Computational Modeling for Psychology

Britt Anderson PhD & MD <britt@uwaterloo.ca>

September 2, 2022

Chapter 1

Preface

Experimental psychology was founded as an empirical *sub*jective science. It merged experimental methods, such as repeating measurements, control conditions, and systematic variation of conditions, with the subjective content of human experience. Physics tells us how intense a light or sound is. Psychology tells us how bright and how loud. The convention is to name Wilhelm Wundt as the discipline's founder and 1879, the year he established his independent experimental laboratory, as the date for the founding. It is only in the 1800s that we see the emergence of scientific experiments that look like modern psychology: Weber's weights, Helmholtz's mercury lamp flash experiments on attention, and Wundt's own experiments on attention.

YouTube Link

While Wundt was merging the experimental methods of physics and physiology with the content of human awareness, it was a generation before that Gustav Fechner, a physicist, demonstrated the power of expressing mathematically psychological ideas. His *psychophysics*

For several years I have taught a class to help psychology undergraduates develop the skills needed to pursue computational modelling. In the beginning, around 2010, most of the students had a real fear of programming and came to the course with no knowledge of the most basic programming concepts. I prioritized teaching them some mathematical terminology used with different modeling approaches (e.g. differential equations and some linear algebra), and prepared them to transition to programming by starting with implementing some basic algorithms in spreadsheets, something almost all were familiar with.

The course therefore had multiple goals and really became multiple courses braided together into one offering. There was some mathematics, some computational neuroscience, a bit of neural networks, fundamentals of programming, basic python or R, and instruction in ancillary tools such as version control and command line tools.

The students finding their way to the course evolved too. In the beginning I had a mostly homogeneous group of psychology undergraduates with curiousity and no programming experience, but over the years more came in with some experience with one or another programming language already, and I also started to see students from more quantitative faculties who had lots of programming experience and just wanted

to implement models. It is hard to teach these two groups together. The result was a decision to divide the course into two.

I now teach a course introducing psychology students to the use of computers as research tools. I hope to work on more of a traditional linear textbook like presentation of that, but the course itself is basically done and structured to allow directed self-study. There are a mixture of online video lectures, as well as notes in textual form and exercises set for the students to work through.

Now, I am rethinking what the other half of the course should become if I am freed of the constraints of teaching git and python, and if I still want to welcome more advanced students and try to get them something new. The goal is to be able to still have the students pick up some mathematical terminology while also learning some of the simple algorithms and thinking of programming more generically, rather than getting anchored in the specific syntax of a specific language. And then there are still some of the same pragmatic issues: how do I get them all up and running quickly with a similar technical set-up acknowledging that some are running Windows, some OSX (both Intel Macs and M1 macs), and that I don't want to be providing that sort of low level technical support. Some of the interesting options, Haskell or Common-Lisp, can be rather a bear to get started installing, and then you may need to spend another day getting comfortable with a strange IDE like emacs/sly/slime. Putting these together I decided to try Racket. It is a language designed to support teaching, and the DrRacket IDE seems to work pretty much out of the box on Linux, Windows, and OSX systems. And it even has a documentation system, scribble, built-in, and which I am using to write this guide.

At the time of this writing (June 2022) I am uncertain how Racket will work out. I used some example Common-Lisp code in a prior offering, and it seemed to mostly lead to mass confusion. My intention was that I could show the algorithm implemented in one language and it would serve as a concrete example that the students could adapt to any other language of their choice. But most of them just tried to rotely adapt, line by line, my code into Python. That was often not successful. By forcing every student to use a common language, I hope I can more easiliy support different skill levels, and the students can provide each other better peer support. Racket lets me mix in and demonstrate some functional programming concepts as well. The students who learn to `filter` or `fold` will be able to look for how to do the same in R or Python or Julia (or whatever becomes the *hot* datascience language of next year or next decade).

In summary, the goals for this course and book are to give students a familiarity with the mathematical terminology and domains that form the backdrop to modeling in psychology. I still feel some basic understanding of what certain mathematical gadgets are is important, e.g. what a differential equation is is something psychologists modeling should know, but they do not, most of them, need to know how to analytically solve them. The basic constructs of linear algebra, matrices and vectors, are important for understanding neural networks, but they mostly need to learn some facts about them (such as the non-commutativity of matrix multiplication), without needing to know the formal rules of vector spaces. Probability comes up more now than it used to, and the mathematical ideas of events and measure considered informally, and not via Lebesgue integration, is a good tool to have in one's tool box.

I am not sure that hand coding the backpropagation algorithm is still that helpful

in 2022. It does give an appreciation of the nuts and bolts of a common learning algorithm, but it is error prone to code, and no one uses a hand coded version in practice anymore. There are perhaps better neural network features to talk about and emphasize that better map to psychological concepts, e.g. the Kohonen network which groups and categorizes in an unsupervised way might be a better example. The agent based models too deserve more treatment in the course than I usually give them, and without depending on a commercial lisp implementation to run things.

In my earlier book I tried to give a little bit of everything for everyone. What I hope to do here is not a new version of that book, but a reconception of how to teach some of the same concepts. In order to get started let's try to make neural network