Rattus binomialis

MATLAB concepts covered:

1. 'rand'
2. simulations for answering questions of probability
3. 'for' loops
4. indexing arrays
5. 'tic' and 'toc' to time code execution
6. binomial distribution
7. pdf and cdf

The file BinoRat.m contains one solution to this exercise. We refer to it at several points in this document, but we will only make it available to you after you have completed the exercise, so you can compare your solution to it. In the meantime, you can access http://www.springerlab.org/qmbc/hints.php or the bottom of this document for hints.

A rat is performing a "2-alternative forced choice" (2AFC) task in which it must identify an odor presented at a central port. If it detects odor 'A' it should choose the right-hand port for a reward; if it detects odor 'B' is should choose the other port.

See movie: uchida-mainen-rat_odor_discrimination.mov

The rat in the movie clip is highly trained and gets it right almost every time. But if we go back in time to the early training period, we would find that the rat seems to get it wrong as often as he gets it right. So you decide to do a test and keep track of his correct rate for a block of 50 trials. After 50 trials, we see that the rat has gotten 31 trials correct (19 trials wrong) for an average of 62% correct. **You want to know if the rat has learned the task or if he is still guessing.**

Let's say that we know nothing about probability theory or the binomial distribution, so we will just use the brute force power of a simple simulation.

What is the null hypothesis?

Since there are only two possible answers, this corresponds to a correct probability of 0.5, which is identical to the rate at which tossing a "fair" coin should produce a "head" (or a "tail"). So our previous question can be put as: **how often would one expect to get 31 (or more) heads out of 50 coin tosses**?

Why is the parenthetical "or more" added to the previous sentence?

Well, we're not really interested in the probability of the rat getting *exactly* 31 of 50 trials correct. We're looking for conditions under which we would *reject* the null hypothesis of guessing. So we would clearly count any performance that was better than 31 as well. To answer with a real coin would take a long time, but computers are fast, so we can let the computer do the flipping. How?

Step 1: Write a single line of MATLAB code to simulate a coin toss. It should return a '1' for heads and a '0' for tails.

How many ways can you think of to do this?

See Hint 1 below for help
Also see Step 1 in BinoRat.m

To address our particular question, we need to find the number of occurrences of tails in runs of 50 tosses.

How might we do this?

Step 2: Write a 'for' loop to simulate 50 coin tosses and store the results in an array called 'tosses'.

We want to know *how often* a certain number of tails is likely to occur in a run of 50 tosses, so we need to repeat the coin flip 50 times. Then we can ask whether, in those 50 tosses, we observed 31 or more tails. One way to solve this problem is by writing a 'for' loop.

What is a 'for' loop?

See Hint 2 below for help
Also see Step 2 in BinoRat.m

Let's start by initializing some variables.

What types of variables will we need for this simulation?

Thus far, you've learned how to "grow" arrays within 'for' loops, but it's better form to have your arrays pre-declared, provided it's predictable in advance how many values you will be generating. In this case it is, so declare variables to hold the results of your simulations:

```
tosses = zeros(n,1); % results of individual coin tosses on one simulation
```

Why did we initialize this variable with zeros?

So, now we can write the 'for' loop to simulate 50 coin flips.

See Hint 2a below for help
Also see Step 2 in BinoRat.m

Step 3: Code a pair of nested 'for' loops that will do *nsim* simulations of 50 coin tosses and store the resulting number of heads on each simulation in *results*.

Step 2 had us simulate flipping a coin 50 times. But we want to flip a coin 50 times *lots of times* so that we can generate an understanding of how likely we are to turn up 31 tails.

How can we simulate 50 coin flips 1000 times?

See Hint 3 below for help
Also see Step 3 in BinoRat.m

What additional variables will we need?

To run our 1000 simulations of 50 coin flips each, we will write a nested for loop. Now we can ask how often a given result (31 or more tails) happens in 1000 simulations.

See Hint 3a below for help
Also see Step 3 in BinoRat.m

If you run your nested for loops again, do you get the same answer? How can you get a progressively "better" answer?

Step 4: Write a script that will do the same thing without 'for' loops.

Because MATLAB is a vector-based language, we can often do things much more efficiently using indexing and operators (such as '>') that work across entire arrays or functions that work along a specific dimension, such as down columns.

How might you solve this problem without relying on for loops?

See Hint 4 below for help
Also see Step 4 in BinoRat.m

This code will be much more satisfying code (fewer lines!), and it will execute much more quickly. You can test the speed using MATLAB's 'tic' and 'toc' functions. You execute a 'tic' at the beginning of your code, and then a 'toc' at the end, and you'll get the amount of elapsed time. Try this for your loops in Step 3 and compare it to the code in Step 4.

See Hint 4a below for help
Also see Step 4 in BinoRat.m

Step 5: Using a built in binomial function, write code that will give the probability of getting 5 or fewer heads in 10 tosses.

You might vaguely recall that events like coin tosses, where there are only two possible outcomes, are described by the binomial distribution[1]. Based on your knowledge of MATLAB, you might expect that there is an existing function that will make our task easier. Remembering only this key word, 'binomial', find some useful sounding functions.

Using one of these functions, what is the probability of getting (exactly) 5 heads from 10 tosses? Is there a single built-in function that can do this for us?

See Hint 5 below for help

Also see Step 5 in BinoRat.m

What are cumulative distribution functions (CDFs) and probability density functions (PDFs)?[2] *Which function would be appropriate to use to determine the probability of getting 5 or fewer heads from ten tosses?*

Step 6: Using 'binocdf', calculate the probability that the rat gets it right 31 times or more by just guessing.

Finally, let's return to our rat question. Remember, we want to know the probability of getting 31 tails or more out of 50 tosses.

How can we use binopdf to solve this question?
How can we use binocdf to solve this question?

See Hint 6 below for help
Also see Step 6 in BinoRat.m

OK, so what do we think? Is the rat "getting it" or is he just guessing? How many would he need to get right (out of 50) before you believed it?

How can we use the above functions to answer this?

EXTRA Step 7: Visualize the results from your simulation

Use the array named 'results' from Step 3 and plot its values.
Now, calculate the mean, standard deviation and standard error of 'results'. How do these values change as a function of the number of simulations?
Remake a histogram plot of 'results' but change the Y-axis values to percent occurrence rather than count number.

EXTRA Step 8: What if the distribution we cared about wasn't a binomial?

What other functions does Matlab make available? Inspect how they are distinct.
Find the relationship between a poisson distribution's mean and its variance.
Is there an easy way to generate random numbers from non-uniform distributions? Exponential? Normal? Plot them and compare them to the output of rand.
When would these other distribution types be useful?

EXTRA Step 9: Compute the 95% upper and lower confidence bounds for:

a) 30 correct out of 50
b) 60 of 100
c) 300 of 500
Is there a MATLAB function that does this auto-magically?

Footnotes

[1] The **binomial distribution** is the discrete probability distribution of the number of successes in a sequence of *n* independent yes/no experiments, each of which yields success with probability *p*.

[2] In probability theory, a **probability density function** (**pdf**), or **density** of a continuous random variable is a function that describes the relative likelihood for this random variable to occur at a given point. The probability for the random variable to fall within a particular region is given by the integral of this variable's density over the region. The probability density function is nonnegative everywhere, and its integral over the entire space is equal to one.

Likewise, the **cumulative distribution function** (**cdf**), describes the probability that a real-valued random variable *X* with a given probability distribution will be found at a value *less than or equal to x*. Intuitively, it is the "*area so far*" function of the probability distribution. Because the area under the pdf must sum to one, we can get the *"area beyond"* as: 1 – cdf(x).

Hints

Hint 1

Type `help rand`. How could you use the rand function to output zeros and ones only?

Hint 2

A for loop repeats a statement a specific number of times. Remember, a for loop is completed by an `end` statement.

Hint 2a

Once you have initialized the variables you need, you can use a for loop and the random coin toss function you used in step 1 to simulate a total of 50 coin tosses. The number of correct guesses will correspond to the number of times you get 1 in your coin toss. how can you keep track of that?

Hint 3

You have already coded a for loop that simulates 50 coin tosses. You can repeat this 1000 times by creating another for loop around this one. At the end of the series of 1000 experiments, you will probably want to have an array of length 1000 storing the result of each experiment. What exactly is the outcome (for each round of 50 coin tosses) you want to keep track of?

Hint 3a

You have encountered methods of comparing two numbers earlier (e.g. <, ==, >=). Use those to find out which of the simulations did as well as or better than your actual rat. How can you convert this number into a probability?

Hint 4

The key here is understanding that going through a loop *n* times is the same thing as performing the same operation or set of operations on all the elements of a vector of length n. For instance, the following two are equivalent

Code 1:

```
randomNumbers=[];
for i=1:5
    randomNumbers(end+1)=10*rand(1,1);
end
```

Code 2:

```
myArray=rand(1,5);
randomNumbers=10*myArray;
```

For a nested loop consisting of one loop with *m* iterations and one loop with *n* iterations, you can use a 2-dimensional array of length m×n. In your case, you will want a 50×1000 array. Think carefully about what the entries in your array should be. Once you have figured that out, the other operations are similar to what you have done in the first part of the exercise.

Hint 4a

Use `help tic` for help. If you add a tic command at the start of a block of code and a toc command at the end, MATLAB will compute the time it takes to execute the block of code in between. The output will contain a line similar to this:

```
Elapsed time is 10.360045 seconds.
```

Hint 5

Remember you can use `help` if you are interested in a specific function and `lookfor` if you are interested in a specific topic (e.g. the binomial). The functions `binopdf` and `binocdf` will be of interest to you.

Hint 6

Refer to footnote 2 for explanations of pdf and cdf. You can also use the help function to find out more.

Revisions:
RTB wrote it, 23 April 2011 as "binodemo.m"
RTB revised it, 23 May 2011 as "binorat.m"
DAR updated it, 4 August 2011 as two files. This pdf and corresponding “binorat.m"
JW updated and reformatted on 8 August 2011
RTB added exercise 10 (number picking), 27 May 2012, saved as "BinoRat.m"
MIS updated, added link to hints on course website, removed number picking part to go into new file, 5 August 2013
MIS added hints at the end of the file, 15 August 2013