slope of a line that can be used to poredict y from x. But there is much more info available from R. To capture it, we need to provide a name for the output of the "lm()" command.

```
hold=lm(y~x)  # "hold" is just my name for the output. You can use whatever
```

In R, to find out what is in "hold" we use

```
attributes(hold) # this shows how to get more detail from "hold"
```

For example, you can find the predicted values of y from either of the commands

```
hold[5]
hold$fitted
```

Now it is natural to graph the data along with the predicting line:

```
plot(x,y) # just to replot the data
lines(x,hold$fitted) # add the regression line to the plot
pause()
```

Since we arranged the straight line to be a good predictor, it is not surprising that it does a decent job. But in practice we sometimes want to look at what are called "residuals" which are the errors of prediction IF we were to try to predict y from x for those x that are in our data set (and we know y).

```
hold$residuals
```

and we can plot these as well:

```
plot(x,hold$residuals)
```

Note that if these is a pattern to the residuals (y-pred(y)) as a function of x, then we can improve our prediction of y from x. So a good prediction model would show no pattern in this residual plot.

Lets illustrate this by forming a new predicted variable w.

```
w=x+2*x^2+10*rnorm(10)
plot(x,w)
hold.new=lm(w~x)
lines(x,hold.new$fitted)
pause()
```

This fit is lousy, and the residual plot would emphasize the shortcoming of the fit.

```
plot(x,hold.new$residuals)
```

The plot shows that we need to add a curved portion to the model for a good predictor.

How is the prediction line chosen? It is chosen to minimize the errors of prediction, in some sense. Since it is impossible to reduce all the errors to zero, with a straight line model (in most data sets), we need a way to make the errors small as a group. The usual way to do this is to minimize the sum of squared errors. i.e. $\sum_{i=1}^{n}(y_i - pred(y_i))^2$

This quantity (in our linear model example) can be computed as

```
sum((hold$residuals)^2)
```

although we do not need it for anything.

What is more useful is the size of the prediction errors. Note that the average prediction error is 0.

```
mean(hold$residuals)
```

since if it were not zero, we could move the line a bit to reduce the sum of squared errors.

So we cannot use the mean residual to measure how good our prediction is.

But the SD of residuals does tell us the size of a "typical" residual (error).

```
sd(hold$residuals)
```

and this tells us how far off the true y we are likely to be if we use x to predict y.

In judging the size of the errors, since we have an estimate of the mean and sd of them, it would be helpful to know the distribution – is it normal?

```
hist(hold$residuals)
```

If we can assume normality of the residuals (which will depend on the data itself) we can make statements about the size of error that would not be exceeded 95% of the predictions.

What follows is an example of the outputs from the commands listed above: