Overview

January 15, 2019

Why R?

- Free of charge (though paid support options are available).
- Open source and extensible.
- Over 13,000 available libraries for all kinds of specialized tasks (and they are all free!).
- Very popular *programming language* for statistics. "It promotes experimentation and exploration, which improves data analysis."
- Very easy to write and share custom code.
- Great for visualization. Excellent packages for graphics, e.g., ggplot2.
- A very active and helpful community.
- Very flexible: Good support for metaprogramming
  - Functions, environments, and expressions are first-class citizens.
  - Call stack directly accessible.
  - "Non-standard evaluation" can be used to build domain-specific languages (DSLs) within R.
  - Supports custom operators.
  - May "shadow" built-in functions/operators, e.g., you may write your own assignment operator to replace the built-in one.
- Supports array-based programming.
- Easy administration. Installing and updating packages is a breeze.
- Easy to parallelize.
- Supported on Windows, Linux, and OS X (Mac).
- Has a very nice interface to Spark for Big Data tasks (cluster computing platform).

A few disadvantages

- Can be a little slow.
- Assumes data fits into main memory. Special packages required to work with larger data sets.
- Weakly typed. Implicit type conversions are common, and these can lead to hard-to-find bugs.
- Documentation could be better.
- No official support line.
- Good support for multiprocessing but weak support for multithreading.

What is R anyway?

"R was born as Lisp with some syntactic sugar on top so that it would look like S" – Pat Burns

R is a dialect of the S statistical language and is largely inspired by Lisp. If you are coming from SAS or Stata, you are better off thinking about it as a programming language and not as a statistical environment. Although statistics is its primary use case, it is a "proper" programming language, so it may be used for more general tasks.

- Interpreted: Slower execution than compiled languages but potentially faster development time (immediate feedback, no compilation step).
  - May be pre-compiled into bytecode (rarely done outside of packages) for faster execution.
  - Includes a Just-In-Time (JIT) bytecode compiler. In recent versions of R, this is turned on by default.
  - Bottlenecks sometimes written in a compiled language such as C.
- Dynamically typed: Data types are associated with values, not variables. Type checking occurs on as as-needed basis at runtime. Harder to identify bugs but makes metaprogramming easier, less "language bureaucracy."
- Lexically scoped: variable lookup is based on where a function is defined (lexical), not where it is called (dynamic).

  - Under lexical scoping, the structure of your running program matches the structure of your source code. Under dynamic scoping, they differ, which makes it hard to follow program logic.
- Generally favors value semantics, not reference semantics.
- Dynamic scoping and reference semantics can be emulated as needed. Do so sparingly.
- Multi-paradigm: Supports different styles of programming. Idiomatic R is mostly functional but with elements of object-oriented and procedural programming.
- Interactive or batched: May run scripts and/or enter commands through a REPL (Read-Eval-Print Loop). Interactive programming useful for experimentation and debugging.
- May be interfaced with a number of other languages: C, C++, Fortran, Tcl, Java, Python. The C++ interface is especially popular and easy to use.
- Function parameters are evaluated lazily (as needed). Variable assignments are eager (on demand).
  - These are merely defaults. You can do lazy variable assignments and eager parameter evaluation upon request.
- Variables/functions are late-bound. Before running your code, R only checks that your syntax is correct. It doesn't verify that variables exist until you attempt to use them.
  - This can delay the discovery of bugs, but it makes some things easier: no need to "forward declare" variables.

R source files are usually marked by the `.R` file extension. This is merely a common convention (R doesn't care what extension is used). Other R-related file extensions include .Rc (bytecode), .RData, .Rda, and .RDS.

Functional programming in R

R is a functional language, meaning that computation involves evaluation of functions. It is not a so-called *pure* functional language like Haskell but is instead functional in a more traditional sense, like Lisp.

- Writing R code mostly involves writing and calling functions. Functions take a number of inputs and return an output.
  - Functions in R are not required to be "pure."" That is, they may involve side effects like I/O and non-local assignment. However, it's good practice to make most of your functions pure and to avoid non-local assignment without good justification.
- Functions in R are first class citizens, i.e., functions are just another data type.
- Idiomatic R code involves the extensive use of higher order functions, i.e., functions that take other functions as parameters and/or return a function.
  - Explicit loops are supported by the language, but their use in R is generally regarded as bad practice. Use recursion or higher order functions instead.
- With a few exceptions, values are immutable (e.g., object methods generally cannot mutate an object), but variables are mutable (can be rebound to new values).
- If you have used Lisp before, think of R as Lisp without all the parentheses. It's not exactly true, but it's close!

Object-oriented programming in R

R is also an object-oriented language. In fact, it is a *purely* object-oriented language. Object-oriented programming is a paradigm which utilizes *objects* that contain both data (*fields*), as well as the functions (*methods*) that act upon that data. It provides several benefits that can help tame complexity in software. Object-oriented programming will not be a major topic in this course, but be aware that object-oriented features are available.

SAS vs. R

- SAS: Licensed, closed source. R: Free, open source. New methods are almost always released in R first.
- User code in SAS is written "on top" of the SAS system. User code in R is (generally) on the same level as the base libraries. Imagine a version of SAS where your code consists of writing your own PROCs.
- SAS has centralized support; R does not.
- SAS is split into many sub-languages: DATA step, PROCs, macros, IML, SCL, ODS, etc. R is a little more uniform: processing of all kinds is handled by function application.
- SAS is very structured–its historical roots in sequential batch processing are obvious. R is more freeform.

- SAS is very unique. R more closely resembles other modern languages (e.g., Python, Matlab, Javascript), which makes it easier to transfer skills.
- Unlike SAS, R assumes data fit into main memory. Special packages required for larger data sets.
- R is a common tongue between statisticians and data scientists. Most data scientists don't use SAS. Most statisticians don't use Python. Both groups use R.
- Output in R is usually more terse. If you want something, you have to ask for it.
- Idiomatic R does not involve the use of macros (there is a "defmacro" function, but no one uses it). Write functions instead of macros.
  - No big loss. R doesn't need macros.
- SAS is mostly a procedural language. R is mostly functional with some elements of object-oriented & procedural programming.

RStudio

R can be downloaded from the Comprehensive R Archive Network, CRAN. We will be using RStudio, a popular IDE. It is important to keep in mind that R (the language) and RStudio (the GUI) are separate things, and it is entirely possible to use different workflows with other tools or text editors:

- `emacs` through ESS.
- `vim` with the Vim-R-Plugin.
- Sublime Text.
- Scite.
- Notepad++.

A few useful resources

There is a constantly growing collection of materials available offline and online to learn R. The Journal of Statistical Software and the Use R! series from Springer regularly publish applications of R to different domains.

A good overview for beginners is Learning R.

SAS users may find useful R for SAS and SPSS users, although I have never used it myself.

For the analysis of compley survey data, you may want to take a look to "Complex Surveys. A Guide to Analysis Using R".

The official documentation in CRAN (The Comprehensive R Archive Network) is available to read but goes well beyond the scope of this class.

Looking around

RStudio offers four basic windows.

- Console (R interpreter)
- Code, where we will write our code.
- History/Environment
- Plots/Packages/Help

Getting help

Documentation in R can be accessed through the interpreter. For instance, if we wanted to get information about what `lm` does, or what paramaters it takes or some examples of usage, we would type:

```
?lm
```

To search for a topic, one can type:

```
??"nonlinear regression"
help.search("nonlinear regression") # alternative syntax
```

Note that the above only searches through installed packages. Better search method: Google. ;)

The R community is very helpful and active. If you ever get stuck in a problem, the best solution is to ask in StackOverflow, a very large community of programmers using the `#r` tag.

Like other single-letter languages, R can be tricky to Google. Try: "R programming," "R statistics."

Within Westat, there is a growing community of users and we have a number of resources for Q&A and sharing information or announcements.