

DOKUZ EYLÜL UNIVERSITY
GRADUATE SCHOOL OF NATURAL AND APPLIED
SCIENCES

ROBUST TWO SAMPLE TESTS APPLICATIONS
BY USING STATISTICAL PROGRAMING
LANGUAGE R

by
Mustafa BİNAR

September, 2011

İZMİR

**ROBUST TWO SAMPLE TESTS APPLICATIONS
BY USING STATISTICAL PROGRAMING
LANGUAGE R**

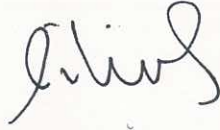
**A Thesis Submitted to the
Graduate School of Natural and Applied Sciences of Dokuz Eylül University
In Partial Fulfillment of the Requirements for the Master of Science in
Statistics, Statistics Program**

**by
Mustafa BİNAR**

**September,2011
İZMİR**

M.Sc THESIS EXAMINATION RESULT FORM

We have read the thesis entitled “**ROBUST TWO SAMPLE TESTS APPLICATIONS BY USING STATISTICAL PROGRAMING LANGUAGE R**” completed by **MUSTAFA BİNAR** under supervision of **ASIST. PROF. DR. A. FIRAT ÖZDEMİR** and we certify that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.



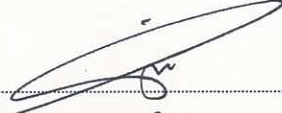
Asist. Prof. Dr. A. Firat ÖZDEMİR

Supervisor



Prof. Dr. Serdar KURT

(Jury Member)



Doç. Dr. Cenk ÖZK

(Jury Member)



Prof. Dr. Mustafa SABUNCU

Director

Graduate School of Natural and Applied Sciences

ACKNOWLEDGMENTS

I would like to firstly express my deep gratitude to my supervisor Ass. Prof. Dr. Abdullah Fırat ÖZDEMİR and Dr. Engin YILDIZTEPE due to inform me with his own informations about R statistical programming language and robustness in statistics.

I would like to thank my father Suat BİNAR, my mother Seyhun BİNAR and my sister Fatma Nur BİNAR for continuous moral and material support and motivating comments throughout this research.

Mustafa BİNAR

ROBUST TWO SAMPLE TESTS APPLICATIONS BY USING STATISTICAL PROGRAMMING LANGUAGE R

ABSTRACT

There are various statistical tests used for performing statistical inferences in the recent statistics literature. The software and algorithms developed for these statistical tests help the researcher in making inferences. These software programs are generally called statistical software. Current statistical software can easily perform many classical statistical tests. However, further studies on these software programs are needed and some novel software and algorithms have been tried to be developed since the current programs are inadequate for the newly developed statistical methods against some hypothesis violations. In this study, the R programming language, which has been widely used especially in academic studies, is mentioned. R, which was designed especially for data processing, computation, graphical display and statistical methods, is a programming language that has been developed by the contributions of researchers from various places in the world since 1997, which is accessible free of charge. In the application chapter of the study, a R function is written for the B square test with one-step M-estimator and bootstrap-t method and a simulation study is conducted for various two independent samples tests. And the results of the simulation are discussed in detail.

Keywords: R programming language, statistical software development, robust tests, robustness

İSTATİSTİKSEL PROGRAMLAMA DİLİ R İLE DAYANIKLI İKİ ÖRNEKLEM TESTLERİ UYGULAMALARI

ÖZ

Günümüz istatistik literatüründe, istatistiksel çıkarımların yapılabilmesi için kullanılan birçok istatistiksel test vardır. Bu istatistiksel testler için geliştirilen yazılım ve algoritmalar da çıkarsamalarda kullanılmak üzere araştırmacıya yardımcı olmaktadır. Bu yazılımlara genel olarak istatistiksel yazılımlar denmektedir. Günümüz istatistiksel yazılımları birçok klasik istatistiksel testi kolaylıkla yapabilmektedir. Fakat, bazı varsayım ihlallerine karşı yeni geliştirilen istatistiksel yöntemler için yetersiz olmaları ve literatürde bulunan yazılımların ücretli olmalarından dolayı yeni bir takım çalışmalara gereksinim duyulmuş, yeni yazılımlar ve algoritmalar geliştirilmeye çalışılmıştır. Bu çalışmada, son yıllarda özellikle akademik çalışmalarda yaygın olarak kullanılan R programlama dilinden bahsedilmiştir. Özellikle veri işleme, hesaplama, grafiksel gösterim ve istatistiksel yöntemler için tasarlanan R, 1997 yılından bugüne dünyanın farklı yerlerindeki araştırmacıların katkıları ile geliştirilmiş ve ücretsiz olarak ulaşılabilen bir programlama dilidir. Çalışmanın uygulama bölümünde ise tek-adım M-tahmincisi ile hesaplanan B kare testi ve bootstrap-t metodu için R fonksiyonu yazılmış ve bazı bağımsız iki örneklem testleri için benzetim çalışması yapılmıştır. Bu benzetim çalışmasının sonuçlarına değinilmiştir.

Anahtar Sözcükler: R programlama dili, istatistiksel yazılım geliştirme, dayanıklı testler, dayanıklılık

CONTENTS

	Page
M.Sc THESIS EXAMINATION RESULT FORM.....	ii
ACKNOWLEDGMENTS	iii
ABSTRACT	iv
ÖZ	v
CHAPTER ONE - INTRODUCTION	1
1.1 Introduction To R.....	1
1.2 Facilities Of The R Environment	2
1.3 Getting Help	3
CHAPTER TWO - OBJECTS IN R	5
2.1 Vectors	6
2.1.1 Numerical Vectors And Assignment	7
2.1.2 Logical Vectors And Assignment	8
2.1.3 Character Vectors And Assignment.....	9
2.2 Matrices and Arrays	11
2.3 Lists	13
2.4 Data Frames	14
2.5 Functions	16
2.6 NULL Object	16
2.7 Environments	17
2.8 Classes.....	18
CHAPTER THREE - FUNCTIONS AND GRAPHICAL PROCEDURES.....	20
3.1 Recalling A Built-in Function In R.....	21
3.2 Writing Functions.....	23
3.3 Conditional And Repetitive Statements	25
3.3.1 Conditional Statements	25

3.3.2 Repetitive Statements.....	27
3.3.2.1 The for Loop	27
3.3.2.2 The while And repeat Loop	28
3.4 Simulation In R Programming Language	31
3.4.1 Simulation Tools In R Programming Language	31
3.4.2 The Bootstrap Resampling Method	32
3.5 Graphical Functions In R	33
3.5.1 Arguments Of The High Level Graphical Functions.....	33
3.5.2 Low Level Graphical Functions.....	34
3.5.3 High Level Graphical Functions	35
3.5.3.1 Bar Charts And Dot Charts	36
3.5.3.2 Pie Charts	37
3.5.3.3 Histograms	38
3.5.3.4 Box Plot.....	39
3.5.3.5 Scatter Plot	40
3.5.3.6 Q-Q Norm	41
3.6 Special Argument “...”	43
CHAPTER FOUR - AN APPLICATION WITH R: B² TEST WITH ONE-STEP M ESTIMATOR AND BOOTSTRAP-T METHOD.....	46
4.1 M-Estimators And One-Step M-Estimator	46
4.2 Estimating The Standard Error Of $\hat{\mu}_m$	48
4.3 B^2 Test With One-Step M-Estimator And Bootstrap-t Method.....	50
4.4 Yuen’s Test	53
4.5 Design Of The Simulation Study	54
4.6 Simulation Results	55
CHAPTER FIVE - CONCLUSIONS.....	58
REFERENCES.....	61

APPENDIX63

CHAPTER ONE

INTRODUCTION

Statistics is a discipline which comprises of all methods used for collecting, summarizing and analyzing the data, making inferences from these data. In practice, researchers need some software in order to make inferences easily and in a rapid way. These softwares are generally called, in the statistics literature, as statistical software or statistical software packages.

There are many programs in the statistical software literature. Most of these software packages include nearly all of the classical statistical methods. However, these software packages' being inadequate for the new methods in statistics, or having high prices gave rise to the need for new software. "R" statistical programming language is a statistical software development environment, which was developed to satisfy this need and its user group has been increasing continuously.

1.1 Introduction To R

R language was first written by Ross Ihaka and Robert Gentleman from Auckland University Statistics Department, in New Zealand. Later, a group comprising of various researchers from all over the world, continued to develop R language and this group was named "R Core Team" in 1997.

Two programming languages were influential in the design of R language. These were the S language developed by Becker, Chamber and Wilks, and the Scheme language developed by Sussman. R is similar to the S language in appearance, but close to Scheme in terms of syntax and semantics. The first version of S was completed in 1976 (R Core Team, 2009). It was a programming language developed as an alternative to FORTRAN, the preferred programming language in statistical calculations in that period. In 1988, S-PLUS was introduced after improving S language. S-PLUS is now being marketed by a software company named TIBCO, and its licence belongs to this company. S-PLUS software being expensive for

academic research and education, Ross Ihaka and Robert Gentleman, two statisticians from New Zealand, developed the R programming language in pursuit of new software. The codes of R were released under General Public Licence-GPL in 1995, and the first version developed by the “core team, assembled in a short time, was released on 29th of February, 2000 (R Core Team, 2009).

R statistical programming language was designed for data manipulation, calculation and graphical display. Applications to be used for the newly developed data analysis methods can easily be written in R, since it is under GPL and it has the related packages built-in. The command line syntax of R is bears resemblance to C language. At the same time, it is a functional programming language. Therefore, it has some functions facilitating the code writing for statisticians and mathematicians. It has some special functions a statistician can use in data analysis and graphical display.

R and S-PLUS languages have similar command lines. Most of the codes written in any of these languages can be implemented in the other. Both of them produce successful data analysis results. However, R is an opensource code and can be accessed by a researcher anywhere in the world.

It is possible, in R language, to use many methods developed recently in addition to classical statistical analysis methods and graphical tools. R can be easily obtained via internet and is free of charge.

1.2 Facilities Of The R Environment

R language has objects in different types for an efficient data manipulation. The most significant features of R language can be summarized as below:

- It has different object types for efficient data analysis and storage.
- It has operators and mathematical functions required for performing operations on arrays and matrices.

- It includes the functions required for descriptive statistics and classical statistical methods.
- It has graphic functions which are efficient and can be defined in detail.
- The functions written for novel methods can easily be distributed using packages.
- It is still being improved and supported by the R core team.
- It can be obtained free of charge.

R language bears resemblance to C language in terms of the syntax of the commands. But R is a functional programming language. This feature facilitates the statistical modeling and graphical display.

R programming language can be rapidly updated for newly developed methods, and the packages written for these new methods can be easily imported.

1.3 Getting Help

There are three types of help in R language. These are; the help menu under the R tools section, electronic user's manuals, and the r-project.org website. The electronic user's manuals can be accessed from the Manuals section under the help menu, and the documentation section in the following internet address: www.r-project.org.

One of the help features in R is the access to help about the functions in R. This help topic can be accessed in two ways. First one is the code written in the command line of R, and the other one is the functional help under the help menu. For instance, if arithmetic mean is at issue and the arguments required for this are not known, a functional help can be accessed by entering the code below into the command line. The other way of getting such help is using the help menu.

```
>?mean
```

```
or
```

```
>help(mean)
```

The subject based search in order to find the function to be used can be performed with the “help.search” function. For instance, in order to find a function on data input, the command

```
>help.search(“data input”) can be used (R Core Team, 2009).
```

In the second and third chapters of this study the R programming language is introduced in many aspects. In Chapter Two object structure, the use of matrices and arrays, and examples regarding these are presented. In Chapter Three, the function concept of R, the graphical properties, and the control and loop structures are explained.

In Chapter Four, a simulation application is performed using a R function written for the use of the method developed for comparing independent groups with one-step M-estimator and bootstrap-t method (Özdemir & Kurt, 2006). In the last chapter the results of the simulation are evaluated.

CHAPTER TWO

OBJECTS IN R

R programming language is an object-oriented language as many other programming languages today. Object-oriented programming languages cannot provide direct access to the data stored in the memory. Therefore, another option is needed in order to access the data stored by R. This option is called the objects. The object of R may be symbols or variables.

In R, the objects also have the structure which facilitates performing operations on the data. For instance, an operation which needs loops can be performed, in R, by functions that operate on matrices or arrays in an easy and efficient way. As one could understand, matrices, arrays and functions are, too, objects for storing data in R. For example about this issue in R; let `x_examp` vector be a variable comprising only of numbers, the `ls()` function in R returns the objects present in the environment. If the example below is examined, it can be seen that `x_examp` variable is an object.

```
>x_examp<-1:4
>ls(pattern="x_examp")
[1] "x_examp"
```

As it can be seen in the example, `x_examp` variable is listed as an object. If the argument between quotation marks is not added to the argument in the `ls()` function, all objects present in R global environment would be listed. Some widely used objects in R and their return values are presented in Table 2.1.

Table 2.1 Return type of objects

Objects	Return type
"Vectors"	A vector
"matrices and arrays"	A matrix or array

Table 2.1 (Continue)

"list"	a list object
"data frames"	a data frame
"functions"	A function
"NULL"	A NULL object
"environment"	an environment
"Classes"	A class

R does not have only these types of objects. There are other object types also. However, only these types' objects are mentioned in this study.

2.1 Vectors

Vectors are one of the most important objects in R. In this object type, each data is placed in their cell. The data in the vector can be accessed via an index operator. For instance, a data in the 5th cell of x vector can be accessed as such:

```
>x<-1:5
>x[5]
[1] 5
```

As it is seen in the example above, the [] operator written to the right of the variable is the vector index operator of R. There are six types of vectors in R. These vector types, their logical return values and their data storage modes are given in Table 2.2. These vector types can be categorized into three groups generally, and these are widely used in statistical calculations. The vector types used are as below:

- Numeric Vectors and Value Assignment
- Logical Vectors and Value Assignment
- Character Vectors and Value Assignment

Table 2. 2 The modes and storage modes for the different vector types

type	mode	storage.mode
Logical	logical	logical
Integer	numeric	integer
Double	numeric	double
Complex	complex	complex
Character	character	character
Raw	raw	raw

It is, of course, an important problem that how the data would be entered into these vector types mentioned above. The solution to this problem is given in subsections of this chapter.

2.1.1 Numerical Vectors And Assignment

The numerical vectors in R language are vectors in which only numbers are placed. The data can be assignmented into this vector type in a few different ways.

The assignment can be performed using the “c()” function. An example is given below on how the c() function is used in R, and how the data is assignmented to the vector using the function. Let x vector be a numerical vector;

```
>x<-c(1,4,3,5) or
>c(1,4,3,5)->x
```


Both situations are valid for R programming language. Using the code above, 1, 4, 3 and 5 are transferred to the data list of x vector.

If the data to be entered into x vector is a sequential array, R provides convenience. This convenience is presented below.

If the value to be entered in x vector is a sequence, R provides a convenience for this matter. This situation is shown in the example below:

```
>x<-1:5 or x<-seq(5)
>1:5->x or seq(5)->x
```

Numeric vectors are a type of vector which is significant for researches studying computational sciences such as statistics and mathematics; because R has some functions which facilitate the operations on vectors. `sum()`, `mean()`, etc. functions can be given as examples to this function type.

2.1.2 Logical Vectors And Assignment

Logical vectors are important and useful vector type in R. This vector type can only have two values. These values are “TRUE” and “FALSE”. In R, “TRUE” value equals to 1 numerically, and “FALSE” equals to 0. In logical vectors, just as in numerical vectors, the data can be added with “`c()`” function. Below is an example:

```
>x<-c(T,F,T) or
>c(T,F,T)->x
```

In R, logical vectors are generally used to compare one or more variables. If the return value is 0 or FALSE for a vector at the end of the comparison, it means that the compared values are not overlapping.

While entering “TRUE” or “FALSE” values into logical vectors “T” or “F” values can also be used. However, in R, “T” and “F” arguments may be different objects so they can include different values. In such a condition the data entered into the logical vector will not be “TRUE” or “FALSE” (R Core Team, 2009).

2.1.3 Character Vectors And Assignment

Another vector type in R is the character vectors. This vector type may include a string as well as any character of the string. There are several ways to enter data into this vector. One of them is again the “c()” function. Let x vector be a character vector:

```
>x<-c(“a”, “b”, “c”) or x<-c(‘a’, ‘b’, ‘c’)
> c(“a”, “b”, “c”)->x or c(‘a’, ‘b’, ‘c’)->x
```

Single quote or double quote character may be used while entering data into character vector. “c()” function is not compulsory for entering data into the vector. Entering the string will be enough. Single quotes or double quotes should be used while performing this action.

```
>x<-“abc” or x<-‘abc’
> “abc”->x or ‘abc’->x
```

Generally speaking, one should avoid using single quotes while entering data into character vectors. Double quotes are more useful for the researcher for avoiding errors.

There are some mathematical operators in R which can operate on vectors. These operators are given in Table 2.3.

Table 2. 3 The mathematical operators

+: (plus)	Operator of plus (addition)
-: (subtraction)	Operator of subtraction
*: (multiplication)	Operator of multiplication
/: (division)	Operator of division
^: (power)	Operator of power
%/%	Whole number division
%%	Modular arithmetic
%in%	Matching operator

Using the operators in the table, it is possible to carry out operations on vectors, without using the loops. In order to explain the operators in more detail, the examples below are given. Here, let x, y and t be numerical vectors;

(+) operator:

```
>x<-c(1,5,6) and
```

```
>y<-c(8,5,4)
```

```
>t<-x+y
```

```
> t
```

```
[1] 9 10 10
```

%in% operator:

```
>x %in% y
```

```
>FALSE TRUE FALSE
```

The command line usage of the (+) operator is valid for all other operators which are not illustrated here. As it is seen, the results were obtained without using loop commands for the vector sums.

2.2 Matrices and Arrays

One of the most important objects of the R programming language is the arrays. As it is for the vectors, there are three types of arrays. These are;

- Numerical Arrays
- Logical Arrays
- Character Arrays

These arrays have the same features with vectors. The data transfer into these arrays is done using the “c()” function or the other methods mentioned for vectors. There is a relationship between vectors, arrays and matrices. This relationship is shown in Figure 2.1.

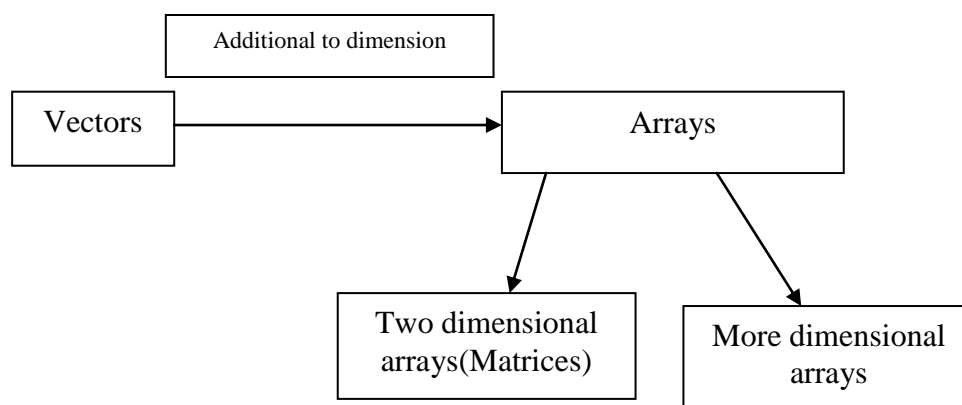


Figure 2.1 Relation between arrays, vectors and matrices.

The most important property that distinguishes arrays from vectors is that vectors do not have a dimension but arrays may have one or more dimensions (Crawley, 2007). This difference will be understood well in R environment. Let x be a vector, and xd be an array with dimension;

```
>x<-1:5
```

```
>xd<-array(1:5, dim=c(1,5))
```

x and xd are now two vectors with the same value. However, the difference of xd vector is understood as below:

```
>is.array(x)
[1] FALSE
```

If the same situation is applied to xd

```
>is.array(xd)
[1] TRUE
```

The difference between arrays and vectors are shown above. A vector cannot be seen as an array, but array may become vectors. Therefore, operators which can operate on vectors may also operate on arrays.

Another important object in R is the matrices. In mathematics, matrices are defined as two dimensional arrays. The data in each row and column of the matrices are in the same storage mode (R Core Team, 2009). In other words, it is not possible for a data to be logical, and another to be character. All the data is either character or logical.

In order to generate a matrix object in R, the `matrix()` function is used. The arguments of this `matrix()` function are as below:

```
>matrix(data, nrow, ncol, byrow=TRUE,...)
```

In this function “...” is a special argument and this will be dealt with in functions section. The generation of a numerical matrix is presented below:

```
>dat<-matrix(c(1:36),nrow=4, ncol=9, byrow=T)
> dat
```

```

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]  1   2   3   4   5   6   7   8   9
[2,] 10  11  12  13  14  15  16  17  18
[3,] 19  20  21  22  23  24  25  26  27
[4,] 28  29  30  31  32  33  34  35  36

```

In mathematics, matrices have some properties. These properties are maintained in R too. Some of these properties are matrix multiplication, transpose, eigenvalue and eigenvector, etc. The matrix multiplication operator in R is “%*%”. The transpose operation in matrices is carried out with the “t()” function. Eigenvalues or eigenvectors are found with the “eigen()” function. The determinant of the matrix is found with the “det()” function.

The points to take into consideration matrix operation are explained here. Let A be a matrix of $n \times y$ dimensions. In this case, in order to carry out $A \%*\% B$, B matrix should be of $y \times z$ dimension. In order to eigen() function to operate, the matrix’ being a square matrix is compulsory. This situation is valid for the “det()” function.

2.3 Lists

One of the most widely used object types in R is the lists. Lists are objects which comprise of the elements of a known object and the object itself. At the same time lists are a special vector type. In R, lists can store data. A simple example on this property is;

```

> mylist <-list(first=c(1,3,5),second=c(“one”, “three”, “five”),
third=matrix(1:6,3))

```

In this way, the elements of the object called mylist includes all arguments and values of the list() function. As it can be seen from this example, mylist object is a list object which can store data.

2.4 Data Frames

Another important object type in R is the data frames. Data frames are another means of storing data in R. Data frames works like matrices. The results of the operations are placed into a data frame's rows. In the columns, on the other hand, the arguments of the operation are placed. If a data frame is required in R, it is generated by `data.frame()` function. A generating data frame can be understood with the example below.

```
>df<-data.frame(isim=c("ali","yeşim","murat",
"hakan","gülay"),yas=c(21,20,22,21,19),
bol=c("tarih","fizik","mat","kim","ist"),puan=c(93,78,88,91,90))
> df
  isim   yas  bol  puan
1  ali   21  tarih  93
2 yeşim  20  fizik  78
3  murat  22   mat  88
4  hakan  21   kim  91
5  gülay  19   ist  90
```

The data frames in R are not used only in this way. The data frames mostly used in research are the ones which are read from external files. In order these data to be read the `read.table()` function is required in R. With this function, data from a data file in any directory of the computer can be transferred into R environment as data frame.

This function can read data from the file extensions below:

- *.dat file extension
- *.txt file extension

The data read from the files with these file extensions are now a data frame in R. With these data, the result inference can be carried out using the related operators and functions. Let us establish the issue with an example. Let there be a data file under the name of “dat.txt” in C section of the computer, and let the data in this file be as below:

x_var	y_var
5	10
8	15
9	17
10	25

The code should be entered into the command line in order to read this file:

```
>dat<-read.table("C:/dat.txt", header=T)
```

```
> dat
```

	x_var	y_var
1	5	10
2	8	15
3	9	17
4	10	25

If the “header” argument in read.table() function is not TRUE, this function would not return the headers x_var and y_var. Instead, R would assign names to these variables. If the dat object, from which the data is transferred, it would be seen that this object is a data frame.

```
>is.data.frame(dat)
```

```
[1] TRUE
```

As it can be seen from the result above, dat object is a data frame. Therefore, the functions required for the inference can be used as it was mentioned before. For

instance, if the summarizing statistics of these two variables, taken as data, are wanted to be calculated, the `summary()` function is needed.

```
>summary(dat)
  x_var      y_var
Min.   :5.00   Min.   :10.00
1st Qu.: 7.25   1st Qu.:13.75
Median : 8.50   Median :16.00
Mean   : 8.00   Mean   :16.75
3rd Qu.: 9.25   3rd Qu.:19.00
Max.   :10.00   Max.   :25.00
```

2.5 Functions

Functions are also important objects in R programming language. Manipulations can be made on functions. They have four basic components:

- Function name
- “function” statement
- Arguments
- Body

Functions will be discussed in detail in following chapters.

2.6 NULL Object

NULL object is generally used for vectors. For a vector, NULL means that the vector does not have any data in it; sayingly, the vector has a length of zero. There are two types of functions in the usage of NULL object. The first one is the `as.null()`, used for rendering the vector “null”. The second one, `is.null()`, returns a logical value for the vector. The usage of NULL object is exemplified below:

```
>x<-as.null(x)
>is.null(x)
[1] TRUE
```

2.7 Environments

Another R object is the “environments”. R work area is a global environment. The objects in this global environment are called the global objects. There is a global object example below. If x object is entered in the global environment,

```
>x<-1.00034^3
```

Here x is a global object, and can be used anywhere in R. R also has functional environments. The objects in this environment are called local or functional objects, and they are only valid in the environments they are in. Functional objects can be placed in the arguments and body components of the function.

```
>envr<-function(x) x^2
```

Examining the example above, x is a functional object, and can only be used in this function. This x object cannot be used in another environment. The content of the envr function is a functional environment. If x object is not defined elsewhere in the global environment, the “can not find symbol” error message is returned when x is entered into the command line. The distinction between global and functional environments are given below with a broad example.

```
>z <- 5
>func1<- function(xx) {
z <- 10
z^2 + func2(xx)
}
>func2 <- function(xx) {
```

```
xx * z
}
```

First of all, the code segment is written in the command line of R. As it can be seen, z object appears in both global and functional, func1 and func2, environments. However, z objects take the value in the environment it appears. In other words, it has the value of 5 for global environment, the value of 10 for func1 environment, and again the global value of 5 in func2, since any value is not assigned to z in func2 environment. Therefore, when the code is run as func2(2), the result below is obtained:

```
>func2(2)
[1] 110
```

2.8 Classes

In R language, classes are also objects. The class object is returned in R user interface as return type. In order to see which class an object or a variable belongs, there is a class() function in R. Classes may return different classes for the different objects present in R. For instance, the return type for data frames is “data frame”, for linear models “lm”, and for matrices “matrix”. For an example about this issue in R;

```
>x<-0; x<-as.matrix(x)
>class(x)
[1] “matrix”
```

As it can be seen in the example above, x object belongs to the matrix class. Put another way, x object has the features of the matrices. Therefore, this object can be assigned the values necessary for matrices, and with these values the result inference can be carried out using operators which operate on matrices.

About objects in R can be finally said that objects limited here. R language has more object types; i.e., pairlist, factors, dot dot dot object, etc. For other object types,

CRAN or documents about R (Crawley, Dalgaard, Braun & Murdoch, ...) are good reference materials.

CHAPTER THREE

FUNCTIONS AND GRAPHICAL PROCEDURES

In this chapter the functions in R and some graphical tools will be explained. Functions are one of the most important objects of R. With this object type, researchers may perform data manipulation more easily. For instance, R has many facilitating functions which can operate on matrices and arrays. Making inferences has become easy with these functions. In fact, many operations and data analyses are conducted by recalling functions in R. For instance, $\log(x)$ is the recall of a function in R, which is required for obtaining the logarithm of the elements of an object.

There are some prerequisites for recalling a built-in function in R. Functions have some certain format, and while recalling a function, this format should be taken into consideration. For instance, $\text{plot}(x,y)$. Here $\text{plot}()$ function is recalled with its arguments and this recall is compatible with the format of the function. Here x and y are vectors.

R statistical programming language has many effective and useful functions for statisticians and mathematicians. These functions can be used efficiently in data analyses by writing them into the command line. Some functions that are useful in data analysis and their meaning are presented in Table 3.1.

Table 3.1 Some mathematical and statistical functions in R

Operation	Meaning
$\text{max}(x)$	maximum value in x
$\text{min}(x)$	minimum value in x
$\text{sum}(x)$	total of all the values in x
$\text{mean}(x)$	arithmetic average of the values in x

Table 3.1 (Continue)

median(x)	median value in x
range(x)	vector of min x and max_ x
var(x)	sample variance of x
cor(x,y)	correlation between vectors x and y
sort(x)	a sorted version of x
rank(x)	vector of the ranks of the values in x
order(x)	an integer vector containing the permutation to sort x into ascending order
quantile(x)	vector containing the minimum, lower quartile, median, upper quartile, and maximum of x
cumsum(x)	vector containing the sum of all of the elements up to that point
cumprod(x)	vector containing the product of all of the elements up to that point
history()	displays previous commands used
is.na()	returns TRUE if vectors have na
length()	number of elements in a vector or of a list
print()	prints a single R object

(R Core Team, 2009)

3.1 Recalling A Built-in Function In R

Recalling a built-in function in R commandline is an important subject. R programming language is case sensitive, since it is an object-oriented language. In other words X and x are different objects in R. As it can be understood from this, a

function called “abc” can be written in 8 different ways. Therefore, in order to run the required function correctly, the function’s name and its arguments should be written properly into the command line. The running of the abc function with its arguments is presented below:

```
>abc<-function(arg_1, arg_2) { body }
```

The abc function is given above. Recall of the function in R command line is given below:

```
>abc( value of arg_1, value of arg_2)
```

The abc function above does not have any other recall type. Put another way, when this function is recalled as Abc or ABC, and if there is not any function with these names, R would return this error message: “can not find function”.

Let us give an example to one of the built-in functions mentioned in Table 3.1. The functions in this table are built-in functions of R. For instance, mean() function is a function that calculates the arithmetic mean and a function in which a compatible argument may added. The example is below:

```
>mean(rnorm(100))
```

Here, rnorm() function, too, is a built-in one. With the function above, a sample of 100 units is generated from the standart normal distribution, and the arithmetic mean of this sample is calculated with mean() function. As a different argument, trimming percentile argument can also be added to the mean() function.

```
>mean(rnorm(100), tr=0.2)
```

As it can be seen in the example above, the trimming ratio is designated with “tr” argument in R.

3.2 Writing Functions

R language is a functional programming language. This programming language has many built-in functions other than the ones given in Table 3.1. However, each researcher should develop original functions and use these functions in data analysis. The elements required for writing a function are presented below:

```
>function_name<-function (arg1, arg2, ..., argn){body}
```

As it can be seen in the example, in order to create a function in R, four basic elements should be paid attention.

First of them is the name of the function. Name of the function is a function element required to recall that function. As it was mentioned before, R is case sensitive. Therefore, this issue should be considered. In other words, function_name element and Function_Name element are different from each other.

The second element of the function is the “function” statement. While defining a function in R, this keyword is used, as it can be seen in the example above.

The third element of a function is the “arguments” part. If there are arguments to be used in the functions, these arguments are written between brackets after the “function” statement. If there is a default value for one of these arguments, this value is assigned into the argument in the arguments part.

The fourth element of a function is the body. The commands to be used in the function are placed in this section. Generally, this section is written between two braces (“{body}”). If the function has a code comprising of one single line, the use of these two braces is not necessary.

Lastly, in this part is performed an example R code including the defining of a function and the recalling the defined function.

With the function written below, the two sample t-test is performed. The basic elements of this function are indicated in the example. The vectors necessary for the function to run are entered.

```
>x<-c(5,6,8,2,3,5); y<-c(8,9,4,5,6,1)
```

Function_name	Argument lists
---------------	----------------


```
> twosam <- function(x1, x2) {
  n1 <- length(x1); n2 <- length(x2)
  xb1 <- mean(x1); xb2 <- mean(x2)
  s1 <- var(x1); s2 <- var(x2)
  s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
  tst <- (xb1 - xb2)/sqrt(s*(1/n1 + 1/n2))
  tst
}
```

Body

The name of the function above is “twosam”. The argument of this function are “x1” and “x2”. The objects and the operation used in the environment of the function, namely in the body of the function, are presented too. This function returns the test statistics “tst” as a result. The code below is needed in order to recall this function in R global environment.

```
> tstat <- twosam(x,y); tstat
```

With the code above, “twosam” function conducts the two sample t-test using the vectors defined above. The result is then transferred to the tstat object, created in R global environment. Later, the “tstat” object is recalled in global environment and the result is presented in the screen.

3.3 Conditional And Repetitive Statements

Generally, conditional and repetitive statements are required when writing a program in a programming language. R programming language too has some conditional and repetitive statements. These statements are used frequently while a researcher writes a function. This section mentions the conditional statements first, and then the repetitive statements.

3.3.1 Conditional Statements

The conditional statement in R language is generally the “if” keyword. Conditional statement is generally used to compare one or more conditions in functions. The working mechanism of this statement is generally the returning of the result, in case some certain condition is met, as TRUE and then running the codes in this conditional statement. Generally, if a single line code is written when the condition is met, the codes are not written in the curly braces (“{ body }”). Below, the working mechanism of this statement is presented:

```

program section      A
                    if (condition ) B
                    D

```

The code presented schematically above is the application style of a condition in the program sections. Program sections comprise of A, B, and D. Section A and D are not conditional, but the running of section B depends on whether the “if (condition)” is TRUE or not. If the return value of the condition is “FALSE” B section would not run and D section would run.

“if” keyword is generally used in a single way. However, sometimes there can be another condition when the first condition is not met. In order to add an extra condition the keyword “else” is used. The use of this keyword is given below:

```

program section      A
                    if (condition ) B else C
                    D

```

Program sections comprise of four sections. These are A, B, C and D sections. A and D are nonconditional sections. B and C sections, in which the “if” and “else” statements are used, are conditional sections. In cases where “if” condition is not realized, the section with “else” condition would run.

There are some operators for the use of conditional statement. These are called the logical and comparison operators. Table 3.2 presents these operators and their meanings.

Table 3. 2 The comparison and logical operators for if conditonal statement

	Operators	Meaning
Logical	&	Logical “and”
		Logical “or”
	!	Logical “not”
	&&	Logical “and”
		Logical “or”
Comparison	<	Less than
	>	Greater than
	==	Equal to
	<=	Less than and equal to
	>=	Greater than and equal to
	!=	Not equal to

Here & and && or | and || are the same operators, and perform the same logical comparisons. However, while & and | operators compare a single cell in a vector versus cells in the other vector, && and || operators compare the whole elements in the vectors (R Core Team, 2009).

3.3.2 Repetitive Statements

While writing codes in programming language, sometimes, some operations are needed to be written many times. In such conditions, some repetitive statements are added to the programming languages in order to facilitate the programmer's job. These statements are generally called loops. Loops help the programmer to perform the jobs repeatedly in a specific number of times. The general working principles of loops are presented below:

```

program section  A
                  loop B
                  C
order of execution A B B ...B C

```

As it can be seen in the mechanism above, loops do not include any conditions as conditional statements. In the example above there are three program sections. These are A, B and C. A and C sections do not include any loops. However, B section has a loop. A programming language using such a loop runs the operation in B and then starts performing the operation in C.

The mechanism above shows the running of a loop in R. There are three different loops in R programming language. These are;

- for loop
- While loop
- Repeat loop (Braun & Murdoch, 2007)

3.3.2.1 The for Loop

The “for” loop is the loop command, that enables the same operations to be performed with a sequence, which is defined specifically. The structure of a “for” loop is presented below.

for (loopvariable in sequence) expression1

Here, the loopvariable is the variable required to enable the loop. This variable increases or decreases in the sequence and controls the progression. At the same time, it controls the end of the loop, when the end of the sequence is reached. Sequences may be increasing or decreasing arrays. Here, the important point is that the sequence should include sequential values such as 1, 2, 3, ..., 10. An example on the use of a “for” loop is given below:

> for (i in 1:4) print(i)	or	>for (i in 4:1) print(i)
[1] 1		[1] 4
[1] 2		[1] 3
[1] 3		[1] 2
[1] 4		[1] 1

In the example above the loopvariable is i, and 1:4 and 4:1 are the sequences.

3.3.2.2 *The while And repeat Loop*

R programming language has two other loop statements except for “for” loop. These are the loops performed with while and repeat keywords.

“While” loop includes a condition differently from “for” loop. Using this condition it creates a loop. However, “while” loop controls this condition in the body section of the loop. The working mechanism of a “while” loop is presented below:

```
while (condition) { (control of the condition) expr }
```

If a while loop is set during program writing, the operations in the body section would continue until the condition is returned “FALSE”. If the condition is not controlled in the body, an infinite loop would be formed. This is explained in the example below:

```
>i<-0  
>while (i<5) print(i)
```

The loop above is an infinite loop, because it is not controlled anywhere in the code.

```
>i<-0  
>while (i<5) {  
  i<-i+1  
  print(i)  
}
```

In this loop, on the other hand, the result will be displayed as 2, 3, 4 and 5.

Another loop can be created in R, apart from while and for, with the repeat keyword. Repeat loop, similar to while loop, requires control in the body section. There is a small difference between repeat loop and while loop in terms of the position of the condition. The working mechanism of repeat loop is given below:

```
repeat {expr  
  (condition) break }
```

As it can be seen in the example above, differently from while loop, in repeat loop the conditional statement places in body part. For this condition another keyword, “break”, should be used. If the keyword “break” is not used, an infinite loop would be created. The example above for the while loop can be written for repeat loop as below:

```
>i<-0  
>repeat {  
  i<- i+1  
  print(i)
```

```
if (i==5) break }
```

Below is an example of using while loop and repeat loop in a function. The factors of a number are found in this example.

```
fac<-function(x) {
f <- 1
t <- x
while(t>1) {
f <- f*t
t <- t-1 }
return(f) }

fac<-function(x) {
f <- 1
t <- x
repeat {
if (t<2) break
f <- f*t
t <- t-1 }
return(f) }
```

If in this part is writed a function that performs inference with twosam function mentioned before, the “if” keyword should be used. Below is an example that performs inference with twosam function:

```
> twosam <- function(x1, x2, alpha=0.05) {
n1 <- length(x1); n2 <- length(x2)
xb1 <- mean(x1); xb2 <- mean(x2)
s1 <- var(x1); s2 <- var(x2)
s <- ((n1-1)*s1 + (n2-1)*s2)/(n1+n2-2)
tst <- (xb1 - xb2)/sqrt(s*(1/n1 + 1/n2))
crit_point<-qt(1-alpha/2,n1+n2-2)
if (abs(tst) > crit_point) {
print(“ null hypothesis is rejected”)
} else { print(“null hypothesis is not rejected”)}
}
```

In order to perform the inference in the R command line, a code similar to the one below would be sufficient:

```
> twosam(rnorm(15),rnorm(15,mean=2))  
[1] "null hypothesis is rejected"
```

3.4 Simulation In R Programming Language

Simulation study is one of the most frequently used research and inference methods in applied sciences. Conducting simulation studies manually takes too much time. Therefore, software packages or programming languages try to offer help to researchers. R programming language has functions and packages which facilitates the researcher in simulation studies.

A simulation study can be divided into three phases:

- Obtaining the data to be simulated
- Determining and the analysis of the quantities the researcher is interested in (determining the rejection area, estimation of parameters, model estimation, etc.)
- Results from the first two phases and the interpretation of these results.

This section will explain how the data required for the simulation could be obtained in R programming language.

3.4.1 Simulation Tools In R Programming Language

One of the most important topics in R for a statistician or a researcher is the use of the simulation tools in R. R programming language has some tools for performing simulation research easily. For instance, the generation of the distribution from which the sample would be selected can be performed very easily in R. R can generate data from almost all statistical distributions. This issue is explained in detail in the following section.

Prefix the name given here by ‘d’ for the density, ‘p’ for the CDF, ‘q’ for the quantile function and ‘r’ for simulation (random deviates). The first argument is x for dxxx, q for pxxx, p for qxxx and n for rxxx (except for rhyper and rwilcox, for which it is nn). In not quite all cases is the non-centrality parameter ncp are currently available: see the on-line help for details (R Core Team, 2009, p.39).

Depending on the explanations above, if researcher is to generate data from a uniform distribution, the code below should be examined:

```
>runif(n=15, min=2, max=5)
```

With this code segment 15 data set with 2 as minimum value and 5 as maximum value is derived. The arguments of the runif() functions are n which is the size of the sample, min and max values.

3.4.2 The Bootstrap Resampling Method

Deriving samples from a population is an important issue in simulation. The following paragraph the bootstrap resampling method is explained.

One practical alternative, known as the bootstrap, is to treat the original sample of values as a stand-in for the population and to resample from it repeatedly, with replacement, computing the desired estimate each time (Good, 2006, p.21).

Below are the codes required for deriving a sample using the bootstrap sampling method in R:

```
> bootsamp<-function(){
  x<-rnorm(2000,5,2)
  for (i in 1:100){
    samplex<-matrix(sample(x,20*599,replace=T),nrow=599)
  }
}
```

In this example, there is a population with a mean of 5 and standard deviation of 2 with normal distribution. From this population, 599 samples of size 20 are derived by replacement. This deriving operation is repeated 100 times with a “for” loop and 100 bootstrap resampled samples are obtained. The sample derivation operation in the example above is performed by assigning “TRUE” value to the replace argument via the function below. The sample() function in R is given below without changing the default values.

```
>sample(x, size, replace=FALSE)
```

With the function above a sample of “size” size is derived from the population x without replacing.

3.5 Graphical Functions In R

It was mentioned before that R programming language has many built-in functions. Most of the built-in functions in this language are graphical. R language has many graphical tools. Not all of them are mentioned in this study, but these graphical tools limited to the confines of this study.

The graphical tools of R language comprises of two different function groups. These are called high-level and low-level functions (Murrel, 2006). R programming language has some arguments to increase the competence of high-level functions in analyses.

3.5.1 Arguments Of The High Level Graphical Functions

The arguments of the high-level functions used in R are given in Table 3.3.

Table 3. 3 Arguments of the high level graphical procedures

axes=FALSE: Suppresses generation of axes—The default, axes=TRUE, means include axes.

Table 3. 3 (Continue)

log="x"	
log="y"	
log="xy"	Causes the x, y or both axes to be logarithmic. This will work for many, but not all, types of plot.

type=	The type argument controls the type of plot produced, as follows:
type="p"	Plot individual points (the default)
type="l"	Plot lines
type="b"	Plot points connected by lines (both)
type="o"	Plot points overlaid by lines
type="h"	Plot vertical lines from points to the zero axis (high-density)
type="s"	
type="S"	Step-function plots. In the first form, the top of the vertical defines the point; in the second, the bottom.
type="n"	No plotting at all. However axes are still drawn (by default) and the coordinate system is set up according to the data. Ideal for creating plots with subsequent low-level graphics functions.

xlab=string	
ylab=string:	Axis labels for the x and y axes. Use these arguments to change the default labels, usually the names of the objects used in the call to the high-level plotting function.

main=string	Figure title, placed at the top of the plot in a large font.
sub=string	Sub-title, placed just below the x-axis in a smaller font.

These arguments are frequently used in high-level functions. They are informing arguments which inform the researcher in the course of his/her study.

3.5.2 Low Level Graphical Functions

In some cases, high-level functions can not be fully explanatory. In such cases, low-level functions are used to give extra information after the use of high-level

functions. For instance, low-level functions such as points, lines, texts, etc. can be used in such cases. Table 3.4 presents some of the low-level functions of R language.

Table 3. 4 Some low level graphical functions

points(x, y),lines(x, y):	Adds points or connected lines to the current plot. plot()'s type argument can also be passed to these functions (and defaults to "p" for points() and "l" for lines().)
text(x, y, labels, ...):	Add text to a plot at points given by x, y. Normally labels is an integer or character vector in which case labels[i] is plotted at point (x[i], y[i]). The default is 1:length(x). Note: This function is often used in the sequence
polygon(x, y, ...):	Draws a polygon defined by the ordered vertices in (x,y) and (optionally) shade it in with hatch lines, or fill it if the graphics device allows the filling of figures.
title(main, sub):	Adds a title main to the top of the current plot in a large font and (optionally) a sub-title sub at the bottom in a smaller font.
segments(x0, y0, x1, y1, ...):	draws line segments
arrows(x0, y0, x1, y1, ...):	draws arrows
symbols(x, y, ...):	draws circles, squares, thermometers, etc.

3.5.3 High Level Graphical Functions

R programming language has high-level graphical functions to enable the researcher to see which properties the data have in statistical analyses. These procedures in R language are originated from the Sussman's Scheme language (R Core Team, 2009). Table 3.5 below presents the graphical procedures frequently used in statistical research.

Table 3. 5 High level graphical functions of R programming language

plot(x)	plot of the values of x (on the y-axis) ordered on the x-axis
plot(x, y)	bivariate plot of x (on the x-axis) and y (on the y-axis)
pie(x)	circular pie-chart

Table 3.5 (Continue)

boxplot(x)	“box-and-whiskers” plot
dotchart(x)	if x is a data frame, plots a Cleveland dot plot (stacked plots line-by-line and column-by-column)
hist(x)	histogram of the frequencies of x
barplot(x)	histogram of the values of x
qqnorm(x)	quantiles of x with respect to the values expected under a normal law

R also has many high-level functions not present in the table above. Help about these functions can be obtained from the R website on the internet.

3.5.3.1 Bar Charts And Dot Charts

Bar charts are the statistical graphics in which each data range is represented with a bar. If the data set at hand is a cluster data set, the bar chart is quite efficient. This function has some arguments which can compare each set with another set or the elements in the set with each other. There is an example below about barplot and its graphical displaying is given in Figure 3.1.

```
>aa<-matrix(c(1:24),nrow=4);barplot(aa,beside=T,main= “example of barplot”)
```

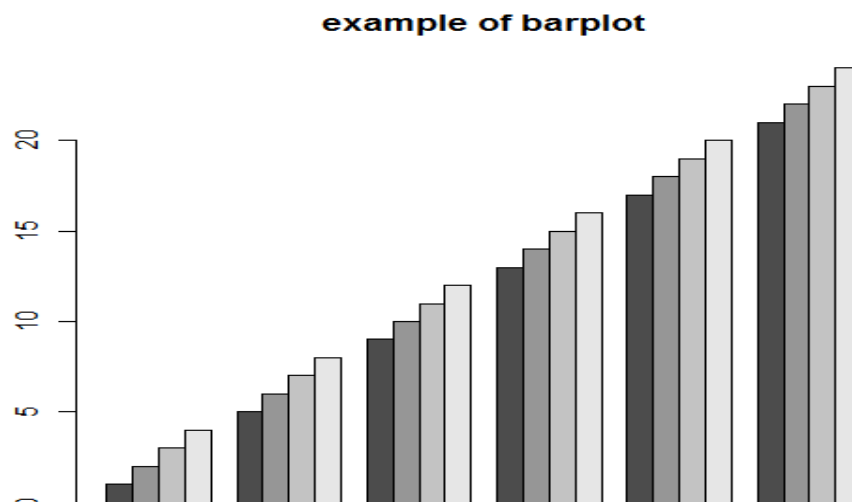


Figure 3. 1 Bar chart for example

At the results in Figure 3.1, due to “beside” argument taking “TRUE” value, a inset drawing is present in the graphic. At the same time, a main title is given to the graphic with the high-level function argument “main”.

The dotchart can be used as an alternative to barchart. It is a graphical displaying which draws dots on a simple scale using the data. The graphical displaying of the aa matrix above is given in dotchart in Figure 3.2.

```
>dotchart(aa, main="example of dotchart")
```

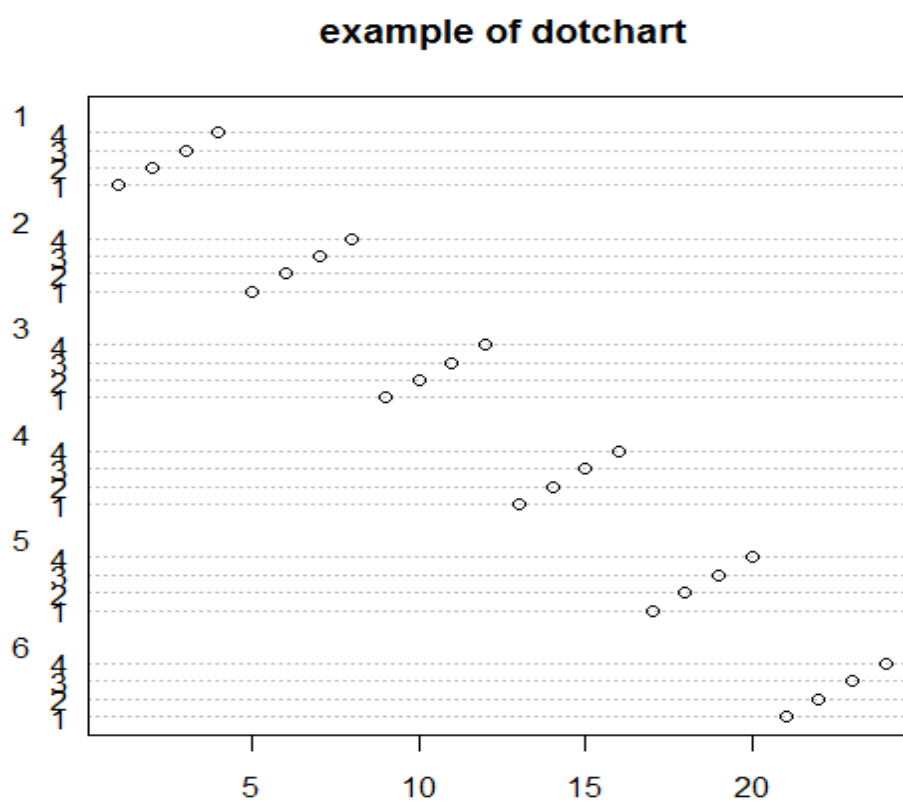


Figure 3. 2 Dot chart for example

Barchart has a histogram displaying, while dotchart has a pointwise one.

3.5.3.2 Pie Charts

Pie charts are used to show the proportion of each value to the whole. The code below shows how a pie chart is generated in R.

```
>pie.gra<- c(0.11, 0.3, 0.3, 0.10, 0.04, 0.12);names(pie.gra) <- c("Banana",
"Cherry", "Apple", "Cococa", "Other", "Vanilla"); pie(pie.gra,main="Pie chart for
example")
```

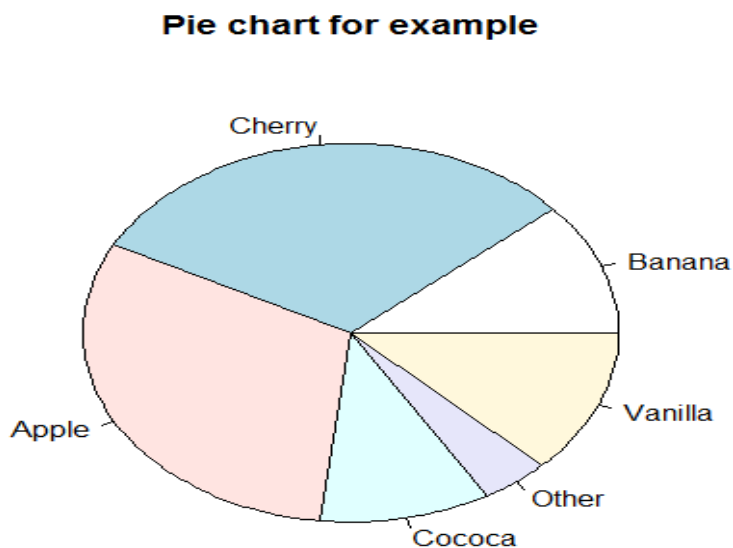


Figure 3. 3 Pie chart of R programming language

3.5.3.3 Histograms

Histograms are barcharts that display the distribution of an individual measure taken from a part or a process. It is also called the frequency distribution, since it shows the frequency of a value with the length of the bar. Histogram is a graphical tool frequently used by statisticians in data analysis. Below is the function in R to draw a histogram:

```
>hist(x,...)
```

For instance:

```
>x<-rnorm(100);hist(x)
```

Histogram is a specialized version of the barchart which shows the distribution of the frequency of the numbers in the data. Each bar includes the frequency of the values which are defined at a certain range in the data.

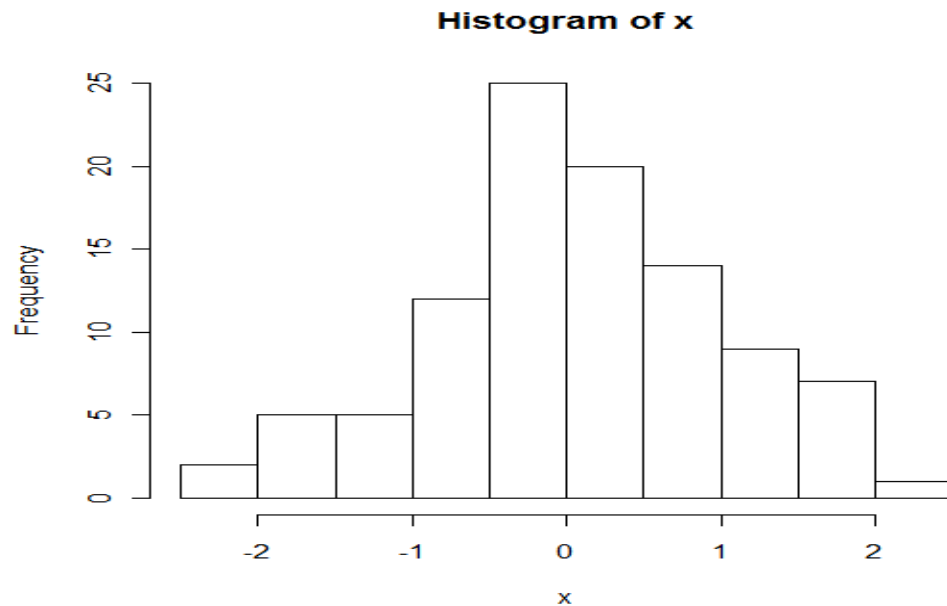


Figure 3. 4The histogram of R programming language

3.5.3.4 Box Plot

Box plot depends on summarizing graphically the 5-tuple representation prepared for the data in terms of the related variable. Especially, it is used to summarize the data in terms of the central position, spread, skewness and kurtosis and to define the skewed values. These five values are given below:

- X_{\min} : minimum of the sample “minimum observation value”
- Q_1 : Quarter “first of lowerquarter”
- X_{med} : Median “median or mean of sequential data”
- Q_3 : Quarter “third or upper quarter”
- X_{maks} : maximum of sample “maximum observation value”

Box plot can be used as an alternative to histogram graphic. A box-plot example is given below:

```
>d<-rnorm(1000,5,2)
>boxplot(d)
```

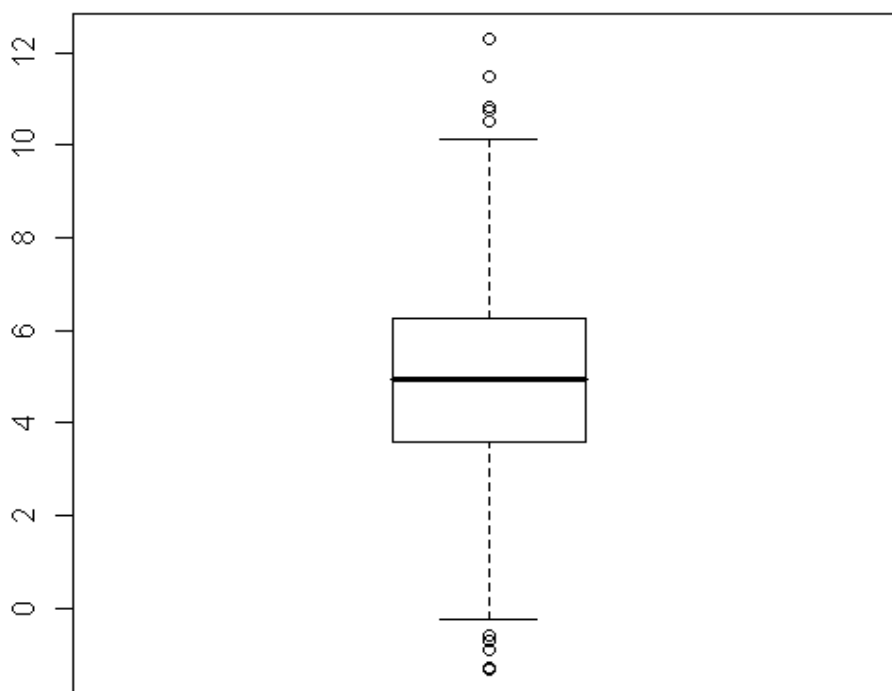



Figure 3. 5Box plot of R programming language

3.5.3.5 Scatter Plot

Scatter plot graphic is a graphical tool that shows the distribution of the data set in dots. It is a frequently used method in statistical data analyses. Below is the scatter plot function in R:

```
>plot(x,...)
```

The example below shows how the scatter plot function is used in R. At the same time an argument is added to the function.

```
>d<-rnorm(1000,5,2);d1<-rnorm(1000,5,2)
>plot(d,d1);lines(d,d1,col="grey40")
```

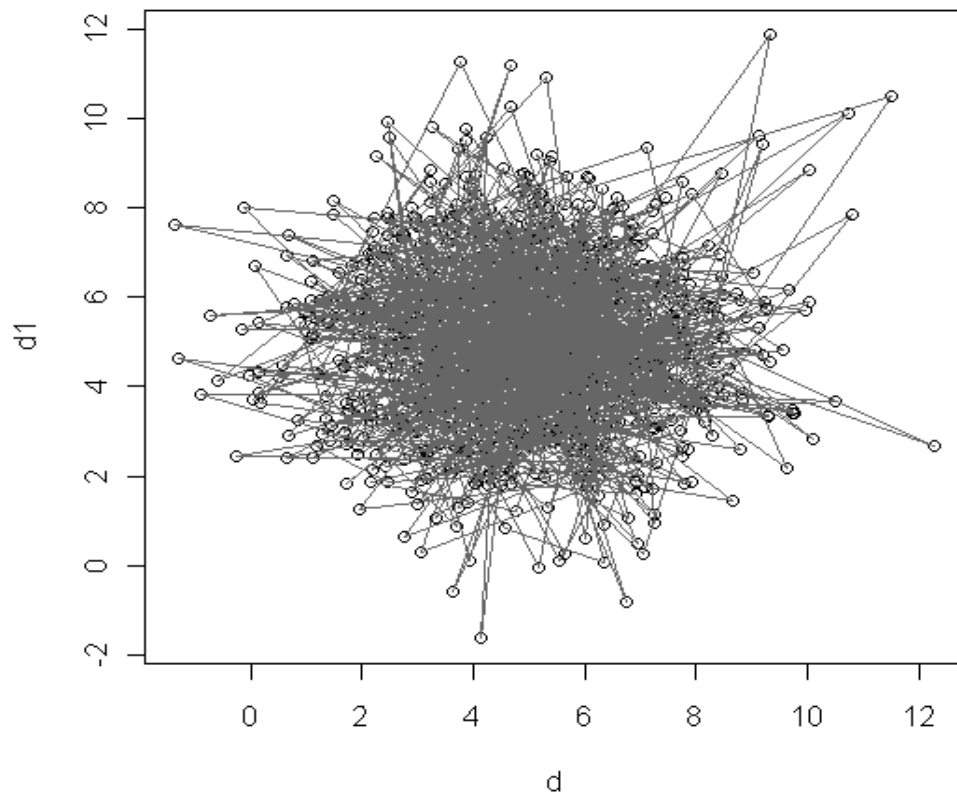


Figure 3. 6 Scatter plot normal versus poisson distribution

3.5.3.6 Q-Q Norm

Q-Q norm is one of the best ways that can show the researcher whether the data at hand is distributed normally or not. The function draws the quantiles of the elements in the sample taken against the theoretical quantiles of the normal distribution. This function has the arguments below, in its general usage in R:

```
>qqnorm(x,...)
```

The code below can be given as an example to Q-Q norm.

```
>d<-rnorm(1000,5,2); qqnorm(d)
```

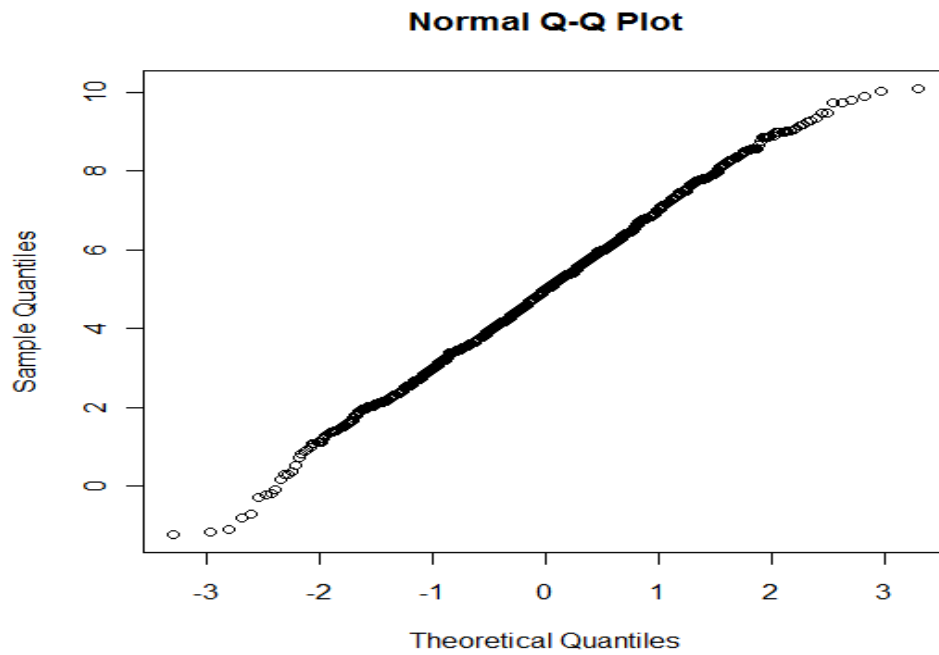


Figure 3.7 Q-Q plot of R programming language

The sections above tried to give examples to the high-level graphical functions of R while providing examples to the use of the arguments. Below is an example in which low-level and high-level functions are used together with the arguments.

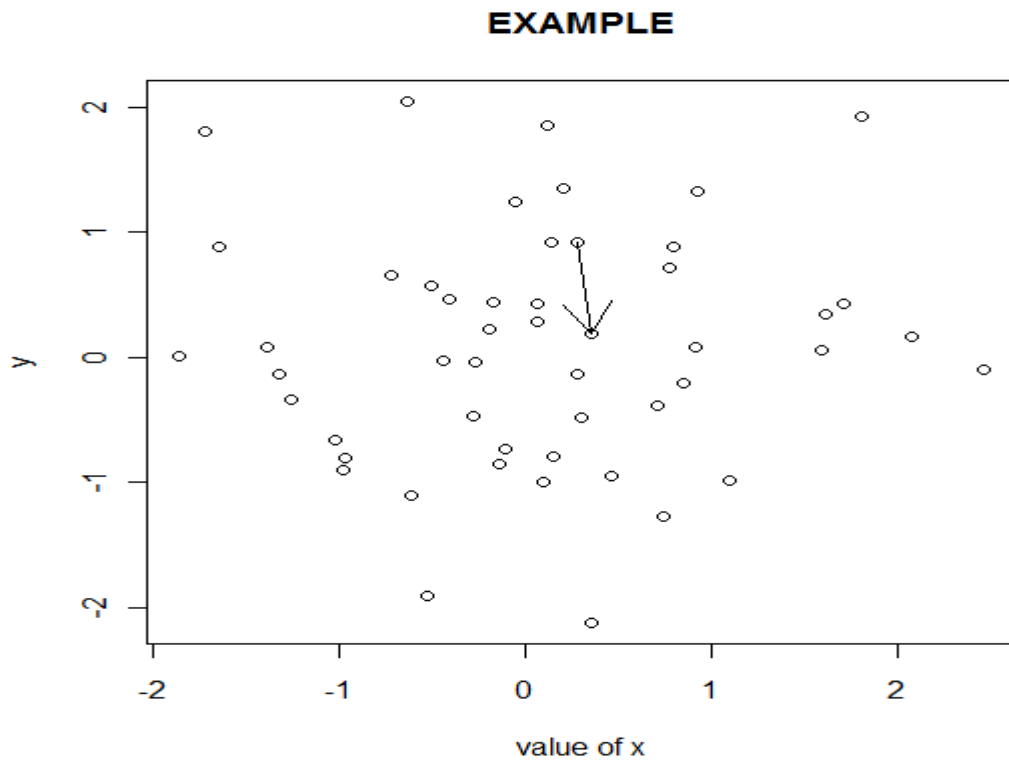


Figure 3.8 General example of graphical procedure of R language

```
>x<-rnorm(50);y<-rnorm(50);plot(x,y, main="EXAMPLE", xlab="value of
x",axes=TRUE);arrows(x[2],y[2],x[5],y[5])
```

In the previous sections the procedures used in data analysis, such as barchar, box-plot, etc., were mentioned. The important point here is the choice of the graphical tools. This depends on the choices of the researcher.

3.6 Special Argument “...”

“...” argument is one of the object types in R. This argument is used mostly in function environments, so it is mentioned in this section. The functions in which the “...” argument is used can be added different arguments by the researcher. “...” is used in the argument part of the functions and is added to the appropriate functions in the body part of that function. The pairing of the function used should be paid attention. Otherwise, an error message may return. A simple example about the usage of this argument is given below:

```
>grap<-function(x,...){drw<-hist(x,...);return(drw)}
```

Here, “...” is added as an argument to grap function, and in the body section “...” argument is also used in hist() function. In this case, examining the code below, the “main” and “col” arguments are added to the hist() function by means of “...” argument.

```
> grap(rnorm(100),main= “dot dot dot example”, col= “black”)
```

If the code above and Figure 3.9 below are examined, it can be seen that more than one compatible argument may be added in the area in which this object type is used.

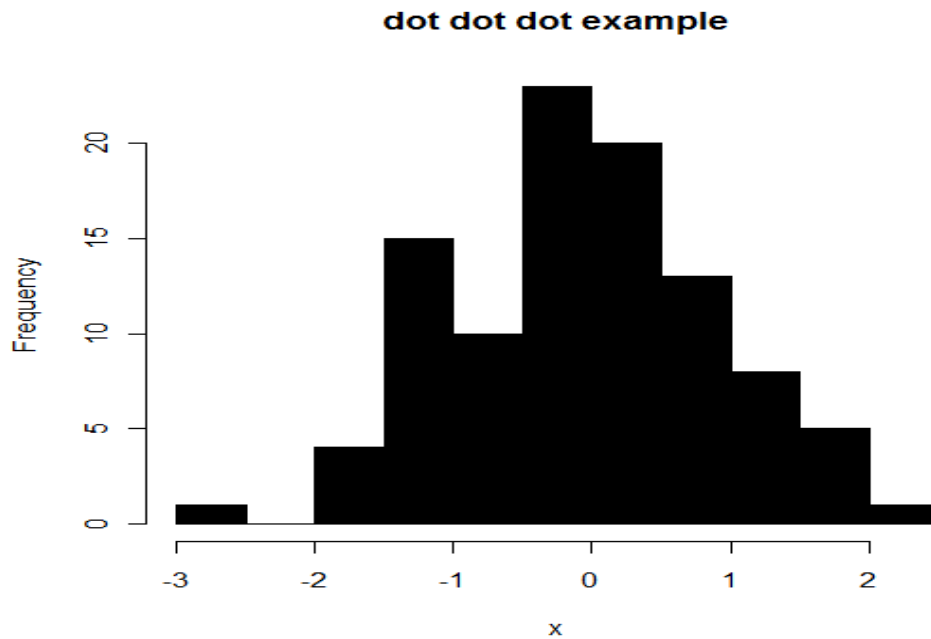


Figure 3.9 Dot Dot Dot example of R language

When Figure 3.9 is examined, it can be seen that two arguments are added to the `hist()` function by means of “...” argument. “...” object type’s being a special object type in R can be understood from the example above.

Lastly, this section will mention the slow operational speed of loops in R. Therefore, they can lower the efficiency of the functions in which they are used. However, there are some cases in which loop usage is inevitable. The examples below are the evidence to the inefficiency of loops in R:

```
>dat<-rnorm(100000000)
> system.time(max(dat))
> pc<-proc.time()
> max<-dat[1]
> for (i in 2:100000000)
+ if (max<dat[i]) max<-dat[i]
> proc.time()-pc
```

With the codes above the operation of finding the max value in the “dat” data set. The total wait time for the first code written in R is obtained by the `system.time()` function and found 1.27 minutes. In the second code, on the other hand, the max

finding operation is performed using the loops and the total wait time lasts more than 7 minutes.

As it can be seen from the examples above, loops in R are not efficient, especially in large data sets. In this case, researcher may try logical methods for comparison and loops. Below is an example of performing logical comparison without using “if” keyword. The sample code which finds the elements smaller than 5 in x vector is given below:

```
>x<-3:8  
>length(x[x<5])
```

The operation performed by the code above can be rewritten using the “if” keyword, but the code above is faster in terms of operational efficiency.

CHAPTER FOUR

AN APPLICATION WITH R: B^2 TEST WITH ONE-STEP M ESTIMATOR AND BOOTSTRAP-T METHOD

The previous chapters discussed the definition, history of R, its functions used on arrays and matrices, and the graphical properties of this programming language. The differences of this programming language from other languages were also mentioned. The specific arguments and statements used in R were introduced and examples on these were provided in order to intensify the use of these arguments, statements and functions.

This chapter presents a statistical simulation study which uses the aforementioned features of R and its means. The hypotheses that normal distribution and population variance being homogeneous are prerequisites in all parametric methods that compare two or more groups using a measure of location. However, in application there are many cases in which these hypotheses are not met (Wilcox, 2005). In this chapter of the study, the R function written for the B^2 test with one-step M-estimator and bootstrap-t method, which are influenced less from the hypothesis violations and the resulting extreme values, is introduced. By the function written, this method is compared to the Yuen trimmed mean test and t-test, which are used in the literature for the same purposes, in terms of maintaining type I error on different sampling orders.

4.1 M-Estimators And One-Step M-Estimator

When searching for a measure of location, one strategy is to use some value, say c , that is in some sense close, on the average, to all the possible values of the random variable X . One way of quantifying how close a value c is from all possible values of X is in terms of its expected squared distance from c . If c intended to characterize the typical subject under study, a natural approach is to use the value c that minimizes $E(X - c)^2$. Viewing $E(X - c)^2$ as a function of c , the value of c

minimizing this function is obtained by differentiating, setting the result equal to 0, and solving for c .

Let $\xi(X - \mu_m)$ be some function that measures the distance between X and some unknown constant μ_m , and let ψ be its derivative with respect to μ_m . Attention is restricted to those functions for which $E[\xi(X - \mu_m)]$, viewed as a function of μ_m , has a derivative.

A general approach to defining a measure of location is to take μ_m to be the value that minimizes $\xi(X - \mu_m)$. So in general, μ_m satisfies

$$E[\psi(X - \mu_m)] = 0.$$

When $\xi(X - \mu_m) = (X - \mu_m)^2$, $E[\psi(X - \mu_m)] = 0$ and $\mu_m = \mu$, the population mean. Various choices for ξ and ψ have been proposed. Here we focus on Huber's well known choice for ψ (Huber, 1981).

$$\psi(x) = \max\{-K, \min(K, x)\} \quad (1)$$

A common choice for K is 1.28, the 0.9 quantile of the standard normal distribution, and this will be used unless stated otherwise. In a given situation some other choice might be more optimal, but $K=1.28$ guards against relatively large standard errors while sacrificing very little when sampling from normal distribution. For a detailed discussion about choosing K , see Huber (1981). Estimation of Huber's M-measure of location is usually accomplished with an iterative estimation procedure such as the Newton-Raphson method. Even with only a single iteration, the resulting estimator has good asymptotic properties (Serfling, 1980). One iteration of this procedure yields the one-step M-estimator

$$\hat{\mu}_m = \frac{1.28MADN(i_2 - i_1) + \sum_{i=i_1+1}^{n-i_2} X_{(i)}}{n - i_1 - i_2}, \quad (2)$$

where

$$MADN(\omega_N) = MAD/0.6745,$$

$$MAD(\omega) = MED\{|X_1 - M|, |X_2 - M|, \dots, |X_n - M|\}$$

is the median absolute deviation statistic and M is the usual median. Here i_1 is the number of observations X_i such that

$$\frac{X_i - M}{MADN} < -1.28 \quad (3)$$

and i_2 is the number of observations X_i such that

$$\frac{X_i - M}{MADN} > 1.28 \quad (4)$$

A fundamental difference between the one-step M-estimator and trimmed mean is that an M-estimator empirically determines the amount of trimming whereas a trimmed mean is based on a predetermined amount of trimming. A seemingly natural appeal of the one-step M-estimator is that if sampling is from a light tailed distribution, it might be reasonable to trim very few observations or none at all. If a distribution is skewed to the right, a natural reaction is to trim more observations from the right versus the left tail of the empirical distribution.

M-estimators have highest possible breakdown point 0.5 which make them very insensitive to outliers (Wilcox, 2001).

4.2 Estimating The Standard Error Of $\hat{\mu}_m$

The influence function of M-estimator which measures the relative extent a small perturbation in underlying distribution has on μ_m , has somewhat complicated form. It depends in part on the measure of scale used in ψ here this is $MADN(\omega_N)$ and

it's influence function. The influence function of ω_N is just the influence function of $MAD(\omega)$ divided by 0.6745. Let

$$A(X) = \text{sign}(|X - \theta| - \omega)$$

where θ is the population median. Let

$$B(X) = \text{sign}(X - \theta)$$

and

$$C(X) = A(X) - \frac{B(X)}{f(\theta)} \{f(\theta + \omega) - f(\theta - \omega)\}$$

The influence function of ω_N is

$$IF_{\omega_N}(X) = \frac{C(X)}{2(0.6745)\{f(\theta + \omega) + f(\theta - \omega)\}}. \text{(Wilcox, 2005)}$$

Estimating $IF_{\omega_N}(x_i)$, the value of the influence function of ω_N at x_i , requires an estimate of the probability density function $f(x)$. Here, adaptive kernel estimator was used. Denoting the estimate of the probability density function $f(x)$ with $\hat{f}(x)$ and computing

$$\hat{A}(x_i) = \text{sign}(|x_i - M| - MAD),$$

$$\hat{B}(x_i) = \text{sign}(x_i - M),$$

$$\hat{C}(x_i) = \hat{A}(x_i) - \frac{\hat{B}(x_i)}{\hat{f}(M)} \{\hat{f}(M + MAD) - \hat{f}(M - MAD)\},$$

an estimate of $IF_{\omega_N}(x_i)$ is

$$V_i = \frac{\hat{C}(x_i)}{2(0.6745)\{\hat{f}(M + MAD) + \hat{f}(M - MAD)\}}.$$

Letting $Y = (X - \mu_m)/\omega_N$, the influence function of μ_m is

$$IF_m(X) = \frac{\omega_N \psi(Y) - IF_{\omega_N}(X) \{E(\psi'(Y)Y)\}}{E[\psi'(Y)]}$$

Since the estimate of $IF_{\omega_N}(x_i)$ and ω_N is known, all that remains when estimating $IF_m(x_i)$ is estimating $E[\psi'(Y)Y]$ and $E[\psi'(Y)]$ where $Y = (X - \mu_m)/\omega_N$. Set

$$y_i = \frac{x_i - \hat{\mu}_m}{MADN},$$

and

$$D_i = \begin{cases} 1, & \text{if } |y_i| \leq K \\ 0, & \text{otherwise.} \end{cases}$$

Then, $E[\psi'(Y)]$ is estimated with

$$\bar{D} = \frac{1}{n} \sum_{i=1}^n D_i$$

Finally, $E[\psi'(Y)Y]$ is estimated with

$$\bar{C} = \frac{1}{n} \sum_{i=1}^n D_i y_i$$

The sum in this last equation is just the sum of the y_i values satisfying $|y_i| \leq K$. The value of the influence function of μ_m , evaluated at x_i , is estimated with

$$U_i = \{(MADN)\psi(y_i) - V_i \bar{C}\} / \bar{D}.$$

Then, the squared standard error of $\hat{\mu}_m$ can be estimated as follows.

$$\hat{\sigma}_m^2 = \frac{1}{n(n-1)} \sum_{i=1}^n U_i^2. \quad (\text{Wilcox, 2005}) \quad (5)$$

The computations described here are tedious so a built in R function “mestse” given by (Wilcox, 2005) was used with the simulation study.

4.3 B^2 Test With One-Step M-Estimator And Bootstrap-t Method

This test uses one-step M-estimator and bootstrap-t procedure with the method derived by Ozdemir and Kurt (2006).

Let X_{1j}, \dots, X_{n_jj} be a random sample from the j th distribution ($i=1, \dots, n_j; j=1,2$) and let $\hat{\mu}_{mj}$ be one-step M-estimator of group j as given in Eq (2). The squared standard error of the sample one-step M-estimator for group j ($j=1,2$) is estimated as in Eq (5).

$$\hat{\sigma}_{mj}^2 = \frac{1}{n_j(n_j-1)} \sum_{i=1}^{n_j} U_i^2$$

Let

$$\omega_j = \frac{\frac{1}{\hat{\sigma}_{mj}^2}}{\frac{1}{\hat{\sigma}_{m1}^2} + \frac{1}{\hat{\sigma}_{m2}^2}},$$

$$X^+ = \omega_1 \hat{\mu}_{m1} + \omega_2 \hat{\mu}_{m2}$$

and

$$T_j = \frac{\hat{\mu}_{mj} - X^+}{\hat{\sigma}_{mj}}$$

A normalizing transformation is applied to T_j so that it has, approximately, a standard normal distribution. There are several normalizing transformations for the t statistics in the literature. Here, the method derived by Bailey (1980) is used. Here, the transformation is

$$z_j = \pm \frac{4\nu_j^2 + \frac{5(2z_c^2 + 3)}{24}}{4\nu_j^2 + \nu_j + \frac{(4z_c^2 + 9)}{12}} \nu_j^{1/2} \left\{ \ln \left(1 + \frac{T_j^2}{\nu_j} \right) \right\}^{1/2} \sim N(0,1)$$

where $\nu_j = h_j - 1$, $h_j = n_j - i_1 - i_2$. i_1 and i_2 were given in Eq(3) and Eq(4) respectively and should be calculated for each sample separately. z_c is the critical value of related significance level under standard normal distribution. The sign of

z_j transformation is the same as T_j . Then the test statistics of the proposed method under H_0 is

$$B_m^2 = \sum_{j=1}^2 z_j^2 = \sum_{j=1}^2 \left(\frac{4v_j^2 + \frac{5(2z_c^2 + 3)}{24}}{4v_j^2 + v_j + \frac{(4z_c^2 + 9)}{12}} v_j^{1/2} \ln \left[1 + \frac{\left(\frac{\hat{\mu}_{mj} - X^+}{\hat{\sigma}_{mj}} \right)^2}{v_j} \right] \right)^{1/2} \quad (6)$$

To decide whether to accept or reject $H_0 : \mu_{m1} = \mu_{m2}$. The bootstrap-t procedure is used with the following steps;

1. Let $X_{1j}^*, \dots, X_{n_jj}^*$ be a bootstrap sample of size n_j generated from X_{1j}, \dots, X_{n_jj} by random sampling with replacement and set $C_{ij}^* = X_{ij}^* - \hat{\mu}_{mj}$, $i = 1, \dots, n_j$.
2. Using the $C_{ij}^* = X_{ij}^* - \hat{\mu}_{mj}$, $i = 1, \dots, n_j$ values just obtained compute $\hat{\mu}_{mj}^*$, $\hat{\sigma}_{mj}^{*2}$, ω_j^* , X^{*+} , T_j^* and

$$B_m^{*2} = \sum_{j=1}^2 z_j^{*2} = \sum_{j=1}^2 \left(\frac{4v_j^2 + \frac{5(2z_c^2 + 3)}{24}}{4v_j^2 + v_j + \frac{(4z_c^2 + 9)}{12}} v_j^{1/2} \ln \left[1 + \frac{\left(\frac{\hat{\mu}_{mj}^* - X^{*+}}{\hat{\sigma}_{mj}^*} \right)^2}{v_j} \right] \right)^{1/2}$$

3. Repeat first two steps B times, yielding $B_{m1}^{*2}, \dots, B_{mB}^{*2}$. B=599 appears to suffice in most situations when $\alpha = 0.05$.
4. Put $B_{m1}^{*2}, \dots, B_{mB}^{*2}$ values in ascending order, yielding $B_{m(1)}^{*2} \leq \dots \leq B_{m(B)}^{*2}$. The $B_{m(B)}^{*2}$ values provide an estimate of the distribution of B_m^{*2} defined in Eq (6).

5. Set $l = \alpha B/2$, rounding to the nearest integer, let $u = B - l$ and chose $B_{m(l+1)}^{*2}$,

$$B_{m(u)}^{*2}.$$

6. Reject H_0 if $B_m^2 < B_{m(l+1)}^{*2}$ or $B_m^2 > B_{m(u)}^{*2}$.

4.4 Yuen's Test

Let $X_{1j}, \dots, X_{n_j j}$ be a random sample from the j th distribution ($i = 1, \dots, n_j; j = 1, 2$).

Yuen's procedure tests $H_0: \mu_{t1} = \mu_{t2}$ the hypothesis of equal population trimmed means, using the test statistic

$$T_y = \frac{\bar{X}_{t1} - \bar{X}_{t2}}{\sqrt{d_1 + d_2}} \quad (7)$$

assuming that the null distribution of T_y is a Student's t distribution with a degrees of freedom

$$v_y = \frac{(d_1 + d_2)^2}{d_1^2/(h_1 - 1) + d_2^2/(h_2 - 1)}. \quad (8)$$

and

$$\bar{X}_{tj} = \frac{1}{h_j} \sum_{i=g_j+1}^{n_j-g_j} X_{(i)j} \quad (9)$$

is the j th sample's trimmed mean and $X_{(1)j} \leq X_{(2)j} \leq \dots \leq X_{(n_j)j}$ represent the ordered observations associated with the j th group. Let $g_j = \lfloor \gamma n_j \rfloor$ indicate that γn_j is rounded down to the nearest integer and γ represents the proportion of observations that are to be trimmed in each tail of the distribution. Here,

$$d_j = \frac{(n_j - 1) s_{wj}^2}{h_j (h_j - 1)} \quad (10)$$

is Yuen's estimate of the squared standard error, where s_{wj}^2 is the gamma-Winsorized variance and $h_j = n_j - 2g_j$ is the effective sample size, that is, the size after trimming ($j = 1, 2$). The gamma-Winsorized variance is calculated as follows.

$$s_{wj}^2 = \frac{1}{n_j - 1} \sum_{i=1}^{n_j} (Y_{ij} - \bar{Y}_{wj})^2 \quad (11)$$

where

$$\begin{aligned} Y_{ij} &= X_{(g_j+1)j} \quad \text{if } X_{ij} \leq X_{(g_j+1)j} \\ &= X_{ij} \quad \text{if } X_{(g_j+1)j} < X_{ij} < X_{(n_j-g_j)j} \\ &= X_{(n_j-g_j)j} \quad \text{if } X_{ij} \geq X_{(n_j-g_j)j} \end{aligned}$$

are the Winsorized scores and

$$\bar{Y}_{wj} = \frac{1}{n_j} \sum_{i=1}^{n_j} Y_{ij} \quad (12)$$

is the sample Winsorized mean (Yuen, 1974).

4.5 Design Of The Simulation Study

Simulations were performed by generating observations from a g-and-h distribution (Hoaglin, 1985). This is done by generating Z from a standard normal distribution and then computing

$$X = \frac{(\exp(gZ) - 1) \exp\left(\frac{hZ^2}{2}\right)}{g}.$$

When $g = 0$, this last expression is taken to be

$$X = Z \exp\left(\frac{hZ^2}{2}\right).$$

The reason for using the g-and-h distribution is that it provides a simple method for generating observations from a wide variety of distributions, which include extreme departures from normality as measured by skewness and kurtosis. Because it

is not clear just how non-normal distributions might be in applied work, the strategy here is to include distributions with skewness and kurtosis values that surely exceed what can be expected in practice. For $g = h = 0$, X has a standard normal distribution and for $g=1, h=0$ X has a lognormal distribution. When $g = 0$, X is symmetric. Table 4.1 shows the estimated skewness and kurtosis values corresponding to the g and h values used in the simulations. Skewness is measured with $\kappa_1 = \mu_3 / \mu_2^{3/2}$ and kurtosis is measured with $\kappa_2 = \mu_4 / \mu_2^2$ where $\mu_k = E(X_k - \mu)^k$.

Table 4.1 Some properties of g and h distribution

g	h	$\hat{\kappa}_1$	$\hat{\kappa}_2$
0	0	0	3
0.5	0.2	10.554	200.736
0.5	0.5	102.533	20405.6

4.6 Simulation Results

The three tests were compared in terms of their ability to control the probability of a Type I error when testing at the 0.05 level. Although the importance of a Type I error depends on the situation, Bradley (1978) has suggested that ideally, the actual level should be between 0.045 and 0.055 when testing at the 0.05 level. And at a minimum the actual level should be between 0.025 and 0.075. Here the focus is on Bradley's more conservative criterion that the actual level should be between 0.045 and 0.055.

All simulations were done in R with 10000 replications for each case. Regarding the Yuen Test, both 10%, 20% and 25% trimming were used.

Table 4.2 gives the type I errors of the tests used in the simulations.

Table 4.2 Simulation results

Distributions	n_1 n_2	σ_1 σ_2	yuen (0.1)	yuen (0.2)	yuen (0.25)	B_m^2	T
Dist.1: Normal	10 10	1 1	0.0445	0.0455	0.0455	0.0454	0.0476
		1 4	0.0560	0.0594	0.0594	0.0458	0.0613
	10 20	1 4	0.0526	0.0533	0.0562	0.0505	0.0103
		4 1	0.0550	0.0607	0.0611	0.0463	0.1643
	20 20	1 4	0.0468	0.0478	0.0518	0.0488	0.0549
		4 1					
Dist.1: Normal	10 10	1 4	0.0528	0.0492	0.0459	0.0461	0.0955
		4 1	0.0526	0.0571	0.0570	0.0459	0.0606
	10 20	1 4	0.0480	0.0482	0.0487	0.0464	0.0384
		4 1	0.0538	0.0557	0.0560	0.0414	0.1637
	20 20	1 4	0.0528	0.0485	0.0513	0.0523	0.0915
		4 1	0.0529	0.0549	0.0579	0.0513	0.0545
Dist.1: Normal	10 10	1 4	0.0476	0.0537	0.0537	0.0480	0.0759
		4 1	0.0475	0.0525	0.0524	0.0466	0.0584
	10 20	1 4	0.0521	0.0595	0.0598	0.0486	0.0921
		4 1	0.0520	0.0515	0.0514	0.0404	0.1738
	20 20	1 4	0.0463	0.0493	0.0532	0.0487	0.0853
		4 1	0.0478	0.0505	0.0545	0.0492	0.0529
Dist.1: g=0.5 h=0.5	10 10	1 4	0.0437	0.0407	0.0387	0.0394	0.1526
		4 1	0.0438	0.0398	0.0393	0.0482	0.0778
	10 20	4 1	0.0480	0.0439	0.0414	0.0447	0.2314
		1 4	0.0459	0.0421	0.0433	0.0516	0.1537

In the simulation study the probability of producing type I error was calculated generating random data for 21 different sampling orders. The sampling orders started from the ideal conditions in which the normal distribution and homogenous variance

hypotheses were met, and gradually deviated from these hypotheses by using the distributions given in Table 4.1 for skewness and kurtosis values.

CHAPTER FIVE

CONCLUSIONS

This study aimed at introducing the R language, which has been used frequently in recent academic research, and conducting an application with this language. The strengths of this programming language are;

- Obtaining free of charge,
- Being an object-oriented programming language,
- Ease of increasing the functionality by adding different packages, developed for different purposes,
- Having 2-D, 3-D and more dimensional developed graphic tools.

Its weaknesses are;

- It is hard to learn.
- Although it has advanced data processing features, it needs to master array and matrix operations.
- It is not suitable for working with large data files. Insufficient memory errors may arise if files over several hundreds of megabytes are tried to open (R Development Core Team, 2008).
- It does not have any customer support unit to deal with the problems encountered, since it is not a commercial product.

According to Wegman and Solka SAS, which is defined as the Microsoft of the statistical software packages, is more like a program used in corporate-scale (Wegman & Solka, 2005). SAS is an extensive software package with application tools that can be used in many areas, and these features make SAS one of the most expensive software in its class. Statistical Packages for Social Sciences (SPSS) is another statistical software package which has the power of global competition, and it is especially used in social and educational sciences. The most basic difference between R and the widely used statistical software packages such as SAS and SPSS

is that R is not a software package, but a software development environment and a programming language that can be used for statistical calculations and graphics.

S-Plus which can be used on Windows and Unix platforms and R which can be used on Windows, Unix, and MacOS platforms are very similar to each other, except for their user interface, and a code written in one of them can generally be used in the other. However, some differences in syntax rules, which seem small, are so important that they may return different results after running the commands (Ihaka & Gentleman, 1996). On the contrary to R, in S-Plus the data objects are stored in the files created on the disk; therefore it is possible to recover the current environment after a problem. In R, on the other hand, everything is stored internally and it is not possible to recover the current environment after a problem. The most important feature that differentiates these two languages is R's being free of charge.

R, which is a open source and free of charge programming language preferred by researchers for having functions required for using the robust statistical methods, an important discipline in statistics, and with its easily addable packages for newly developed statistical methods, poses a powerful alternative that can meet the requirements in statistics and mathematics education, as well as academic research, without the need of licencing.

In the application chapter of the study, an R function for the M-estimator and bootsrap-t method version (B_m^2) of the method developed by Özdemir and Kurt (2006) was written, and this method was compared to the Yuen Test, Student t Test and (B_m^2) tests, which are the most important robust two sample tests, in terms of maintaining levels of type I errors, using narrow sample ranges.

Here, the type I error values for the Yuen and B_m^2 tests, with 10% trimming ratio, went out of Bradley's conservative criterion in 4 out of 21 orders. This number is observed as 9 for Yuen test with 20% trimming ratio, 10 for Yuen Test with 25% trimming ratio, and 17 for Student-t test.

Robust inference methods are expected to maintain the significance levels defined by the researcher in cases where the relative hypotheses about the method are met. When the results of the first group, in which both population distributions are normal, are investigated, it is seen that Yuen Test with a 10% trimming ratio does not meet this condition in 2 sampling orders.

B_m^2 test uses an estimator like M-estimator with a breakdown point higher than the trimmed mean. Therefore, it is affected less from the outlier values in determining the point where the observation values in the population clustered, compared to the trimmed mean, and gives closer estimations to this clustering point. Trimmed mean discards a fixed amount of the sample data, according to the trimming ratio, whatever the symmetry structure of the population is. M-estimator, on the other hand, uses an empirical outlier value determination measure. Therefore, it defines some variable number of the data as outlier value and discards them from the right, from the left or sometimes from both sides according to the symmetry structure of the population.

The Student-t test, developed almost 100 years ago, can not maintain the nominal significance levels defined by the researcher, as it deviates from the normal theory hypotheses, especially in non-balanced experiment orders (Wilcox, 2005). Therefore, B_m^2 and Yuen test should be considered as alternatives for comparing two independent groups, when these hypotheses are not met.

REFERENCES

- Bailey, B.J.R. (1980). Accurate normalizing transformations of Student's t Variate. *Applied Statistics*, 29, 304-306.
- Bradley, J.V. (1978). Robustness? *British Journal of Mathematical and Statistical Psychology*, 31, 144-152.
- Braun, W. J., & Murdoch, D.J. (2007). *A first course in statistical programming with R*. Cambridge Uni. Press.
- Crawley, M.J. (2007). *The R book*. Johns Willey & Sons LTD.
- Dalgaard, P.(2008). *Introductory statistics with R(2th Ed.)*. Springer Science Business Media, LLC.
- Good, P.I. (2006). *Resampling methods(2th Ed.)*. Birkhäuser Boston, Springer Science Business Media, LLC. 21
- Huber, P. J. (1981). *Robust statistics*. New York: John Wiley And Sons.
- Hoaglin, D.C., Mosteller, F., & Tukey, J.W. (1985). *Exploring data tables, trends and shapes*. Newyork: Wiley
- Ihaka, R., & Gentleman, R.(1996), R: A language for data analysis and graphics, *Journal of Computational and Graphical Statistics*, 5(3). 299-314
- Maindonald, J., & Braun, J.(2006). *Data analyses and graphics using R(2th Ed.)*. Cambridge Uni. Press.
- Murrel, P. (2006). *R graphics*. CRC Press.

- Ozdemir, A.F., & Kurt.S (2006). One-way fixed effect analysis of variance under variance heterogeneity and a solution proposal. *Selcuk Journal of Applied Mathematics*, 7, 81-91
- R Core Team. (2008). *R language defination*. Retrieved October, 25, 2009, from <http://www.r-project.org>.
- R Core Team. (2009). *An introduction to R*. Retrieved January, 25, 2010, from <http://www.r-project.org>. 2, 39.
- R Core Team (2009). *R data import/export*. Retrieved January, 25, 2010, from <http://www.r-project.org>. 13.
- R Core Team (2009). *R internals*. Retrieved January, 25, 2010, from <http://www.r-project.org>.
- Serfling, R.J. (1980). *Approximation Theorems of Mathematical Statistics*. Newyork: Wiley
- Wegman E. J., & Solka J. L. (2005). “*Statistical Software for Today and Tomorrow*” in *encyclopedia of statistics*. John Wiley. 3.
- Wilcox, R.R. (2001). *Fundamentals of modern statistical methods: Substantially improving power and accuracy*. Newyork: Springer.
- Wilcox, R.R. (2005). *Introduction to robust estimation and hypothesis testing*(2th Ed.). Elsevier Academic Press.
- Yuen, K.K. (1974). The two-sample trimmed t for unequal population variances. *Biometrika*, 61, 165-170.

APPENDIX

Simulation codes for the B^2 test with one-step M-Estimator and bootstrap-t method.

```

boot_t2<-function(n1=10, n2=10, mu=0, sigma1=1, sigma2=4){
sum<-0
curr<-0
stats<-as.vector(0)
for (i in 1:10000){
x<-rnorm(n1,mu,sigma1)
onex<-onestep(x)
y<-rnorm(n2,mu,sigma2)
oney<-onestep(y)
#test stat
T<-osbt2(x,y)$teststat
samplex<-matrix(sample(x,length(x)*599,replace=T),nrow=599)
sampley<-matrix(sample(y,length(y)*599,replace=T),nrow=599)
farx<-samplex-onex
fary<-sampley-oney
madxx<-apply(farx,1,mad)
madyy<-apply(fary,1,mad)
#mad control of x and y
for (i in 1:599){
if (madxx[i]==0){
dat<-gener(n1,sigma1)$x
farx[i,]<-dat
}
}
for (i in 1:599){
if (madyy[i]==0){
dat<-gener(n2,sigma2)$x
fary[i,]<-dat
}
}
}

```



```

}
}
#osbt2 test
for (i in 1:599) stats[i]<-osbt2(farx[i,],fary[i,])$teststat
stats<-sort(stats)
results<- if (T<=stats[16] || T>=stats[583]) { 1 } else { 0 }
sum<-sum+results
}
print(sum)
}

```

```

gener2<-function(x,sigma=1){
if (gener(x,sigma)$madx==0) gener(x,sigma)
}
gener<-function(nn,sigma=1){
x<-rnorm(nn,sd=sigma)
x<-x-onestep(x)
madx<-mad(x)
if (madx==0) gener2(nn,sigma)
else return(list(x=x,madx=madx))
}

```

```

osbt2<-function(x,y,k=2,alpha=0.05,bend=1.28){
##one-step m estimator ile B2t testi yapar
#s ler mestse ile hesaplandi
x<-x[!is.na(x)] # Remove any missing values in x
y<-y[!is.na(y)] # Remove any missing values in y
zc<-qnorm(alpha/2)
x2<-(x-median(x))/mad(x)
y2<-(y-median(y))/mad(y)
C<-length(x[abs(x2)>bend])
D<-length(y[abs(y2)>bend])

```

```

e<-c(C,D)
alist<-list(x,y)
f<-(sapply(alist,length))-e
s=sapply(alist,mestse)^2
wden=sum(1/s)
w=(1/s)/wden
yplus<-sum(w*(sapply(alist,onestep)))
tt<-((sapply(alist,onestep))-yplus)/sqrt(s)
v<-(f-1)
z<-
  ((4*v^2)+(5*((2*(zc^2))+3)/24))/((4*v^2)+v+(((4*(zc^2))+9)/12))*sqrt(v)*(sqrt(1
  og(1+(tt^2/v))))
teststat<-sum(z^2)
crit<-qchisq(1-alpha,k-1)
bt2pvalue<-1-(pchisq(teststat,k-1))
list(p.value=bt2pvalue,teststat=teststat,crit=crit,e=e,f=f,s=s,w=w,tt=tt)
}

```