Advanced R: Random Cool Stuff with R

Keegan Hines

R for speed

Rcpp

multicore

R for fun

knitr

shiny

slidify: HTML slide decks

Advanced R: Random Cool Stuff with R

Keegan Hines

Outline

1. R for speed
   - Rcpp: integration with C++
   - multicore: parallelization
2. R for fun
   - knitr: dynamic documents
   - shiny: interactive web apps
   - slidify: HTML slide decks

Rcpp

```
library(Rcpp)
cppFunction('
int add(int x, int y, int z){
int sum= x+y+z;
return sum;
}
')
```

```
add(1, 3, 9)
```

```
## [1] 13
```

Rcpp

```
randWalk<-function(N){
walk<-c(0)
for (i in 2:N){
walk[i]<-walk[i-1] + runif(1)
}
return(walk)
}
```

```
randWalkVec<-function(N){
walk<-cumsum(runif(N))
}
```

```
cppFunction(
NumericVector randWalkCpp(int N){
NumericVector walk(N);
walk[0]=0;
for (int i = 2; i<N ; i++){
walk[i]=walk[i-1] + rand();
}
return walk;
}
')
```

Rcpp

```
library(microbenchmark)
microbenchmark(randWalk(1000), randWalkVec(1000), randWalkCpp(1000))
```

```
## Unit: microseconds
##                expr     min      lq  median      uq     max neval
##     randWalk(1000) 7406.98 7545.35 7733.66 8516.17 37796.4   100
##  randWalkVec(1000)   40.95   42.34   44.59   45.54    63.1   100
##  randWalkCpp(1000)   10.37   10.83   14.36   15.58   850.3   100
```

Rcpp

Using `cppFunction` in your code might be a little cumbersome, standalone C++ is preferable.

```
#include Rcpp.h
namespace Rcpp;
// [[ Rcpp::export]]
int myfunction(){...}
```

```
sourceCpp("path/to/myfunction.cpp")
myfunction(myVariables)
```

multicore

For use with embarassingly parallel problems which can be distributed across the multiple cores you might have access to.

```
## Relies on the apply functions
squares <- c()
for (i in 1:5) {
    squares[i] <- i^2
}
squares
```

```
## [1]  1  4  9 16 25
```

```
squares <- lapply(1:5, function(x) x^2)
squares
```

```
## [[1]]
## [1] 1
##
## [[2]]
## [1] 4
##
## [[3]]
## [1] 9
##
## [[4]]
## [1] 16
##
## [[5]]
## [1] 25
```

The apply functions are not actually faster than for-loops, they just make for more compact code.

multicore

If you have access to multiple cores, there is a multiple-core-version of lapply.

```
library(multicore)
multicore:::detectCores()


## [1] 4
```

multicore

```
fast_computation <- function(x) {
    return(x^2)
}
slow_computation <- function(x) {
    walk <- c(0)
    for (i in 2:x) {
        walk[i] <- walk[i - 1] + rnorm(1)
    }
}
```

multicore

```
F1 <- function(lis) {
    lapply(lis, slow_computation)
}
parF1 <- function(lis) {
    mclapply(lis, slow_computation)
}
```

```
microbenchmark(F1(100:200), parF1(100:200), times = 20)
```

```
## Unit: milliseconds
##             expr   min    lq median    uq   max neval
##     F1(100:200) 223.5 231.5  244.9 253.1 301.9    20
##  parF1(100:200) 102.6 105.7  109.3 113.8 212.8    20
```

multicore

```
F2 <- function(lis) {
    lapply(lis, fast_computation)
}
parF2 <- function(lis) {
    mclapply(lis, fast_computation)
}
```

```
microbenchmark(F2(100:200), parF2(100:200), times = 20)
```

```
## Unit: microseconds
##            expr     min       lq  median    uq   max neval
##     F2(100:200)   345.9   441.1   708.6  1542  3680    20
##  parF2(100:200) 12498.7 13959.5 15340.4 16540 18089    20
```

Parallelization isn't always a good idea.

knitr

- generate dynamic documents
- document and manage workflows
- can use with markdown, LaTex, and more

```
# doc1.Rmd
Title
=======
This is an R markdown document.
Essentially identical to the markdown you *already* know.
We can represent chunks of code.
```
x<-rnorm(100)
hist(x)
```
We can embed images.
![](./image.png)
```

```
library(knitr)
knit2html("doc1.Rmd")
```

knitr

But notice that none of that code was actually evaluated. And the image we embedded was just a static image that we previously created.

```
# doc2.Rmd
Using knitr to make dynamic documents
===
  Now we embed code chunks in a special syntax.
This sends the code off to an R process, and the result is shown in the document.
 ```{r}
x<-rnorm(1000)
hist(x)
```
```

```
knit2html("doc2.Rmd")
```

Whatever code we want to run gets wraped in: “‘{r} blah blah “‘

knitr

- Now we can use simple markdown documents to keep track of all the analysis we do. Great for documentation and explanation.
- The analysis and visualization of data is repeated on the fly, every time we compile the document, and the document is easily shared – reproducible research.
- The data, the analysis, the visualization, and the text all stay in one place. Makes it a lot easier to write papers (really, trust me) .

knitr

- Generating html documents is useful for keeping notes or documenting your analysis, but it doesn't really look *awesome*. What if you want a more professional look?
- Can also use knitr with LaTeX, to get beautiful layout of text and equations, and also dynamic generation of figures and tables.

knitr

```
#texDoc1.tex
\documentclass{article}
\usepackage{graphicx}
\title{Nice Lookin Document}
\author{Keegan}
\begin{document}
\maketitle
Here is some text. Let us not forget to use some equations.
\[
p(\theta|y_N) \propto \prod{i=1}^N \theta^k (1-\theta)^{n-k}
\]
And with TeX, we include images in this way...
\includegraphics[width=10cm]{image.png}
\end{document}
```

knitr

Again, the limitation of that approach is that we embed a previously created static image to use as a figure. With knitr, we can run that analysis and create figure as the TeX document is compiled.

```
#texDoc2.Rnw
\documentclass{article}
\title{Nice Lookin Document}
\author{Keegan}
\begin{document}
\maketitle
Here is some text. Let us not forget to use some equations.
\[
p(\theta|y_N) \propto \prod{i=1}^N \theta^k (1-\theta)^{n-k}
\]
With knitr, we send some code off to an R process in order to generate the figure.
'<<FigureName,fig.width=4>>=
x<-rnorm(100)
hist(x)
@  '
```

Whatever R code we want to embed gets wrapped in <<>>= blah blah @ and compile the document using `knit()`.

shiny

- shiny allows us to build interactive and immersive web-based applications
- Importantly, we do so using *only* R, and need no knowledge of html, css, or javascript.
- shiny apps are very easy to create and are incredibly useful for scientific communication, explanation, and data sharing.

shiny

We just need to write two scripts:

- ui.R : Defines the layout and the interactive elements that the user can access.
- server.R : Defines what computations are done in response to user interactions.

shiny

```
#ui.R
library(shiny)
shinyUI(pageWithSideBar( #what the page looks like
headerPanel('Shiny Apps'), #name the app
#make a sidebar layout
sidebarPanel(
#let's have interactive sliders
sliderInput('obs', 'Number of observations:',
min=1,max=1000,value=500)
),
#in the main panel, plot a variable called distPlot
mainPanel( plotOutput('distPlot')  )
)
)
```

shiny

```
library(shiny)
shinyServer(function(input, output) {
    output$distPlot <- renderPlot({
        # generate random variables and plot
        dist <- rnorm(input$obs)
        hist(dist)
    })
})
```

Those two scripts are all we need.

```
library(shiny)
runApp()
```

shiny

So now we're running web apps on our local machine. Pretty useful, but to share with others, we have to send around R files and the user needs to have R and know a little bit about it.

We can remotely host shiny apps and then just send people links. Go get a free account at `shinyapps.io/signup.html`

shiny

Interactive Apps

- Scientific communication, explaining complex concepts
- Sharing data and results with colleagues

Remote Hosting

- People don't need R to interact with your scientific story, just a web browser
- Interactive conference posters, talks

slidify

Generate HTML slide decks using only Rmarkdown.

```
library(slidify)
author("MySlides")
```

(Package isn't in CRAN yet, install from github)
This creates an Rmarkdown template for your slides, just fill in what you want in each slide.

```
slidify("index.Rmd")
```

slidify

Cool trick - Any github repo with a branch called gh-pages will get served as a website. If the content of that repo is the stuff of websites (html,css), then you get free web hosting. So, create a branch called gh-pages and push to it.

```
git branch gh-pages
git checkout gh-pages
git add .
git commit -m 'MY WEBPAGE!!!'
git push origin gh-pages
```

Read up on this– `pages.github.com`

slidify

HTML slides - Why?

- Remote hosting, cross platform - just need a web browser
- Modern web browsers have gotten really good at stuff that we would like to have in presentations - embedding rich media, interactivity
- Interactive Presentations! Embed a shiny app by adding this line to the html

```
<iframe src='remoteWebsite.com'> </iframe>
```