<u>Python Exercise</u>

Hakwan Lau @ RIKEN CBS
(with input from his lab members over the years, especially Brian Maniscalco & Siva Rajananda)

Here I describe 3 problem sets for you to solve with Python. They are very close to what you will have to do in the lab on a day-to-day basis, if you are going to lead your own projects. I have found this the best way for one to learn Python, i.e. to learn by actually using it. There are courses, and introductory books. But most of us learned how to use it by just reading the documentations, watching online tutorials, and figuring out how others code in the forums, etc. So I suggest you try this route. You can ask other lab members for the occasional odd technical questions, but you really should try to figure it out yourself. Some people never manage to learn it this way. In that case, I'd suggest you take a course. But even if you take a course, or already have some experience in programming before, to run projects in the lab, you need to complete these exercises first.

Preparation Work

You should be able to get access to Python 2, given that one particularly useful package, PsychoPy, is now only compatible with Python 2 but not 3. Besides, you should have packages such as numpy, scipy, matplotlib, etc., installed beforehand. These are important packages that Python programs count on.

You can download Python from its official site: https://www.python.org/, where you can also find official (basic) syntactical instructions. You should be able to use Python if you are working on MacOS, with which Python 2 is already installed.

Here's a list of documentations that you may want to refer:
PsychoPy: http://www.psychopy.org/documentation.html
matplotlib: http://matplotlib.org/contents.html
numpy (esp. random): https://docs.scipy.org/doc/numpy/reference/routines.html
scipy (some statistical tools): https://docs.scipy.org/doc/
For simple installation, please consult Google.

Here's a list of useful YouTube playlists where you can find clear online instructions:
matplotlib: https://www.youtube.com/watch?v=q7Bo_J8x_dw

Here's a list of basic things you should be familiar with in Python:

- How to create, revise, and access with different data structures. The main ones are lists, dictionaries, and numpy arrays.
- How to load and save different kinds of data files (most common ones are csv., xlsx., and mat.).
- How to use different kinds of conditionals and loops: if, for, and while, esp. in cases of list comprehensions (list comprehension is a powerful syntax in Python – you should be good at it).
- How to write and use functions.

<u>Problem 1 – Analyzing A Behavioral Dataset</u>

Please find a mat. file in this Dropbox link.

This is a MATLAB data file. As the field is moving towards Python, which involves many open source packages and is simple and flexible, it is expected that you should master it. The data file is from a simple psychophysics experiment. Subject saw circles of visual noise on the left and right sides of the screen, and one of the patches contained a sinusoidal visual grating. Subjects had to indicate which stimulus had the grating (left or right) and rate how confident they were that their answers were correct, on a scale of 1 to 4.

The format of the data file is as follows (or you can open it to have a look). “data” is the main variable of interest. It is a “struct” variable, which means that it is basically a package that holds a bundle of variables inside of it. “data” holds several arrays with 1 row and 1,000 columns. In these arrays, the i’th column contains information pertaining to the i’th trial of the experiment. The main variables of interest in “data” are:

- **data.stimID**, which codes what type of stimulus was shown. 0 = grating was on the left, 1 = grating was on the right.
- **data.response**, which codes the subject’s stimulus classification. 0 = responded that grating was on the left, 1 = responded that grating was on the right. Negative values indicate trials where the subject didn’t enter this information within the given time limit.
- **data.rating**, which codes the subject’s confidence rating, from 1 to 4. 1 is the lowest confidence, while 4 is the highest. Negative values indicate trials where the subject didn’t enter this information within the given time limit.
- **data.responseRT**, which codes the time between the stimulus offset and the subject’s response.

1. Plot the mean reaction time for the first and the second halves of the experiment respectively. Put error bars (standard error of the mean) on the two bars representing the reaction times. Put labels and everything to make it look nice. Make the graph black and white only and print it out to your screen.
2. Run a t-test to see if the RTs for the first and the second halves of the experiment differed significantly. If you do not know what a t-test is, try Wikipedia or any elementary statistics book.
3. Plot the median reaction times for the 4 confidence levels respectively (you no longer need error bars for this one and the following ones).
4. Calculate and plot d’ for the 4 confidence levels respectively. If you do not know what d’ is, check it out in this handout by David Heeger (this video by Devin Burns can also help you understand some concepts of SDT if it's new to you). We use SDT a lot in the lab. Be familiar with it (something to think about: now that you have calculated the 4 d’ values for each confidence level, do you think this is theoretically a correct thing to do? Answer: according to SDT, not really. Think about why not.)
5. Calculate and plot meta-d’ for the first half and the second half of the experiment respectively, using the in-house functions located in this Dropbox folder. Read through the relevant documentation on this Columbia University website to get at least a conceptual understanding of what meta-d’ measures are.

Problem 2 – Programming Your Own Experiment

1. Set up an experiment using PsychoPy that does the following: Display a screen to ask the subject to get ready. When they are, they press the spacebar to start the experiment. In each trial, you present a white square on either the right or the left side of the screen. If the square is on the right, the subject has to press the "M" key as quickly as possible. If the square is on the left, the subject has to press the "Z" key. If the subject presses the wrong key, make a beep sound, and flash the word "error" on top of the square. After the subject presses the key, clear the screen, wait for 1.5 seconds, and start a new trial. If the subject presses "Q", the experiment terminates.
2. Set the white square to be exactly 3 degrees visual angle wide. To figure out visual angle, you need to do some calculations based on the distance from the subject to the screen (which we usually just assume to be a constant even though the subject moves a bit), and the size of the screen, etc. Figure out how to do it.
3. Change to flash the white square for only 50 ms no matter when the subject presses the key.
4. Replace the white square with a Gabor patch (figure out what it is). In every trial, the Gabor patch is going to be presented in a random orientation. Set the luminance contrast of the Gabor to be exactly 50%. You need to understand something about Gamma correction (this one you can ask other lab members) and figure out how to calculate contrast for the Gabor.
5. Now, randomly interleave the Gabor trials and white square trials. Specify and display the following instruction on the beginning screen: If it is a Gabor, the subject has to press the left key when the Gabor is on the right, and vice versa. If it is a white square, they press the key on the same side as the white square. (left key = Z, right key = M)
6. Record the data for an experiment of 100 trials (exactly 50 Gabor and 50 white squares, in random order; in general, always randomize the order of a list of trials, rather than randomly decide what each trial is. Think about why the former is better than the latter). Easiest would be to use yourself as a subject. Then plot the RTs and error rates for the two conditions (Gabor with reverse left-right responding vs. white square with straightforward responding).

BONUS - Online version of Problem 2

Once you're done with the above, try making a similar experiment for an online study too. The online experiment version (written in JavaScript) will be exactly the same as the Python version described above except that:

1. You will use jsPsych instead of PsychoPy. This will run in your browser. These two tutorials are a good place to start.
2. There will be no beep sound if the subject makes an error (but you still flash the word "error")
3. You will display a Random Dot Kinematogram (figure out what it is) instead of a Gabor patch. Set the number of dots to be 100 and make the aperture a circle. Instead of a random orientation, it will be a random coherent direction.
4. Once you get the data, plot it in Python.

Problem 3 – Running Some Simulations

Make sure you are familiar with SDT. If this SDT handout by David Heeger is not sufficient, try MacMillan and Creelman's User's Guide Book, especially the first few chapters. There should be a few lab copies of the book.

1. Simulate an experiment in which the "subject" (i.e. a simulated agent) detects a target. In every trial, there's a 50% chance that the subject will receive a sensory input (a value) from the target distribution (randomly sampled from a Gaussian distribution), and a 50% chance that the subject will receive a sensory input from the non-target distribution (another Gaussian of equal variance but lower mean). The subject sets a certain criterion (midpoint between the two means of the Gaussians), such that a "Yes" output is generated if the sensory input crosses this criterion, and "no" if it is below. Run it for 50 trials, count the proportion of Ys and Ns generated.
2. Set d' = 1.5. Run the simulation for 30 trials. From the simulated data, calculate d' as if they are real data, as in problem 1.4. Repeat this simulation several times to get a feel of how d' behaves with only 30 trials. Now run the simulation again for 200 trials, and calculate d' again.
3. Change the criterion *c* such that it is 1 (conservative). Simulate 200 trials, and calculate from the simulated data to make sure you changed the criterion by the right amount (i.e. the empirically estimated c should be close to 1). Calculate d' too. Now lower the criterion to -1. Calculate d' again. d' should be relatively stable at different levels of c.
4. Increase the variance of the target distribution by 50%, but keep the non-target distribution the same. Simulate 200 trials, and see the effect on d'. Now simulate 200 trials for c = 1 and c = -1 respectively. Note how d' is no longer constant when you change c. Try to think about why if you don't already understand.
5. Run the simulation, 200 trials each, at different levels of c, so as to construct an ROC plot. Now run the simulation again, with only 30 trials each, also at the same 5 levels of c, and construct an ROC plot and compare.