Advanced Statistical Computing

Modern statistics: the computer is crucial

- Data manipulation
- Calculation of estimates
- Simulation to get SEs and study the behavior of estimators
- Exploratory analysis
- Graphics

C — computer-intensive calculations
Perl — data manipulation; automation
R — exploratory analysis & graphics, prototyping code

C $\leftrightarrow$ Fortran
Perl $\leftrightarrow$ Sed, Awk, shell scripts, "by hand" manipulation
R $\leftrightarrow$ S-plus, Matlab, SAS, etc.

What we need to be able to do

- Programs which automate the entire process of data manipulation and analysis

  Raw data $\rightarrow$ Final output

- Never "manually" edit data files
  - Can easily repeat things
  - Reduce the chance of errors
  - More efficient
- Get the right answers in reasonable time
- Understand our options
- Be able to determine the performance of a procedure
- Write code that may be reused (keep things modular and relatively general)
- Don't (have to) rely on anyone

My approach

Code/C	
Code/Perl	Programs of general use
Code/R	
Projects/GM00	
…./Prostate	
…./Wolf	Different projects
…./Konecki	
…./Tar	[old projects as .tar.gz files]
…./Wolf/Rawdata	Data as I receive it
…./Wolf/Data	Data ready for analysis
…./Wolf/Output	Summary plots and output
…./Wolf/R	R code specific to project
…./Wolf/Perl	Perl code specific to proj.

[Sometimes I have subdirectories for different aspects of analysis; always keep everything!]

Common problems and my solutions

- Many projects w/ similar data but it arrives in many forms
  - Each has program to convert data to a standard form
  - Have all analysis programs work with the standard form
- Keeping track of what you've done
  - Notebook: plastic folder $\rightarrow$ file folder $\rightarrow$ 3-ring binder
  - Comment all code!
  - R/S-plus analyses: source from a file; work within emacs
- Picking names for files [I have no solution]
- Finding bugs
  - Avoid them: plan $\rightarrow$ comment $\rightarrow$ code
  - Perl, C, R all have debugging facilities
  - I use print statements
  - **Always** inspect your results carefully

Commenting code

- What the program does (you'll be surprised how soon you forget)
- Form of data input and output
- Write basic structure of program as comments before you begin coding
- Explain all complex parts (but not the simple stuff)
- Include references
- Books:

Kerninghan and Pike (1999) The practice of programming. Addison-Wesley, Reading, MA

Oualline (1992) C elements of style. M&T Books, San Mateo, CA

How I spend my time

- Manipulating data (25%)
- Figuring out what to do (20%)
- Coding (15%)
- Commenting code (10%)
- Running programs (5%)
- Debugging (15%)
- Studying results (10%)

Of course, this all varies a great deal depending on the project and the data.

Look at (a) how you spend your time and (b) your quality of life and maybe adjust your approach or seek new tools.

Goals of this course

- Convince you to learn to code in C, perl & R
- Basic issues in programming/analysis
- Basics of C, perl, R (mostly R)
- Basic algorithms/recipes:
  what, how, why, properties
- Not covered:
  How to work with the computer, how to code, and many important topics

Homework:

- Problem set to learn R
- Two programming projects (in R)

Office hours: whenever you can find me

C & perl

Why not force you to learn C & perl?

- Time
- Most people around here don't use them anyway

Why should you learn C & perl?

- Perl: efficiency, avoid errors
- C: the big guns
- Don't need to rely on anyone

Other languages:

- Matlab — faster but less complete/slick?
- S-plus — not free
- SAS — I don't want to know it

- Fortran — get with the 80's!
- C++ — unnecessary?
- Java — leave this for last

Basics of a programming language

- Data types
- Data structures
- Syntax
- Loops
- Conditionals
- Functions/subroutines
- Input/Output
- Libraries
- Compiling
- Running
- Debugging

Once you learn one [good] language well, others are not *too* hard to pick up:

- Take a weekend with a good book
- Have a good set of problems at hand
- Force yourself to use your new tool (despite the pain)

C: super brief

- Major use: Speed
- Compiled language:

```
gcc -o program program.c -lm
```

- Pains to avoid
  - Text processing
  - Windows
  - Graphics
- Pains to conquer
  - Pointers
  - Multi-dimensional arrays
  - Dynamic allocation of memory
  - Use of others' code (e.g. IMSL, R)
  - Calling C from R
- Books:

  The C programming language
  Practical C programming
  The standard C library
  Numerical recipes in C
  Programming with GNU software

Perl: super brief

- Major uses:
  - Manipulation of text and data
  - Automation of tasks
- Interpreted language (vs compiled)

```
#!/usr/local/bin/perl
chmod +x myprogram.pl
```

- Scalars, vectors, hashes, multiply indexed vectors/hashes
- Perl takes care of integers/doubles/strings
- C-like syntax, but always need {}
- Library: `/opt/Lwperl`
  `www.cpan.org`
- O'Reilly books:
  - Learning perl
  - Perl CD bookshelf
  - Perl cookbook
  - Programming perl