Biology 381 - Computational Biology

Course Syllabus

Spring 2018

Meetings

Lecture: Tuesday, Thursday 2:50 - 4:05 pm 129 Marsh Life Science
Lab: Wednesday, 1:00 pm - 3:30 pm 129 Marsh Life Science

Instructors

Lecture: Nick Gotelli
209 Marsh Life Science
ngotelli@uvm.edu
Office hours by appointment

Lab: Lauren Ash
211 Marsh Life Science
Lauren.V.Ash@uvm.edu
Office hours by appointment

Pre-Requisites

- A personal laptop computer that you bring to all lecture and lab meetings
  - Mac or Windows, with battery power to last 1.5 hours
  - all required software packages installed and working
  - wireless access and an active UVM mail account
- Graduate standing as a PhD or MS student at UVM
- Undergraduates by invitation only after consultation with the instructor
- An ongoing research project for which you are collecting and analyzing data
- Knowledge of basic statistics (p-values, means, variances, hypothesis testing)
- No prior knowledge of computer programming is required

Course Content

This course is designed to teach you four things:

1. How to "think on paper" to organize your ideas into explicit hypotheses and interpretations of your data. These methods will allow you to focus and improve your entire thesis in a single morning.

2. How to use modern computational tools to make your analysis, writing, and presentations more efficient and attractive, and to make your analyses transparent and repeatable.
3. Foundational methods in computer programming in R (data structures, functions, control structures, input and output) that will allow you to construct computer models and to easily learn other computer languages (Python, C++).
4. How to recognize, use, and analyze 4 archetypal experimental designs for biologists, apply them to real and simulated data, and create publication quality graphs with the `ggplot2()` package in R.

Traditional and Modern Computational Tools

- Traditional
  - Word/Excel/Powerpoint
  - SAS/SPSS/JMP for stats
  - Sigmaplot
- Commercial
- Proprietary
- Expensive, availability of updates
- Code not available to modify, improve
- File formats usually not compatable
- Poor tools for documentation, transparency
- Modern
  - LaTeX/Markdown/Typora/Overleaf (instead of Word)
  - plain-text editors (instead of Excel)
  - Beamer slides (instead of Powerpoint)
  - R (instead of stats and graphics packages)
- Advantages of Modern
  - free, open source
  - files always usable and readable
  - huge network of expert users (Wikipedia style)
  - less vulnerable to hackers and malware
- Learning to code
  - deeper understanding of problem
  - hypothesis testing
  - life long habit of learning and staying abreast of field

Thinking On Paper

- the logic tree (the basis for "strong inference" that is rarely used explicitly)
- the look-up table (organize the results of multiple statistical analyses with a "dip-switch" test)
- the path diagram (for specifying variables and being explicit about cause-and-effect)
- the simulated data analysis (complete all of your statistical analyses and know how you will interpret the results before your experiment is even finished)

Computational Tools

- Git and GitHub (for tracking and documenting your work)
- RMarkdown (to easily build webpages and pdfs with or without R code)
- Beamer (pdf slides that blow away Powerpoint presentations)
- Regular Expressions (find-and-replace on steroids)
- LaTeX (for type-setting equations and improving your slides)
- learning new tools (how to learn how to do things in R)

Programming Skills

- Basic coding in R studio (functions, scripts, sourcing, console)
- Foundational Programming Methods (annotating code, for loops, control structures, ifelse, and much more)
- Data Files (creating and annotating your data files for safe storage and archival use)
- Stochastic Modeling (using random sampling to add realism to models)
- Null Models (simple permutation tests for estimating p-values)
- Markov Models (transition matrices for population growth, succession, climate change and more)

Experimental Designs

- Graphics (creating publication quality graphics with `ggplot2()`)
- Basic experimental designs (ANOVA, regression, logistic regression, contingency table)
- Data simulation (normal, uniform, poisson, beta, gamma distributions)

Lecture Activities

Once we finish with the preliminaries, the format for the lecture each day is simple. I will open Rstudio from my computer, project the screen so you can see it, and begin coding. I will comment and explain everything as I go along. You will copy what I am typing on the screen into your own computer and the run the code along with me. This is the fastest way for you to learn how to code.

Lab Activities

Each week, there will be a programming assignment to go along with the lecture. Your primary in the laboratory section will be to do this assignment. Using the lab time this way forces you to set aside time each week to get learn the language and get the coding done, minimizes your time spent coding outside of class, and gives you the benefit of being able to ask for help from Lauren and from your fellow students.

Lauren will begin each class with a brief overview of the assignment, and possibly a few suggestions or tips for how to code the exercise.

Student Assignments

In addition to the weekly homework assignments, each student will make a major "teaching presentation" at the end of the semester. You will teach a 30-minute class on how to use an R package to carry out analysis, create graphics, organize data, generate an interactive webpage, or do something else useful in R. You will prepare a script ahead of time, and then lead the entire class (and me) through your exercise, just as I do in each of the lectures.

The last ~3 weeks of lecture and laboratory sessions will be reserved for these student presentations.

Student Responsibilities

- Attend nearly all of the lectures and lab meetings
- Do the coding exercises in class when they are assigned
- Regularly update your course webpage with completed and partially completed assignments
- Study the code of your classmates on their webpages to see how they have solved the same problems you are working on
- Annotate your code so it is useful to you and others
- Surf the internet to find solutions to programming problems

- Share code and ideas, and help others who are struggling
- Prepare well ahead of time for your oral presentations at the end of the semester
- Use the course work to complete some concrete work that will go towards your thesis

Grading

The undergraduate grading paradigm of taking examinations in a limited time frame and doing all of your work exclusively by yourself is antithetical to the programming world. Unless you are at NASA trying to guide an emergency landing for Apollo 13, you will not be writing computer code under a strict time deadline. And, in contrast to the strictures against plagarism in most of science and academia, we will regularly use computer code that others have written and put it to our own use. When you do so, try to include some brief annotation so that you know where the code came from and can get back to the source if you need to. Copying and repeating what others have said or written is how we learn any language, including computer languages. The only thing you should not do is to copy an entire program wholesale without documentation and without any understanding of what it is doing.

Although there will be weekly homeworks, there is no specific due date on these assignments. As soon as you finish them, you will upload them to your individual website. This website is your public "portfolio" that shows your progressive work through the semester. Course grades will be based on an evaluation of your overall portfolio (quality of coding and annotation), and your teaching presentation (quality of oral presentation, depth and thoroughness of coding and explanations)