UNIT III

- Doing Math and Simulation in R
- Math Function
- Extended Example Calculating Probability-
- Cumulative Sums and Products-
- Minima and Maxima- Calculus
- Functions For Statistical Distribution
- Sorting,
- Linear Algebra Operation on Vectors and Matrices
- Extended Example: Vector cross Product
- Extended Example: Finding Stationary Distribution of Markov Chains
- Set Operation
- Input /out put
- Accessing the Keyboard and Monitor
- Reading and writing Files

Math Functions

- R includes an extensive set of built-in math functions. Here is a partial list:
- exp(): Exponential function, base e
- log(): Natural logarithm
- log10(): Logarithm base 10
- sqrt(): Square root
- abs(): Absolute value

Math Functions

- sin(), cos(), tan()and so on: Trig functions, the arguments will be in radians,
- asin(), acos(), atan() inverse trignometry functions.
- > tan(45*pi/180)
- [1] 1
- > a<-tan(45*pi/180)
- > b<-atan(a)
- > b
- [1] 0.7853982
- > b*180/pi
- [1] 45

Math Functions

- sum(): sum returns the sum of all the values present in its arguments.
  - sum(..., na.rm = FALSE)
- ... : numeric or complex or logical vectors.
- na.rm : logical. Should missing values (including NaN) be removed?

- prod(): prod returns the product of all the values present in its arguments.
  - sum(..., na.rm = FALSE)
- ... : numeric or complex or logical vectors.
- na.rm : logical. Should missing values (including NaN) be removed?

Math Functions

- cumsum() and cumprod(): Cumulative sum and product of the elements of a vector
  - > x <- c(12,5,13)
  - > cumsum(x)
  - [1] 12 17 30
  - > cumprod(x)
  - [1] 12 60 780
- round(), floor(), and ceiling(): Round to the closest integer, to the closest integer below, and to the closest integer above
- factorial(): Factorial function

Math Functions

- min() and max(): Minimum value and maximum value within a vector.
- which.min() and which.max(): Index of the minimal element and maximal element of a vector.
- pmin() and pmax(): Element-wise minima and maxima of several vectors
  - There is quite a difference between min() and pmin(). The former simply combines all its arguments into one long vector and returns the minimum value in that vector. In contrast, if pmin() is applied to two or more vectors, it returns a vector of the pair-wise minima, hence the name pmin.
- The max() and pmax() functions act analogously to min() and pmin().

Math Functions

- x<-c(2,4,7,8,1,9,5)
- > y<-c(23,2,54,1,4,67,3)
- > min(x)
- [1] 1
- > min(x,y)
- [1] 1
- > max(x)
- [1] 9
- > max(x,y)
- [1] 67
- > pmin(x,y)
- [1] 2 2 7 1 1 9 3
- > pmax(x,y)
- [1] 23 4 54 8 4 67 5

Math Functions

- Find the smallest value of $f(x)=x^2-\sin(x)$
- Function minimization/ maximization can be done via nlm() or optim() functions in R.
- > nlm(function(x) return(x^2-sin(x)),8)
- $minimum
- [1] -0.2324656
- $estimate
- [1] 0.4501831
- $gradient
- [1] 4.024558e-09
- $code
- [1] 1
- $iterations
- [1] 5

Math Functions

- Here, the minimum value was found to be approximately−0.23, occurring at x = 0.45. A Newton-Raphson method (a technique from numerical analysis for approximating roots) is used, running five iterations in this case. The second argument specifies the initial guess, which we set to be 8.

Calculus

- R also has some calculus capabilities, including symbolic differentiation and numerical integration.
- > D(expression(exp(x^2)),"x") # derivative
- exp(x^2) * (2 * x)
- > integrate(function(x) x^2,0,1)
- 0.3333333 with absolute error < 3.7e-15

Sort, order, rank

- Numerical sorting of a vector can be done with the sort() function,
  - > x <- c(13,5,12,5)
  - > sort(x)
  - [1] 5 5 12 13
  - >x
  - [1] 13 5 12 5
- Note that x itself did not change.
- Order(), this function gives the indices of the sorted values in given vector.
  - > order(x)
  - [1] 2 4 3 1
- This means x[2] is the smallest value, x[4] is second smallest value
- Sort the given data frame

Sort, order, rank

- > rupees<-c(2,5,1,4,9,8)
- > kids<-c("Manoj","Ganesh","Sravani","Nikitha","Jyothika","Laya")
- > df<-data.frame(y,x)
- > df
- kids rupees
- 1 Manoj 2
- 2 Ganesh 5
- 3 Sravani 1
- 4 Nikitha 4
- 5 Jyothika 9
- 6 Laya 8

Sort, order, rank

- > df[order(df$rupees),]
- kids money
- 3 Sravani 1
- 1 Manoj 2
- 4 Nikitha 4
- 2 Ganesh 5
- 6 Laya 8
- 5 Jyothika 9

Sort, order, rank

- A related function is rank(), which reports the rank of each element of a vector.
  - > x<-c(2,5,1,4,9,8)
  - > rank(x)
  - [1] 2 4 1 3 6 5
- When we have duplicate values
  - > rank(y)
  - [1] 4.0 1.5 3.0 1.5
- This says that 13 had rank 4 in x; that is, it is the fourth smallest.
- The value 5 appears twice in x, with those two being the first and second smallest, so the rank 1.5 is assigned to both.

Linear Algebra Operations on Vectors and Matrices

- Multiplying a vector by a scalar
  - > z<-c(25,15,47,84,96,68,59,53,62,42,48)
  - > z*3
  - [1] 75 45 141 252 288 204 177 159 186 126 144
- To compute the inner product (or dot product) of two vectors, use crossprod(),
  - > a<-c(3,7,2)
  - > b<-c(2,5,8)
  - > crossprod(a,b)
  - [,1]
  - [1,] 57
- The function computed 3·2+7·52 + 2·8 = 57.
- Note that the name crossprod() is a misnomer, as the function does not compute the vector cross product.

Linear Algebra Operations on Vectors and Matrices

- For matrix multiplications, the operator to use is %*% not *.
  - > c<-matrix(1:4,ncol=2)
  - > c [,1] [,2]
  - [1,] 1 3
  - [2,] 2 4
  - > d<-matrix(rep(1,4),ncol=2)
  - > d
  - [,1] [,2]
  - [1,] 1 1
  - [2,] 1 1
  - > c%*%d
  - [,1] [,2]
  - [1,] 4 4
  - [2,] 6 6

Linear Algebra Operations on Vectors and Matrices

- The function solve() will solve systems of linear equations and even find matrix inverses. For example, let's solve this system:
- x1 + x2 =2
- −x1 + x2 =4
- > a<-matrix(c(1,1,-1,1),ncol=2,byrow=T)
- > b<-c(2,4)
- > solve(a,b)
- [1] -1 3
- > solve(a)
- [,1] [,2]
- [1,] 0.5 -0.5
- [2,] 0.5 0.5
- In the second call solve(), we are not giving second argument so it computers inverse of the matrix.

Linear Algebra Operations on Vectors and Matrices

- Few other linear algebra functions are,
- t(): Matrix transpose
- qr(): QR decomposition
- chol(): Cholesky decomposition
- det(): Determinant
- eigen(): Eigenvalues/eigenvectors
- diag(): Extracts the diagonal of a square matrix (useful for obtaining variances from a covariance matrix and for constructing a diagonal matrix).
- sweep(): Numerical analysis sweep operations

Linear Algebra Operations on Vectors and Matrices

- Note the versatile nature of diag(): If its argument is a matrix, it returns a vector, and vice versa. Also, if the argument is a scalar, the function returns the identity matrix of the specified size.
- > x<-matrix(1:9,ncol=3)
- > diag(x)
- [1] 1 5 9
- > a<-c(1,2,3)
- > diag(a)
- [,1] [,2] [,3]
- [1,] 1 0 0
- [2,] 0 2 0
- [3,] 0 0 3

Linear Algebra Operations on Vectors and Matrices

```
> diag(3)
     [,1] [,2] [,3]
[1,]    1    0    0
[2,]    0    1    0
[3,]    0    0    1
> diag(9)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    0    0    0    0    0    0    0    0
[2,]    0    1    0    0    0    0    0    0    0
[3,]    0    0    1    0    0    0    0    0    0
[4,]    0    0    0    1    0    0    0    0    0
[5,]    0    0    0    0    1    0    0    0    0
[6,]    0    0    0    0    0    1    0    0    0
[7,]    0    0    0    0    0    0    1    0    0
[8,]    0    0    0    0    0    0    0    1    0
```

Linear Algebra Operations on Vectors and Matrices

- The sweep() function is capable of fairly complex operations. As a simple example, let's take a 3-by-3 matrix and add 1 to row 1, 4 to row 2, and 7 to row 3.
- > a<-matrix(1:9,ncol=3)
- > a
- [,1] [,2] [,3]
- [1,] 1 4 7
- [2,] 2 5 8
- [3,] 3 6 9

Linear Algebra Operations on Vectors and Matrices

- > sweep(a,1,c(3,4,5),"+")
- [,1] [,2] [,3]
- [1,] 4 7 10
- [2,] 6 9 12
- [3,] 8 11 14
- > sweep(a,2,c(3,4,5),"+")
- [,1] [,2] [,3]
- [1,] 4 8 12
- [2,] 5 9 13
- [3,] 6 10 14
- The first two arguments to sweep() are like those of apply(): the array and the margin, which is 1 for rows in this case. The fourth argument is a function to be applied, and the third is an argument to that function.

Vector Cross Product

- The cross product of vectors (x1,x2,x3) and (y1,y2,y3) in three-dimensional space is a new three-dimensional vector, as shown
  - (x2y3 −x3y2,−x1y3 +x3y1,x1y2 −x2y1)
- M<-matrix(c(NA, NA, NA, x1, x2, x3, y1, y2, y3), ncol=3, byrow=T)
- The cross product vector can be computed as a sum of sub-determinants.
- For instance, the first component in above Equation, x2y3 − x3y2, is easily seen to be the determinant of the submatrix obtained by deleting the first row and first column of matrix M.

Vector Cross Product

```
xprod <- function(x,y) {
m <- rbind(rep(NA,3),x,y)
xp <- vector(length=3)
for (i in 1:3)
xp[i] <- -(-1)^i * det(m[2:3,-i])
return(xp)
}
```

Set Operations

- R includes some handy set operations, including these:
  - union(x,y): Union of the sets x and y
  - intersect(x,y): Intersection of the sets x and y
  - setdiff(x,y): Set difference between x and y, consisting of all elements of x that are not in y
  - setequal(x,y): Test for equality between x and y
  - c %in% y: Membership, testing whether c is an element of the set y
  - choose(n,k): Number of possible subsets of size k chosen from a set of size n

Set Operations

- Write your own binary operations.
- Code the symmetric difference between two sets— that is, all the elements belonging to exactly one of the two operand sets. Because the symmetric difference between sets x and y consists exactly of those elements in x but not y and vice versa.
- Note the code consists of easy calls to setdiff() and union().

Set Operations

- > symdiff
- function(a,b) {
- sdfxy <- setdiff(x,y)
- sdfyx <- setdiff(y,x)
- return(union(sdfxy,sdfyx))
- }
- >x
- [1] 1 2 5
- >y
- [1] 5 1 8 9
- > symdiff(x,y)
- [1] 2 8 9

Set Operations

- Write a binary operand for determining whether one set u is a subset of another set v.
- Hint: A bit of thought shows that this property is equivalent to the intersection of u and v being equal to u.

Set Operations

- > "%subsetof%" <- function(u,v) {
- + return(setequal(intersect(u,v),u))
- +}
- > c(3,8) %subsetof% 1:10
- [1] TRUE
- > c(3,8) %subsetof% 5:10
- [1] FALSE

Set Operations

- The function combn() generates combinations. Let's find the subsets of {1,2,3} of size 2.
- > c32 <- combn(1:3,2)
- > c32
- [,1] [,2] [,3]
- [1,] 1 1 2
- [2,] 2 3 3
- > class(c32)
- [1] "matrix“
- The results are in the columns of the output. We see that the subsets of {1,2,3} of size 2 are (1,2), (1,3), and (2,3).

Set Operations

- The function also allows you to specify a function to be called by combn() on each combination. For example, we can find the sum of the numbers in each subset, like this:
  - > combn(1:3,2,sum)
  - [1] 3 4 5
- The first subset, {1,2}, has a sum of 3, and so on.

Input/Output

- R features a highly versatile array of I/O capabilities. We'll start with the basics of access to the keyboard and monitor, and then go into considerable detail on reading and writing files, including the navigation of file directories.

Accessing the Keyboard and Monitor

- R provides several functions for accessing the keyboard and monitor. Few of them are scan(), readline(), print(), and cat() functions.
- Using the scan() Function
  - You can use scan() to read in a vector or a list, from a file or the keyboard.
  - Suppose we have files named z1.txt, z2.txt.
  - z1.txt contains the following
    - 123
    - 4 5
    - 6

Accessing the Keyboard and Monitor

  - z2.txt contains the follwing
    - abc
    - de f
    - g
- > scan("z1.txt")
- Read 4 items
- [1] 123 4 5 6
- > scan("z2.txt")
- Error in scan("z2.txt") : scan() expected 'a real', got 'abc’

Accessing the Keyboard and Monitor

- > args(scan)
- function (file = "", what = double(), nmax = -1L, n = -1L, sep = "",
- quote = if (identical(sep, "\\n")) "" else "'\\"", dec = ".",
- skip = 0L, nlines = 0L, na.strings = "NA", flush = FALSE,
- fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "", allowEscapes = FALSE,
- fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
- NULL
- > scan("z2.txt",what="")
- Read 4 items
- [1] "abc" "de" "f" "g"

Accessing the Keyboard and Monitor

- The scan() function has an optional argument named what, which specifies mode, defaulting to double mode.
- So, the nonnumeric contents of the file z2 produced an error. But we then tried again, with what="". This assigns a character string to what, indicating that we want character mode.
- By default, scan() assumes that the items of the vector are separated by whitespace, which includes blanks, carriage return/line feeds, and horizontal tabs. You can use the optional sep argument for other situations.

Accessing the Keyboard and Monitor

- > x1<-scan("z2.txt",what="")
- Read 4 items
- > x2<-scan("z2.txt",what="",sep="\n")
- Read 3 items
- > x1
- [1] "abc" "de" "f"   "g"
- > x2
- [1] "abc" "de f" "g"

Accessing the Keyboard and Monitor

- You can use scan() to read from the keyboard by specifying an empty string for the filename:
- > scan("")
- 1: 43 23 65 12
- 5:
- Read 4 items
- [1] 43 23 65 12
- > scan("",what="")
- 1: "Rama Krishna" "Bala Sai" "Geethika" "srikanth" "Preethi" "Murty"
- 7:
- Read 6 items
- [1] "Rama Krishna" "Bala Sai" "Geethika" "srikanth"
- [5] "Preethi" "Murty"

Accessing the Keyboard and Monitor

- readline() function
  - If you want to read in a single line from the keyboard, readline() is very handy.
  - readline() is called with its optional prompt.
  - > readline()
  - Hai how are u
  - [1] "Hai how are u“
  - > readline("Enter your Name")
  - Enter your Name Dr. P. Sita Rama Murty
  - [1] "Dr. P. Sita Rama Murty"

Accessing the Keyboard and Monitor

- Print() function
  - At the top level of interactive mode, you can print the value of a variable or expression by simply typing the variable name or expression. This won't work if you need to print from within the body of a function. In that case, you can use the print() function,
  - print() is a generic function, so the actual function called will depend on the class of the object that is printed. If, for example, the argument is of class "table", then the print.table() function will be called.

Accessing the Keyboard and Monitor

- > print("Hai")
- [1] "Hai"
- > x<-matrix(1:4,ncol=2)
- > print(x)
- [,1] [,2]
- [1,] 1 3
- [2,] 2 4
- > print(x%*%x)
- [,1] [,2]
- [1,] 7 15
- [2,] 10 22

Accessing the Keyboard and Monitor

- cat() function:
  - We can use cat() instead of print function, for cat() function, we have to supply our own end of line character.
  - > cat("abc\n")
  - abc
  - > cat("geetha venkat", "sravana Kumar", "phani\n", sep="\n")
  - geetha venkat
  - sravana Kumar
  - Phani
  - > cat("geetha venkat", "sravana Kumar", "phani\n", sep="")
  - geetha venkatsravana Kumarphani
  - > cat("geetha venkat", "sravana Kumar", "phani\n", sep="-")
  - geetha venkat-sravana Kumar-phani

Accessing the Keyboard and Monitor

- Reading and Writing files
  - read.table() is used to read a data frame from the file.
  - > read.table("z1.txt",header=TRUE)
  - name nature
  - 1 Hemant Obidient
  - 2 Sowjanya Hardworking
  - 3 Girija Friendly
  - 4 Preethi Calm

Accessing the Keyboard and Monitor

- Reading and Writing files
  - Scan() would not work here, as our data-frame has mixture of character and numeric data.
  - We can read a matrix using scan as
  - Mat<-matrix(scan(“abc.txt”), nrow=2, ncol=2, byrow=T)
  - We can do this generally by using read.table() as
  - read.matrix<-function(filename){
  - as.matrix(read.table(filename))}

Accessing the Keyboard and Monitor

- Reading a Text-File:
  - readLines() is used to read in a text file, either one line at a time or in a single operation.
  - For example, suppose we have a file z1 with the following contents:
    - John 25
    - Mary 28
    - Jim 19
  - We can read the file all at once, like this:
    - z1 <- readLines("z1")

Accessing the Keyboard and Monitor

- Reading a Text-File:
  - Since each line is treated as a string, the return value here is a vector of strings—that is, a vector of character mode.
  - There is one vector element for each line read, thus three elements here.
  - Alternatively, we can read it in one line at a time. For this, we first need to create a connection, as described next.

Accessing the Keyboard and Monitor

- Introduction to Connections:
  - Connection is R's term for a fundamental mechanism used in various kinds of I/O operations.
  - The connection is created by calling file(), url(), or one of several other R functions.
  - ?connection
    - > c <- file("z1","r")
    - > readLines(c,n=1)
    - [1] "John 25"
    - > readLines(c,n=1)
    - [1] "Mary 28"
    - > readLines(c,n=1)
    - [1] "Jim 19"

Accessing the Keyboard and Monitor

- Introduction to Connections:
  - > readLines(c,n=1)
  - character(0)
  - We opened the connection, assigned the result to c, and then read the file one line at a time, as specified by the argument n=1.
  - When R encountered the end of file (EOF), it returned an empty result.
  - We needed to set up a connection so that R could keep track of our position in the file as we read through it.

Accessing the Keyboard and Monitor

- Introduction to Connections:
  - c <- file("z","r")
  - > while(TRUE) {
  - + rl <- readLines(c,n=1)
  - + if (length(rl) == 0) {
  - + print("reached the end")
  - + break
  - + } else print(rl)
  - +}
  - [1] "John 25"
  - [1] "Mary 28"
  - [1] "Jim 19"
  - [1] "reached the end"

Accessing the Keyboard and Monitor

- Introduction to Connections:
  - > c <- file("z1","r")
  - > readLines(c,n=2)
  - [1] "John 25" "Mary 28"
  - > seek(con=c,where=0)
  - [1] 16
  - > readLines(c,n=1)
  - [1] "John 25”

Accessing the Keyboard and Monitor

- The argument where=0 in our call to seek() means that we wish to position the file pointer zero characters from the start of the file—in other words, directly at the beginning.
- The call returns 16, meaning that the file pointer was at position 16 before we made the call. The first line consists of "John 25" plus the end-of-line character, for a total of eight characters, and the same is true for the second line. So, after reading the first two lines, we were at position 16.
- To close a connection call close(). This lets the system know that the file you have been reading is complete.

Accessing the Keyboard and Monitor

- Accessing files on remote machines via urls:
  - Certain I/O functions, such as read.table() and scan(), accept web URLs as arguments.
  - uci <- "http://archive.ics.uci.edu/ml/machine-learning-databases/echocardiogram/echocardiogram.data”
  - > ecc <- read.csv(uci)

Accessing the Keyboard and Monitor

- Writing to a file:
  - The function write.table() works very much like read.table(), except that it writes a data frame instead of reading one.
    - > kids <- c("Jack","Jill")
    - > ages <- c(12,10)
    - > d <- data.frame(kids,ages,stringsAsFactors=FALSE)
    - >d kids ages
    - 1 Jack 12
    - 2 Jill 10
    - > write.table(d,"kds.txt")

Accessing the Keyboard and Monitor

- Writing to a file:
  - In the case of writing a matrix to a file, just state that you do not want row or column names, as follows:
  - write.table(xc, "xcnew", row.names=FALSE, col.names=FALSE)
  - The function cat() can also be used to write to a file, one part at a time.
  - > cat("abc\n",file="u")
  - > cat("de\n",file="u",append=TRUE)
  - The first call to cat() creates the file u, consisting of one line with contents "abc". The second call appends a second line. The file is automatically saved after each operation.

Accessing the Keyboard and Monitor

- You can also use writeLines(), the counterpart of readLines(). If you use a connection, you must specify "w" to indicate you are writing to the file, not reading from it:
  - > c <- file("www","w")
  - > writeLines(c("abc","de","f"),c)
  - > close(c)
- The file www will be created with these contents:
  - abc
  - de
  - f

Probability

- Now we see how to find the probability that exactly one event occur:
- If three friends x, y, z appeared for an examination x has17% chance of failure, y has 7% chance of failure, and Z has 26% chance of failure.
- What is the probability that exactly one of them will fail in the exams?
- P(X fails, but not others) = 0.17 * 0.93 * 0.74, P(Y fails, but not others) = 0.83 * 0.07 * 0.74, P(Z fails, but not others) = 0.83 * 0. 93 * 0.26.

Probability

- Suppose we have n independent events, and the ith event has the probability pi of occurring. What is the probability of exactly one of these events occurring?
- Suppose that n = 3 and our events are named A, B, and C. Then we break down the computation as follows:
- P(exactly one event occurs) = P(A and not B and not C) + P(not A and B and not C) + P(not A and not B and C)
- P(A and not B and not C) would be pA(1−pB)(1−pC), and so on.
- For general n, that is calculated as follows: Sum(pi(1−p1)...(1−pi−1)(1−pi+1)...(1−pn)) where i is 1 to n

Probability

- exactlyone <- function(p) {
- notp <-1-p
- tot <- 0.0
- for (i in 1:length(p))
- tot <- tot + p[i] * prod(notp[-i])
- return(tot)
- }
- The expression notp[-i] computes the product of all the elements of notp, except the ith—exactly what we need.