Introduction

Syllabus and Background

Basics

- Review the course syllabus

  `http://www.stat.uiowa.edu/~luke/classes/STAT7400/syllabus.pdf`

- Fill out info sheets.
  - name
  - field
  - statistics background
  - computing background

Homework

- some problems will cover ideas not covered in class
- working together is OK
- try to work on your own
- write-up must be your own
- do not use solutions from previous years
- submission by GitHub at `http://github.uiowa.edu` or by **Icon** at `http://icon.uiowa.edu/`.

Project

- Find a topic you are interested in.
- Written report plus possibly some form of presentation.

Ask Questions

- Ask questions if you are confused or think a point needs more discussion.
- Questions can lead to interesting discussions.

Computational Tools

Computers and Operating Systems

- We will use software available on the Linux workstations in the Mathematical Sciences labs (Schaeffer 346 in particular).
- Most things we will do can be done remotely by using `ssh` to log into one of the machines in Schaeffer 346 using `ssh`. These machines are

  `l-lnx2`*xy*`.stat.uiowa.edu`

  with $xy = 00, 01, 02, \ldots, 19$.
- You can also access the CLAS Linux systems using a browser at

  `http://fastx.divms.uiowa.edu/`

  - This connects you to one of several servers.
  - It is OK to run small jobs on these servers.
  - For larger jobs you should log into one of the lab machines.
- Most of the software we will use is available free for installing on any Linux, Mac OS X, or Windows computer.
- You are free to use any computer you like, but I will be more likely to be able to help you resolve problems you run into if you are using the lab computers.

Git and GitHub

- Git is a *version control system* that is very useful for keeping track of revision history and collaboration.
- We will be using the University's GitHub server.
- *Today* you should log into the page `http://github.uiowa.edu` with your HawkID.
- I will then create a repository for you to use within the class organization at `https://github.uiowa.edu/STAT7400-Spring-2019`.
- A brief introduction to Git is available at `http://www.stat.uiowa.edu/~luke/classes/STAT7400/git.html`.

What You Will Need

- You will need to know how to
  - run R
  - Compile and run C programs
- Other Tools you may need:
  - text editor
  - command shell
  - `make`, `grep`, etc.

Class Web Pages

The class web page

`http://www.stat.uiowa.edu/~luke/classes/STAT7400/`

contains some pointers to available tools and documentation. It will be updated throughout the semester.

Reading assignments and homework will be posted on the class web pages.

Computing Account Setup: Do This Today!

- Make sure you are able to log into the CLAS Linux systems with your HawkID and password. The resources page at

  `http://www.stat.uiowa.edu/~luke/classes/STAT7400/resources.html`

  provides some pointers on how to do this. If you cannot, please let me know immediately.

- If you have not done so already, log into the page

  `http://github.uiowa.edu`

  with your HawkID to activate your GitHub account.

Computational Statistics, Statistical Computing, and Data Science

Computational Statistics: Statistical procedures that depend heavily on computation.

- Statistical graphics
- Bootstrap
- MCMC
- Smoothing
- Machine lerning
- ...

Statistical Computing: Computational tools for data analysis.

- Numerical analysis
- Optimization
- Design of statistical languages
- Graphical tools and methods
- ...

Data Science: A more recent term, covering areas like

- Accessing and cleaning data
- Working with big data
- Working with complex and non-standard data
- Machine learning methods
- Graphics and visualization
- ...

Overlap: The division is not sharp; some consider the these terms to be equivalent.

Course Topics

- The course will cover, in varying levels of detail, a selection from these topics in *Computational Statistics*, *Statistical Computing*, and *Data Science*:
  - basics of computer organization
  - data technologies
  - graphical methods and visualization
  - random variate generation
  - design and analysis of simulation experiments
  - bootstrap
  - Markov chain Monte Carlo
  - basics of computer arithmetic
  - numerical linear algebra
  - optimization algorithms for model fitting
  - smoothing
  - machine learning and data mining
  - parallel computing in statistics
  - symbolic computation
  - use and design of high level languages
- Some topics will be explored in class, some in homework assignments.
- Many could fill an entire course; we will only scratch the surface.
- Your project is an opportunity to go into more depth on one or more of these areas.
- The course will interleave statistical computing with computational statistics and data science; progression through the topics covered will not be linear.
- Working computer assignments and working on the project are the most important part.

- Class discussions of issues that arise in working problems can be very valuable, so raise issues for discussion.
- Class objectives:
  - Become familiar with some ideas from computational statistics, statistical computing, and data science.
  - Develop skills and experience in using the computer as a research tool.

Thumbnail Sketch of R

- R is a language for statistical computing and graphics.
- Related to the S language developed at Bell Labs.
- High level language
  - somewhat functional in nature
  - has some object-oriented features
  - interactive
  - can use compiled C or FORTRAN code
- many built-in features and tools
- well developed extension mechanism (packages)
  - tools for writing packages
  - many contributed packages available.

Some examples:

- Fitting a linear regression to simulated data:

```
> x <- c(1,2,3,4,3,2,1)
> y <- rnorm(length(x), x + 2, 0.2)
> lm(y ~ x)

Call:
lm(formula = y ~ x)

Coefficients:
(Intercept)            x
      1.887        1.019
```

- A function to sum the values in a vector

```
> mysum <- function(x) {
+      s <- 0
+      for (y in x) s <- s + y
+      s
+ }
> mysum(1:10)
[1] 55
```

Thumbnail Sketch of C

- C is a low level language originally developed for systems programming
- Originally developed at Bell Labs for programming UNIX
- Can be used to write very efficient code
- Can call libraries written in C, FORTRAN, etc. on most systems
- A reasonable book on C is *Practical C Programming, 3rd Edition*, By Steve Oualline. The publisher's web site is

  `http://www.oreilly.com/catalog/pcp3/`

  There are many other good books available.
- A simple example program is available at

  `http://www.stat.uiowa.edu/~luke/classes/STAT7400/examples/hello`.

Example: summing the numbers in a vector:

```
#include <stdio.h>

#define N 1000000
#define REPS 1000

double x[N];

double sum(int n, double *x)
{
    double s;
    int i;

    s = 0.0;
    for (i = 0; i < N; i++) {
        s = s + x[i];
    }
    return s;
}

int main()
{
    double s;
    int i, j;

    for (i = 0; i < N; i++)
        x[i] = i + 1;

    for (j = 0; j < REPS; j++)
        s = sum(N, x);

    printf("sum = %f\n", s);
    return 0;
}
```

Speed Comparisons

Consider two simple problems:

- computing the sum of a vector of numbers
- computing the dot product of two vectors

The directory

```
http://www.stat.uiowa.edu/~luke/classes/STAT7400/
                     examples/speed
```

contains code for these problems in C, Lisp-Stat, and R. Timings are obtained with commands like

```
time ddot
```

for the C versions, and

```
x<-as.double(1:1000000)
system.time(for (i in 1:1000) ddot(x,x))
```

for R.

The results:

Sum	Time (sec)	base = C	base = C -O2
C sum	2.33	1.00	2.21
C sum -O2	1.05	0.45	1.00
R sum	0.81	0.35	0.77
R mysum	21.42	9.21	20.40
C sumk	7.92	3.41	7.54
C sumk -O2	4.21	1.81	4.00
R mysumk	83.15	35.76	79.19

Dot Product	Time (sec)	base = C	base = C -O2
C ddot	2.34	1.00	2.25
C ddot -O2	1.04	0.45	1.00
R ddot	47.85	20.47	46.01
R crossp	1.46	0.63	1.40

Notes:

- R sum means built-in `sum`; R crossp means `crossprod`
- `sumk` and `mysumk` use Kahan summation.

Some conclusions and comments:

- Low level languages like C *can* produce much faster code.
- It is much easier to develop code in an interactive, high level language.
- Usually the difference is *much* less.
- Improvements in high level language runtime systems (e.g. byte compilation, runtime code generation) can make a big difference.
- Using the right high level language function (e.g. `sum`) can eliminate the difference.
- High level language functions may be able to take advantage of multiple cores.
- Speed isn't everything: accuracy is most important!