STAT 385: Statistics Programming Methods

James Balamuta

Department of Statistics
University of Illinois at Urbana-Champaign
balamut2@illinois.edu

June 13, 2016
Summer Session 2 - 2016

On the Agenda

1. Intro
   - Profiles
   - Course Structure
2. Intro to Programming
   - Motivation
   - Basics
3. Intro to R
   - Background of R
   - RStudio IDE
   - Sample Code
4. Types of Languages
   - Interpreters
   - Compilers

On the Agenda

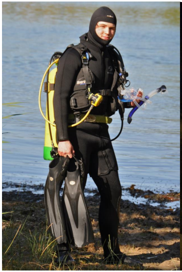
1. Intro
   - Profiles
   - Course Structure
2. Intro to Programming
   - Motivation
   - Basics
3. Intro to R
   - Background of R
   - RStudio IDE
   - Sample Code
4. Types of Languages
   - Interpreters
   - Compilers

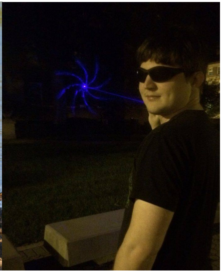
Hello

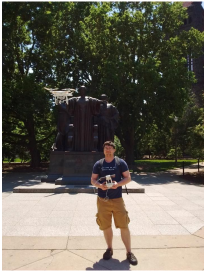
my name is

James

Who am I?

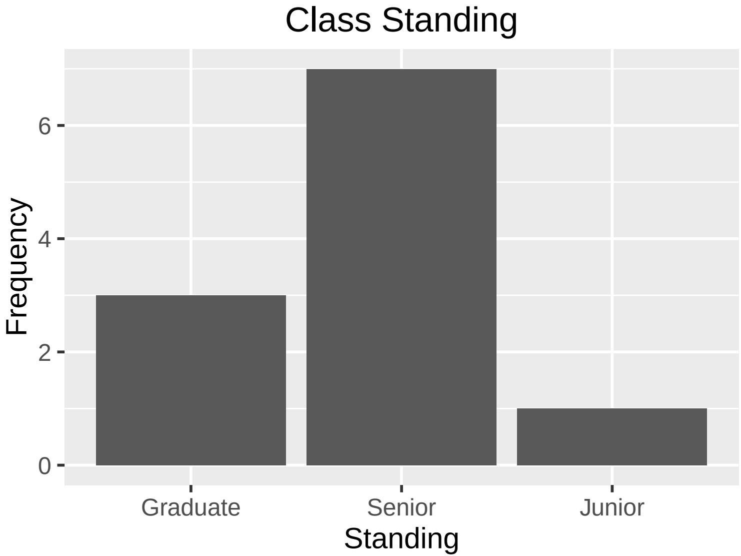




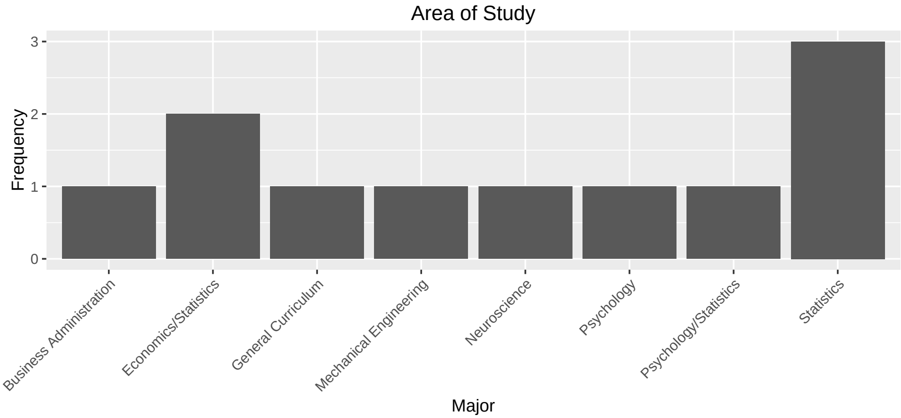


- 3rd Year PhD Statistics/Informatics
- Research
  - NASA Carbon Monitor System Project
  - Time Series Latent Variable Estimation
  - Choice in Psychometric Models
- Teaching
  - List of Excellent Teachers (SU 2014)
  - Created three courses:
    - STAT 330: Data Visualization
    - STAT 385: Statistics Programming Methods (yes, **this** course!)
    - STAT 480: Data Science Foundations

You are...


Class Standing
Frequency
0
2
4
6
Graduate
Senior
Junior
Standing

You are...


Area of Study
Frequency
0
1
2
3
Business Administration
Economics/Statistics
General Curriculum
Mechanical Engineering
Neuroscience
Psychology
Psychology/Statistics
Statistics
Major

Survey says...

Entrance Survey Responses

Now you!

What's your name?

Why are you here?

Course Websites

Material will be posted to

http://stat385.thecoatlessprofessor.com/

Source code: https://github.com/coatless/stat385/

Course Websites

Discussion and Questions should be posted to

https://piazza.com/illinois/summer2016/stat385

Online Analytical Environment

https://rstudio.stat.illinois.edu/

About the Course

- Emphasize computing theory and methods for statistical algorithms
- Learn about computing to use in a future career or graduate school
- Will primarily cover R and C++ (through Rcpp)

About the Course

- Lots of work is ahead of you!
  - Writing code, reading chapters, and working in a group.
- Focus is on creating content rather than consuming it.
- Lots of open spaces, invite your friends! (Just fill out the audit form)

Course Objectives

- View different statistical concepts presented from STAT 200 (and select foundational topics from 400-level STAT courses) from a programming perspective instead of a purely theoretical framework.
- Implement different statistical algorithm.
- *Explain* the underlying algorithm.
- Use version control
- Distributed computing
- Handle a Whiteboard interview
- Group capstone project

Group Projects

The group capstone project is meant to showcase the knowledge that you have acquired throughout the course. This is an example that will be invaluable to you when applying for a job or to graduate school.

- Defines your experience in this course;
- Must be fully functional;
- Choose groups carefully!

Types of Projects

- Changes pre-existing functions in a statistical package
- Enables *new* features in a statistical computing environment
- You pick!
  - Graphics? Fantasy Sports Reader? Grading Application?

Point Distribution

Type	Points Per	Total Points
Participation	25	25
Homework	8 × 25	200
Exam	125	125
Group Project	4 × 37.5	150
Course Total		500

Table: Course Point Distribution

One last thing...

This course has been made possible by the encouragement, understanding (of the many errors), and enthusiasm from the following folks:

- Prof. Jeffrey Douglas
- Dr. Alexey Stepanov
- Prof. Steven Culpepper
- Prof. Douglas Simpson
- Prof. Stephane Guerrier

Thank you!

On the Agenda

1. Intro
   - Profiles
   - Course Structure
2. Intro to Programming
   - Motivation
   - Basics
3. Intro to R
   - Background of R
   - RStudio IDE
   - Sample Code
4. Types of Languages
   - Interpreters
   - Compilers

Age Old Question

"If a tree falls in a forest and no one is around to hear it, does it make a sound?"

Age Old Question Redux

Original:

"If a tree falls in a forest and no one is around to hear it, does it make a sound?"

Changes to:

"If a *statistical algorithm* exists and no one *uses* it, does it really exist?"

Or more critically:

"If a *statistical result* exists and no one *understands* it, does it really exist?"

Technology?

Today, most people are *users* of a computer.

They do **not** need to know how a computer works.

The majority of folks simply turn on technology and immediately see a graphic that they can click or tap with a finger.

Take for example getting a sports score from ESPN.

How to interact with a computer program is only **known**.

Computer == Scary?

This rationale has existed for awhile since computers are a bit scary...
Like spiders...

What is programming?

Definition:

Programming is the art of instructing a computer to do exactly what you say through an *algorithm*.

Definition:

Algorithms are a process or set of rules to be followed in calculations or other problem-solving operations

Programming Defines the 21st Century Tool

"Humans are tool builders and we build tools that can dramatically amplify our innate human abilities. Of all of the inventions of humans, the computer is going to rank near if not at the top as history unfolds and we look back. And it is the most awesome tool that we have ever invented."

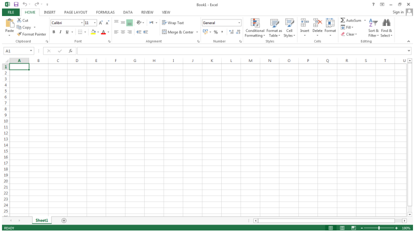
– Steve Jobs (from the Lost Interview)

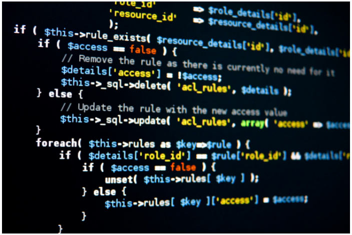
Why study programming now?

Programming has been available from the advent of computers. But, why am I hearing about it now?

What's changed?

A shift from working within **excel** to **automation**





```
                        'role_id'            => $role_details['id'],
                        'resource_id'        => $resource_details['id'],
                        );
if ( $this->rule_exists( $resource_details['id'], $role_details['id
    if ( $access == false ) {
        // Remove the rule as there is currently no need for it
        $details['access'] = !$access;
        $this->_sql->delete( 'acl_rules', $details );
    } else {
        // Update the rule with the new access value
        $this->_sql->update( 'acl_rules', array( 'access' => $acces
    }
    foreach( $this->rules as $key=>$rule ) {
        if ( $details['role_id'] == $rule['role_id'] && $details['r
            if ( $access == false ) {
                unset( $this->rules[ $key ] );
            } else {
                $this->rules[ $key ]['access'] = $access;
            }
        }
```

Plus, a lot more CPU power on a traditional desktop.

Benefits

The adoption of programming methods has several benefits:

1. Speed
2. Consistency
3. Resources
4. Computer Savviness
5. Logic

On the Agenda

1. Intro
   - Profiles
   - Course Structure
2. Intro to Programming
   - Motivation
   - Basics
3. Intro to R
   - Background of R
   - RStudio IDE
   - Sample Code
4. Types of Languages
   - Interpreters
   - Compilers

Learning a new language is HARD!

Ready?

What is R?

- R is a language designed specifically for statistical computing and graphics
- R is an interactive interface to many different tools.
- R is based on the S language, which was developed by Bell laboratories
- R is an open source (e.g. free) project that is cross platform (OS X, Windows, and Linux)
- R is available on *The R Project for Statistical Computing* website http://www.r-project.org

Why R?

Pros:

- It's free!
- Large repository of packages that often contain the latest breakthrough statistical methods
- Able to integrate Fortran, C, C++, and Python code via wrappers
- UIUC STAT Dept. Standard

Cons:

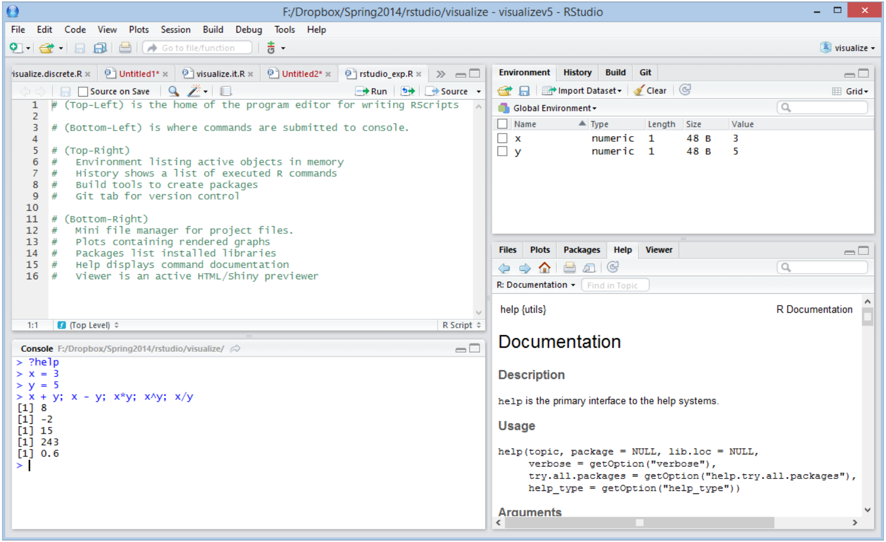
- Objects are always kept in RAM leading to Total System RAM constraining tasks
- Pass by value (e.g. make a copy) quickly eats up available RAM
- Very steep learning curve
- Skin and bones UI

RStudio

RStudio is an Integrated Developer Environment (IDE) for R.

- Advanced GUI that emphasizes a project workflow
- Provides support for a novice user and an advanced user
- Open source (e.g. free) project that is cross platform (OS X, Windows, and Linux)
- Download RStudio via http://www.rstudio.com
- OR use RStudio `https://rstudio.stat.illinois.edu`

RStudio View



F:/Dropbox/Spring2014/rstudio/visualize - visualizev5 - RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
visualize
isualize.discrete.R × Untitled1* × visualize.it.R × Untitled2* × rstudio_exp.R ×
Source on Save
Run
Source
1:1
(Top Level)
R Script
Console F:/Dropbox/Spring2014/rstudio/visualize/
Environment History Build Git
Import Dataset
Clear
Grid
Global Environment
Files Plots Packages Help Viewer
R: Documentation
Find in Topic

```
# (Top-Left) is the home of the program editor for writing RScripts

# (Bottom-Left) is where commands are submitted to console.

# (Top-Right)
#   Environment listing active objects in memory
#   History shows a list of executed R commands
#   Build tools to create packages
#   Git tab for version control

# (Bottom-Right)
#   Mini file manager for project files.
#   Plots containing rendered graphs
#   Packages list installed libraries
#   Help displays command documentation
#   Viewer is an active HTML/Shiny previewer
```

```
> ?help
> x = 3
> y = 5
> x + y; x - y; x*y; x^y; x/y
[1] 8
[1] -2
[1] 15
[1] 243
[1] 0.6
> 
```

□	Name	Type	Length	Size	Value
□	x	numeric	1	48 B	3
□	y	numeric	1	48 B	5

help {utils} R Documentation

Documentation

Description

`help` is the primary interface to the help systems.

Usage

```
help(topic, package = NULL, lib.loc = NULL,
     verbose = getOption("verbose"),
     try.all.packages = getOption("help.try.all.packages"),
     help_type = getOption("help_type"))
```

Arguments

Warming up to R

To begin our exploration of the R language, we'll use R to mimic a scientific calculator. Scientific calculators are able to:

- Compute mathematical expressions.
- Temporarily store values in a variable.

Explanations of the code, are given by comments predated by a #.

Output from the code is given by two ##.

Storing Values and Calculations

```r
# Create numeric object with values
x = 3
y = 5

# Perform calculations
x + y

## [1] 8

x - y

## [1] -2

# x*y; x/y; x^y;
```

Vectors

In R, a number like **5** is treated as a vector, or a collection of values, with **length of 1**.

We will see at a later time that this behavior, while odd, is actual pretty great to *vectorize* computations.

Vectors

```
x = c(1,2,3,4,5) # Create vector
y = 6:10         # Shorthand

cbind(x, y)      # Combine Columns to form Matrix: 5 x 2

##      x  y
## [1,] 1  6
## [2,] 2  7
## [3,] 3  8
## [4,] 4  9
## [5,] 5 10

rbind(x, y)      # Combine Rows to form Matrix: 2 x 5

##   [,1] [,2] [,3] [,4] [,5]
## x    1    2    3    4    5
## y    6    7    8    9   10
```

Built in Functions and Loops

Like any good programming language before it, R has built in functions to aide in the workflow.

A small sampling of functions is:

- `sum` - Summation over elements $\sum_{i=1}^{n} x_i$
- `mean` - Average over elements $\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$
- `sd` - Standard Deviation over elements $\sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2}$
- `sample` - Random sample from $x_1, x_2, \ldots, x_i, \ldots, x_n$

To get help, use

```
?function_name
```

Built-in Functions & Loops

```
x = seq(1, 10, by = 2)   # 1, 3, 5, 7, 9
y = seq(10, 30, by = 5)  # 10, 15, .. , 30

result = numeric(1)      # Storage
for(i in 1:length(x)){   # (variable in sequence)
  result = x[i] + result
}                        # Loop (slow)

(out = sum(x))           # Vectorized Function (faster)

## [1] 25

all.equal(result, out)   # Same value

## [1] TRUE
```

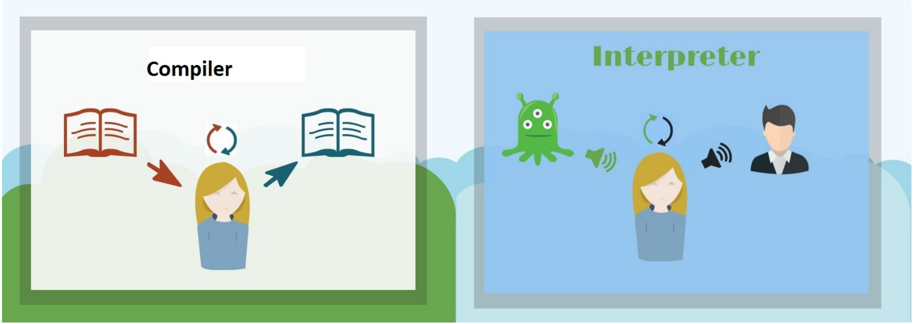
On the Agenda

1. Intro
   - Profiles
   - Course Structure
2. Intro to Programming
   - Motivation
   - Basics
3. Intro to R
   - Background of R
   - RStudio IDE
   - Sample Code
4. Types of Languages
   - Interpreters
   - Compilers

Talking to a Computer

- In order to talk to a computer, you must speak its dialect.
- The dialect though is normally in 1's and 0's (or binary).
- Until Rear Admiral Grace M. Hopper came along...

Now, we have the option of:

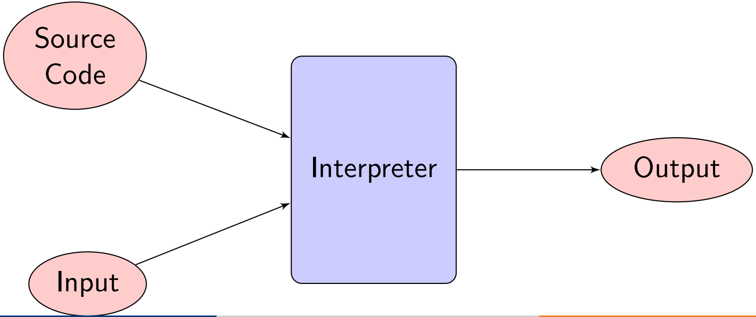


What is an Interpreter?

An *interpreter* is a program that translates a high-level language into a low-level one, but it does it at the moment the program is run.

So, the interpreter takes the source code, one line at a time, and translates each line before executing it. Every time the program runs.

Think of like a person providing a “real time translation” to a conversation.



Interpreters: Pros & Cons

As a result of the program being instantly translated, it is able to immediately provide feedback (e.g. output, errors, etc).

For example, entering the following into R yields:

```
3+4

## [1] 7
```

The downside to this approach are:

1. The lack of optimized code
2. Constant translation

A looping example

```
bad.loop = function(){
  sum = 0
  for(i in 1:1000){
    a = 1/sqrt(2) # In loop
    sum = (sum+i)*a
  }
  sum
}
```

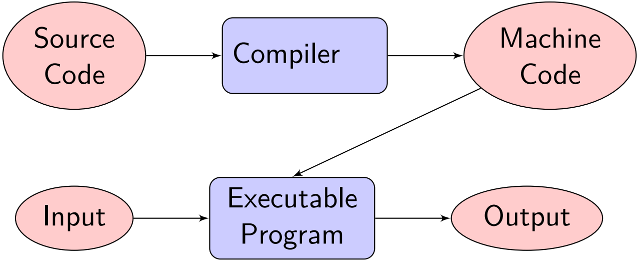
```
good.loop = function(){
  sum = 0
  a = 1/sqrt(2) # Out of Loop
  for(i in 1:1000){
    sum = (sum+i)*a
  }
  sum
}
```

test	replications	elapsed	relative	user.self	sys.self
good.loop()	100	0.045	1.000	0.041	0.003
bad.loop()	100	0.059	1.311	0.057	0.001

What is a Compiler? Can it do my taxes?

A compiler takes source code tries to optimize it before converting it into machine language **once**. After it is done compiling, the code can then be ran again and again without ever needing to be recompiled.

So, a compiler is like an editor who is asked to look over a paper. If it thinks something can be better, then it will take the initiative and implement that option.



Compilers: Pros

Compilers will attempt to optimize the code that they are given.

After succesful compiled, code will not need to be compiled again*.

Base R

```r
good.loop = function(){
  sum = 0
  a = 1/sqrt(2) # Out of Loop
  for(i in 1:1000){
    sum = (sum+i)*a
  }
  sum
}
```

Bytecompiled R

```
## list(.Code, list(8L, LDCONST.OP, 1L, SETVAR.OP, 2L, POP.OP, LDCO
##     3L, GETFUN.OP, 4L, PUSHCONSTARG.OP, 5L, CALL.OP, 6L, DIV.OP,
##     7L, SETVAR.OP, 8L, POP.OP, LDCONST.OP, 3L, LDCONST.OP, 9L,
##     COLON.OP, 10L, STARTFOR.OP, 12L, 11L, 42L, GETVAR.OP, 2L,
##     GETVAR.OP, 11L, ADD.OP, 13L, GETVAR.OP, 8L, MUL.OP, 14L,
##     SETVAR.OP, 2L, POP.OP, STEPFOR.OP, 29L, ENDFOR.OP, POP.OP,
##     GETVAR.OP, 2L, RETURN.OP), list({
##     sum = 0
##     a = 1/sqrt(2)
##     for (i in 1:1000) {
##         sum = (sum + i) * a
##     }
##     sum
## }, 0, sum, 1, sqrt, 2, sqrt(2), 1/sqrt(2), a, 1000, 1:1000, i,
##     for (i in 1:1000) {
##         sum = (sum + i) * a
##     }, sum + i, (sum + i) * a))
```

Compilers: Cons

Compilers spend a lot of time analyzing and processing the program.

They require the **ENTIRE** program to be sent.

There is additional storage required to handle the machine generated code.

Errors will only appear **AFTER** the entire program is analyzed.

Did I mention compilers take a lot of time?

A looping example redux

After applying a compiler, there should be a noticeable change...

```
library("compiler")
good.comp = cmpfun(good.loop)
bad.comp = cmpfun(bad.loop)
```

test	replications	elapsed	relative	user.self	sys.self
good.comp	100	0.007	1.000	0.007	0.000
bad.comp	100	0.020	2.857	0.019	0.001
good.loop	100	0.039	5.571	0.035	0.001
bad.loop	100	0.057	8.143	0.054	0.001

To summarize..

Compilers Pros:

1. Code is optimized.
2. Program runs really fast... (Vroom Vroom fast)
3. Compiles once.*
4. Conditional statements are faster

Cons:

1. Lots of time spent analyzing and optimizing code
2. More storage required due to machine code.
3. Errors only show at the end

Interpreters Pros:

1. Immediate Feedback
2. Less storage required
3. Friendlier

Cons:

1. Takes single instruction as input.
2. Constant compiles from high-level to low-level
3. Slow program execution
4. Conditional statements are slower.

*no errors, changes, etc.

Questions? Comments?

James Balamuta

balamut2@illinois.edu

stat385.thecoatlessprofessor.com