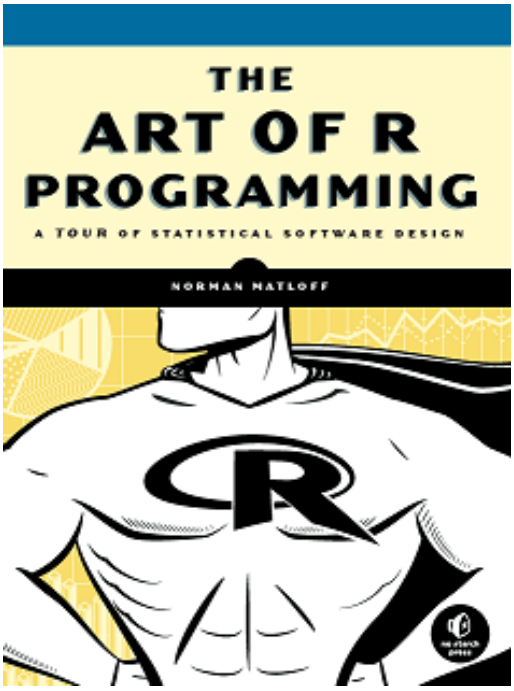
The Art of R Programming - Review

Radha Krishna

Dec 28 , 2011

Abstract

The purpose of this document is to document my learnings from the book,'The Art of R Programming'



Contents

1 Introduction

It has been one month Nov 26 2011 since I had decided that I will never use MS Word in life except to transfer content from Lyx document so that I can post it on some blog. Infact it would be better if I can post the content in a pdf format itself. As I reflect today about my learnings in the last one month, one thing I can say is that I am extremely happy with the decision of not using word at all. I have read 6 books since then and I have used LaTeXin writing stuff. I am much more confident now that I will be able to mix R code and LaTeXand create memorable documents.

This book is written by Normal Matloff who has worked in both Computer science department and Statistics department at UCLA. Hence this book is markedly different from the books that are available on R. You get a nice blend of point of views. Also the author clearly states in his preface that this book is essentially a book for those who want to DEVELOP SOFTWARE in R. If you want to write some adhoc code for doing some adhoc analysis, this book is definitely a stretch. However if you consider using R for doing your day to day work as well as doing research, then this book is an awesome reference. Let me summarize my learnings from this book.

2 Getting Started

My learnings from this chapter are

- R is polymorphic. A single function can take in variety of input classes.
- Always make the code efficient and elegant

```
> oddcount <- function(x) {
      k <- 0 # assign 0 to k
      for (n in 1:length(x)) {
      if (x[n] %% 2 == 1) k <- k+1 # %% is the modulo operator
      }
      return(k)
      }
> oddcount <- function(x) {
      k <- 0 # assign 0 to k
      for (n in x) {
      if (n %% 2 == 1) k <- k+1 # %% is the modulo operator
      }
      return(k)
      }
```

  Obviously the last function is elegant .
- Instances of S3 classes are lists with an attribute `class`
- If class is a just a list with additional attribute, why do we need them ? Well, because genericFunctions in R take classes as input and invoke functions specific to that class.
- To do a google search in all your packages installed use `help.search("multivariate")`
- Use `sos` package to get help. Obviously missing from the list of help resources mentioned in this book is www.stackoverflow.com

3 Vectors

What's covered in this chapter ?

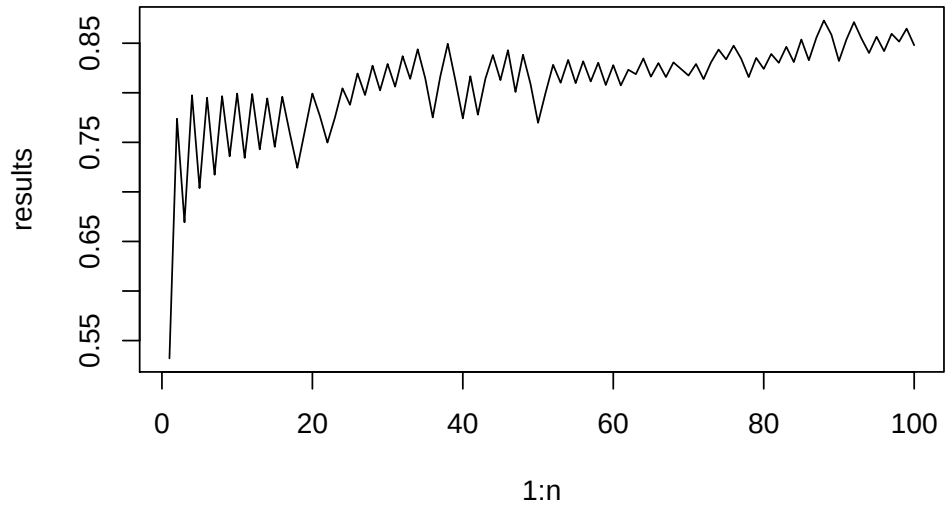
Trial Find consecutive runs

```
> set.seed(1977)
> x                                <- rbinom(100, 1, 0.5 )
> runs                          <- which(x[2:100]==1 & x[1:99]==1)
> print(runs)
 [1]  2 12 15 18 19 20 23 24 27 28 41 74 81 94 95
> x                                <- c(1,0,0,1,1,1,0,1,1)
> runs                          <- which(x[2:9]==1 & x[1:8]==1)
> print(runs)
[1] 4 5 8
```

I completely misread this example and ended up writing a completely different function.

Came across an interesting problem of weather prediction. Let me code this up. This is exactly what is called deliberate practice. I need to find a way to predict the weather based on training data

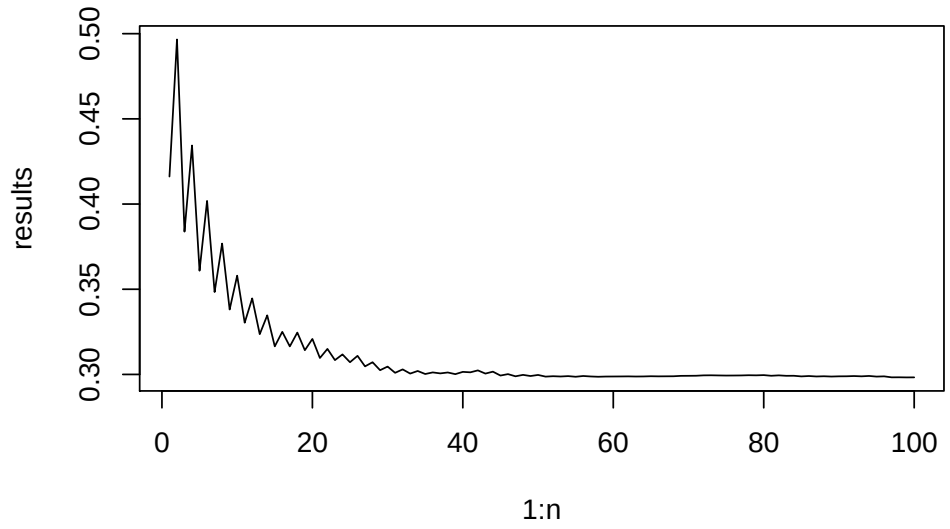
```
> set.seed(1977)
> x                                <- rbinom(1000, 1, 0.5 )
> k                                <- 3
> getErrorRate        <- function(x,k) {
         y                <- c( filter(x, rep(1/k,k), method="convolution",sides=1) )
         z                        <- cbind(x,y)
         z                        <- z[!is.na(z[,2]),]
         er.rate                <- 1- sum(z[,2]>0.5 &z[,1]==1)/dim(z)[1]
         return(er.rate)
 }
> n                                <- 100
> results                        <- numeric(n)
> for(k in 1:n){
      results[k]        <- getErrorRate(x,k)
 }
```



The book has a nice function that is written in an optimal way

```
> predc                          <- function(x,k) {
         n                 <- length(x)
         k2                 <- k/2
         pred         <- vector(length=n-k)
         csx                 <- c(0, cumsum(x))
         for(i in 1:(n-k)) {

           if(csx[i+k] - csx[i] >= k2) pred[i] <- 1 else pred[i] <- 0
         }
         return( mean(abs(pred-x[(k+1):n] ) ) )
 }
> x                                <- rbinom(10000, 1, 0.3 )
> k                                <- 3
> n                                <- 100
> results                         <- numeric(n)
> for(k in 1:n){
         results[k]         <- predc(x,k)
         }
```



In the above figure , the realization is a sample path where the odds of getting heads is 0.3. After a sample size of 30 , the error rate stabilizes. Ok, enough. I don't think there is a point in spending more time here.

My learnings from this chapter are

- There is no scalar in R
- When we assign something to x, x is basically a pointer to a data structure
- Try to use all() any() functions whenever possible
- sapply gives out a matrix when the entry is a vector
- NULL is a special R object with no mode
- It is better to assign a variable NULL and then use it in a loop to grow the variable /vector
- sign function is a damn useful function. Keep this in working memory always
- `:` produces integers whereas `c()` produces double
- `names(x) <- NULL` removes the name of the elements in a vector
- sapply converts an input vector to an output matrix.
- In R, we can ask the function to skip over NA values.
- Understand the imporatance of NULL. If you assingn a variable as NULL, then you can dynamically grow the vector in a loop. NULL is a special object with no mode. This can be a good teaser question in an interview
- `:` produces integers whereas `c()` produces double

```
>         x        <- 1:2
>     y        <- c(1,2)
>     identical(x,y)
[1] FALSE
```

  The above result is false precisely because of type mismatch.
- I never knew that you could remove remove names of the vector

```
>         x        <- 1:3
```

```
>     names(x) <- letters[1:3]
>     x
a b c
1 2 3
>     names(x) <- NULL
>     x
[1] 1 2 3
```

- Smart use of diff function is better I coded it in a much simpler way than the book

```
> x        <- c(5,12,13,3,6,0,1,15,16,8,88)
> y        <- c(4,2,3,23,6,10,11,12,6,3,2)
> mean(as.numeric(diff(x)>0) == as.numeric(diff(y)>0))
[1] 0.4
```

  Oops , I was mistaken . The author did a much better job of coding with diff

```
> x        <- c(5,12,13,3,6,0,1,15,16,8,88)
> y        <- c(4,2,3,23,6,10,11,12,6,3,2)
> mean(sign(diff(x)>0) == sign(diff(y)>0))
[1] 0.4
```

4 Matrices and Arrays

What's covered in this chapter ?

```
> x        <- matrix(data = rnorm(4) , nrow = 2)
> y        <- matrix(data = rnorm(4) , nrow = 2)
> z        <- array( data = c(x,y), dim = c(2,2,2))
> z
, , 1

           [,1]      [,2]
[1,] -0.1111392 1.1618350
[2,] -0.1146646 0.1620229

, , 2

         [,1]       [,2]
[1,] 1.311291  1.7578950
[2,] 1.555177 -0.8925723
```

My learnings from this chapter are

- R uses column-major order
- Learnt about `pixmap` package that gives the grayscale of the image in a matrix format.
- apply will generally not speed up the code. It makes for a compact code, that is easier to read and modify
- when you subset a matrix, you get a vector. The original properties of matrix go missing. Hence use `drop = FALSE` , then the matrix nature is preserved in subsetting.

5 Lists

What's covered in this chapter ?

I would like to do some deliberate practice on one of the blog posts

```
> results               <- list()
> for(i in 1:length(bp)) {
        wrd        <- bp[i]
        results[[wrd]] <- c(results[[wrd]],i)
 }
> freq        <- sapply(results, length)
> results        <- results[order(freq, decreasing=T)]
> x1                <- names(results[1:30])
> x2                <- as.vector(sapply(results[1:30],length))
> x3                <- data.frame(words=x1,count = x2)
> print(x3)
         words count
1          the   280
2           of   163
3           to   150
4           is   141
5            a   112
6          and   102
7         that    77
8           in    71
9          The    54
10           I    45
11         are    43
12          it    37
13        this    35
14         for    35
15          as    31
16     chapter    31
17         can    30
18         one    28
19          be    28
20        from    26
21 probability    26
22         you    26
23      random    25
24         not    24
25        then    23
26       about    22
27        with    22
28          on    22
```

```
29    function    19
30        have    17
```

My learnings from this chapter are

- component names in a list are called tags
- names of the list can be abbreviated to whatever extent is possible with out causing ambiguity
- list with single bracket means you are working on another list. list with double brackets means you are working with the elements of the list
- you can remove a component from the list by setting it to NULL
- use length function to get the size of the list
- R chooses least common denominator for unlisting operations
- There is an unname() function that can be used to remove names from a vector. I had never used this function till date.
- lapply() gives you back a list whereas sapply() gives a vector or a matrix
- There is a nice example of text parsing where lists are used to generate the word count,freq count etc somehow makes me feel that I should do something like this for one of my blog posts and see how it works May be I will parse the content of the book summary of 'How I became a quant' and do some basic deliberate practice on lists
- lists can be recursive too.

6 Data Frames

My learnings from this chapter are

- The way lists are heterogenous counterparts to vector, the same way data frames are heterogenous counterparts to matrix
- There are usually three ways to access a data frame `df$a , df[,1] and df[[1]]`
- use $drop = FALSE$ so that extracted elements have data.frame attribute
- Never had a chance to use `complete.cases()` function which basically removes all rows whichever has NA.
- Learnt how to create dataframes dynamically in a loop using the assign function
- count.fields is a function that counts the number of attributes in a row. Useful at data preparation stage
- Remember lapply on a dataframe will sort each of the individual columns . That is disastrous as all the data gets mixed up

7 Factors and Tables

```
>         g        <- c("A","B","A","C")
>     split(1:4,g)
$A
[1] 1 3

$B
```

```
[1] 2

$C
[1] 4
```

The above function can be used for text parsing.

My learnings from this chapter are

- Factors can be thought of vectors with a bit more information added like levels.
- output of split is a list.
- If you want to run regression analysis for every factor in a data frame, then use by to sudivide the big data frame in to smaller data frames.
- use table argument to get cross tabs.
- aggregate() calls tapply() once for each variable in a group.
- cut() function to generate factors from data.

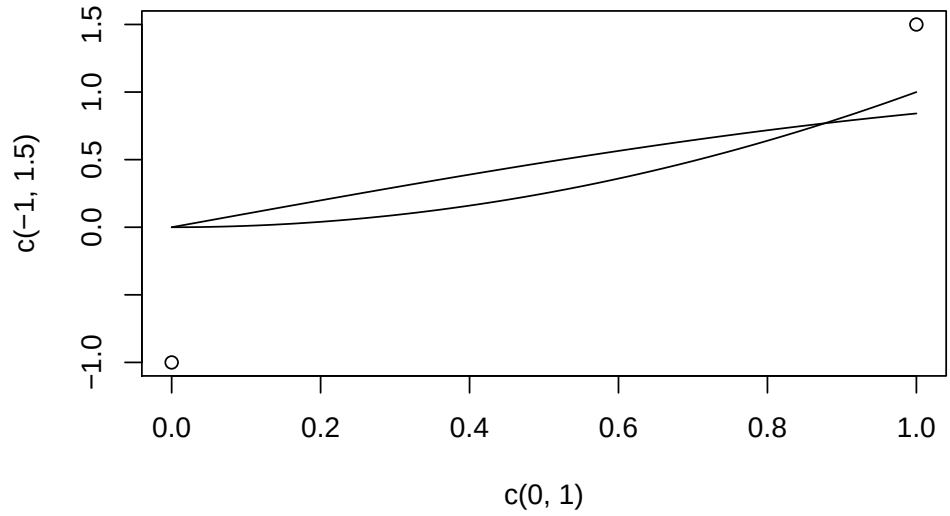
8 R Programming Structures

My learnings from this chapter are

- Avoid nested ifs with `next`
- Use `get` to have a control on the pointer

```
>         x        <- matrix(data=rnorm(4),nrow=2)
>     z        <- get("x")
>     z
           [,1]       [,2]
[1,] -2.006889  0.2482093
[2,] -1.771152 -1.4853836
```

- Using `&&` in `x && y` always checks the first element of both variables
- functions are objects. Hence you can assign them, you can pass them as arguments, loop through them



- Any function defined in top environment has `R_GlobalEnv` as its environment value
- R follows lazy evaluation
- One can use ls() in different ways to print objects from various environments
- References to local variable actually go to the same memory location as the global one, until the value of the local changes. In that case, a new memory location is used.
- Before reading this book I never knew `get` was the most useful function in R.
- You can access the environment using the function parent.frame()
- Use `<<-` to assign values to global variables
- Use `assign` to assign values to non locals
- R uses a ton of global variable assignments in its internal code
- R closure consists of a function's arguments and body together with its environment at the time of the call

```
>     counter        <- function() {
         ctr        <- 0
         f        <- function() {
           ctr        <<- ctr + 1
           cat("value is ",ctr,"\n")

         }
      }
>     c1        <- counter()
>     c2        <- counter()
>     c1
function() {
          ctr        <<- ctr + 1
          cat("value is ",ctr,"\n")
```

```
        }
<environment: 0x000000000319fbe8>
>     c2
function() {
          ctr        <<- ctr + 1
          cat("value is ",ctr,"\n")

        }
<environment: 0x00000000031799c0>
>     c1();c1();c1();c1();
value is  1
value is  2
value is  3
value is  4
>     c2();c2();c2();c2();c2()
value is  1
value is  2
value is  3
value is  4
value is  5
>     c1()
value is  5
```

- Any assignment statement in which teh left side is not just an identifier is considered a replacement function.
- The biggest learning from this chapter is replacement function

```
>         x        <- 1:10
>     x[2:3]
[1] 2 3
>     "["(x,3:8)
[1] 3 4 5 6 7 8
>     test        <- function(x) x^2
>     "test"(4)
[1] 16
```

- There is another way to invoke the function `"function.name"(arguments)`
- `"["(x,y) == x[y]`
- `"[<-(x,2,3)" == x[2] <- 3`
- This section helped me understand the replacement functions that I have been using since the time I have started programming R. I mean I never ever paused to understand whats happening under the hood.
- I have also learnt the concept of anonymous functions. Whenever you write function(x) in sapply, apply etc what one sees is an example of anonymous function.

9 Doing Math and Simulations in R

My learnings from this chapter are

- sweep function is used to add a specific vector to all rows or columns
- setdiff(x,y) gives all elements in x that are not in y

10 Object-Oriented Programming

My learnings from this chapter are

- R promotes encapsulation, allows inheritance , classes are polymorphic
- R has two types of classes S3 and S4. I have a good pneumonic to remember how S3 works. S3 is like a general manager in a company. Basically he does nothing . He merely delegates work. so a print function in R is like a general manager which uses dispatch function to call the relevant function in the input object and make it to do the work. These are called genericFunctions that have nothing internally but a dispatch feature.
- Always remember that , by having an access to the object name, one might get an error by invoking objectname.print, objectname.summary as these objects might be different name spaces. So, you must prefix the object name by namespace and then invoke the generic function
- getAnywhere is a function to get all the namespaces and objects for which a specific function is defined
- S3 classes are written by specifiying a list and then assigning a class attribute to that list. That's about it. So , in one way this simplicity has a flip side. Lot of errror might creep in. This is one of the reasons for the existence of S4 classes which are more structure in nature
- If you want to specify inheritance, a simple vector of names can be assigned to class attribute.It is as simple as that.
- S4 classes are considerably richer than S3 classes
- S4 classes are defined using setClass function and functions are defined using setMethod
- John Chambers advocated S4 classes where google R style guide advocates S3 classes.
- If you are writing a general purpose code, then exists might be a very useful function

11 Input/Output

My learnings from this chapter are

- Try to use cat instead of print as the latter can print only one expression and its output is numbered, which may be a nuisance.
- scan function is useful to read stuff from a file
- There are lot of functions related to files like file.info, dir, file.exists that can be used in data preparation activities
- There are ton of functions for gleaning directories, files etc. Infact I revamped all my chaotic iTunes collection using file and directory functions in R
- came to know about `snow` package and `Rdsm` package that can be used for parallel computation in R

12 String Manipulation

My learnings from this chapter are

- grep(), grepl(), sub(), gsub(),nchar(), paste() , sprintf(), substr(), strsplit(), regexpr(), grepexpr() are basic functions that are needed for text analysis

13 Graphics

My learnings from this chapter are Nothing new learnt here.

14 Debugging

The highlight of the book is this chapter where the author deals with debugging for one full chapter **My learnings from this chapter are**

- Starting from R 2.10, there is a new function , debugonce() that can be used.Very useful as you don't need to type debug and undebug always
- You can put conditional breakpoints also `browser(s>1)`
- Starting from R 2.10,setBreakpoint(filename,linenumber)
- `trace(gy,browser)` is another way of placing a browser() at the start of the function with out modifying the function
- `option(errors=recover)` is very useful to trace back the error for the first time
- Need to check out debug package by Mark Bravington
- Also StatET new version has debugging facility. All my R code is written on an older form of StatET version. Lookign at the number of people hitting the support list of StatET with questions, I have an inertia to move to the new StatET version that promises debugging
- If you are writing very important code /algo, it is better to keep `options(warn=2)` so that it instructs R to convert them to errors and makes the location of warnings easier to find

15 Chapter14-Chapter16

For an experience R user, the last three chapters of the book cover very important topics like

- How to enhance performance of the code ?
- How to make code efficient ?
- How to make R talk to C++, Python and other languages ?
- How can one perform parallel computations in R ?

16 Takeaway

This book is a valuable addition to the R literature and has something new to offer to any R programmer, be it a beginner or a seasoned programmer.