A Third Presentation

on

R

Edgar Hassler
Arizona State University
Hold on a minute! What happened to the second presentation on R?
Okay, I guess we'll call it
R

Spooktacular!

Edgar Hassler
Arizona State University
OUTLINE

- My research
- How R Is Different Than <Programming Language>
- Error Handling
  - tryCatch() and friends
- Parallelism
  - Cheaty, Truish, and True parallelism
- Using C code
  - inline package, an actual package, CUDA
- Packaging It Up
My Research
**Overview**

- Experimental Design
  - Statistics, but really an optimization problem
  - Simulation to find long run behavior, so really the same optimization problem like a billion times.
  - Embarrassingly parallel, but often not amenable to GPUs

- Combined I and G optimal designs
  - CUDA and Python and C
  - G criterion amenable to GPU

- Experimental Design for Generalized Linear Models
  - R and CUDA and C
How R is Different Than <Language>
How R is Different

- R is very similar to Python and Javascript (ECMA-262)
- It has several different object systems with varying amount of structure
- If the reference count is larger than 1 you get pass-by-value behavior, with no easy way around it
- You can peek at the calling code from within the callee
- Every damn thing is a function
- Formulas and Data Frames
**CALLS AND ENVIRONMENTS**

- You can peek at the calling code from within the callee

```r
> f <- function(a) print(str(sys.call()))
> g <- function(a) print(str(match.call()))
> f(1)
  language f(1)
  NULL
> g(1)
  language g(a = 1)
  NULL
> f(a=1)
  language f(a = 1)
  NULL
> g(a=1)
  language g(a = 1)
  NULL
```

- And the environment, n is the number of generations to go back

```r
parent.frame(n = 1)
```
Calls and Environments

Environments:

http://adv-r.had.co.nz/Environments.html
Calls and Environments

There are two different types of assignment*

```
> f <- 6
In the current environment, f is set to 6

> f <<- 6
R searches up the environment path until it finds f, and then sets it to 6. If not found, it settles at global scope.
```

Actually there are more, but they're also more bizarre.

See http://adv-r.had.co.nz/Environments.html about delayed and active bindings.
Double Factorial

Look-up is faster than recalculation

```
.double.factorial.cache <- new.env(parent=emptyenv())

double.factorial <- Vectorize(function(k){
    # Default case
    if(k < 1){
        return(1);
    }
    # Check the cache
    n.name <- sprintf("%d",k)
    if(exists(n.name,envir=.double.factorial.cache)){
        val <- get(n.name,envir=.double.factorial.cache)
    } else {
        val <- k*double.factorial(k-2)
        assign(n.name,val,envir=.double.factorial.cache)
    }
    return(val)
})
```

Alternatively, we could do this with `<<-` and lists. But that makes and modifies a copy...
Helper Function

Utility function I use to map lists to local variables

```r
var.export <- function(v.list){
    envir <- parent.env(environment())
    for(v.label in labels(v.list)){
        assign(v.label, v.list[[v.label]], envir=envir)
    }
}
```
Non-Standard Evaluation

We can dig into the actual arguments used

```r
> f<-function(x){
+   print(str(substitute(x))
+ }
> f(1:10)
language 1:10
```

We can convert that into strings

```r
> g<-function(x){
+   print(str(deparse(substitute(x)))))
+ }
> g(1:10)
chr "1:10"
```
Everything is a Function

Every damn thing is a function

```r
> X <- matrix(rnorm(4),ncol=2)
> Y <- matrix(rnorm(4),ncol=2)
> X %*% Y
   [,1]       [,2]
[1,] 2.60082343 -2.230194
[2,] 0.09116735 -0.195290
> `%*%`(X,Y)
   [,1]       [,2]
[1,] 2.60082343 -2.230194
[2,] 0.09116735 -0.195290
```

For real, everything

```r
> `colnames<-(X,c("C","D"))
   C  D
[1,] -0.5604756 1.55870831
[2,] -0.2301775 0.07050839
```
Backtick for Bad Names

The backtick character can let us address elements of the environment that have illegal names

```r
> `1` <- 2
> 1
[1] 1
> `1`
[1] 2
> ls()
[1] "1"
```
Formulas and Data Frames

- Formulas are first class objects
  - \~A*B
  - \~1+A*B
  - \~A+B+A:B
  - \~1+A+B+A:B

- They're evaluated with the names of the data frame as local variables
  - I(something)
  - A|B
Error Handling
**Error Handling**

- Why handle errors? Why not do what Go does?

```r
x <- -4
x <- slow.factorial(x)
print(x)
```

- Arguably bad practice, but R is meant for convenient use in exploration. Programming bad practices should be expected!

```r
slow.factorial <- function(x){
  if(x < 0){
    print("Error")
    return(NA)
  } else if(x==0){
    return(1)
  } else {
    return(slow.factorial(x-1)*x)
  }
}
```
Error Handling

Better to raise an error

```r
slow.factorial <- function(x){
  if(x < 0){stop("x must be positive")}
  if(x==0){
    return(1)
  } else {
    return(slow.factorial(x-1)*x)
  }
}
```

Execution results in

```r
> x <- slow.factorial(x)
Error in slow.factorial(x) : x must be positive
> print(x)
[1] -4
```
Error Handling

- Using `stop` halts execution and loads R with the error information for debugging.

  ```r
  > traceback()
  2: stop("x must be positive") at #2
  1: slow.factorial(x)
  ```

- It also allows us to handle the error in an appropriate way using the error control functions.

- RStudio has some new slick debugging features. Self Study Prompt: Learn them and master the R language!
We don't have nice exceptions like Java, but R fakes it...

```r
> tryCatch(
    expression1,
    warning=expression2,
    error=expression3,
    finally=expression4
);
```

We can return values from `tryCatch`

If we just want to suppress errors and continue

```r
> try(expression, silent=FALSE)
```
Using C Code
Using C Code

- C code lets you deal with R objects as they are internally
  - You can do things in much less memory and much faster
  - You can use existing code in Fortran or C rather than rewriting it to be slower and in R
  - Debugging can be hard, especially on Windows

Note: Hadley Wickham suggests using the RCpp package to write C++ code and connect it to R. This is also the topic of Randy Eubank's book.

- I write C directly because I abhor package dependencies, not necessarily the “best” way.
Using C Code

- Two conventions for calling C code
  - The .C convention is the nice way that passes traditional C types
  - The .Call convention gives you SEXP's and access to internals.
  - If you can use the former, you'll save yourself some torment.
Inline Package
**Inline Package**

- In Windows you need Rtools, and add the Rtools bin and gcc directory to your system path.

- Write C code in a string in R and let Rtools compile it

```r
x <- as.numeric(1:10)
n <- as.integer(10)

## A simple Fortran example
code <- "
int i;
for(i=1; i<*n; i++){
    x[n] += x[n-1];
}
"
weirdfn <- cfunction(signature(n="integer", x="numeric"),
code, convention=".C")
weirdfn(n, x)$x
```
The inline package generated this code and compiled it

Program source:
1: #include <R.h>
2: 
3: 
4: extern "C" {
5:   void file20081ffe57c6 ( int * n, double * x );
6: } 
7: 
8: void file20081ffe57c6 ( int * n, double * x ) {
9: 
10:   int i;
11:   for(i=1; i<*n; i++){
12:     x[i] = x[i] + x[i-1];
13:   } 
14: 
15: }
CALLING SHARED LIBRARIES
With RTools (or just any Linux or Mac install) you can use R to compile binary compatible shared libraries from your C code, and you can load these and access them through R

R CMD SHLIB funcs.c
dyn.load("funcs.dll")
.C("f", n=as.integer(5), x=as.double(rnorm(10)))
R Packages

with

C Code
R Packages and C Code

- RStudio has a very nice manager for building packages.
  - Handles compilation to package for source or binary distribution.
  - I use this to make a binary package for all of my Windows machines and then manually install and launch worker threads on each workstation.
  - Starting a new project and choosing extension makes a skeleton package that you just have to update and drop in your code...
R Packages and C Code

- The directories have different purposes
  - inst – Things that are just installed into the package folder (unit tests, binary resources)
  - R – R code for your package
  - src – compiled source code for your package, and instructions on how to compile it.
  - DESCRIPTION – a file containing a description of your package.

- The remaining directories and files I use for other purposes.
R Packages and C code

- There are additional directories for stored data and other resources.
- Data is stored in a format that makes it fast to import as an R object – important point for authors out there.
Parallelism
Parallelism

Cheap parallelism

Package parallel for Linux and Mac

> library(parallel)
> cores <- detectCores()
> mclapply(1:10, func, mc.cores=cores)

There's a way to use it under Windows using sockets... Everything is harder on Windows.

Run many copies and use a shared data source (I use MySQL)

Works on all systems

Other packages:

Snow, Rmpi, and many more...
Parallelism with CUDA

- NVIDIA CUDA
  - Allow you to do computation with double precision on the GPU
  - Good analogy is a loom. If you want to do the same thing many many times to different data, GPU will far outpace CPU.
  - If you have code that has many conditionals it can be slower (sometimes much slower) to run on the GPU
  - At present, the GPU doesn't have the kind of optimization that multi-core CPUs do
  - It's easier on Mac or Linux. On Windows you have to compile to a DLL and then link into it. You also have to:
CUDA

with

Windows
CUDA with Windows

- You pretty much have to use Visual Studio and the NVIDIA CUDA Toolkit
- Generate Relocatable Device Code allows you to do linking
- Remember to use 64bit if you use 64bit R, 32 bit otherwise R.Version()
CUDA with Windows

- You pretty much have to use Visual Studio and the NVIDIA CUDA Toolkit.
- Set the code generation arguments to match your requirements.
  - Default is compute_10, sm_10, but this doesn't allow things like malloc within a __device__...
  - I used compute_20, sm_21
CUDA with Windows

- Add registry settings to keep Windows from killing the display driver if it holds for more than (2 seconds?)

  HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\GraphicsDrivers

- TdrLevel indicates how the GPU is monitored.
  - 0 indicates no monitoring
  - 3 indicates timer

- TdrDelay is the number of seconds to wait under 3

ACTION ITEMS
**Action Items**

- **Buy these books:**
    - Much of this book is available free online (http://adv-r.had.co.nz/). Covers a lot and has its own great references.
    - Has a lot of good introductory and advanced material.
    - Discusses package development better than the others. Also covers much of the language.
    - Very concise description of the R and S-plus languages and how things work.
**Action Items**

- Buy these books:
    - Focuses on C++ in R for statistical computing (monte carlo, bootstrap, EM algorithm, and much more)
    - Dr. Eubank is a cool dude.
- Use stackoverflow.com for R questions and answers
- Download R, Rstudio, and Rtools.
- Decide if the pain of Windows is less than the pain of dual-booting. Record this decision in your last will and testament.