week 00: introduction

preamble

Many areas of biology have become data-intensive, requiring computational analysis of large data sets from genomics, imaging, and other technologies. Some people say that the way forward is to work in interdisciplinary teams because they think it's too hard for one person to do both biological experiments and computational data analysis. Large scale data analysis sure does look like it requires computer science, statistics, software engineering, and mathematics skills. Can you really be a biologist, a computer scientist, a statistician, a software engineer, and a mathematician all at the same time?

The central idea of this course is that if you're doing experiments, you can and should be analyzing your own data. The course is designed with experimental biologists in mind, but if you consider yourself something else, like a statistician or computer scientist, that's good too. A main point of the course is that distinguishing between "biologists" and "computer scientists" and "statisticians" is counterproductive. We're not going to worry about what we're called, or what we're already trained in. We're going to work from the perspective of solving biological problems as scientists, using experiments that happen to generate large data sets.

Of course, there's only so much one person can do. OK, sure, we can't expect biologists to also be full fledged computer scientists, mathematicians, or statisticians. But there are three ideas we can take advantage of:

- **Scripting is not software engineering:** You can learn to use the command line and to

write analysis scripts without being a computer scientist. You can do a lot of data analysis with just basic knowledge of a scripting language. You can learn a lot by cribbing code examples from the internet and modifying them for your uses. It's not a *great* way of writing code, mind you -- but it suffices, especially to get you started. Experimentalists start their training by getting working protocols and learning how to make modifications. You can learn scripting the same way. Over time, you'll find yourself learning and adopting better software engineering techniques naturally.

- **Computational analysis is more like experimental science than you might think:** We'll do positive and negative controls, using scripts to generate synthetic test data. We'll learn techniques for taking small samples from large data sets so we can actually look at data by eye to find problems, and for identifying and looking at outliers. We'll learn to be paranoid about the many ways that biological data can fool you. In a way, biologists already have a superpower when it comes to analyzing big complicated data sets. Biologists are trained to ask incisive questions of invisible systems, and to anticipate being blindsided by unexpected answers. Large biological data sets can be as impenetrably complicated as living organisms. You can't look at all the data at once. At any one time, you only get to make a specific probe into the data system, asking a specific question, much like doing an experiment on a living system. You're looking into the data through a straw. If you're used to the clean provability of a computer science algorithm or a mathematical equation, the messiness of biological data analysis may come as a shock.

- **Statistical analysis can be done by**

simulation. By doing positive and negative controls, we can do statistics intuitively. "How do I calculate a P-value" becomes "how often do I get a false positive result from my negative control?". "What's my statistical power?" becomes "how often does my positive control work?" We will start by posing a biological question we want to ask, and we'll design controlled experiments to answer it. By doing it from this direction, we're forced to think about what our "null hypothesis" is (what are our negative controls) and what effect we're actually expecting to see (what are our positive controls), before we start worrying about stuff like whether we should be doing a Student t-test or something else. We'll approach the statistical testing from an intuitive and experimental perspective, driven by simulations and control experiments where we know the answer. That way, when we go to use a statistical test, we already know why we're doing it (we're trying to replace a time-consuming approximate simulation with a fast analytic calculation); we can check the assumptions we were willing to make in our controls against the assumptions of the statistical test; and we can doublecheck that our analytical results match simulations. Even when we do use fancy statistical tests, we'll learn to use simulations to check that we're using them correctly.

There's a couple of hidden agendas. One is that we're going to take the fundamentals seriously. We'll bump up against the limitations of a script-hacking, simulation-driven approach. That's what I want to happen. I think it's easier to learn the formal and correct stuff when you reach a point that you actually need it for your work. You'll wish you could do things faster and cleaner, with crisp, provable analytics instead of stochastic simulations. You'll see intuitive, biologically-

motivated reasons that we need to develop quantitative skills. We'll see some fundamentals in algorithms and data structures; in engineering and testing robust software; in probabilistic inference and statistics. You'll learn work patterns for doing reliable and reproducible computational science. There are whole courses in these things, so we won't be able to cover everything; far from it! Instead, the aim in this course is to lower your energy barriers for learning in areas you aren't comfortable with yet. We will favor depth over breadth. The idea is that if you learn *one* algorithm, *one* mathematical derivation, *one* statistical test, or *one* computer language, truly and deeply, then that next one is so much easier to learn on your own.

The field of biological data analysis is fluid and fast-evolving. No one is an expert. Including me. The breadth required is too large, and things change too fast. Throughout your career, you're always going to have to learn new things. There isn't a single body of work to teach you biological data analysis, not like we can teach you a curriculum of statistical thermodynamics or multivariable calculus. Whatever *content* I teach this semester, a lot of it will soon be obsolete. What I want to teach you is an *approach* and a *mindset*. The important biology questions will change. The experiments that generate the data will change. Today's trendy computer languages will die. Mathematical and statistical techniques will evolve as more computer power becomes available.

The aim of this course is to help you learn how to learn on your own: to be fearless and skeptical, and to know enough to bootstrap your way to useful solutions. I am not "trained" as a mathematician, or a computer scientist, or a statistician, or a software engineer. If you were to quiz me, you would find shocking, embarrassing holes in my knowledge base. My theory is that everyone working in biological data analysis has

the same problem -- whether they will admit it to you or not.

If you're like me, you may find yourself looking around the class going, wow, everyone knows so much more than me, how am I ever going to learn all this stuff? I want you to learn not to be intimidated. It is natural and common in this field to not know something. Ask questions. Work hard. Figure stuff out. Share your knowledge. Know when you know something, and know when you don't.

For example, we're going to be using Python in this course, but I'm a Python novice. Professionally, I'm a C programmer, but I also use a mix of Python, awk, Perl, and the UNIX command line in my data analyses. I started learning Python for the first year of MCB112, and I'm still teaching myself as we go. If you see me doing something in Python that you know how to do better, speak up!

Pythonistas: Perl is a *perfectly fine* scripting language, and you can't convince me otherwise; the only thing that killed Perl is fashion.

structure of the course

The course is built around 12 weekly data analysis problems. The course is structured so you actually learn to analyze data, rather than just listening to me talk about it. It's not a survey course. We're going to do a few things, and do them well and deeply. Most of the time you spend on the course will be spent outside class, writing code to solve the week's analysis problem. It's essentially a lab course, except that the lab is your laptop.

We have three lectures a week, 75 minutes each. Roughly speaking, the Monday lecture is about fundamentals, and the Wednesday lecture is about applications. Whatever the theme of that week's data analysis problem is, I'll use Monday to set the stage, giving you the background and theory you need, and recommending reading. I'll expect you to have a look at the week's problem and start thinking about it. The Wednesday lecture

will tend to be more interactive. I'm expecting you to ask questions about where you think you need more background to solve the problem, and I'll give you more practical advice on what you're going to need to do. The Friday lecture is given by one of the teaching fellows. It's even more practical; they will typically walk through an example that will help you understand the steps of doing that week's problem.

gene expression analysis

The course focuses on RNA-seq gene expression analysis as its unifying thread. A very basic question arises quickly in an RNA-seq experiment: which genes are expressed differently in condition A versus condition B? From that "simple" question, a framework of biological data analysis emerges, from data exploration to visualization to statistical modeling.

If you don't have much biological background (what's a gene? what's gene expression?), not to worry. We'll introduce all necessary biological concepts. Next week, week 1, is all about gene expression and RNA-seq as background.

the sand mouse

Essentially all the data sets we use in the course will use simulated synthetic data, from fictitious experiments on a fictitious organism: the sand mouse, *Mus silicum*. My sand mouse is inspired by a challenge issued by John Hopfield in 2001, in which Hopfield generated synthetic data from a known neural network and challenged computational neuroscientists to solve the "inverse problem", back-engineering the principles of how the network worked from the observed data. To pretty much everyone's surprise, I think, Hopfield's sand mouse challenge was won by a group led by David MacKay, using a probabilistic inference approach. You'll hear a lot about

(That is, an in silico mouse.)

probabilistic inference in this course - and a fair amount about MacKay's work and writing in particular, partly because he's a hero of mine.

python

We're going to use Python3 for the course, with Python modules for data analysis and data visualization including numpy, SciPy, matlibplot, and pandas, and Jupyter Notebook.

That's Python3, not Python2. When you install Anaconda, make sure you get Python3. There are important differences and our code examples will all be Python3.

You'll do the psets in Jupyter Notebook, and turn in your answers as jupyter pages.

If you don't have much background in Python, or Jupyter, or in writing code at all, not to worry. The course is designed to bring you up to speed.

bring a laptop

You need a computer (laptop or desktop) and an internet connection - obviously! - and you'll need a Python scientific data environment installed on the computer. (We'll show you how.) If you don't have a computer, let us know immediately! The course has a couple of Mac laptops available for lending.

Don't be shy about asking for one. When I was an undergrad, there was no way on earth that I could have afforded a laptop, and I'm still stunned seeing that everyone seems to have one.

prepare for next week

First and most importantly: your first task is to install an up-to-date Anaconda distribution, for Python3. This will automagically install pretty much everything you need for the course.

If you're already a Python aficionado and you have your favorite Python environment lovingly installed already, that's fine too - just make sure it's Python3 or There Will be Trouble.

Second, if you aren't yet a Python coder, you can get ahead of the game and get started. There are plenty of on-line tutorials, including the one from the Python Software Foundation. There's also plenty of books for learning Python, including (funnily enough) *Learning Python* by Mark Lutz (O'Reilly and Associates, 2013). (Don't be daunted by its doorstopping size!) Figure out what kind of

resources you think you learn best from (online, books), and get your resources lined up. The folks in the Harvard Library have made electronic copies of *Learning Python* and *Python for Data Analysis* available to you, linked at our Canvas site.

There aren't any required books for the course. I'll post lecture notes every week, and everything else we need, such as PDFs for some reading assignments. Python tutorials and documentation, I just talked about above.

a Jupyter page you can try now

To start getting used to Jupyter Notebook and Python code, here's an example page you can download: [download w00-first-jupyter.ipynb] [view it in nbviewer].

If you download the .ipynb file and save it to your disk, then type the command

```
jupyter notebook
```

and your web browser should open to Jupyter, showing you a list of files in the current directory. Navigate to where you've got the .ipynb file if you need to, then click on it, and it'll open.

This example walks through a small but real-life statistical problem of deciding how many sand mice to use to distinguish a significant experimental effect; instead of just using magic (a t-test), we do it by simulation with positive and negative controls.

peek behind the curtain

- Python script that generated the intro animation: w00-intro.py

This script uses some additional tools (animation

and generating MP4 movie files) that are a little advanced, and for which you might even need to install some extra tools to run the script (namely, you might have to install the FFMPEG library with `brew install ffmpeg` or the like), but feel free to have a look.
