**2.3 Basic math and stat using R**

This section is devoted to transform your math or stat knowledge in computational form. R will serve you as a scientific calculator with programming functionality.

**2.3.1 Perform simple arithmetics using R.**

In this section (section [2.3.1](https://bookdown.org/siju_swamy/Stat_Lab/intro.html#experiment2)) we will focus on the functionality of R as a calculator. We will begin with simple addition, multiplication, and power computations. The codes/programs in R are read from left to right, and executed in that order.

57 + 89

## [1] 146

45 - 87 *# find difference*

## [1] -42

60 \* 3 *# find product*

## [1] 180

7/18 *# find quotient*

## [1] 0.3888889

4^4 *# calculating power*

## [1] 256

It is implicitly assumed (and implemented too) that any reliable computing software must have included the brackets, orders, division, multiplication, addition, and subtraction, *BODMAS* rule. It means that if the user executes 4×334×33, the answer is 108, that is, order is first executed and then multiplication, and not 1728, multiplication followed by order. We verify the same next.

4\*3^3

## [1] 108

**2.3.2 Perform basic R functions.**

In this section we will discuss R functions related to basic math and stat.

The absolute value of elements or vectors can be found using the abs command. For example:

l1=-4:3 *# creating a sequence of numbers from -4 to 3 with step size 1*

l1

## [1] -4 -3 -2 -1 0 1 2 3

abs(l1) *# returns absolute value of l1*

## [1] 4 3 2 1 0 1 2 3

Remainders can be computed using the R operator %%.

(-4:3) %% 3

## [1] 2 0 1 2 0 1 2 0

The integer divisor between two numbers may be calculated using the %/% operation.

(-4:4) %/% 3

## [1] -2 -1 -1 -1 0 0 0 1 1

The sign operator tells whether an element is positive, negative, or neither.

sign(-4:3)

## [1] -1 -1 -1 -1 0 1 1 1

The number of digits to which R gives answers is set at seven digits by default. There are multiple ways to obtain our answers in the number of digits that we actually need. For instance, if we require only two digits accuracy for 7/18, we can use the following:

round(7/18,2)

## [1] 0.39

It is often of interest to obtain the greatest integer less than the given number, or the least integer greater than the given number. Such tasks can be handled by the functions floor and ceiling respectively. For instance:

floor(3.1415)

## [1] 3

ceiling(0.618)

## [1] 1

Sum of values store in variables can be found using the sum function in R.

a=5;b=10

paste0("sum is:", sum(a,b))

## [1] "sum is:15"

Note: An array in R usually created with the combine (syntax :c(variables)) function.

val=c(1,2,6,7,8,3,5,7)

sum(val)

## [1] 39

Other similar math functions are: all, any, prod, min, max, and range. The last five of these is straightforward for the user to apply to their problems. This is illustrated by the following.

prod(val)

## [1] 70560

min(val)

## [1] 1

max(val)

## [1] 8

range(val)

## [1] 1 8

Now we are left to understand the R functions any and all. The any function checks if it is true that the array under consideration meets certain criteria. As an example, suppose we need to know if there are some elements of (−1,3,4,−9,4)(−1,3,4,−9,4) less than 0.

any(c(-1,3,4,-9,4)<0)

## [1] TRUE

all(c(1,6,-14,-154,0)<0) *# all checks if criteria is met by each element*

## [1] FALSE

Trigonometric functions are very useful tools in statistical analysis of data. It is worth mentioning the emerging areas where this is frequently used. Wavelet analysis, functional data analysis, and time series spectral analysis are a few examples. Such a discussion is however beyond the scope of this current book.We will contain ourself with a very elementary session. The value of 𝜋 is stored as one of the constants in R.

sin(pi/2)

## [1] 1

atan(0) *# atan calculate inverse tan*

## [1] 0

log(exp(1)) *# log calculate natural logarithm*

## [1] 1

Arc-cosine, arc-sine, and arc-tangent functions are respectively obtained using acos, asin, and atan. Also, the hyperbolic trigonometric functions are available in cosh, sinh, tanh, acosh, asinh, and atanh.

**2.3.3 Complex numbers in R**

Complex numbers can be handled easily in R. Its use is straightforward and the details are obtained by keying in ?complex or ?Complex at the terminal. As the arithmetic related to complex numbers is a simple task, we will look at an interesting case where the functions of complex numbers arise naturally. To input a complex number 1+i1+i, write c=1+1i.

1+1i

## [1] 1+1i

exp(1i\*pi)

## [1] -1+0i

Note that eiπ=cosπ+isinπ=−1+i0eiπ=cos⁡π+isin⁡π=−1+i0.

For the illustration purpose, the characteristic function of a uniform distribution in [−1,1][−1,1] (φ(t)=eitb−eitait(b−a)φ(t)=eitb−eitait(b−a)) can be simulated in R as follows:

*# Plot of Characteristic Function of a U(-1,1) Random Variable*

a <- -1; b <- 1

t <- seq(-20,20,.1)

chu <- (exp(1i\*t\*b)-exp(1i\*t\*a))/(1i\*t\*(b-a))

plot(t,chu,"l",ylab=(expression(varphi(t))),main="Characteristic Function of Uniform Distribution [-1, 1]")

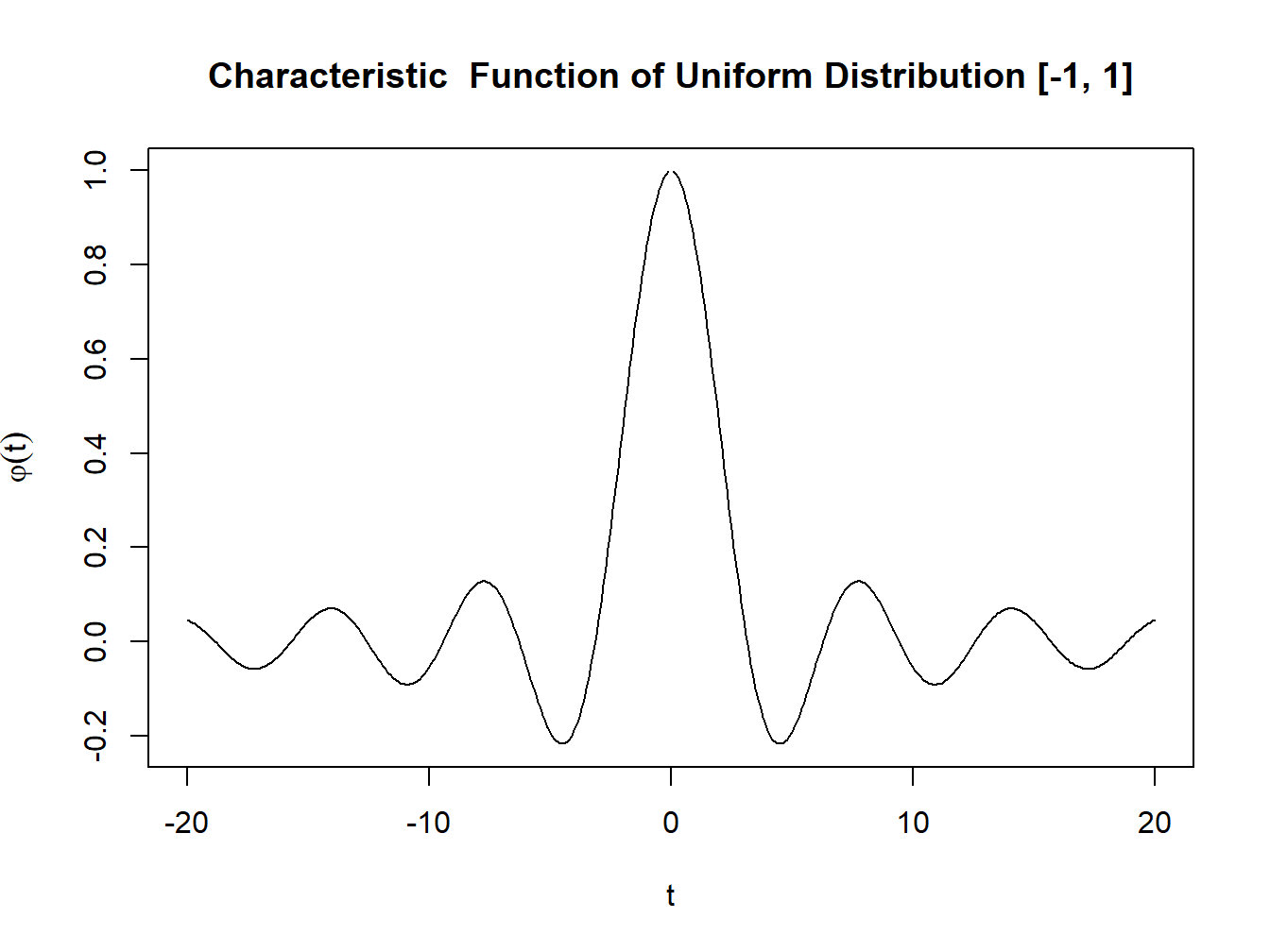


Figure 2.2: Characteristic function of Uniform Distribution

**2.3.4 Special Mathematical Functions**

Here, by special functions, we mean certain mathematical entities that are difficult to calculate as the series/range increases. Factorials do have an in-built function in factorial. A few examples are in order.

**Factorial**

factorial(3)

## [1] 6

**Combination**

Consider the classical problem of selecting r-out-of-n objects. If we have to select k objects with replacement, that is the first drawn object is replaced in the pack before the second object is drawn, the number of ways of accomplishing the task is (nk)(nk). If n = 10 and r = 4, the choose function gives us the desired answer through the arguments n and k. That is, (nk)(nk) is calculated in R with choose(n,k).

n=10;k=4

choose(n,k)

## [1] 210

**Permutation**

The permutation operation is defined as nPr=n×(n−1)×⋯×(n−r+1)nPr=n×(n−1)×⋯×(n−r+1). So we can use prod function in R to find 10P410P4 as:

prod(10:(10-4+1)) *# permutations of 10 p 4*

## [1] 5040

## 4.1 Basic statistical functions in R

**Summary Statistics**

Consider the Youden and Beale dataset- AD-9 avalable in R distribution. Here, we have two preparations of virus extracts and the data is available on the number of lesions produced by each extract on the eight leaves of tobacco plants. We read this data and assigned it to a new object yb. The summary statistics for the two extracts are as follows ([Tattar 2015](https://bookdown.org/siju_swamy/Stat_Lab/descriptive-statistics-probability-using-r.html" \l "ref-ACSWR)):

library(ACSWR)

## Warning: package 'ACSWR' was built under R version 4.1.3

data(yb)

summary(yb)

## Preparation\_1 Preparation\_2

## Min. : 7.00 Min. : 5.00

## 1st Qu.: 8.75 1st Qu.: 6.75

## Median :13.50 Median :10.50

## Mean :15.00 Mean :11.00

## 3rd Qu.:18.50 3rd Qu.:14.75

## Max. :31.00 Max. :18.00

The summary considers each variable of the yb data frame and appropriately returns the summary for that type of variable. What does this mean? The two variables Preparation\_1 and Preparation\_2 are of the integer class. In fact, we need these two variables to be of the numeric class. For each class of objects, the summary function invokes particular methods, check methods from the utils package, as defined for an S3 generic function. We will not delve more into this technical aspect, except that we appreciate that R has one summary for a numeric object and another for a character object. For an integer object, the summary function returns the minimum, first quartile, median, mean, third quartile, and maximum. Minimum, mean, median, and maximum are fairly understood and recalled easily.

Quartiles are a set of (three) numbers, which divide the range of the variables into four equal parts, in the sense that each part will have 25% of the observations. The range between minimum and first quartile will contain the smallest 25% of the observations for that variable, the first and second quartile (median) will have the next, and so on. For the yb data frame, it may be seen from the mean and median summaries that the Preparation\_2 has a lesser number of lesions than Preparation\_1. The first and third quantiles can also be obtained by using the codes quantile(x,.25) and quantile(x,.75) respectively. The percentiles, nine percentiles which divide the data into ten equal regions, can be obtained using the function quantile:

quantile(yb$Preparation\_1,seq(0,1,.1)) *# here seq gives + 0, .1, .2, ...,1*

## 0% 10% 20% 30% 40% 50% 60% 70% 80% 90% 100%

## 7.0 7.7 8.4 9.1 9.8 13.5 17.2 17.9 19.2 23.3 31.0

The code lapply(yb,summary) also gives us the same values. The lapply will be considered in detail later.

lapply(yb,summary)

## $Preparation\_1

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 7.00 8.75 13.50 15.00 18.50 31.00

##

## $Preparation\_2

## Min. 1st Qu. Median Mean 3rd Qu. Max.

## 5.00 6.75 10.50 11.00 14.75 18.00

Tukey’s fivenum gives another set of important summaries, which are detailed data analysis section. The fivenum summary, containing minimum, lower hinge, median, upper hinge, and maximum,

fivenum(yb$Preparation\_1)

## [1] 7.0 8.5 13.5 19.0 31.0

We now consider some measures of dispersion. Standard deviation and variance of a sample are respectively obtained by the functions sd and var. Range is obtained using the range function.

sd(yb$Preparation\_1); sd(yb$Preparation\_2) *# finding sd*

## [1] 8.176622

## [1] 4.956958

var(yb$Preparation\_1); var(yb$Preparation\_2) *# finding variances*

## [1] 66.85714

## [1] 24.57143

range(yb$Preparation\_1); range(yb$Preparation\_2) *# list out ranges of data*

## [1] 7 31

## [1] 5 18

Note: In general, the median is more robust than the mean. A corresponding measure of dispersion is the median absolute deviation, abbreviated as MAD. The R function mad returns MAD. Another robust measure of dispersion is the inter quartile range, abbreviated as IQR, and the function available for obtaining it in R is IQR. These measures for the variables of the Youden and Beale data yb are thus computed in the next code chunck.

mad(yb$Preparation\_1); mad(yb$Preparation\_2)

## [1] 7.413

## [1] 5.9304

IQR(yb$Preparation\_1); IQR(yb$Preparation\_2)

## [1] 9.75

## [1] 8

The skewness and kurtosis are also very important summaries. Functions for these summaries are not available in the base package. The functions skewcoeff and kurtcoeff from the ACSWR package will be used to obtain these summaries. The functions skewness and kurtosis from the e1071 package are more generic functions. We obtain the summaries for the Youden and Beale experiment data below.

skewcoeff(yb$Preparation\_1); kurtcoeff(yb$Preparation\_1)

## [1] 0.8548652

## [1] 2.727591

skewcoeff(yb$Preparation\_2); kurtcoeff(yb$Preparation\_2)

## [1] 0.2256965

## [1] 1.6106

## 4.2 Basic set operations

Note that the sample space in probability theory may be treated as a super set. It has to be exhaustive, covering all possible outcomes. Loosely speaking we may say that any subset of the sample space is an event. Thus, we next consider some set operations, which in the language of probability are events. Let X,YX,Y be two subsets of the sample space Ω. The union, intersection, and complement operations for sets is defined as follows:

### 4.2.1 Set operations in R

Performs set union, intersection, (asymmetric!) difference, equality and membership on two vectors.

Syntax: union(x, y),intersect(x, y), setdiff(x, y), setdiff(y, x), setequal(x, y). **Example:**

x <- c(sort(sample(1:20, 9)), NA)

y <- c(sort(sample(3:23, 7)), NA)

union(x, y)

## [1] 1 2 6 7 10 13 14 15 19 NA 3 8 18 21 23

intersect(x, y)

## [1] 13 15 NA

setdiff(x, y)

## [1] 1 2 6 7 10 14 19

setdiff(y, x)

## [1] 3 8 18 21 23

setequal(x, y)

## [1] FALSE

*## True for all possible x & y :*

setequal( union(x, y),c(setdiff(x, y), intersect(x, y), setdiff(y, x)))

## [1] TRUE

is.element(x, y) *# length 10*

## [1] FALSE FALSE FALSE FALSE FALSE TRUE FALSE TRUE FALSE TRUE

is.element(y, x) *# length 8*

## [1] FALSE FALSE TRUE TRUE FALSE FALSE FALSE TRUE