Scientific Computing for Biologists

Biology 723

Fall 2013

Tue 2:50-5:20

Instructor: Paul M. Magwene

TA: Colin Maxwell

Email: paul.magwene@duke.edu

Phone: 613-8159

27 August 2013

Overview of Lecture

- Course Mechanics
  - Goals of course
  - Structure of lectures
  - Grading
- Introduction to R
  - R resources
  - Important programming concepts
  - Introduction to data types and data structures in R
- Literate programming
- Hands-On Session

Class Structure

- Lectures
  - Typically 60-75 minutes
  - Emphasize the mathematical basis of the methods/approaches from both a geometric and algebraic basis
  - Discuss algorithms underlying the methods
- Hands-on
  - Walk through some examples
  - Apply the techniques and concepts to real data
  - Highlight available R/Python libraries

Syllabus

Date	*Topic*
August 27	Introduction; Getting Acquainted with R
September 3	Data as Vectors: Geometry of Correlation and Regression; Visualizing bivariate data in R
September 10	Descriptive statistics as matrix operations; Visualizing and working with multivariate data in R
September 17	Multiple regression and introduction to biplots; Regression in R
September 24	Non-linear regression models
October 1	Eigenvectors and Eigenvalues; Principal Components Analysis
October 8	Singular Value Decomposition, Biplots, and Correspondence Analysis
October 15	Discriminant analysis and Canonical Variate Analysis
October 22	Fall Break
October 29	Analyses based on Similarity/Distance I; Hierarchical and K-means clustering
November 5	Analyses based on Similarity/Distance II; Multidimensional scaling
November 12	Randomization and Monte Carlo Methods; Jackknife, Bootstrap, Permutation
November 19	Building Bioinformatics Pipelines I; Pipes, redirection, subprocesses
November 26	Building Bioinformatics Pipelines II; Putting the concepts to work

Course Goals

1. Introduce multivariate statistics from a geometric perspective, emphasizing the geometry of vector spaces.
2. Develop a good working knowledge of R (a statistical computing environment), Python (a general purpose programming language), and Unix-like command line (de facto standard for building bioinformatics pipelines).
3. Provide the tools and knowledge to conduct reproducible computational and statistical research.

Texts for Course

- Wickens, T. D. 1995. The geometry of multivariate statistics.
- Matloff, N. 2011. The Art of R Programming.
- Downey, A. B., J. Elkner, and C. Meyers. How to think like a computer scientist: learning with Python.
  - Available at `http://www.ibiblio.org/obp/thinkCSpy/`

Supplementary Texts

- Statistics
  - Krzanowski, W. J. 2003. Principles of multivariate analysis. Oxford University Press.
  - Sokal, R. R. and F. J. Rohlf. 1995. Biometry. W. H. Freeman.
- Math
  - Hamilton, A. G. 1989. Linear algebra: an introduction with concurrent examples. Cambridge University Press.

Grading

- Problem sets/programming assignments
  - 10-12 homeworks over the course of the semester
  - Programming assignments should not be submitted until they produce correct results; I will provide you with scripts/corresponding data to check the correctness of your code
  - No credit for late assignments

Why Both R and Python?

The first half of the course is built around these use of the R programming language; in the second half of the course we will use both R and Python.

- R is geared toward statistical computing
  - Great set of built-in facilities for statistically oriented tasks
  - Somewhat cumbersome syntax for non-statistical tasks
- Python is a general programming language
  - Clearer syntax
  - Wider range of modules
    - web programming, databases, numerical analysis, etc.
  - More natural language for simulation
  - More suitable as a 'glue' language
    - building bioinformatics pipelines

Introduction to R

What is R?

- 'A language and environment for statistical computing and graphics'
- First developed in the mid-90s
- Derives from the S language
  - S was developed at Bell Labs in the mid-80s
- Advantages
  - Free and open-source
  - Much of the academic statistical community has adopted it
  - Active developer and user community
  - Wealth of built-in and user contributed libraries available for all types of analyses
- Disadvantages
  - GUI not as well developed as commercial statistical packages
    - S-Plus; site licensed by Duke - see OIT website
  - Has higher learning curve than some other simpler statistical software
  - Command-line can be intimidating

R Resources on the Web

- Home Page
  - `http://www.r-project.org`
- Comprehensive R Archive Network (CRAN)
  - `http://cran.r-project.org/mirrors.html`
  - See especially the 'Task Views'
    - Statistical and population genetics
    - Environmental and ecological analysis
    - Spatial statistics
- Introductions and Tutorials
  - see `http://cran.r-project.org/other-docs.html`

Some R Packages of Interest

- Bioconductor – software package geared towards analysis of genomic data, especially microarray data, `http://www.bioconductor.org/`
- ape – 'Analysis of Phylogenetics and Evolution', `http://ape.mpl.ird.fr/`
- ade4 – Analysis of Ecological Data : Exploratory and Euclidean methods in Environmental sciences, `http://pbil.univ-lyon1.fr/ADE-4/home.php?lang=eng`

Some Important Programming Concepts

- Data Types
  - refer to the types of values that can be represented in a computer program
  - determine the representation of values in memory
  - determine the operations you can perform on those values
  - Examples: integers, strings, floating point values
- Data Structures
  - a way of storing collections of data
  - different structures are more efficient for particular types of operations
  - Examples: lists, hash tables, stacks, queues, trees
- Variables
  - Variables are references to objects/values in memory
  - Think of them as labels that point to particular places in a computer's memory

More Important Programming Concepts

- Statement
  - an instruction that a computer program can execute
  - Example: `print("Hello, World!")`
- Operators
  - Symbols representing specific computations
  - Example: `+`, `-`, `*` (addition, subtraction, multiplication)
- Expression
  - a combination of values, variables, and operators
  - Example: `1 + 1`
- Functions (subroutines, procedures, methods)
  - A piece of code that carries out a specific task, set of instructions, calculations, etc.
  - Typically used to encapsulate algorithms

Basic Data Types, Data Structures and Operators in R

Numeric Data Types in R

- Floating point values ('doubles')

```
> x <- 10.0
> typeof(x)
[1] "double"
```

- Complex numbers

```
> x <- 1+1i
> typeof(x)
[1] "complex"
```

- Integers
  - Default numeric type is double, must explicitly ask for integers if single values

```
> x <- as.integer(10)
> typeof(x)
[1] "integer"
```

Additional Data Types in R

- Boolean('logical')

```
> x <- TRUE # or x <- T
> x <- F # or x <- FALSE
> typeof(x)
[1] "logical"
```

- Character strings

```
> x <- 'Hello' # or x <- "Hello"
> typeof(x)
[1] "character"
```

Arithmetic Operators and Mathematical Functions in R

```
> 10 + 2 # addition
[1] 12
> 10 - 2 # subtraction
[1] 8
> 10 * 2 # multiplication
[1] 20
> 10 / 2 # division
[1] 5
> 10 ^ 2 # exponentiation
[1] 100
> 10 ** 2 # alternate exponentiation
[1] 100
> sqrt(10) # square root
[1] 3.162278
> 10 ^ 0.5 # same as square root
[1] 3.162278
> pi*(3)**2  # R knows some useful constants
[1] 28.27433
> exp(1) # exponential function
[1] 2.718282
```

Simple Data Structures in R: Vectors

Vectors are the simplest data structure in R

- vectors represent an ordered list of items

```
> x <- c(2,4,6,8)
> y <- c('joe','bob','fred')
```

- vectors have length (possibly zero) and type

```
> typeof(x)
[1] "double"
> length(x)
[1] 4
> typeof(y)
[1] "character"
```

Simple Data Structures in R: Vectors

Accesing the objects in a vector is accomplished by 'indexing':

- The elements of the vector are assigned indices `1`...n where n is the length of the vector

```
> x <- c(2,4,6,8)
> length(x)
[1] 4
> x[1]
[1] 2
> x[2]
[1] 4
> x[3]
[1] 6
> x[4]
[1] 8
```

Simple Data Structures in R: Vectors

- Single objects are usually represented by vectors as well

```
> x <- 10.0
> length(x)
[1] 1
> x[1]
[1] 10
```

- Every element in a vector is of the same type
  - If this is not the case the the values are coerced to enforce this rule

```
> x <- c(1+1i, 2+1i, 'Fred', 10)
> x
[1] "1+1i" "2+1i" "Fred" "10"
```

Arithmetic Operators Work on Vectors in R

Most arithmetic operators work element-by-element on vectors in R

```
> x <- c(2, 4, 6, 8)
> y <- c(0, 1, 2, 3)
> x + y
[1]  2  5  8 11
> x - y
[1] 2 3 4 5
> x * y
[1]  0  4 12 24
> x^2
[1]  4 16 36 64
> sqrt(x)
[1] 1.414214 2.000000 2.449490 2.828427
```

Simple Data Structures in R: Lists

Lists

- Lists in R are like vectors but the elements of a list are arbitrary objects (even other lists)

```
> x <- list('Bob',27, 10, c(720,710))
> x
[[1]]
[1] "Bob"

[[2]]
[1] 27

[[3]]
[1] 10

[[4]]
[1] 720 710
```

Simple Data Structures in R: Lists

Accessing objects in Lists:

- Items in lists are accessed in a different manner than vectors.
  - Typically you use double brackets (`[[]]`)to return the element at index `i`
  - Single brackets always return a list containing the element at index `i`

```
> x <- list('Bob', 27, 10, c(720,710))
> typeof(x[1])
[1] "list"
> typeof(x[[1]])
[1] "character"
```

Simple Data Structures in R: Lists

- Objects in R lists can be named

```
> x <- list(name='Bob',age=27, years.in.school=10)
> x
$name
[1] "Bob"

$age
[1] 27

$years.in.school
[1] 10
```

- Named list objects can be accessed via the $ operator

```
> x$years.in.school
[1] 10
> x$name
[1] "Bob"
```

- The names of list objects can be accessed with the names() function

```
> names(x)
[1] "name"  "age"  "years.in.school"
```

Literate Programming

“Literate programming” is a concept coined by Donald Knuth, a preeminent computer scientist:

- Programs are useless with descriptions
- Descriptions should be literate, not comments in code or typical reference manuals.
- The code in the descriptions should work.

Literate Programming and Reproducible Research

How literate programming can help to ensure your research is reproducible:

- The steps of your analyses are explicitly described, both as written text and the code and function calls used.
- Analyses can easily checked for correctness and reproduced from your literate code.
- Your literate code can serve as a template for future analyses, saving you time and the trouble of remembering all the gory details.

Tools for literate programming in R and Python

How literate programming can help to ensure your research is reproducible:

- R – knitr; tool for weaving together R code and text to produce 'computable' documents that can be output as HTML or LaTeX.
- Python – Ipython "notebooks"; Mathematica like interactive computing environments that intermingle code, graphics, and text.

A Literate programming Example

Literate programming tools typically use simple markup conventions in which you weave your code into your description by putting it between delimiter blocks

Example:

```
Here are some trivial R examples that will help to
illustrate how knitr works:

<<>>=
z <- 1:10
mean(z)
summary(z)
z[z > 5]
@

The above text was a code block woven into my
description. It gets evaluated and integrated into
the output. Cool, eh?
```

knitr output

Output produced by knitr and LaTeX for the code on the previous slide:

Here are some trivial R examples that will help to illustrate how knitr works:

```
z <- 1:10
mean(z)

## [1] 5.5

summary(z)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.25    5.50    5.50    7.75   10.00

z[z > 5]

## [1]  6  7  8  9 10
```

The above text was a code block woven into my description. It gets evaluated and integrated into the output. Cool, eh?

Fancier knitr output

Here's a somewhat fancier example of what knitr can produce:

A knitr example that incorporates graphics is always nice. First, let's generate the data by drawing 1000 observations from the standard normal ($\mu = 0, \sigma = 1$).

```
data <- rnorm(1000)  # 1000 obs. drawn from standard normal
```

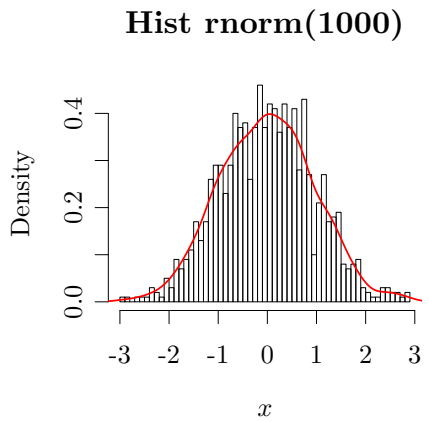
Next, we create a summary table:

```
summary(data)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.9800 -0.6810  0.0011 -0.0046  0.6400  2.8700
```

Finally, we create a nice figure in which a density estimate is superimposed on a histogram:

```
hist(data, breaks = 50, freq = F, main = "Hist rnorm(1000)", xlab = "$x$")
lines(density(data), col = "red", lwd = 2)
```



Things to Remember

- Try it out - programming involves experimentation
- Practice - learning to program, like learning a foreign language, requires lots of practice.
- Persist - many new tools/concepts can be hard to grasp at first. Keep plugging away until you get that ‘Aha!’ moment

You might be surprised to find that...

- Programming is fun! (at least sometimes)
- Math is fun! (at least sometimes)
- Statistics is fun! (at least sometimes)

- Gaining new insights into how your biological system of interest works is fun! (always)