Computational Statistics in R Lab

Computer Science PG Study Material
Paper: CS-292 Module - II

Part-III: R Programs

Anupam Pattanayak[1]
Assistant Professor,
Department of Computer Science,
Raja N. L. Khan Women's College (Autonomous),
Midnapore, West Bengal

May 5, 2020

[1]anupam.pk@gmail.com

Contents

1

R Programming Basics

In the previous lecture notes, we have seen usage of some basic commands and doing input-output in R, and then commands involving vectors and matrices, and then statistical commands. We will now see how to do programming in R.

As earlier, we will refer the book by Cotton[1], and the book by Matloff[2].

1.1 Creating & Executing Our First R Program

To create a R program, we first open a text editor to do coding of the program and use the extension *.R* to save the program. Now, we write our first program to display the string *Hello World!*. The one-line program is shown below.

```
# Program to display a string
print("Hello World!",quote=FALSE)
```

The first line is *comment line*, indicated by # at the begining. This program uses the command print( ) with two arguments. First argument is the string within double quote or single quote that we want to display. Second argument says whether we want to display the quotation around the string or not. Note that, if we do not provide the firt argument within quote, the program will generate error. Although we can work with both single quote or double quote, it is good practice to double quotes for strings.

[1]Learning R - A Step by Step Functional Guide by Richard Cotton, Orielly
[2]The Art of R Programming- A Tour by Norman Matloff, No Starch Press

Now, we save the program as *HelloWorld.R* and next we execute the program by invoking the command *Rscript HelloWorld.R* from the terminal. Note that, this command is issued from user command prompt, *not from R command prompt*. The execution is shown below.

```
$ Rscript helloworld.R
[1] Hello World!
```

Here onwards, we use the term *program* to mean R program.

1.2 Program to Display a Variable

Suppose, we have a variable *n* which we want to initialize with value *50*. To initialize and display it's value, we write the program as shown below.

```
# Program to display value of an object
n<-50
print(n)
```

We save the program as *DisplayNumber.R* and next we execute the program by invoking the command *Rscript DisplayNumber.R* from the terminal. The execution is shown below.

```
$ Rscript DisplayNumber.R
[1] 50
```

1.3 Program to Display a Variable with Informative Message

In the previous program, we have displayed the value of the variable *n*. It would be of help for end users to get output like *n=50*. This we can achieve using cat( ) function which concatenates two strings. To see details about this function use cat( ) in the R command prompt. The program for this is as shown below.

```
# Prog. to Display a Variable along with Informative Message
n <- 50

    ## print an informative message
```

```
cat("n = ", n, "\n")
```

We save the program as *cat1.R* and next we execute the program by invoking the command *Rscript cat1.R* from the terminal. The execution is shown below.

```
$ Rscript cat1.R
n =  50
```

1.4 Program to Read a Number and Display it

To read a number from keyboard, we can use the function scan( ) function. Usage of this function is illustrated in the following program which reads a number from keyboard and display it.

```
# Prog. to Read a number and Display it
print("Enter a number: ",quote=FALSE)
num<-scan("stdin",integer(),n=1)
cat("You have entered:",num,sep="\n")
```

Note the usage of scan( ) function. The first argument *stdin* specifies that input is taken from keyboard which is standard input device. Second argument says type of input we want, and third one says how many inputs we want to read. If we want double number, then we use double( ) as the second argument . In case, we want a character string as input, then use integer( ) as second argument. It will enable us to read a string upto a space.

We save this program as *Read1.R* and next we execute the program by invoking the command *Rscript Read1.R* from the terminal. The execution is shown below.

```
$ Rscript Read1.R
[1] Enter a number:
150
Read 1 item
You have entered:
150
```

1.5 Program to Read a Vector and Display it

To read a vector from from keyboard, we can use the function scan( ) function as we have seen earlier. We have to set value of n to the length of vector. This is illustrated in the program below.

```
# Read a Vector and display it
print("Enter an integer vector os size 5: ",quote=FALSE)
vec<-scan("stdin",integer(),n=5)
cat("You have entered: ",vec,sep='\n')
```

We save the program as *ReadVector.R* and execute the program by the command *Rscript ReadVector.R* from the terminal. The execution is shown below.

```
$ Rscript ReadVector.R
[1] Enter an integer vector os size 5:
50 10 47 63 20
Read 5 items
You have entered:
50
10
47
63
20
```

1.6 Program to Initialize a Matrix and Display it

To initialize a 2×3 matrix row-wise with given values and to display the matrix, we write the following program.

```
# Example of 2x3 matrix initialization row-wise

mat <- matrix(c(1,2,3, 11,12,13), nrow = 2, ncol = 3, byrow =
    TRUE)
print(mat)
```

We save the program as *Matrix1.R* and execute the program by the command *Rscript Matrix1.R* from the terminal. The execution is shown below.

```
$ Rscript Matrix1.R
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   12   13
```

1.7 Program for Even - Odd

Here, we will use the *if* construct. It's syntax is shown below:

```
if(condition) true_part else false_part
```

The program for this even odd checking is shown below. As the logic is very well known to you, we are not explaining it to you.

```
#even-odd program in R
readline("Enter a number: ")
n<-scan("stdin",integer(),n=1)
r<-n%%2
if(r==0) {print(n); print(" is Even ")} else {print(n); print("
    is Odd ")}
```

Note the usage of if( ) construct. Particularly, how we have embedded multiple statements within *true_part* and *else_part*. We have use multiple print( ) function multiple times, you can use the cat( ) function instead of multiple `if` in *true_part* and similarly in *else_part*. We save this program as *Even_Odd.R* and then we execute the program by invoking the command *Rscript EvenOdd.R* from the terminal. The execution is shown below.

```
$ Rscript Even_Odd2.R
Enter a number:
[1] ""
55
Read 1 item
[1] 55
[1] " is Odd "
```

1.8 Program for Looping

Here, we will use the *for* loop construct. It's syntax is shown below:

```
for(loop_control_variable in range) loop_body
```

To display value from 1 to 10 using for loop, we write the following one-line program and is shown below.

```
# use of for loop
for(i in 1:10) print(i)
```

Note the usage of for( ) loop construct. We save this program as *loop1.R* and then we execute the program by invoking the command *Rscript loop1.R* from the terminal. The execution is shown below.

```
$ Rscript loop1.R
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

1.9 User Defined Function

Here, we will see how to define our own user-defined functions that we can use in the program. It's syntax is shown below:

```
FunctionName <- function(){
   function_body
}
```

Let us define a simple function that displays a simple message *R Programming*. It's definition and call is shown in the following.

```
printMessage <- function(){
   print("R Programming ")
}

printMessage()
```

We save this program as *func1.R* and then run it as we have done earlier. See the output yourself.

1.10 User Defined Function with Arguments

Here, we will see how to define our own user-defined functions that takes argument. We pass a string to the user defined function and display that after appending some other text. It's definition and call is shown in the following.

```
printMessage <- function(str) {
   cat(str," morning ",sep=" ")
}

printMessage("Good")
```

We save this program as *func1.R* and then run it as we have done earlier. See that you obtain the following output.

```
$ Rscript func2.R
Good  morning
```

2

R Programs for Statistical Operations

In this chapter, we will see some programs illustrating use of some built-in statistical and matrix operations. We will also define some user-defined functions here and use those for our specific purposes.

2.1 Program to Find Mean of a Vector

First, suppose we want to write a program that computes mean of a vector of some specific length. For this, we use the built-in function *mean( )* in our program, and it is shown below.

```
# Prog to Compute Mean of a Vector using existing func.
print("Enter an integer vector os size 6: ",quote=FALSE)
vec<-scan("stdin",integer(),n=6)    # read the vector
cat("You have entered: ",vec,sep='\n')
mn<-mean(vec)
cat("Mean of the Vector=: ",mn,sep="  ")  # display the vector
```

Note the usage of for( ) loop construct. We save this program as *loop1.R* and then we execute the program by invoking the command *Rscript loop1.R* from the terminal. The execution is shown below.

```
$ Rscript VectorMean.R
[1] Enter an integer vector os size 6:
5 4 15 20 8 12
Read 6 items
You have entered:
5
4
```

```
15
20
8
12
Mean of the Vector=:    10.66667
```

2.2 Program to Check for Symmetric Matrix

First, we will read size of the input matrix and then read the matrix. Then we define a function that checks if this matrix is symmetric. For this we use a bulit-in function *isSymmetric( )* that returns if the object is symmetric. The program is shown below.

```
## Example of checking symmetricc matrix using user-define
    function

MatSymmetric <- function(mat) {
n<-isSymmetric(mat)
if(n==TRUE) print("symmetric matrix") else print("matrix is not
    symmetric")
}


print("Read Matrix Row Size: ",quote=FALSE)
row<-scan("stdin",integer(),n=1)
print("Read Matrix Column Size: ",quote=FALSE)
col<-scan("stdin",integer(),n=1)
m<-row*col
cat("Enter an integer matrix of size: ", m, sep="\n")
vec<-scan("stdin",integer(),n=m)

mat1 <- matrix(vec, nrow = row, ncol = col, byrow = TRUE)

print("Matrix ",quote=FALSE)
mat1

MatSymmetric(mat1)
```

Note the usage of for( ) loop construct. We save this program as *loop1.R* and then we execute the program by invoking the command *Rscript loop1.R* from the terminal. The execution is shown below.

```
$ Rscript MatrixSymmetric.R
```

```
[1] Read Matrix Row Size:
2
Read 1 item
[1] Read Matrix Column Size:
2
Read 1 item
Enter an integer matrix of size:  45
3
3
5
Read 4 items
[1] Matrix
     [,1] [,2]
[1,]    5    3
[2,]    3    5
[1] "symmetric matrix"
```

2.3 Program For Binomial Distribution

Binomial distribution is a discrete probability distribution which is very helpful to obtain result of of n independent trials in an experiment. Each trial has only two outcomes: *true* or *false*. If the probability of successful is p, then the probability of obtaining k successful outcomes out of n independent trials is given by $\binom{n}{k} \times p^k \times (1-p)^{n-k}$.

Now suppose, there are ten multiple choice questions (MCQ) in a test. Each question has four options for answer to choose, and just one of them is correct answer. We have to find the probability of obtaing exactly 6 correct answers if a student randomly answers all questions. So, here n=12, k=6, p=0.25. We write the following program for this that uses binomial distribution function.

```
## Binomial Distribution

ans<-dbinom(6, size=10, prob=0.25)
sprintf("Probability of answering 6 questions correctly =%f ",
    ans)
```

Note the usage of dbinom( ) function and use of sprintf( ) function. Use of sprintf( ) function is quite similar to printf( ) function used in C programming language. We save this program as *BinomDistribution.R* and then we execute

the program by invoking the command *Rscript BinomDistribution.R* from the terminal. The execution is shown below.

```
$ Rscript BinomDistribution.R
[1] "Probability of answering 6 questions correctly =0.016222 "
```

2.4 Program For Poisson Distribution

Poisson distribution is the probability distribution of an independent event occurring in a given interval. If λ is the mean occurrence per interval, then probability of having k occurrences within a given interval is given by $\frac{\lambda^k \times e^{-\lambda}}{k!}$.

Now suppose, If 80 heavy vehicles crosses a bridge per hour on average. We need to find the probability of 100 or less number of heavy vehicles crossing that bridge in a given hour.

So, here $\lambda = 80$, and $k = 120$. We write the following program for this that uses Poisson distribution function.

```
## Poisson Distribution

ans<-ppois(100, lambda=80)
sprintf("Probability of crossing 100 heavy vehicles =%f ",ans)
```

Note the usage of ppois( ) function. We save this program as *PoissoDistribution.R* and then we execute the program by invoking the command *Rscript PoissoDistribution.R* from the terminal. The execution is shown below.

```
$ Rscript PoissoDistribution.R
[1] "Probability of crossing 100 heavy vehicles =0.986831 "
```

There are many other features of R programming like plotting graphs, and string handling. You should consult the text books for more on this. Before we conclude, we shown you how to plot graphs and save the graphs. We want to draw a curve for the function $f(x) = e - e^{-x}$. We use a user-defined function for this. Following program shows the script.

```
## Plot a curve
x11()    # holds the graph window
pdf(file="plot1.pdf")    # define the file where to save graph o/
    p
f<-function(x) return (1-exp(-x))     # user defined func.
curve(f,0,4)     # generates the curve for f
```

```
dev.off()
```

Note the usage of different functions here. Run the program. An output file with name *plot1.pdf* is generated when we run the program. The figure 2.1 is shown below.

Explore the use of plot() function which is very usefule for plotting graph where we can define the label of X-axis and Y-axis, can give title of the graph etc.

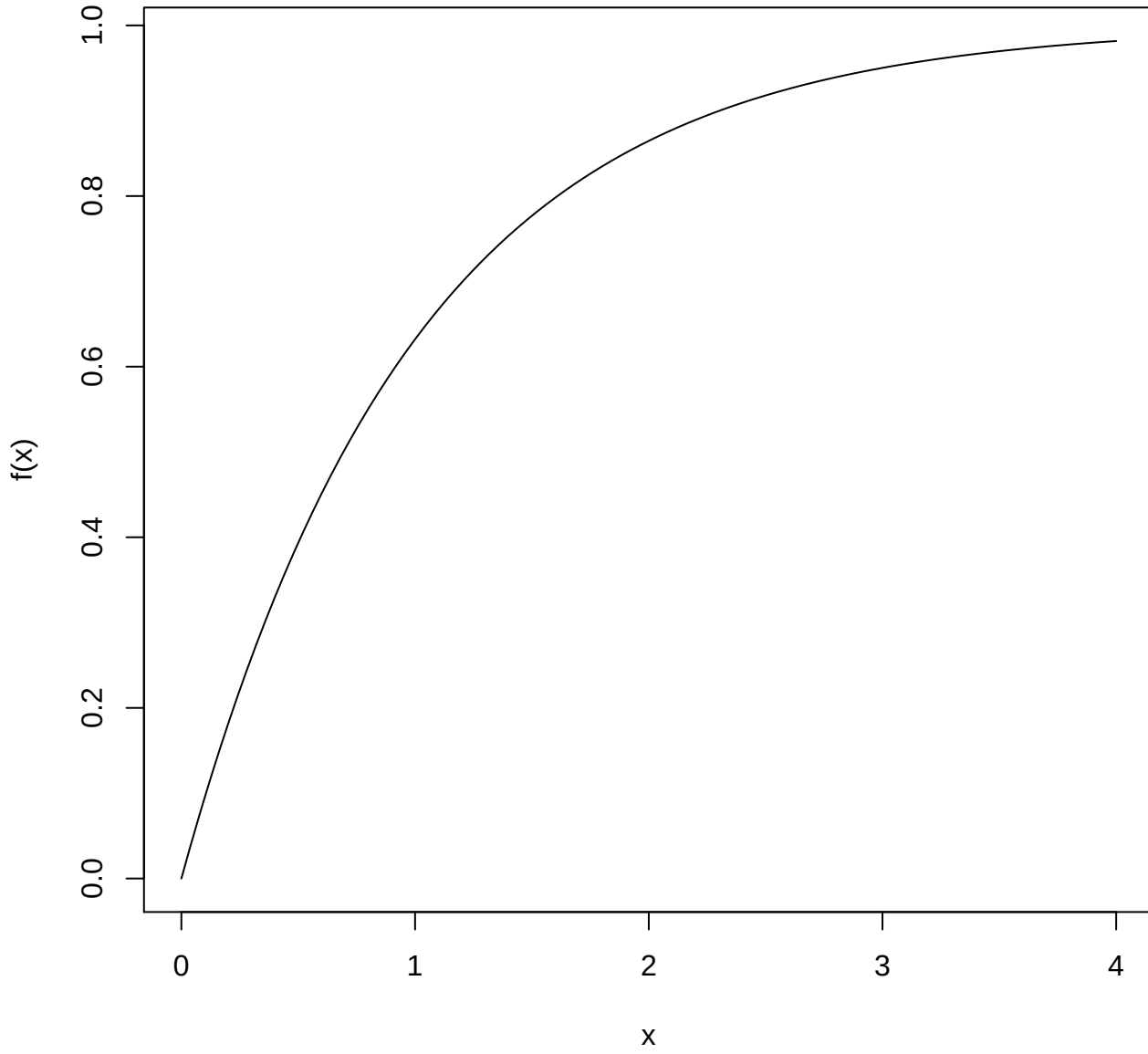

Figure 2.1: Curve Output