Scientific Computing for Biologists

Biology 723
Fall 2012
Tue 2:50-5:20

Instructor: Paul M. Magwene

Email: paul.magwene@duke.edu

Phone: 613-8159

28 August 2012

Overview of Lecture

- Course Mechanics
  - Goals of course
  - Structure of lectures
  - Grading
- Introduction to R
  - Advantages of R and Python
  - R and Python Resources
  - Important programming concepts
  - Introduction to data types and data structures in R and Python
  - 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 28	Introduction; Getting Acquainted with R
September 4	Data as vectors; Vector operations: dot product, correlation, regression as projection
September 11	Linear Algebra Review I; Descriptive statistics as matrix operations, multivariate visualizations
September 18	Linear Algebra Review II; Regression models
September 25	Eigenvectors and Eigenvalues; Principal Components Analysis
October 2	Singular Value Decomposition, Biplots, and Correspondence Analysis
October 9	Discriminant analysis and Canonical Variate Analysis
October 16	Fall Break
October 23	Analyses based on Similarity/Distance I; Hierarchical and K-means clustering
October 30	Analyses based on Similarity/Distance II; Multidimensional scaling
November 6	Randomization and Monte Carlo Methods; Jackknife, Bootstrap, and Permutation Tests
November 13	Building Simulations in Python
November 20	Building Bioinformatics Pipelines I; Pipes, redirection, subprocesses
November 27	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) and Python (a general purpose programming language).

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 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
    - Overall grade: assignments are worth one point each, for a total of 10 points for the semester. Final grades will be assigned based on this scale: 9+ pts = A, 8 = B, 7 = B-, 6 = C, 5 = F

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 $1 \ldots n$ 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
```

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"
```

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)