Tips and tricks for becoming a proficient psychologist programmer
(w/ examples from Matlab/Octave and PsychToolBox)

Author: Cendri Hutcherson (c.hutcherson@utoronto.ca)
Date: May 27, 2018
Version: 1.0
URL: www.decisionneurolab.com/resources/Intro_to_Programming_for_Psychologists/

Step 1: Installing the software package(s)

1. **Installing Matlab**
    a. Matlab can be downloaded and used for a 30-day trial period free of charge, after which purchase of a license is required. Many academic institutions have arrangements with Matlab to allow their members to use it, so check with your technology staff for options.
    b. To download the trial version, visit Matlab's trial version download page.
    c. To check your institution's licensing options, visit the trial version download page above and click on the "Check for campus license" button.
2. **Installing Octave**
    a. Octave is the open-source (free) alternative to Matlab. It is somewhat more limited in the things it can do, but many of the most basic commands translate one-to-one from Matlab to Octave.
    b. To download Octave, visit https://www.gnu.org/software/octave/, click "Download" and select the instructions for your operating system.
3. **Installing PsychToolBox**
    a. PsychToolBox (PTB) is a set of scripts and functions designed for optimizing experiment presentation.
    b. To download PTB visit http://psychtoolbox.org/download/ and follow the instructions for your operating system.

Step 2: Learning the basics of programming

Learning general programming skills

There are some very good free on-line tutorials out there to help you begin to learn the basics of pretty much any computer language. Here are links to a few of the better ones for Matlab:

1. https://www.tutorialspoint.com/matlab/index.htm

2. https://learntocode.mathworks.com/portal.html (very basic, but includes on-line interactive examples and tasks)

Getting a good, comprehensive book for learning is also extremely helpful. There are many, many commands and functions that have already been written and shared that can make your life easier, but remembering their details is not always easy. Having a quick and comprehensive reference on hand can be useful. Three recommendations:

1. Mastering Matlab (Hanselman & Littlefield) - older now, but pretty comprehensive and still good!
2. Matlab: A practical introduction to programming and problem solving (Attaway) – not as comprehensive, but may be an easier introduction for complete newbies
3. The Elements of Matlab Style (Johnson) – a great little book covering some tips for how to write the most element and readable Matlab code. Much of this generalizes to other programming languages.

Basic things to learn to become proficient:

1. Understand the different data/variable types
    a. In Matlab, the most common are scalars/floats, strings, vectors, matrices, cell arrays, and structures
    b. Know how to read and write the contents of a data type
2. Understand how commands are written and executed (i.e., basics of syntax)
3. Understand how to control the order/flow of command execution
    a. if and if-else statements
    b. for loops
    c. while loops
4. Understand the value and strategic use of comments (non-executable code that allows you to state, in English, what code is doing, to provide guides for more readable code)
5. Understand scripts and functions
6. Understand file access and how to import/export data

Psychologist-Specific Skills

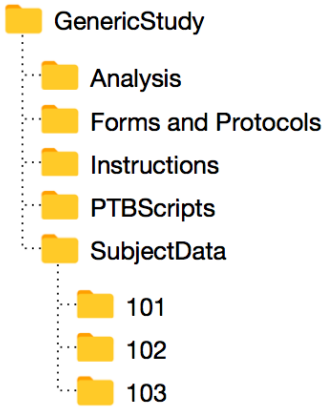
7. If you are hoping to use Matlab/Octave for experiment presentation, stimulus delivery, or data recording, the next step is to start learning how to use PsychToolBox. See Peter Scarfe's excellent set of tutorials (http://peterscarfe.com/ptbtutorials.html) for well-commented and easy-to-understand demonstrations of many of the basic tasks one might want to do when using PTB for experiment presentation and stimulus delivery
8. Use PTB's user forum to search for solutions

a. Go to https://groups.yahoo.com/neo/groups/psychtoolbox/conversations/topics?guccounter=1
b. Check to see if similar questions have been asked before
c. Post specific questions, using suggested etiquette for most effective questions (http://psychtoolbox.org/forum/)

9. Once you have data (woo!) you will want to understand basic commands for data analysis, plotting, and graphics (see Step 4 below)

Step 3: Programming a Study: Tips and Tricks

1. **Set up a study directory with clear and consistent structuring.** In general, I recommend thinking carefully about how you would like study folders to be set up and sticking with a basic structure for every single study you run. This has the advantage that any user familiar with the setup for one study can more quickly and easily find what they are looking for in another study, because they already have some idea of where to look. Over time, I have gravitated toward the following directory setup, though you may benefit from a different setup depending on your particular needs.



2. **Write general-purpose scripts and functions** that can be used over and over in multiple studies. For instance, for my own workflow, I have written the following frequently-used functions (available for download at https://www.decisionneurolab.com/resources/Intro_to_Programming_for_Psychologists/)
    a. InitPTB.m - An initialization script for starting up a study. Gets subject information, creates data files for logging, and sets up a computer screen for displaying images

b. logData.m - A data-logging function capable of storing any kind of data
c. collectResponse.m - A function for collecting responses and RTs
d. showInstruction.m - A function for displaying instruction slides to a subject and waiting for the user to hit a key to advance to the next instruction
e. makeTxtrFromImg.m – A function for quickly importing an image file and putting it into a format accessible for use by PTB
f. findPicLoc.m – A function for taking an image and computing the screen coordinates based on user-defined specifications
g. searchcell.m – A function for searching a cell structure to find a particular value. Useful

3. **Divide and conquer.** Use a master script (e.g. runMain.m) to control the flow of an experiment. Then write stand-alone scripts/functions to divide the study into components. I recommend some or all of the following divisions
    a. Instructions/practice: See runPractice.m for an example.
    b. Study phases, e.g.:
        i. runChoiceTask.m: Asks subject to make choices about whether to eat a food. Can be run independently, or called from runMain.
        ii. runRatingTask.m: Collects subjective ratings for foods that were seen in the study. Can be run independently or called from runMain.
        iii. determineFood.m: At the end of the study, can take choices made by the subject and implement one of them. Can be run independently or called from runMain.m
    c. Repetitive tasks/blocks of code, e.g.:
        i. runChoiceTrial.m – Called by runChoiceTask, it takes user-specified inputs about what to do for each trial, then runs through a specific trial sequence
        ii. runRatingTrial.m – Called by runRatingTask, it takes user-specified inputs about what to do for each trial, then runs through a specific trial sequence
4. **Develop testing procedures.** Nothing hurts more than getting halfway through data collection, only to realize you haven't collected a vital piece of data. Only slightly less painful is getting through your first subject or two and then having the experiment crash partway through. I recommend developing a set of practices that allow you to test various aspects of data collection and data analysis.
    a. Google is your friend. Googling for solutions to particular tasks, or for particular error messages, is often the best way to solve a particular issue.
    b. Develop proficiency using debugging tools. In Matlab, the most commonly used commands for doing debugging are
        i. dbstop if error
        ii. dbquit

iii. dbup & dbdown

iv. dbclear all

c. **Never** assume that just because code works for one trial, it will work for all trials. **Always** test the study multiple times with different input to find weak points.

d. Develop “test-mode” versions of the study. Adding a few lines of code to help you run through a study from beginning to end without having to *actually* run through it can save you a lot of time, effort, and frustration. See collectResponse.m for one example of how to use test-mode for study debugging

5. **Get in the habit of extensive and strategic comments.** It’s easy to write code only for oneself, assuming that you will remember exactly why or how you did certain things. Turns out this is wishful thinking. Pretending you are writing code for a naïve audience is not altruistic, but it *is* an example of delayed gratification. You will thank yourself later!
6. **White space, right space.** There are a number of ways to make code more readable, so that you can quickly parse what is going on. Getting in the habit of using indents, spaces, and new-lines strategically to help illustrate code flow is another way of making the code readable, accessible, and understandable.
7. **Use a platform like GitHub to save changes to code.** It can be very useful to save versions of experiments, so that you can test out new additions of code without worrying about breaking things. GitHub is one of the most popular platforms for doing so. It is also useful because it can allow you to share code and branch off different versions fairly easily. Two introductory tutorials for doing this are:

    a. https://guides.github.com/activities/hello-world/

    b. https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners

Step 4: Analyzing your data

Woo! You’ve learned to program, you’ve written your code, you’ve collected your data. Now what? Do some analyses → [redacted] → Publish a paper in Science!

Just kidding.

But…programming can be an incredibly useful tool in doing data analysis. Writing code to run data analyses using the same kinds of strategies you use for programming experiments (e.g. divide and conquer; white space right space; comment, comment, comment) can be incredibly useful.

To be perfectly honest, I do not love using Matlab for data analysis. You can do it, but there are better and/or easier ways to analyze your data for most types of analyses used by psychologists (unless you are doing neuroimaging with EEG or fMRI, in which case there are incredibly useful Matlab software packages, such as SPM and EEGlab).

I generally prefer writing scripts to export the data in a format that can be read by another statistical programming language like R. See convertDataToText.m for an example.

That being said, it can also sometimes be useful to perform simple analyses directly in Matlab. If you have the Statistics Toolbox for Matlab, here are some of the most basic kinds of tests (and their corresponding Matlab function names):

1. Averages (mean)
2. Standard deviations (std)
3. T-tests (ttest)
4. Correlations (corrcoef)
5. Regressions (glmfit)

You may also want to make charts or plots. Here are examples for two of the most common:

6. Bar plots with standard deviation
    a. See this page for a nice set of demonstrations for simple bar plots: https://www.mathworks.com/help/matlab/ref/bar.html
    b. See the BarPlotDemo.m for an example of a bar plot with error bars and labels
7. Scatter plots with a regression line
    a. See this page for a nice set of demonstrations for simple scatter plots: https://www.mathworks.com/help/matlab/ref/scatter.html
    b. See ScatterPlotDemo.m for an example of a scatter plot with a fitted regression line superimposed on top.

See the StatisticsAnalysisDemo.m script for an example of writing a set of flexible statistical analyses that can be useful for documenting and referencing results when writing up a paper.

Summary

You will undoubtedly develop your own workflow and habits for programming. You may even end up preferring a different programming language from the one you started with. Python, for example, is another popular language for many psychologists.

However, the basic approaches to programming laid out here are broadly applicable across many different languages. Good luck!

```
function [success, satisfaction] = learnToProgram(amountOfEffort)
 % A program designed to output the expected amount of success and
 % satisfaction one is likely to get from a given approach to programming. The
 % only input variable required to calculate these things is the variable
 % “amountOfEffort”
```