**NSCI 360, Fall 2019**

# Computational Assignment 2: Programs that simulate behavior & brain activity

**DUE SUNDAY, November 10, 9 pm**

As in Assignment 1, computational assignments contain questions for all background levels of programming and mathematical experience (Levels 1, 2 and 3). You will be graded only on those questions that correspond to your agreed-upon background level (if you don't remember your level, ask me). However, it may be necessary to answer Level 1 questions (or at least be *able* to answer them) in order to answer Level 2 questions. Similarly, Level 2 capabilities may be needed to answer Level 3 questions.

In general, there is one place you should always check first before asking questions of the instructor or even other students, which is the Help documentation that comes with Matlab. For example, to get help using Matlab's **plot** command, you can type "`help plot`" or "`doc plot`" at the command prompt (looks like this: >>), or type "`doc`" and then search for **plot** in the Search box that you'll see. Try copying the little snippets of code examples that are included in the `doc` articles. Paste them into the Command Window and hit Return, and you'll see what the code does. Then change it a little bit, and see what that does. That seems to be the best, practical way to learn how to do what you need to do.

**Collaboration Policy:** I expect students to try to do these problems individually, and to submit their answers in their own writing (or their own code). Do not copy and paste from any other student. However, you should feel free to discuss problems with other students, and you should feel free to modify code from the `doc` articles as you see fit, as long as you do that yourself (i.e., don't copy code from the `doc` and then send it to another student for their use – they should look that information up themselves).

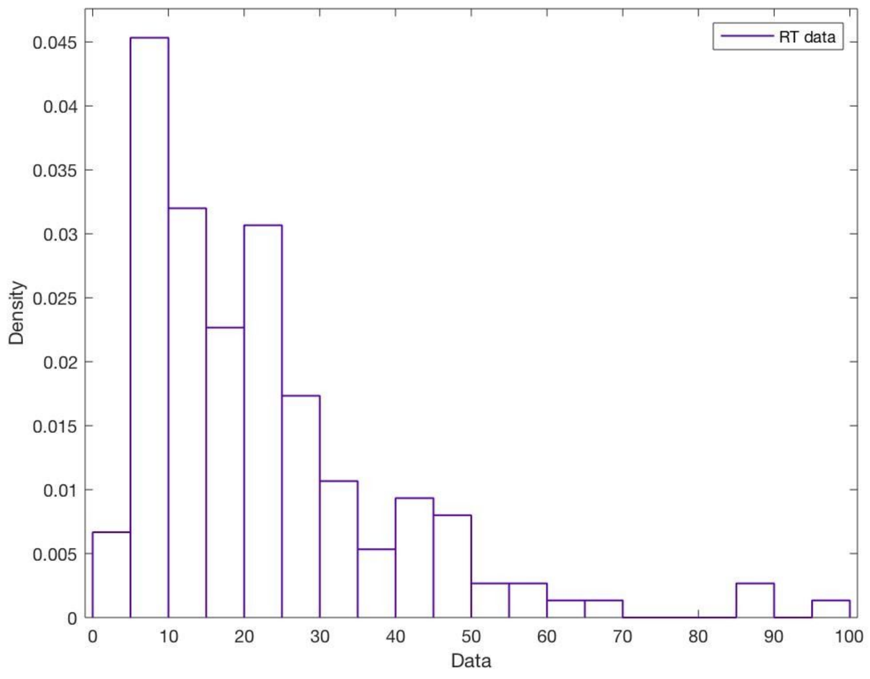
How to submit your work: Just copy and paste your code, your figures, or your written answers into the spaces provided in this document. Then email to me by the deadline at:
psimen@oberlin.edu
Please name the file "*YourLastName_YourFirstName*_NSCI360_HW2". Files can be in Word, text, Pages (Apple), or PDF format.

7/8/9/11

**7. Response Times generated by a random-walk model:**

**Submit your function code as a separate file called random_walk.m. Also, paste your histogram here:**

**. . . . . . . .**

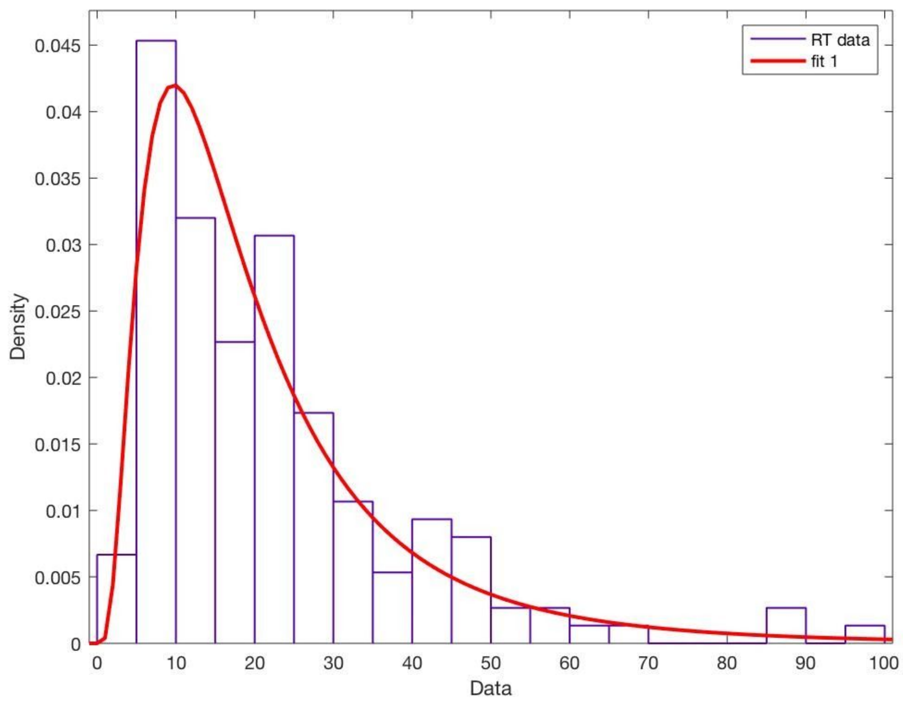


**8. Use Matlab's distribution fitting tool to fit the response time (RT) distributions:**

**Save your fitting window to a pdf file, by selecting from the dfittool's File menu the Print to Figure option. Then save that figure as a pdf, and paste it here.**

**. . . . . . . .**

**Type here the name of the distribution that fit your data best, and report the log likelihood score of the fit.**



log likelihood: -521.22

**9. Compute micro-SAT (speed-accuracy tradeoff) plots**

**Type your answers here, and paste a figure that shows your micro-SAT.**

**. . . . . . . .**

**Paste your code here:**

**. . . . . . . .**

**11. Response-locked vs. stimulus-locked averaging:**

**Submit your function code as a separate file called "averaging.m". Have it create four figures (one each for the stimulus-locked averages of 1-crossing and -1-crossing trajectories, and one each for the response-locked average of 1 and -1-crossing trajectories). Paste those as pdf figures here:**

**. . . . . . . .**

**10. More complex neural network**

**Save your code into its own .m file. Call it "complex_net.m". Make it accept 3-dimensional input vectors, representing how cat-like (how close to 0) or dog-like (how close to 1) a given animal is in terms of the three features. Also, try out some non-binary input vectors, and see how they get**

**classified (they too should produce network decisions that an input animal is a cat, a dog, neither, or even both, whatever that means.)**

**12. Rate-varying Poisson processes**

**Submit your function code as a separate file called "non_homog_Poisson.m" (which stands for "non-homogeneous Poisson", meaning that the rate of Poisson spikes changes over time). Have the function create a single figure. Paste the pdf figure here:**
**. . . . . . . .**

**<u>LEVEL 2 AND LEVEL 3</u>**

**7. Response Times generated by a random-walk model:** Using for-loops and a random number generator, compute a random walk: a variable whose initial value is 0, and which gets incremented by both a deterministic increment (0.01) and a random increment (normal with mean 0, std of 0.25) at each "time" step. Stop the process when it exceeds the value 1, or decreases below the value -1. Keep track of the values of this variable, and plot out the resulting trajectory. Also, keep track of *when* (at which time step) the variable exceeds 1, or decreases below -1. Do this 150 times. Then create a histogram of those threshold-crossing times ("response times", or "RTs"), and save the response time data to a .mat file. Also, save a vector of 1s and -1s that indicate which threshold was crossed first.

**Submit your function code as a separate file called random_walk.m. Also, paste your histogram here:**
**. . . . . . . .**

**8. Use Matlab's distribution fitting tool to fit the response time (RT) distributions:** Load up the RT data from Problem 7, and use the dfittool distribution fitting tool in Matlab to fit a distribution to the data (simply type `dfittool' at the command prompt >>). Try fitting the gamma, lognormal and Weibull distributions to the data. Which one fits best?

For assistance with using dfittool, use the `doc`, and search for "Distribution Fitting Tool". There is a large help page that documents how to use this interactive tool for fitting different probability distributions to the data you've generated. But the overall process is pretty simple:

i. Make sure the display type is set to "Density (PDF)"
ii. Make sure you have your RT data as a variable in the Workspace
iii. Click the Data button
iv. In the popup menu, select your RT data for the "Data" field, then click "Create Data Set" (you can close the Data window when this is done)
v. Then, back in the main dfittool window, click NewFit
vi. Make sure the Data pulldown menu is listing your RT data
vii. Select whichever distribution you like from the pulldown list of many different sorts of probability distributions
viii. Then click "Apply", and observe how well that probability density can be made to fit your data
ix. Report the "log likelihood" score that you observe in the bottom of the popup window. The less negative this number is, the closer the fit to the data is.

**Save your fitting window to a pdf file, by selecting from the dfittool's File menu the Print to Figure option. Then save that figure as a pdf, and paste it here.**

. . . . . . . .

**Type here the name of the distribution that fit your data best, and report the log likelihood score of the fit.**

. . . . . . . .

**9. Compute micro-SAT (speed-accuracy tradeoff) plots:** Take the RT data and response type data (1 or -1) from Problem 7, and examine the proportion of 1s and -1s that occur in each of 10 consecutive time windows, or "bins". Plot out the proportion of 1s in each bin. Is it flat, increasing, or decreasing? Now, what about the average RT for all 1-crossings? Is it different than the average RT for all -1-crossings?

**Type your answers here, and paste a figure that shows your micro-SAT.**

**<u>LEVEL 3 ONLY</u>**

**10. More complex neural network:** For this problem, you should complete Problem 5 first. Then the problem should be modified so that there are 3 input values (a

3-dimensional vector), and two output units. Think of each output unit as a category (e.g., if the output is (1,0), then the system is "thinking of" a cat; if it is (0,1), it is thinking of a dog). The 3 input values (a column vector) can be thought of as the values of different visual features (facial shape, body shape, type of movement). Do the thresholding operation with a threshold of 2 again. Your main job is just to design the weight matrix necessary to compute a function that corresponds to the following "truth table":

| IN1 | IN2 | IN3 | OUT1 | OUT2 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 |

For your information, what you are doing in this problem is essentially finding a plane that divides the three-dimensional input space into two categories, and classifying input vectors that exist on one side of the plane as a "0", and those that fall on the other as a "1". And you are doing this separately for each output unit. Note that it is possible (in fact, easy!) to define functions that are not "linearly separable" in this way. One such linearly inseparable function of two binary variables, call them A & B, is the exclusive-or (XOR) operation, defined as follows: if A is 1, or B is 1, but not both, then the output is 1, otherwise the output is 0. When this was recognized in the 1960s, early neural network research took a huge hit in popularity, funding, and overall progress. We talked about the solution to this problem in class (i.e., using more than just two layers of units), but we will not implement that solution now.

> **Save your code into its own .m file. Call it "complex_net.m". Make it accept 3-dimensional input vectors, representing how cat-like (how close to 0) or dog-like (how close to 1) a given animal is in terms of the three features. Also, try out some non-binary input vectors, and see how they get classified (they too should produce network decisions that an input animal is a cat, a dog, neither, or even both, whatever that means.)**

**11. Response-locked vs. stimulus-locked averaging:**

For this problem, you first need to save several random walk trajectories from Problem 7. You then need to create a single, average signal that averages at each time step across all the separate random walk trials. Do this separately for those trajectories that terminate at the 1-boundary, and then for those that terminate at the -1-boundary. First, make sure that all the trajectories *begin* at the same time: this is an example of stimulus-locked averaging. Second, make sure that all the trajectories *end* at the same time: this is an example of response-locked averaging. These two averages will not be the same, since the random walk takes different amounts of time to cross threshold on different trials.

> **Submit your function code as a separate file called "averaging.m". Have it create four figures (one each for the stimulus-locked averages of 1-crossing and -1-crossing trajectories, and one each for the response-locked average of 1 and -1-crossing trajectories). Paste those as pdf figures here:**
>
> . . . . . . . .

**12. Rate-varying Poisson processes:** Create a spike-generating process as in the previous assignment (note the simple way you can do this that is shown in Problem 6). However, do it in such a way that the rate of spiking changes three times over the course of a simulation (rising and falling over time). Create many simulations of such a process, with the spike-rate changing at roughly the same time on each simulation. Then, create spike rasters as requested in Problem 6. Finally, create a spike-time histogram. This histogram should show how many spikes occurred within a set of small time windows across all simulations.

Break the output figure into two "subplots", by using the "subplot" command. By using the following commands, you are telling Matlab first to draw into the top subplot out of a column of two subplots. The second command tells it to draw into the bottom subplot. Spike rasters should go into the top subplot; histograms should go into the bottom subplot.

```
subplot(1,2,1);
subplot(1,2,2);
```

> **Submit your function code as a separate file called "non_homog_Poisson.m" (which stands for "non-homogeneous Poisson", meaning that the rate of Poisson spikes changes over time). Have the function create a single figure. Paste the pdf figure here:**
>
> . . . . . . . .