Matlab Notes, Part IV

The Matlab “random” command

For lots of purposes it's useful to be able to generate lists of “random” numbers. We say “random” (in quotes) because a computer can only do something according to the instructions of some program that's part of the software. In some sense, numbers “produced according to a recipe” aren't really “random”; truly random events are the result of some process of chance, such as throwing dice or flipping a balanced coin.

Strictly speaking, Matlab generates for us a collection of “pseudo-random” numbers: that is, numbers which “look as if” there were generated by some truly random process. However, since they “act like” random numbers, we will refer to them as random rather than use the longer word “pseudo-random.”

An important point is this. Different sets of random numbers can follow different “patterns”.

Suppose, for example, you had a perfectly balanced roulette wheel where the only possible outcomes were, say, the numbers 0,1,...,9. Repeatedly spinning the wheel and recording the results, you'd expect to see (in the long run) approximately the same number of 0's, 1's, ...,9's: that is, you'd get what's called a uniformly distributed set of random outcomes 0,1,...9.

Suppose, instead, you randomly choose a large number of males from the St. Louis phone book and find out what each one weighs (to the nearest pound): you'd get random numbers like 145,153,185,220,... Tallying up the outcomes, you wouldn't expect to find each of the possible weights occurring with equal frequency. The "middle-sized" male weights would occur more frequently and the more extreme weights (high or low) less frequently. This list of numbers, generated by a random process, is not uniformly distributed: instead, it follows a pattern close to what's called a normal distribution.

Matlab has commands to generate lists of “random” numbers that are distributed “uniformly” or “normally”:

- `rand(1,n)` produces an array with 1 row and n entries which are uniformly distributed numbers chosen from the interval [0,1]. Here, “uniformly distributed” means that if you subdivide the interval [0,1] into k equal subintervals, then (for a large n) you would expect roughly the same number of entries from rand(1,n) to be in each one of the subintervals. More intuitively, we could say that each entry in rand(1,n) is “equally likely” to be in any one subinterval as in any other of the same length.

- `2*rand(1,n)` By doubling the numbers in rand(n), we get a 1 row array with n entries uniformly distributed in the interval [0,2]. We can do similar maneuvers to make the numbers be uniformly distributed in any interval we like.

**Exercise** What command will give an array with 1 row and n entries uniformly distributed in the interval [7,12]?

- `randn(1,n)` produces an array with 1 row and n entries which are “normally distributed” with mean 0 and standard deviation 1. (These numbers may be positive or negative. The normal distribution will be discussed soon in class.)
Because the numbers generated in this way are really pseudo-random, they are “concocted” according to some “recipe” programmed into the software. Such a program must “begin somewhere with some initial ingredients.” Matlab resets the software to the same “initial state” for generating random numbers each time it's loaded. Therefore if you generate a list of 100 uniformly distributed random numbers today, shut down Matlab, and then do it again tomorrow, you'll get the same list (at least if it's the same version of Matlab.)

Within the same Matlab session, however, each time you run “rand” the initial state is changed, so if you execute “rand(1,n)” twice in succession, you'll get different lists.

Use “help rand” and “help randn” for more details.

**Mean, Median and Standard Deviation**

For an array x with 1 row,

- `mean(x)` computes the mean (“average”) of the numbers in x
- `median(x)` computes the "middle-sized" value in x

(If x has an odd number of elements, then median(x) is the middle one in order of size.

If x has an even number of values, then median(x) is the average of the two “in the middle” in order of size.)

```plaintext
x = [5,1,3] median(x) = 3
x = [7,1,5,3] median(x) = (3+5)/2 = 8
```

- `std(x)` computes the standard deviation of numbers in x

The “standard deviation” is a measure of how much or little the numbers in x are “spread out” around their mean.

For example, imagine a math class of size 30. If 15 students score “0” and 15 score “100” on an exam, then the mean exam score is 50.

If, instead, 15 students score “49” and 15 score “51”, the mean exam score is also 50. But these are two dramatically different scenarios: the mean doesn't tell the whole story. In the first case, the scores would have a large standard deviation, and in the second case a small standard deviation.

If, yet again, 29 scored 0 and 1 scored 100, the mean score would be about 3.3. In such a case where the data is badly “skewed” to one side, the median is often a better indication of the “center” of the data than the mean.
Making a bar chart of the numbers in an array

$$\text{hist}(x,n)$$  Puts the numbers in the array $x$ into $n$ “bins” and displays the results in a bar chart. If you don't specify $n$, the command $\text{hist}(x)$ uses a default value of $n=10$.

Exercise

1) Make an array $x$ with 10000 uniformly distributed random numbers in the interval $[0,1]$ Put the elements of $x$ into a bar chart with 10 bins; repeat for 20 bins (Notice in both cases that each bin contains roughly the same number of elements.)

2) Make an array $x$ with 10000 normally distributed random numbers (with mean 0, standard deviation 1). Put these in a bar chart with 50 bins. You should see that the numbers generated roughly follow a “bell-shaped” normal distribution.