R Tutorial

with Bayesian Statistics
Using OpenBUGS

Chi Yau
r-tutor.com

R Tutorial
with Bayesian Statistics
Using OpenBUGS

By Chi Yau

Preface

This text provides R tutorials on statistics including hypothesis testing, ANOVA and linear regressions. It fulfills popular demands by users of *r-tutor.com* for exercise solutions and offline access.

Part III of the text is about Bayesian statistics. It begins with closed analytic solutions and basic BUGS models for simple examples. Then it covers OpenBUGS for Bayesian ANOVA and regression analysis. Finally, it shows how to build more complex Bayesian models and demonstrates CODA for Markov Chain Monte Carlo (MCMC) convergence.

The last part of this text discusses advanced GPU computing in R using the RPUDPLUS package. Topics include hierarchical clustering, Kendall's tau, support vector machines and Bayesian classification. It illustrates the importance of High Performance Computing (HPC) in the future of statistics. The text concludes with an extra section on hierarchical multinomial logit model for marketing research.

This eBook is published in Amazon Kindle format. It is a great tool for self-study. The R source code is available for download in r-tutor.com.

I am grateful to users who sent me donations in the past. It provides me encouragement and support in ways unimaginable. I would also like to thank users who alert me of errors. I really appreciate it if you can email me further feedback to *user.feedback@r-tutor.com*.

Chi Yau
Palo Alto, California
April, 2014

TABLE OF CONTENTS

PART I
R Introduction

PART II
Elementary Statistics with R

PART III
Bayesian Statistics Using OpenBUGS

Full Table of Contents

PART I
R Introduction

CHAPTER 5 Data Frame

PART II
Elementary Statistics with R

CHAPTER 6 Qualitative Data

CHAPTER 7 Quantitative Data

CHAPTER 8 Numerical Measures

PART III
Bayesian Statistics Using OpenBUGS

29.3 Measurement Intervals

PART IV
GPU Computing with R

Appendix

References

Part I
R Introduction

We begin with a series of introductory R tutorials. It serves as background material for our discussions on classical statistics, Bayesian statistics, and GPU computing.

The only hardware requirement for most of the R tutorials is a PC with the latest free open source R software installed. R has extensive documentation and active online community support. It is the perfect environment to get started in statistical computing.

Installation

We can visit the R homepage for its latest update, and find a list of CRAN mirrors that provide installation files in nearby locations.

Using External Data

R offers plenty of options for loading external data, including Excel, Minitab and SPSS files. We have included a tutorial on Data Import for the purpose.

R Session

After R is started, there is a console awaiting for input. At the prompt (>), we can enter numbers and perform calculations right away.

```
> 1 + 2
[1] 3
```

Variable Assignment

We assign values to variables with the assignment operator `"="`. Just typing the variable by itself at the prompt will print out the value. We should note that another form of assignment operator `"<-"` is also in use.

```
> x = 1
> x
[1] 1
```

Functions

R functions are invoked by its name, then followed by the parenthesis, and zero or more arguments. The following apply the function `c` to combine three numeric values into a vector.

```
> c(1, 2, 3)
[1] 1 2 3
```

Comments

All text after the pound sign `#` within the same line is considered a comment.

```
> 1 + 1      # this is a comment
[1] 2
```

Extension Package

Sometimes we need additional functionality beyond those offered by the core R library. In order to install an extension package, we should invoke the `install.packages` function at the prompt and follow the instruction.

```
> install.packages()
```

Getting Help

R provides extensive documentation. For example, entering `?c` or `help(c)` at the prompt gives documentation of the function `c` in R. Please give it a try.

```
> help(c)
```

If we are not sure about the name of the function we are looking for, we can perform a fuzzy search with the `apropos` function.

```
> apropos("nova")
[1] "anova"             "anova.glm"
   ....
```

Finally, there is an R specific Internet search engine at RSeek for more information.

Chapter 1
Basic Data Types

There are several basic R data types that are of frequent occurrence in routine R calculations. Though seemingly innocent, they can still deliver surprises. Instead of chewing through the language specification, we will try to understand them better by direct experimentation with R code.

For simplicity, we defer the concept of *vector* for later discussion. Here, we use only vectors of unit length for demonstration.

1.1 Numeric

Decimal values are called **numerics** in R. It is the default computational data type. If we assign a decimal value to a variable `x` as follows, `x` will be of numeric type.

```
> x = 10.5        # assign a decimal
> x               # print x
[1] 10.5
> class(x)        # print class name
[1] "numeric"
```

Furthermore, even if we assign an integer to a variable `k`, it is still being saved as a numeric value.

```
> k = 1
> k               # print k
[1] 1
> class(k)        # print class name
[1] "numeric"
```

The fact that `k` is *not* an integer can be confirmed with the `is.integer` function. We will discuss how to create an integer in our next tutorial on the integer type.

```
> is.integer(k)  # is k an integer?
[1] FALSE
```

1.2 Integer

In order to create an **integer** variable in R, we invoke the `integer` function. We can be assured that `y` is indeed an integer by applying the `is.integer` function.

```
> y = as.integer(3)
> y             # print y
[1] 3
> class(y)      # print class name
[1] "integer"
> is.integer(y)  # is y an integer?
[1] TRUE
```

Incidentally, we can coerce a numeric value into an integer with the `as.integer` function.

```
> as.integer(3.14)   # integer cast
[1] 3
```

And we can parse a string for decimal values in much the same way.

```
> as.integer("5.27") # parse string
[1] 5
```

On the other hand, it is erroneous trying to parse a non-decimal string.

```
> as.integer("Joe")
[1] NA
Warning message:
NAs introduced by coercion
```

Often, it is useful to perform arithmetic on *logical values*. Just like the C language, `TRUE` has the value 1, while `FALSE` has value 0.

```
> as.integer(TRUE)
[1] 1
> as.integer(FALSE)
[1] 0
```

1.3 Complex

A **complex** value in R is defined via the pure imaginary value *i*.

```
> z = 1 + 2i     # a complex number
> z              # print z
[1] 1+2i
> class(z)       # print class name
[1] "complex"
```

The following gives an error since -1 is not a complex value.

```
> sqrt(-1)       # square root of -1
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
```

Instead, we have to use the complex value -1+0*i*.

```
> sqrt(-1+0i)
[1] 0+1i
```

An alternative is to coerce -1 into a complex value.

```
> sqrt(as.complex(-1))
[1] 0+1i
```

1.4 Logical

A **logical** value is often created via comparison between variables.

```
> x = 1; y = 2   # sample values
> z = x > y      # is x larger?
> z              # print result
[1] FALSE
> class(z)       # print class
[1] "logical"
```

Standard logical operations are `&` (and), `|` (or), and `!` (negation).

```
> u = TRUE; v = FALSE
> u & v          # u AND v
[1] FALSE
> u | v          # u OR v
[1] TRUE
> !u             # negation of u
[1] FALSE
```

Further details and related logical operations can be found in the R documentation.

```
> help("&")
```

1.5 Character

A **character** object represents string values in R. For example, the following is a character string made from a Shakespeare quote:

```
> s = "Brevity is the soul of wit."
```

We can find out its length with the function `nchar`.

```
> nchar(s)
[1] 27
```

We can also convert simple data values into character strings with the function `as.character`.

```
> x = as.character(3.14)
> x              # print x
[1] "3.14"
> class(x)       # print class name
[1] "character"
```

And we can merge two character strings into one with the function `paste`.

```
> fname = "Joe"; lname ="Smith"
> paste(fname, lname)
[1] "Joe Smith"
```

However, it is often more convenient to create a readable string with the `sprintf` function, which has a C language syntax.

```
> sprintf("%s has %d dollars",
+     "Sam", 100)
[1] "Sam has 100 dollars"
```

To extract a substring, we apply the `substr` function. Here is an example showing how to extract the substring between the third and twelfth positions in a character string.

```
> substr("Mary has a little lamb.",
+     start=3, stop=12)
```

```
[1] "ry has a l"
```

And to replace the first occurrence of the word "little" by another word "big" in the character string, we apply the `sub` function.

```
> sub("little", "big",
+     "Mary has a little lamb.")
[1] "Mary has a big lamb."
```

More functions for character string manipulation can be found in the R documentation.

```
> help("sub")
```

1.6 Factor

A factor object represents a categorical data type in R. Its sole purpose is to represent qualitative data, such as colors or shoe sizes.

We can create factor values from simple data types using the function `factor`.

```
> a = factor("A")
```

The following confirms that the object is indeed a factor.

```
> class(a)
[1] "factor"
```

In particular, we can create factors from numbers:

```
> x = factor(1)
> y = factor(2)
```

Since `x` and `y` are factor values, there is no arithmetic operation allowed. Try adding them up, and we get errors instead.

```
> x + y
[1] NA
Warning message:
In Ops.factor(x, y) : + not meaningful
for factors
```

More functions for handling factors can be found in the R documentation.

```
> help("factor")
```

Chapter 2
Vector

A vector is a sequence of data elements of the same basic type. Members in a vector are officially called **components**. Nevertheless, we will just call them **members** here. Here is a vector containing three numeric members 2, 3 and 5.

```
> c(2, 3, 5)
[1] 2 3 5
```

And here is a vector of logical values.

```
> c(TRUE, FALSE, TRUE, FALSE, FALSE)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

A vector can contain character strings.

```
> c("aa", "bb", "cc", "dd")
[1] "aa" "bb" "cc" "dd"
```

And we can find the number of members in a vector with the `length` function.

```
> length(c("aa", "bb", "cc", "dd"))
[1] 4
```

2.1 Combining Vectors

Vectors can be combined via the function `c`. For examples, the following two vectors `n` and `s` are combined into a new vector containing members from both vectors.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd")
> c(n, s)
[1] "2"  "3"  "5"  "aa" "bb" "cc" "dd"
```

Value Coercion

In the code snippet above, notice how the numeric values are being coerced into character strings when the two vectors are combined. This is necessary so as to maintain the same primitive data type for members in a single vector.

2.2 Vector Arithmetics

Arithmetic operations on vectors are performed in a member-by-member fashion, *i.e.*, member-wise. For example, suppose we have two vectors `a` and `b` as follows.

```
> a = c(1, 3, 5, 7)
> b = c(1, 2, 4, 8)
```

Then, if we multiply `a` by 5, we would get a vector with each of its members multiplied by 5.

```
> 5 * a
[1]  5 15 25 35
```

And if we add `a` and `b` together, the sum would be a vector whose members are the sum of the corresponding members from `a` and `b`.

```
> a + b
[1]  2  5  9 15
```

Similarly for subtraction, multiplication and division, we get new vectors via member-wise operations.

```
> a - b
[1]  0  1  1 -1

> a * b
[1]  1  6 20 56

> a / b
[1] 1.000 1.500 1.250 0.875
```

Recycling Rule

If two vectors are of unequal length, the shorter one will be recycled in order to match the longer vector. For example, the following vectors `u` and `v` have different lengths, and their sum is computed by recycling values of the shorter vector `u`.

```
> u = c(10, 20, 30)
> v = c(1, 2, 3, 4, 5, 6, 7, 8, 9)
> u + v
[1] 11 22 33 14 25 36 17 28 39
```

There will be a warning if the longer object length is not an exact multiple of the shorter one.

```
> w = c(10, 20, 30, 40)
> w + v
[1] 11 22 33 44 15 26 37 48 19
Warning message:
In w + v : longer object length is not a multiple of shorter obje
```

2.3 Vector Index

We retrieve values in a vector by declaring an index inside a *single square bracket* "`[]`" operator. For example, the following shows how to retrieve a vector member. Since the vector index is 1-based, we use the index position 3 for retrieving the third member.

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[3]
[1] "cc"
```

Unlike other programming languages, the square bracket operator returns more than just individual members. In fact, the result of the square bracket operator is another vector, and `s[3]` is a **vector slice** containing a single member "`cc`".

Negative Index

If the index is negative, it would strip the member whose position has the same absolute value as the negative index. For example, the following creates a vector slice with the third member removed.

```
> s[-3]
[1] "aa" "bb" "dd" "ee"
```

Out-of-Range Index

If an index is out-of-range, a missing value will be reported via the symbol `NA`.

```
> s[10]
[1] NA
```

2.4 Numeric Index Vector

A new vector can be sliced from a given vector with a **numeric index vector**, which contains intended member positions of the original vector to be retrieved.

Here it shows how to retrieve a vector slice containing the second and third members of a given vector `s`.

```
> s = c("aa", "bb", "cc", "dd", "ee")
> s[c(2, 3)]
[1] "bb" "cc"
```

Duplicate Indexes

An index vector allows duplicate values. Hence the following retrieves a member twice in one operation.

```
> s[c(2, 3, 3)]
[1] "bb" "cc" "cc"
```

Out-of-Order Indexes

An index vector can even be out-of-order. Here is a vector slice with the order of first and second members of `s` reversed.

```
> s[c(2, 1, 3)]
[1] "bb" "aa" "cc"
```

Range Index

To produce a vector slice between two members, we can use the colon operator ":". This can be convenient for situations involving large vectors.

```
> s[2:4]
[1] "bb" "cc" "dd"
```

More information for the colon operator is available in the R documentation.

```
> help(":")
```

2.5 Logical Index Vector

A new vector can be sliced from a given vector with a **logical index vector**, which has the same length as the original vector. Its members are `TRUE` if the corresponding members in the original vector are to be included in the slice, and `FALSE` if otherwise.

For example, consider the following vector `s` of length 5.

```
> s = c("aa", "bb", "cc", "dd")
```

To retrieve the second and fourth members of `s`, we define a logical vector `L` of the same length, and have its second and fourth members set as `TRUE`.

```
> L = c(FALSE, TRUE, FALSE, TRUE)
> s[L]
[1] "bb" "dd"
```

The code can be abbreviated into a single line.

```
> s[c(FALSE, TRUE, FALSE, TRUE)]
[1] "bb" "dd"
```

2.6 Named Vector Members

We can assign names to vector members. For example, the following variable `v` is a character string vector with two members.

```
> v = c("Mary", "Sue")
> v
[1] "Mary" "Sue"
```

We now name the first member as `First`, and the second as `last`.

```
> names(v) = c("First", "Last")
> v
 First   Last
"Mary"  "Sue"
```

Then we can retrieve the first member by its name.

```
> v["First"]
[1] "Mary"
```

Furthermore, we can reverse the order with a character string index vector.

```
> v[c("Last", "First")]
  Last  First
 "Sue" "Mary"
```

Chapter 3
Matrix

A **matrix** is a collection of data elements arranged in a two-dimensional rectangular layout. The following is an example of a matrix with 2 rows and 3 columns.

$$A = \begin{bmatrix} 2 & 4 & 3 \\ 1 & 5 & 7 \end{bmatrix}$$

3.1 Matrix Elements

We create a matrix in R with the namesake function from a vector. The elements in a matrix must be all of the same basic type. By default, matrix elements are arranged along the *column* direction.

```
> A = matrix(
+   c(2, 1, 4, 5, 3, 7),
+   nrow=2)

> A
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

We can also input matrix elements along the *row* direction by enabling the `byrow` option.

```
> B = matrix(
+   c(2, 1, 4, 5, 3, 7),
+   nrow=2,
+   byrow=TRUE)

> B
     [,1] [,2] [,3]
[1,]    2    1    4
[2,]    5    3    7
```

In general, a matrix of M rows and N columns is called a $M \times N$ matrix. An element at the m^{th} row and n^{th} column of `A` can be accessed via the expression `A[m, n]`.

```
> A[2, 3]    # 2nd row, 3rd column
[1] 7
```

The entire m^{th} row of `A` can be extracted as `A[m, ]`.

```
> A[2, ]     # the 2nd row of A
[1] 1 5 7
```

Similarly, the entire n^{th} column of `A` can be extracted as `A[ , n]`.

```
> A[ ,3]     # the 3rd column of A
[1] 3 7
```

Note that the code above produces a vector, instead of a 2x1 matrix. In order to produce the latter outcome, an extra argument, `drop`, must be explicitly set as `FALSE`.

```
> A[ ,3, drop=FALSE]
     [,1]
[1,]    3
[2,]    7
```

We can also extract more than one rows or columns at a time.

```
> A[ ,c(1,3)]
     [,1] [,2]
[1,]    2    3
[2,]    1    7
```

If we assign names to the rows and columns, then we can access matrix elements by names instead of coordinates.

```
> dimnames(A) = list(
+   c("row1", "row2"),
+   c("col1", "col2", "col3"))

> A
     col1 col2 col3
row1    2    4    3
row2    1    5    7

> A["row2", "col3"]
[1] 7
```

3.2 Matrix Construction

There are various ways to construct a matrix.

Transpose

Consider the following `3x2` matrix `B`. It has 3 rows and 2 columns.

```
> B = matrix(
+   c(2, 4, 3, 1, 5, 7),
+   nrow=3)

> B
     [,1] [,2]
[1,]    2    1
[2,]    4    5
[3,]    3    7
```

We construct the **transpose** of a matrix by interchanging its columns and rows using the function `t`. Thus the transpose of `B` is a `2x3` matrix, and has 2 rows and 3 columns.

```
> t(B)          # transpose
     [,1] [,2] [,3]
[1,]    2    4    3
[2,]    1    5    7
```

Combining Matrices

We can combine two matrices having same number of rows into a larger matrix. For example, suppose we have another matrix `C` also of 3 rows.

```
> C = matrix(
+   c(7, 4, 2),
+   nrow=3)

> C
     [,1]
[1,]    7
[2,]    4
```

```
[3,]    2
```

Then we can combine `B` and `C` with the function `cbind`.

```
> cbind(B, C)
     [,1] [,2] [,3]
[1,]    2    1    7
[2,]    4    5    4
[3,]    3    7    2
```

Similarly, we can combine two matrices having same number of columns with the function `rbind`.

```
> D = matrix(
+   c(6, 2),
+   nrow=1,
+   ncol=2)

> D          # D has 2 columns
     [,1] [,2]
[1,]    6    2

> rbind(B, D)
     [,1] [,2]
[1,]    2    1
[2,]    4    5
[3,]    3    7
[4,]    6    2
```

3.3 Matrix Arithmetics

When the dimensions of two matrices are compatible, it is possible to perform various arithmetic operations with them.

Addition and Subtraction

We can add or subtract two matrices when they have the same dimensions. For example, suppose `A` and `B` are both `3x2` matrices.

```
> A = matrix(1:6, nrow=3); A
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6

> B = matrix(5:10, nrow=3); B
     [,1] [,2]
[1,]    5    8
[2,]    6    9
[3,]    7   10
```

Then we can compute their **sum**:

```
> A + B
     [,1] [,2]
[1,]    6   12
[2,]    8   14
[3,]   10   16
```

Similarly, we can compute their **difference**:

```
> A - B
     [,1] [,2]
[1,]   -4   -4
[2,]   -4   -4
[3,]   -4   -4
```

Matrix Multiplication

We can multiply two matrices together if the column dimension of the first matrix is the same as the row dimension of the second matrix.

More specifically, if the dimension of the first matrix is MxN, and the dimension of the second matrix is NxK, then their matrix product will be of dimension MxK. Furthermore, the element at the m^{th} row and n^{th} column of the product is the *dot product* of the m^{th} row of the first matrix with the n^{th} column of the second matrix.

For example, consider the following two matrices `C` and `D`. As `C` is a `3x4` matrix, and `D` is a `4x5` matrix, the column dimension of `C` matches the row dimension of `D`. Hence we can define the matrix product of `C` and `D`.

```
> C = matrix(1:12, nrow=3); C
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> D = matrix(-4:15, nrow=4); D
     [,1] [,2] [,3] [,4] [,5]
[1,]   -4    0    4    8   12
[2,]   -3    1    5    9   13
[3,]   -2    2    6   10   14
[4,]   -1    3    7   11   15
```

We can multiply `C` and `D` together using the operator `%*%` in R. The product is a `3x5` matrix as expected.

```
> C %*% D
     [,1] [,2] [,3] [,4] [,5]
[1,]  -40   48  136  224  312
[2,]  -50   54  158  262  366
[3,]  -60   60  180  300  420
```

Chapter 4
List

A **list** is a generic vector containing other objects. It is a very flexible data structure that allows objects of vastly different data types to reside together in the same container.

4.1 List Index

In the following, `x` is a list containing a *copy* each of a numeric vector `n`, a character vector `s`, a logical vector `b`, and a numeric value 3.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc", "dd")
> b = c(TRUE, FALSE, TRUE, FALSE)
> x = list(n, s, b, 3)
```

List Slicing

We retrieve a list slice with the *single square bracket* "`[]`" operator. The following is a slice containing the second member of `x`, which holds a copy of `s`.

```
> x[2]
[[1]]
[1] "aa" "bb" "cc" "dd"
```

With an index vector, we can retrieve a slice with multiple list members. The following is a slice containing the second and fourth members of `x`.

```
> x[c(2, 4)]
[[1]]
[1] "aa" "bb" "cc" "dd"

[[2]]
[1] 3
```

Member Access

If we would like to directly access a list member, we have to use the *double square bracket* "`[[]]`" operator. For example, the object `x[[2]]` in the following expression is the second member of `x`. More specifically, `x[[2]]` is a copy of `s`, instead of a slice containing a copy of `s`.

```
> x[[2]]
[1] "aa" "bb" "cc" "dd"
```

Therefore, we can modify `x[[2]]` without affecting `s`.

```
> x[[2]][1] = "ta"
> x[[2]]
[1] "ta" "bb" "cc" "dd"
> s
[1] "aa" "bb" "cc" "dd"
```

4.2 Named Members

We can assign names to list members, and reference them by names instead of numeric indexes. For example, in the following, `v` is a list with two members, named "bob" and "john".

```
> v = list(
+     bob=c(2, 3, 5),
+     john=c("aa", "bb"))

> v
$bob
[1] 2 3 5

$john
[1] "aa" "bb"
```

List Slicing

We retrieve a list slice with the *single square bracket* "`[]`" operator. Here is a list slice containing a member of `v` named "bob".

```
> v["bob"]
$bob
[1] 2 3 5
```

With an index vector, we can retrieve a slice with multiple members. Here is a list slice with both members of `v`. Notice how they are reversed from their original positions in `v`.

```
> v[c("john", "bob")]
$john
[1] "aa" "bb"

$bob
[1] 2 3 5
```

Member Access

If we would like to directly access a list member, we have to use the *double*

square bracket "`[[]]`" operator. The following references a member of `v` by name.

```
> v[["bob"]]
[1] 2 3 5
```

A named list member can also be accessed directly with the "`$`" operator in lieu of the double square bracket operator.

```
> v$bob
[1] 2 3 5
```

Search Path Attachment

We can *attach* a list to the R search path and access its members without explicitly mentioning the list.

```
> attach(v)
> bob
[1] 2 3 5
```

It should to be explicitly *detached* for cleanup purpose.

```
> detach(v)
```

Chapter 5
Data Frame

A **data frame** is used for storing data tables. It is a list of vectors of equal length. For example, in the following, `f` is a data frame containing a numeric vector `n`, a character vector `s`, and a logical vector `b`.

```
> n = c(2, 3, 5)
> s = c("aa", "bb", "cc")
> b = c(TRUE, FALSE, TRUE)
> f = data.frame(n, s, b)
```

Build-in Data Frame

We often use built-in data frames in R for illustration in our tutorials. For example, we use the following built-in data frame `mtcars` for this purpose.

```
> mtcars
                mpg cyl disp  hp ...
Mazda RX4      21.0   6  160 110 ...
Mazda RX4 Wag  21.0   6  160 110 ...
Datsun 710     22.8   4  108  93 ...
                ............
```

The top line of the table, called the **header**, contains the column names. Each horizontal line afterward denotes a **data row**, which begins with the name of the row, and then followed by the actual data. Each data member inside a row is called a **cell**.

To retrieve data in a cell, we would enter its row and column coordinates in the *single square bracket* "`[]`" operator. The two coordinates are separated by a comma. Hence, a cell coordinate begins with the row position, followed by a

comma, then ends with the column position. The order is important.

Here is the data value from the cell at the first row and second column of `mtcars`.

```
> mtcars[1, 2]
[1] 6
```

Moreover, we can use the row and column names instead of numeric coordinates.

```
> mtcars["Mazda RX4", "cyl"]
[1] 6
```

Lastly, the number of data rows in the data frame can be found by the function `nrow`.

```
> nrow(mtcars) # number of rows
[1] 32
```

And the number of columns in a data frame is given by the function `ncol`.

```
> ncol(mtcars) # number of columns
[1] 11
```

Further details of the `mtcars` data set is available in the R documentation.

```
> help(mtcars)
```

Tips

Instead of printing out the entire data frame, it is often desirable to have a sneak preview via the function `head` instead.

```
> head(mtcars)
            mpg cyl disp  hp drat ...
Mazda RX4  21.0   6  160 110 3.90 ...
               ............
```

5.1 Column Vector

We retrieve a data frame column with the *double square bracket* "`[[]]`" operator. For example, to retrieve the ninth column of the built-in data set `mtcars`, we write `mtcars[[9]]`.

```
> mtcars[[9]]
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

We can retrieve the same column vector by its name.

```
> mtcars[["am"]]
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

We can also retrieve with the "`$`" operator in lieu of the double square bracket operator.

```
> mtcars$am
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

Yet another way to retrieve the same column vector is to use the *single square bracket* "`[]`" operator. We just have to prefix the column name with a comma character, which signals a wildcard match for the row position.

```
> mtcars[,"am"]
 [1]  1 1 1 0 0 0 0 0 0 0 0 ...
```

5.2 Column Slice

We retrieve a data frame column slice with the *single square bracket* "`[]`" operator.

Numeric Indexing

The following is a slice containing the first column of the built-in data set `mtcars`.

```
> mtcars[1]
                   mpg
Mazda RX4         21.0
Mazda RX4 Wag     21.0
Datsun 710        22.8
            ............
```

Name Indexing

We can also retrieve the same column slice by its name.

```
> mtcars["mpg"]
                   mpg
Mazda RX4         21.0
Mazda RX4 Wag     21.0
Datsun 710        22.8
            ............
```

To retrieve a data frame slice with two columns, say, `mpg` and `hp`, we would pack the column names in an index vector inside the single square bracket operator.

```
> mtcars[c("mpg", "hp")]
                   mpg  hp
Mazda RX4         21.0 110
Mazda RX4 Wag     21.0 110
Datsun 710        22.8  93
            ............
```

5.3 Row Slice

Just like what we did with data frame columns, we retrieve rows from a data frame with the single square bracket operator. However, in additional to an index vector of row positions, we append an extra comma character. This is important, as the extra comma signals a wildcard match for the second coordinate of column positions.

Numeric Indexing

For example, the following retrieves a row record of the built-in data set `mtcars`. Please notice the extra comma in the square bracket operator, and it is *not* a typo. It shows that the 1974 Camaro Z28 has a gas mileage of 13.3 miles per gallon, an eight cylinder 245 horse power engine, …, *etc*.

```
> mtcars[24,]
            mpg cyl disp  hp drat ...
Camaro Z28 13.3   8  350 245 3.73 ...
```

To retrieve more than one rows at once, we use a numeric index vector.

```
> mtcars[c(3, 24),]
            mpg cyl disp  hp drat ...
Datsun 710 22.8   4  108  93 3.85 ...
Camaro Z28 13.3   8  350 245 3.73 ...
```

Name Indexing

We can retrieve a row by its name.

```
> mtcars["Camaro Z28",]
            mpg cyl disp  hp drat ...
Camaro Z28 13.3   8  350 245 3.73 ...
```

And we can pack the row names in an index vector in order to retrieve multiple rows at once.

```
> mtcars[c(
```

```
+     "Datsun 710", "Camaro Z28"),]
            mpg cyl disp  hp drat ...
Datsun 710 22.8   4  108  93 3.85 ...
Camaro Z28 13.3   8  350 245 3.73 ...
```

Logical Indexing

Lastly, we can retrieve rows with a logical index vector. In the following vector `L`, its member value is `TRUE` if the car has automatic transmission, and `FALSE` if otherwise.

```
> L = mtcars$am == 0
> L
 [1]   FALSE FALSE FALSE  TRUE ...
```

Here is the list of vehicles with automatic transmission.

```
> mtcars[L,]
                   mpg cyl  disp ...
Hornet 4 Drive    21.4   6 258.0 ...
Hornet Sportabout 18.7   8 360.0 ...
                 ...........
```

And here is the gas mileage data for automatic transmission.

```
> mtcars[L,]$mpg
 [1] 21.4 18.7 18.1 14.3 24.4 ...
```

5.4 Data Import

It is often necessary to import sample textbook data into R before you start working on your homework.

Excel File

Quite frequently, the sample data is in `Excel` format, and needs to be imported into R prior to use. For this, we can use the function `read.xls` from the `gdata` package. It reads from an Excel spreadsheet and returns a data frame. The following shows how to load an Excel spreadsheet named "mydata.xls". This method requires Perl runtime to be present in the system.

```
> library(gdata)
> help(read.xls)
> mydata = read.xls("mydata.xls")
```

Alternatively, we can use the function `loadWorkbook` from the `XLConnect` package to read the entire workbook, and then load the worksheets with `readWorksheet`. The `XLConnect` package requires Java to be pre-installed.

```
> library(XLConnect)
> wk = loadWorkbook("mydata.xls")
> df = readWorksheet(wk,
+         sheet="Sheet1")
```

Minitab File

If the data file is in `Minitab Portable Worksheet` format, it can be opened with the function `read.mtp` from the `foreign` package. It returns a list of components in the Minitab worksheet.

```
> library(foreign)
> help(read.mtp)
> mydata = read.mtp("mydata.mtp")
```

SPSS File

For the data files in **SPSS** format, it can be opened with the function `read.spss` also from the `foreign` package. There is a `to.data.frame` option for choosing whether a data frame is to be returned. By default, it returns a list of components instead.

```
> library(foreign)
> help(read.spss)
> mydata = read.spss("myfile",
+     to.data.frame=TRUE)
```

Table File

A data table can resides in a text file. The cells inside the table are separated by blank characters. Here is an example of a table with 4 rows and 3 columns.

```
100   a1   b1
200   a2   b2
300   a3   b3
400   a4   b4
```

Now copy and paste the table above in a file named "mydata.txt" with a text editor. Then load the data into the workspace with the function `read.table`.

```
> mydata = read.table("mydata.txt")
> mydata
   V1 V2 V3
1 100 a1 b1
2 200 a2 b2
3 300 a3 b3
4 400 a4 b4
```

For further detail of the function `read.table`, please consult the R documentation.

```
> help(read.table)
```

CSV File

The sample data can also be in **comma separated values** (CSV) format. Each cell inside such data file is separated by a special character, which usually is a comma, although other characters can be used as well.

The first row of the data file should contain the column names instead of the actual data. Here is a sample of the expected format.

```
Col1,Col2,Col3
100,a1,b1
200,a2,b2
300,a3,b3
```

After we copy and paste the data above in a file named "mydata.csv" with a text editor, we can read the data with the function `read.csv`.

```
> mydata = read.csv("mydata.csv")
> mydata
  Col1 Col2 Col3
1  100   a1   b1
2  200   a2   b2
3  300   a3   b3
```

In various European locales, as the comma character serves as the decimal point, the function `read.csv2` should be used instead. For further detail of the `read.csv` and `read.csv2` functions, please consult the R documentation.

```
> help(read.csv)
```

Working Directory

Finally, the code samples above assume the data files are located in the R **working directory**, which can be found with the function `getwd`.

```
> getwd()    # working directory
```

You can select a different working directory with the function `setwd`, and thus avoid entering the full path of the data files.

```
> setwd("<new path>")
```

Note that the forward slash should be used as the path separator even on Windows platform.

```
> setwd("C:/MyDoc")
```

Part II
Elementary Statistics with R

Ever wonder how to finish your statistics homework real fast? Or you just want a quick way to verify your tedious calculations in your statistics class assignment. We provide an answer here by solving statistics exercises with R.

Here, you will find statistics problems similar to those found in popular college textbooks. The R solutions are short, self-contained and requires minimal R skill. Most of them are just a few lines in length. With simple modifications, the code samples can be turned into homework answers. In addition to helping with your homework, the tutorials will give you a taste of working with statistics software in general, and it will prove invaluable in the success of your career.

Chapter 6
Qualitative Data

A data sample is called **qualitative**, also known as **categorical**, if its values belong to a collection of non-overlapping classes. Common examples of qualitative data include student letter grade (A, B, C, D or F), commercial bond rating (AAA, AAB, …) and consumer clothing shoe sizes (1, 2, 3, …).

The tutorials in this section are based on an R built-in data frame named `painters`. As the name suggested, it is a compilation of technical information of a few eighteenth century classical painters. The data set belongs to the `MASS` package, which is often implicitly pre-loaded in the R workspace.

```
> library(MASS)
> head(painters)
              Composition Drawing
Da Udine               10       8
Da Vinci               15      16
Del Piombo              8      13
Del Sarto              12      16
Fr. Penni               0      15
Guilio Romano          15      16
              Colour Expression
Da Udine          16          3
Da Vinci           4         14
Del Piombo        16          7
Del Sarto          9          8
Fr. Penni          8          0
Guilio Romano      4         14
              School
Da Udine           A
Da Vinci           A
Del Piombo         A
Del Sarto          A
```

```
Fr. Penni          A
Guilio Romano      A
```

The last column in the data set `painters` contains the school information of each painter. The schools are named as A, B, …, *etc*. The `School` column is therefore qualitative, and consists of factor values.

```
> painters$School
 [1] A A A A A A A A A A B B B ...
Levels: A B C D E F G H
```

For further details of the `painters` data set, please consult the R documentation.

```
> help(painters)
```

6.1 Frequency Distribution

The **frequency distribution** of a data variable is a summary of the data occurrence in a collection of non-overlapping categories.

Example

In the data set `painters`, the frequency distribution of the `School` variable is a summary of the number of painters in each school.

Problem

Find the number of painters belonging to each school in the data set `painters`.

Solution

We apply the `table` function to compute the frequency distribution of the `School` variable.

```
> school = painters$School
> school.freq = table(school)
```

Answer

The frequency distribution of the schools is:

```
> school.freq
school
 A  B  C  D  E  F  G  H
10  6  6 10  7  4  7  4
```

Enhanced Solution

We can apply the `cbind` function to print the result in column format.

```
> cbind(school.freq)
  school.freq
```

```
A          10
B           6
C           6
D          10
E           7
F           4
G           7
H           4
```

Exercise 1

Find the frequency distribution of the composition scores in `painters`.

Solution of Exercise 1

Step 1: Define a new variable `comp` that holds the composition scores.

```
> comp = painters$Composition
```

Step 2: Find the frequency distribution using the `table` function and save the result in another variable `comp.freq`.

```
> comp.freq = table(comp)
```

Step 3: Format and print the result with `cbind`.

```
> cbind(comp.freq)
   comp.freq
0          1
4          3
5          1
6          3
8          6
9          1
10         6
11         2
12         4
13         5
14         3
15        14
16         2
17         1
18         2
```

Exercise 2

Find programmatically the school that has the most painters.

Solution of Exercise 2

Step 1: Find the maximum frequency using the function `max`, and save the result in `school.freq.max`. It shows there are at most 10 painters in each school.

```
> school = painters$School
> school.freq = table(school)
> school.freq.max = max(school.freq)
> school.freq.max
[1] 10
```

Step 2: Compare the vector `school.freq` with `school.freq.max` and save the result in a logical vector `L`. Use this to create a vector slice `school.freq[L]`.

```
> L = school.freq == school.freq.max
> x = school.freq[L]
```

An alternative for finding `x` is to use the function `which`.

```
> x = which(
+     school.freq == school.freq.max)
```

Step 3: Print out the name of the schools in `x` with the namesake function `names`, and find that schools A and D have the most painters.

```
> names(x)
[1] "A" "D"
```

Note that the obvious function `which.max` is insufficient. It only gives the first school that has the most painters, instead of *every* school that has the most painters.

```
> y = which.max(school.freq)
> names(y)      # missing school "D"
[1] "A"
```

6.2 Relative Frequency Distribution

The **relative frequency distribution** of a data variable is a summary of the frequency proportion in a collection of non-overlapping categories.

The relationship of frequency and relative frequency is:

$$Relative\ Frequency = \frac{Frequency}{Sample\ Size}$$

Example

In the data set `painters`, the relative frequency distribution of the `School` variable is a summary of the proportion of painters in each school.

Problem

Find the proportion of painters distributed among the schools in the data set `painters`.

Solution

We first apply the `table` function to compute the frequency distribution of the `School` variable.

```
> school = painters$School
> school.freq = table(school)
```

Then we find the sample size of `painters` with the function `nrow`, and divide the frequency distribution with it. Therefore the relative frequency distribution is:

```
> school.relfreq =
+     school.freq / nrow(painters)
```

Answer

The relative frequency distribution of the schools is:

```
> school.relfreq
school
       A        B        C        D
0.185185 0.111111 0.111111 0.185185
       E        F        G        H
0.129630 0.074074 0.129630 0.074074
```

Enhanced Solution

We can print with fewer digits and make it more readable by setting the `digits` option.

```
> old = options(digits=1)
> school.relfreq
school
   A    B    C    D    E ...
0.19 0.11 0.11 0.19 0.13 ...
> options(old)
```

In addition, we can apply the `cbind` function to print the percentage of painters in each school in column format.

```
> old = options(digits=3)
> cbind(school.relfreq*100)
   [,1]
A 18.52
B 11.11
C 11.11
D 18.52
E 12.96
F  7.41
G 12.96
H  7.41
> options(old)
```

Exercise

Find the relative frequency distribution of the composition scores in `painters`.

Solution of Exercise

Step 1: Define a new variable `comp` that holds the composition scores.

```
> comp = painters$Composition
```

Step 2: Find the frequency distribution using the `table` function and save the result in another variable `comp.freq`.

```
> comp.freq = table(comp)
```

Step 3: Find the relative frequency distribution by dividing `comp.freq` with the sample size.

```
> comp.relfreq =
+     comp.freq / nrow(painters)
> comp.relfreq
comp
       0        4        5  ...
0.018519 0.055556 0.018519  ...
```

Step 4: Format and print the percentage of painters in the schools via the function `cbind`.

```
> old = options(digits=3)
> cbind(comp.relfreq*100)
    [,1]
0   1.85
4   5.56
5   1.85
6   5.56
8  11.11
9   1.85
10 11.11
11  3.70
12  7.41
13  9.26
14  5.56
15 25.93
16  3.70
17  1.85
18  3.70
> options(old)   # restore
```

6.3 Bar Graph

The **bar graph** of a qualitative data sample consists of vertical parallel bars that shows the frequency distribution graphically.

Example

In the data set `painters`, the bar graph of the `School` variable is a collection of vertical bars showing the number of painters in each school.

Problem

Find the bar graph of the painter schools in the data set `painters`.

Solution

We first apply the `table` function to compute the frequency distribution of the `School` variable.

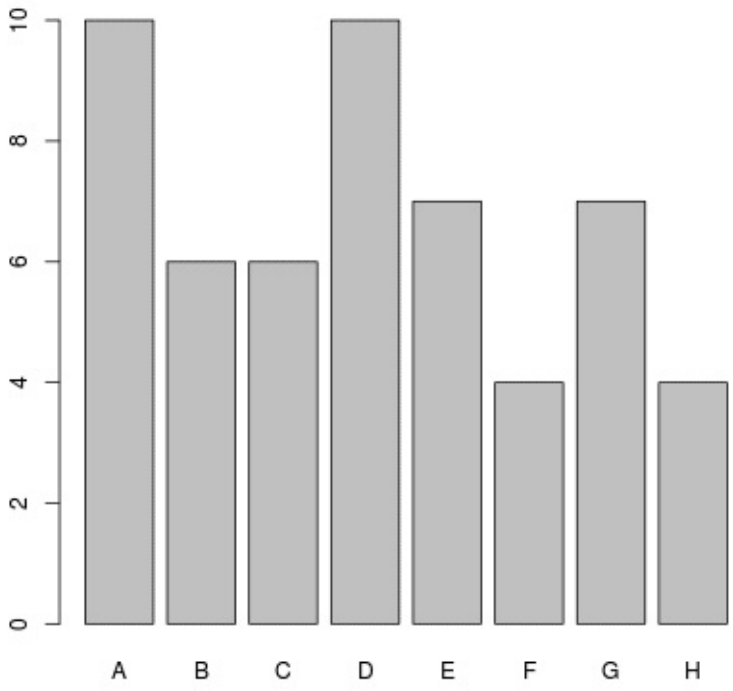
```
> school = painters$School
> school.freq = table(school)
```

Then we apply the function `barplot` to produce its bar graph.

```
> barplot(school.freq)
```

Answer

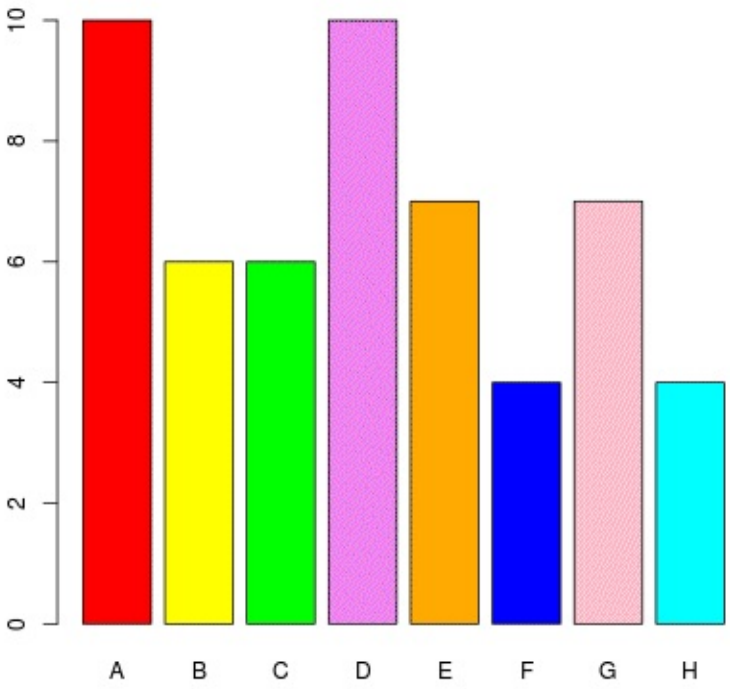
The bar graph of the school variable is:



Enhanced Solution

To colorize the bar graph, we can select a color palette and set it in the `col` argument of `barplot`.

```
> colors = c("red", "yellow",
+   "green", "violet", "orange",
+   "blue", "pink", "cyan")
> barplot(school.freq, col=colors)
```



Exercise

Find the bar graph of the composition scores in `painters`.

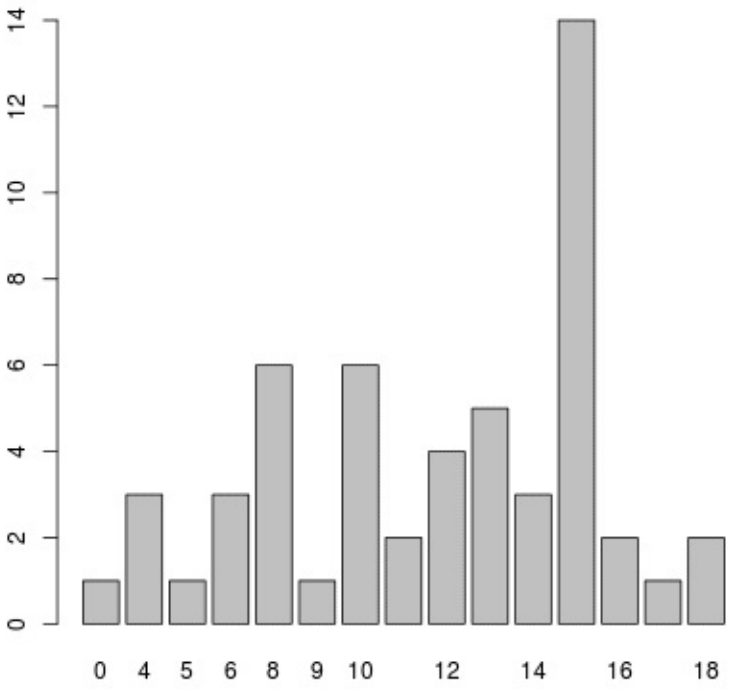
Solution of the Exercise

Step 1: Find the frequency distribution of the painter composition scores with the `table` function.

```
> comp = painters$Composition
> comp.freq = table(comp)
```

Step 2: Then apply the `barplot` function.

```
> barplot(comp.freq)
```


0
2
4
6
8
10
12
14
0
4
5
6
8
9
10
12
14
16
18

6.4 Pie Chart

The **pie chart** of a qualitative data sample consists of pizza wedges that shows the frequency distribution graphically.

Example

In the data set `painters`, the pie chart of the `School` variable is a collection of pizza wedges showing the proportion of painters in each school.

Problem

Find the pie chart of the painter schools in the data set `painters`.

Solution

We first apply the `table` function to compute the frequency distribution of the `School` variable.

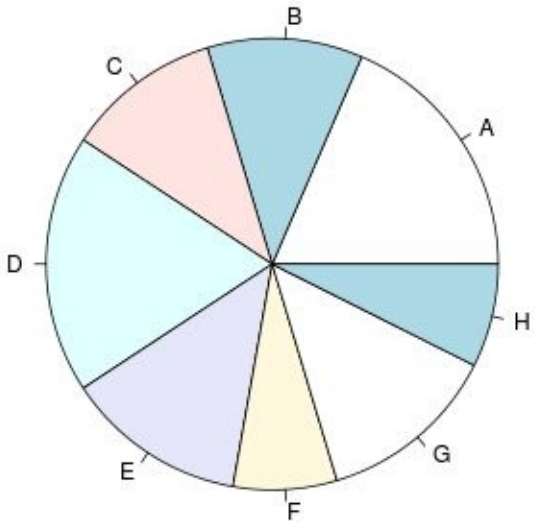
```
> school = painters$School
> school.freq = table(school)
```

Then we apply the function `pie` to produce its pie chart.

```
> pie(school.freq)
```

Answer

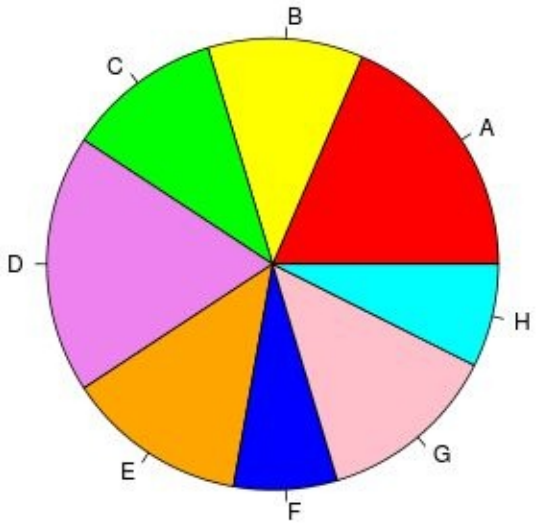
The pie chart of the school variable is:



Enhanced Solution

To colorize the pie chart, we can select a color palette and set it in the `col` argument of `pie`.

```
> colors = c("red", "yellow",
+   "green", "violet", "orange",
+   "blue", "pink", "cyan")
> pie(school.freq, col=colors)
```



Exercise

Find the pie chart of the composition scores in `painters`.

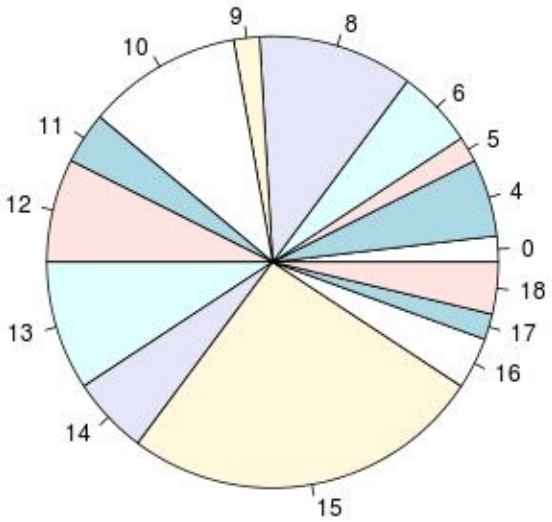
Solution of the Exercise

Step 1: Find the frequency distribution of the painter composition scores with the `table` function.

```
> comp = painters$Composition
> comp.freq = table(comp)
```

Step 2: Then apply the `pie` function.

```
> pie(comp.freq)
```


9
8
10
6
11
5
4
12
0
18
13
17
16
14
15

6.5 Category Statistics

In the data set `painters`, the painters are classified according to the schools they belong. Each school can be characterized by its various statistics, such as its *mean* composition, drawing, coloring or expression scores.

Suppose we would like to know which school has the highest mean composition score. We would have to first find out the mean composition score of each school. The following shows how to find the mean composition score of an arbitrarily chosen school.

Problem

Find out the mean composition score of school C in the data set `painters`.

Solution

Step 1: Create a logical index vector for school C.

```
> school = painters$School
> c_school = school == "C"
```

Step 2: Find the child data set for `painters` of school C. For explanation, please consult our previous tutorial on Data Frame Row Slice.

```
> c_painters = painters[c_school, ]
```

Step 3: Find the mean composition score of school C.

```
> mean(c_painters$Composition)
[1] 13.167
```

Answer

The mean composition score of school C is 13.167.

Alternative Solution

Instead of computing the mean composition score manually for each school, use the function `tapply` to compute them all at once.

```
> tapply(painters$Composition,
+     painters$School, mean)
     A      B      C      D      E
10.400 12.167 13.167  9.100 13.571
     F      G      H
 7.250 13.857 14.000
```

Exercise 1

Find programmatically the school with the highest composition scores.

Solution of Exercise 1

Step 1: Compute the highest composition score of each school with `tapply` and save it in a vector `comp.max`.

```
> comp = painters$Composition
> school = painters$School

> comp.school.max =
+     tapply(comp, school, max)
> comp.school.max
 A  B  C  D  E  F  G  H
17 15 16 15 18  9 18 16
```

Step 2: Find the highest composition score in the data set `painters` with the `max` function.

```
> comp.max.all = max(comp)
> comp.max.all
[1] 18
```

Step 3: Use the function `which` to find those schools whose highest composition score match `comp.max.all`.

```
> x = which(
+     comp.school.max == comp.max.all)
```

Step 4: Print out the names of schools in `x` with the function `names`.

```
> names(x)
```

```
[1] "E" "G"
```

Exercise 2

Find the percentage of painters whose color score is equal to or above 14.

Solution of Exercise 2

Step 1: Use the function `which` to find out painters whose color score is equal to or above 14.

```
> colour = painters$Colour
> x = which(colour >= 14)
```

Step 2: Divide the length of `x` by the number of painters.

```
> length(x)/nrow(painters)
[1] 0.37037
```

Therefore, the percentage of painters with color score 14 or above is about 37%.

Chapter 7
Quantitative Data

Quantitative data, also known as **continuous data**, consists of numeric data that support arithmetic operations. This is in contrast with qualitative data, whose values belong to pre-defined classes with no intrinsic arithmetic operations. We will explain how to apply some of the R tools for quantitative data analysis with examples.

The tutorials in this section are based on a built-in data frame named **faithful**. It consists of a collection of observations of the Old Faithful geyser in the USA Yellowstone National Park. The following is a peek of the data set via the `head` function.

```
> head(faithful)
  eruptions waiting
1     3.600      79
2     1.800      54
3     3.333      74
4     2.283      62
5     4.533      85
6     2.883      55
```

There are two observation variables in the data set. The first one, called `eruptions`, chronicles the duration of individual geyser eruptions. The second one, called `waiting`, is the length of waiting period until the eruption. It turns out there is a correlation between the two variables, as will be seen in a later tutorial

on *Scatter Plot*.

7.1 Frequency Distribution

The **frequency distribution** of a data variable is a summary of the data occurrence in a collection of non-overlapping categories.

Example

In the data set `faithful`, the frequency distribution of the `eruptions` variable is a summary of eruptions classified according to their durations.

Problem

Find the frequency distribution of the eruption durations in `faithful`.

Solution

Step 1: We first find the range of eruption durations with the function `range`. It shows that the observed eruption durations are all between 1.6 and 5.1 minutes.

```
> duration = faithful$eruptions
> range(duration)
[1] 1.6 5.1
```

Step 2: Break the range into non-overlapping sub-intervals by defining a sequence of equal distance break points. If we round the endpoints of the interval `[1.6, 5.1]` to the closest half-integers, we come up with the interval `[1.5, 5.5]`. Hence we set the break points to be the half-integer sequence { 1.5, 2.0, 2.5, … }.

```
> breaks = seq(1.5, 5.5, by=0.5)
> breaks
[1] 1.5 2.0 2.5 3.0 3.5 4.0 4.5 ...
```

Step 3: Classify the eruption durations according to the half-unit-length sub-intervals with the function `cut`. As the intervals are to be closed on the left, and open on the right, we set the `right` argument as `FALSE`.

```
> duration.cut = cut(
```

```
+     duration, breaks, right=FALSE)
```

Step 4: Compute the frequency of eruptions in each sub-interval with the `table` function.

```
> duration.freq = table(duration.cut)
```

Answer

The frequency distribution of the eruption duration is:

```
> duration.freq
duration.cut
[1.5,2) [2,2.5) [2.5,3) [3,3.5)
     51      41       5       7
[3.5,4) [4,4.5) [4.5,5) [5,5.5)
     30      73      61       4
```

Enhanced Solution

We can apply the function `cbind` to print the result vertically in a single column.

```
> cbind(duration.freq)
        duration.freq
[1.5,2)            51
[2,2.5)            41
[2.5,3)             5
[3,3.5)             7
[3.5,4)            30
[4,4.5)            73
[4.5,5)            61
[5,5.5)             4
```

Note

Per R documentation, for performance reasons, you are advised to use the function `hist` to find the frequency distribution. With this approach, the break points become optional, and they do not have to be figured out beforehand. Your work will be much simpler.

Exercise 1

Find the frequency distribution of the eruption waiting periods in the data set

`faithful`.

Solution of Exercise 1

Step 1: Use the `range` function to find the range of waiting period.

```
> waiting = faithful$waiting
> range(waiting)
[1] 43 96
```

Step 2: Round the interval endpoints as `[40, 100]` and setup break points 5 units apart.

```
> breaks = seq(40, 100, by=5)
> breaks
 [1]  40  45  50  55  60  65  70 ...
```

Step 3: Classify the `waiting` values with `cut` function.

```
> waiting.cut = cut(
+     waiting, breaks, right=FALSE)
```

Step 4: Compute the frequency of waiting periods in each waiting class with the `table` function.

```
> waiting.freq = table(waiting.cut)
```

Step 5: Format and print with `cbind` function.

```
> cbind(waiting.freq)
         waiting.freq
[40,45)             1
[45,50)            20
[50,55)            32
[55,60)            24
[60,65)            17
[65,70)             9
[70,75)            23
[75,80)            54
[80,85)            57
[85,90)            23
[90,95)            11
[95,100)            1
```

Exercise 2

Find programmatically the duration sub-interval that has the most eruptions.

Solution of Exercise 2

Step 1: Find the frequency distribution of `duration` as before.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
> duration.freq =
+     table(duration.cut)
```

Step 2: Find the maximum value in `duration.freq`.

```
> duration.freq.max =
+     max(duration.freq)
> duration.freq.max
[1] 73
```

Step 3: Find the member in `duration.freq` that has the maximum value and print out its name.

```
> x = which(
+     duration.freq ==
+     duration.freq.max)
> names(x)
[1] "[4,4.5)"
```

Therefore, the duration period between 4 and 4.5 minutes has the most eruptions.

Exercise 3

Find the frequency distribution of eruption durations in `faithful` using the function `hist`.

Solution of Exercise 3

Step 1: Apply the function `hist` to the `duration` variable with the specified break points and extract the frequency distribution.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
```

```
> h = hist(duration, breaks=breaks,
+     right=FALSE, plot=FALSE)
> duration.freq = h$counts
```

Incidentally, the parameter `breaks` is optional. Hence we can relies on the default break points, and retrieve the frequency distribution with the following shortcut:

```
> h = hist(duration,
+     right=FALSE, plot=FALSE)
> duration.freq = h$counts
> duration.freq
[1] 51 41  5  7 30 73 61  4
```

Step 2 (Optional): Create labels for the sub-intervals terminated by the break points.

```
> len = length(h$breaks)
> a = h$breaks[1:len-1]  # left
> b = h$breaks[2:len]    # right
> labels = paste(
+     "[", a, ", ", b, ")", sep="")
```

Step 3 (Optional): Assign `labels` as the names of the duration frequency in `duration.freq`, and format the result with `cbind`.

```
> names(duration.freq) = labels
> cbind(duration.freq)
         duration.freq
[1.5, 2)            51
[2, 2.5)            41
[2.5, 3)             5
[3, 3.5)             7
[3.5, 4)            30
[4, 4.5)            73
[4.5, 5)            61
[5, 5.5)             4
```

7.2 Histogram

A **histogram** consists of parallel vertical bars that graphically shows the frequency distribution of a quantitative variable. The area of each bar is proportional to the frequency of items found in each class.

Example

In the data set `faithful`, the histogram of the `eruptions` variable is a collection of parallel vertical bars showing the number of eruptions classified according to their durations.

Problem

Find the histogram of the eruption durations in the data set `faithful`.

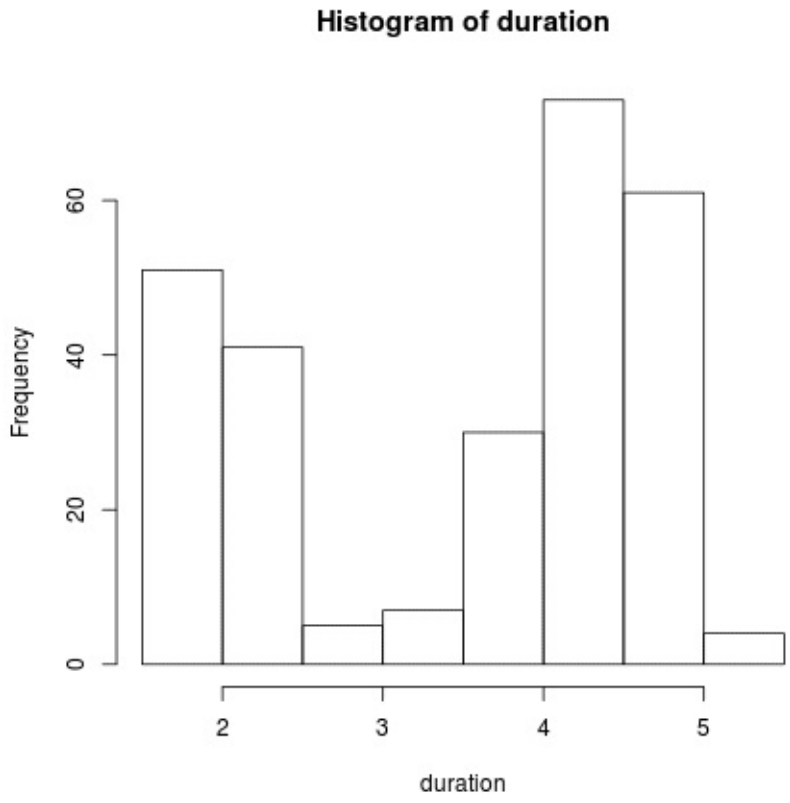
Solution

We apply the `hist` function to produce the histogram of the `eruptions` variable.

```
> duration = faithful$eruptions
> hist(duration,   # apply hist
+   right=FALSE)   # closed on left
```

Answer

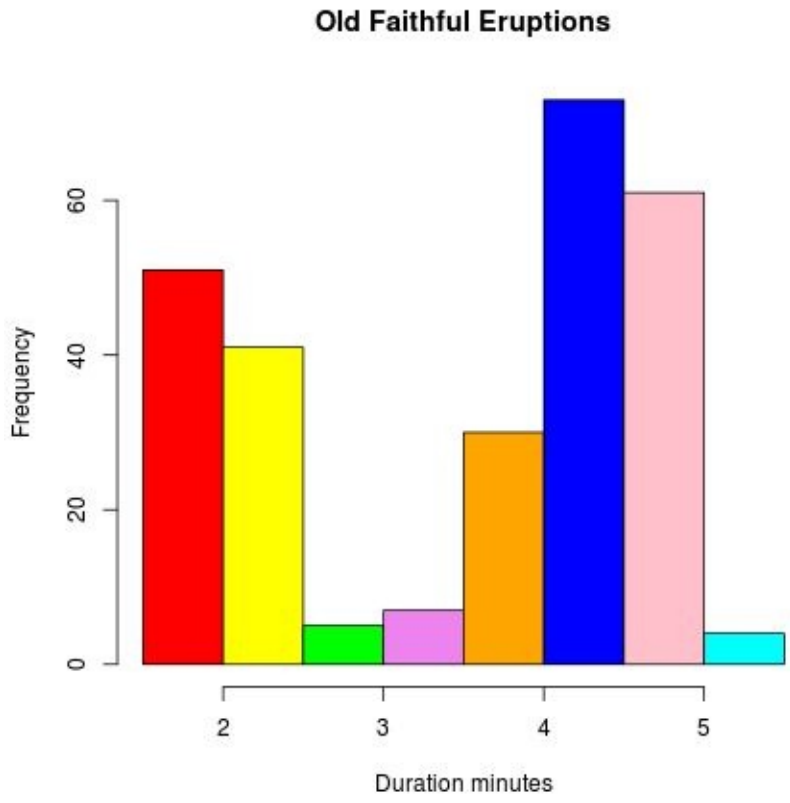
The histogram of the eruption durations is:



Enhanced Solution

To colorize the histogram, we select a color palette and set it in the `col` argument of `hist`. In addition, we select descriptive labels for proper documentation.

```
> colors = c("red", "yellow",
+     "green", "violet", "orange",
+     "blue", "pink", "cyan")
> hist(duration,   # apply hist
+   right=FALSE,   # closed on left
+   col=colors,    # color palette
+   main="Old Faithful Eruptions",
+   xlab="Duration minutes")
```



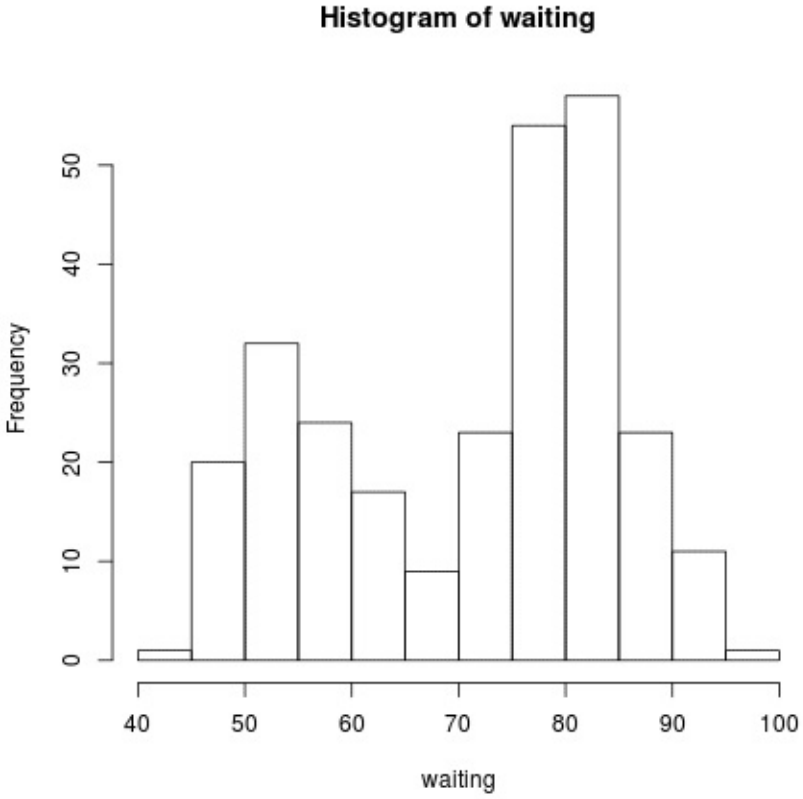
Exercise

Find the histogram of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `hist` function to the `waiting` variable.

```
> waiting = faithful$waiting
> hist(waiting, right=FALSE)
```


Histogram of waiting
Frequency
0
10
20
30
40
50
40
50
60
70
80
90
100
waiting

7.3 Relative Frequency Distribution

The **relative frequency distribution** of a data variable is a summary of the frequency proportion in a collection of non-overlapping categories.

The relationship of frequency and relative frequency is:

$$Relative\ Frequency = \frac{Frequency}{Sample\ Size}$$

Example

In the data set `faithful`, the relative frequency distribution of the `eruptions` variable shows the frequency proportion of the eruptions according to a duration classification.

Problem

Find the relative frequency distribution of the eruption durations in `faithful`.

Solution

We first find the frequency distribution of the eruption durations as follows. Check the previous tutorial on Frequency Distribution for details.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
> duration.freq =
+     table(duration.cut)
```

Then we find the sample size of `faithful` with the function `nrow`, and divide the frequency distribution with it. As a result, the relative frequency distribution is:

```
> duration.relfreq =
+     duration.freq / nrow(faithful)
```

Answer

The frequency distribution of the eruption variable is:

```
> duration.relfreq
duration.cut
 [1.5,2)  [2,2.5)  [2.5,3)  [3,3.5)
0.187500 0.150735 0.018382 0.025735
 [3.5,4)  [4,4.5)  [4.5,5)  [5,5.5)
0.110294 0.268382 0.224265 0.014706
```

Enhanced Solution

We can print with fewer digits and make it more readable by setting the `digits` option.

```
> old = options(digits=1)
> duration.relfreq
duration.cut
[1.5,2) [2,2.5) [2.5,3) [3,3.5)
   0.19    0.15    0.02    0.03
[3.5,4) [4,4.5) [4.5,5) [5,5.5)
   0.11    0.27    0.22    0.01
> options(old)    # restore
```

We then apply the function `cbind` to print both the frequency distribution and relative frequency distribution in parallel columns. We can even multiply the relative frequency by 100 for the more familiar percentage representation.

```
> duration.percentage =
+     duration.relfreq * 100

> old = options(digits=3)
> cbind(duration.freq,
+     duration.percentage)
    duration.freq duration.percentage
[1.5,2)        51               18.75
[2,2.5)        41               15.07
[2.5,3)         5                1.84
[3,3.5)         7                2.57
[3.5,4)        30               11.03
[4,4.5)        73               26.84
[4.5,5)        61               22.43
[5,5.5)         4                1.47
> options(old)
```

Exercise

Find the relative frequency distribution of the eruption waiting periods in `faithful`.

Solution of Exercise

We first compute the relative frequency distribution of `waiting`.

```
> waiting = faithful$waiting
> breaks = seq(40, 100, by=5)
> waiting = cut(
+     waiting, breaks, right=FALSE)
> waiting.freq = table(waiting)
```

Then we divide the frequency distribution of `waiting` by the sample size and find the relative frequency distribution percentages.

```
> waiting.relfreq =
+     waiting.freq / nrow(faithful)
> waiting.percentage =
+     100 * waiting.relfreq
```

Now use `cbind` to print the result.

```
> old = options(digits=1)
> cbind(waiting.freq,
+     waiting.percentage)
     waiting.freq waiting.percentage
[40,45)         1                0.4
[45,50)        20                7.4
[50,55)        32               11.8
[55,60)        24                8.8
[60,65)        17                6.2
[65,70)         9                3.3
[70,75)        23                8.5
[75,80)        54               19.9
[80,85)        57               21.0
[85,90)        23                8.5
[90,95)        11                4.0
[95,100)        1                0.4
> options(old)
```

7.4 Cumulative Frequency Distribution

The **cumulative frequency distribution** of a quantitative variable is a summary of data frequency below given levels.

Example

In the data set `faithful`, the cumulative frequency distribution of the `eruptions` variable shows the *total* number of eruptions whose durations are less than or equal to a set of chosen levels.

Problem

Find the cumulative frequency distribution of the eruption durations in `faithful`.

Solution

We first find the frequency distribution of the eruption durations as follows. Check the previous tutorial on Frequency Distribution for details.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
> duration.freq =
+     table(duration.cut)
```

We then apply the `cumsum` function to compute the cumulative frequency distribution.

```
> duration.cumfreq =
+     cumsum(duration.freq)
```

Answer

The cumulative distribution of the eruption duration is:

```
> duration.cumfreq
[1.5,2) [2,2.5) [2.5,3) [3,3.5)
     51      92      97     104
[3.5,4) [4,4.5) [4.5,5) [5,5.5)
    134     207     268    272
```

Enhanced Solution

We apply the `cbind` function to print the result in column format.

```
> cbind(duration.cumfreq)
        duration.cumfreq
[1.5,2)               51
[2,2.5)               92
[2.5,3)               97
[3,3.5)              104
[3.5,4)              134
[4,4.5)              207
[4.5,5)              268
[5,5.5)              272
```

Exercise

Find the cumulative frequency distribution of the eruption waiting periods in `faithful`.

Solution of Exercise

Apply `cumsum` to the frequency distribution of `waiting`.

```
> waiting = faithful$waiting
> breaks = seq(40, 100, by=5)
> waiting.cut = cut(
+     waiting, breaks, right=FALSE)

> waiting.freq = table(waiting.cut)
> waiting.cumfreq =
+     cumsum(waiting.freq)

> cbind(waiting.cumfreq)
         waiting.cumfreq
[40,45)                1
[45,50)               21
[50,55)               53
[55,60)               77
[60,65)               94
```

[65,70)	103
[70,75)	126
[75,80)	180
[80,85)	237
[85,90)	260
[90,95)	271
[95,100)	272

7.5 Cumulative Frequency Graph

A **cumulative frequency graph** or **ogive** of a quantitative variable is a curve graphically showing the cumulative frequency distribution.

Example

In the data set `faithful`, a point in the cumulative frequency graph of the `eruptions` variable shows the *total* number of eruptions whose durations are less than or equal to a given level.

Problem

Find the cumulative frequency graph of the eruption durations in `faithful`.

Solution

We first find the frequency distribution of the eruption durations as follows. Check the previous tutorial on Frequency Distribution for details.

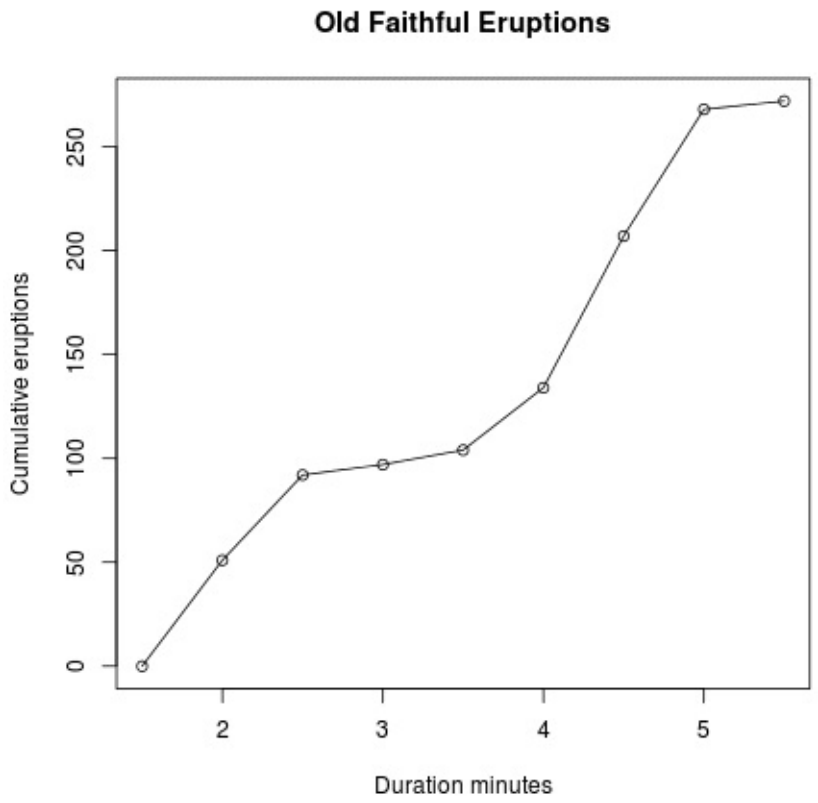
```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
> duration.freq =
+     table(duration.cut)
```

We then compute its cumulative frequency with `cumsum`, add a starting zero element, and plot the graph.

```
> cumfreq0 = c(0,
+     cumsum(duration.freq))
> plot(breaks, cumfreq0,
+   main="Old Faithful Eruptions",
+   xlab="Duration minutes",
+   ylab="Cumulative eruptions")
> lines(breaks, cumfreq0)
```

Answer

The cumulative frequency graph of the eruption durations is:



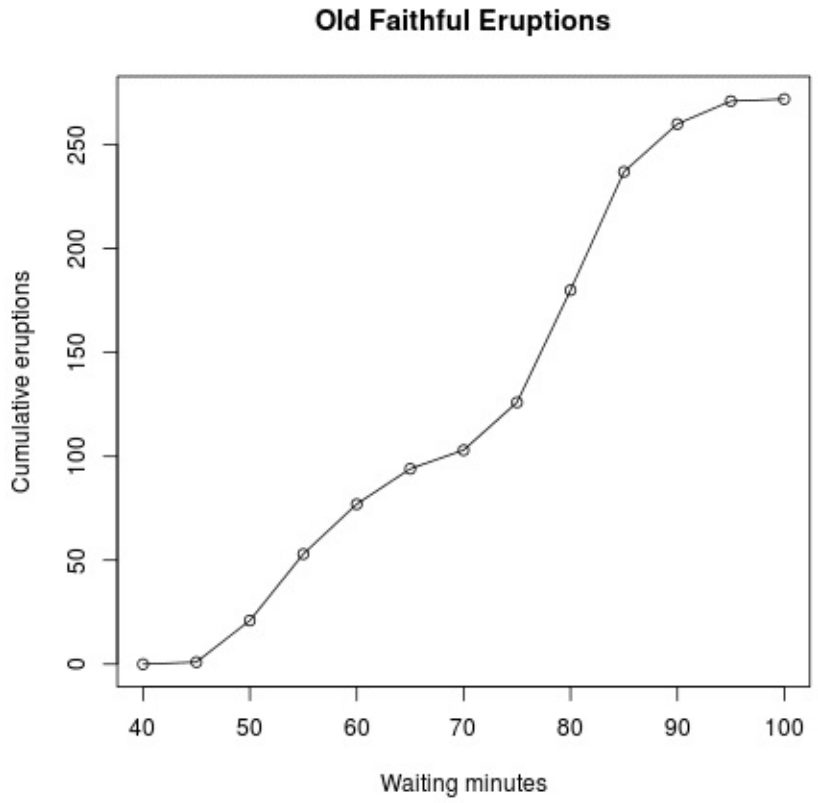
Exercise

Find the cumulative frequency graph of the eruption waiting periods in `faithful`.

Solution of Exercise

```
> waiting = faithful$waiting
> breaks = seq(40, 100, by=5)
> waiting.cut = cut(
+   waiting, breaks, right=FALSE)
>
> waiting.freq = table(waiting.cut)
> waiting.cumfreq0 =
+   c(0, waiting.cumfreq)
>
> plot(breaks, waiting.cumfreq0,
```

```
+   main="Old Faithful Eruptions",
+   xlab="Waiting minutes",
+   ylab="Cumulative eruptions")
> lines(breaks, waiting.cumfreq0)
```



7.6 Cumulative Relative Frequency Distribution

The **cumulative relative frequency distribution** of a quantitative variable is a summary of frequency proportion below given levels.

The relationship between cumulative frequency and relative cumulative frequency is:

$$Cumulative\ Relative\ Frequency = \frac{Cumulative\ Frequency}{Sample\ Size}$$

Example

In the data set `faithful`, the cumulative relative frequency distribution of the `eruptions` variable shows the frequency proportion of eruptions whose durations are less than or equal to a set of chosen levels.

Problem

Find the cumulative relative frequency distribution of the eruption durations in `faithful`.

Solution

We first find the frequency distribution of the eruption durations as follows. Check the previous tutorial on Cumulative Frequency Distribution for details.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
>
> duration.freq =
+     table(duration.cut)
> duration.cumfreq =
+     cumsum(duration.freq)
```

Then we find the sample size of `faithful` with the `nrow` function, and divide the

cumulative frequency distribution with it. Hence the cumulative relative frequency distribution is:

```
> duration.cumrelfreq =
+     duration.cumfreq/nrow(faithful)
```

Answer

The cumulative relative frequency distribution of the eruption variable is:

```
> duration.cumrelfreq
[1.5,2) [2,2.5) [2.5,3) [3,3.5)
0.18750 0.33824 0.35662 0.38235
[3.5,4) [4,4.5) [4.5,5) [5,5.5)
0.49265 0.76103 0.98529 1.00000
```

Enhanced Solution

We can print with fewer digits and make it more readable by setting the `digits` option.

```
> old = options(digits=2)
> duration.cumrelfreq
[1.5,2) [2,2.5) [2.5,3) [3,3.5)
   0.19    0.34    0.36    0.38
[3.5,4) [4,4.5) [4.5,5) [5,5.5)
   0.49    0.76    0.99    1.00
> options(old)    # restore
```

Exercise

Find the cumulative frequency distribution of the eruption waiting periods in `faithful`.

Solution of Exercise

```
> waiting = faithful$waiting
> breaks = seq(40, 100, by=5)
> waiting.cut = cut(
+       waiting, breaks, right=FALSE)

> waiting.freq = table(waiting.cut)
> waiting.cumfreq =
+     cumsum(waiting.freq)
```

```
>
> waiting.cumrelfreq =
+     waiting.cumfreq / nrow(faithful)
> waiting.cumpercent =
+       100 * waiting.cumrelfreq
>
> old = options(digits=3)
> cbind(waiting.cumfreq,
+       waiting.cumpercent)

    waiting.cumfreq waiting.cumpercent
[40,45)           1              0.368
[45,50)          21              7.721
[50,55)          53             19.485
[55,60)          77             28.309
[60,65)          94             34.559
[65,70)         103             37.868
[70,75)         126             46.324
[75,80)         180             66.176
[80,85)         237             87.132
[85,90)         260             95.588
[90,95)         271             99.632
[95,100)        272            100.000
> options(old)
```

7.7 Cumulative Relative Frequency Graph

A **cumulative relative frequency graph** of a quantitative variable is a curve graphically showing the cumulative relative frequency distribution.

Example

In the data set `faithful`, a point in the cumulative relative frequency graph of the `eruptions` variable shows the frequency proportion of eruptions whose durations are less than or equal to a given level.

Problem

Find the cumulative relative frequency graph of the eruption durations in `faithful`.

Solution

We first find the cumulative relative frequency distribution of the eruption durations as follows. Check the previous tutorial on Cumulative Relative Frequency Distribution for details.

```
> duration = faithful$eruptions
> breaks = seq(1.5, 5.5, by=0.5)
> duration.cut = cut(
+     duration, breaks, right=FALSE)
> duration.freq =
+     table(duration.cut)
> duration.cumfreq =
+     cumsum(duration.freq)
> duration.cumrelfreq =
+     duration.cumfreq/nrow(faithful)
```

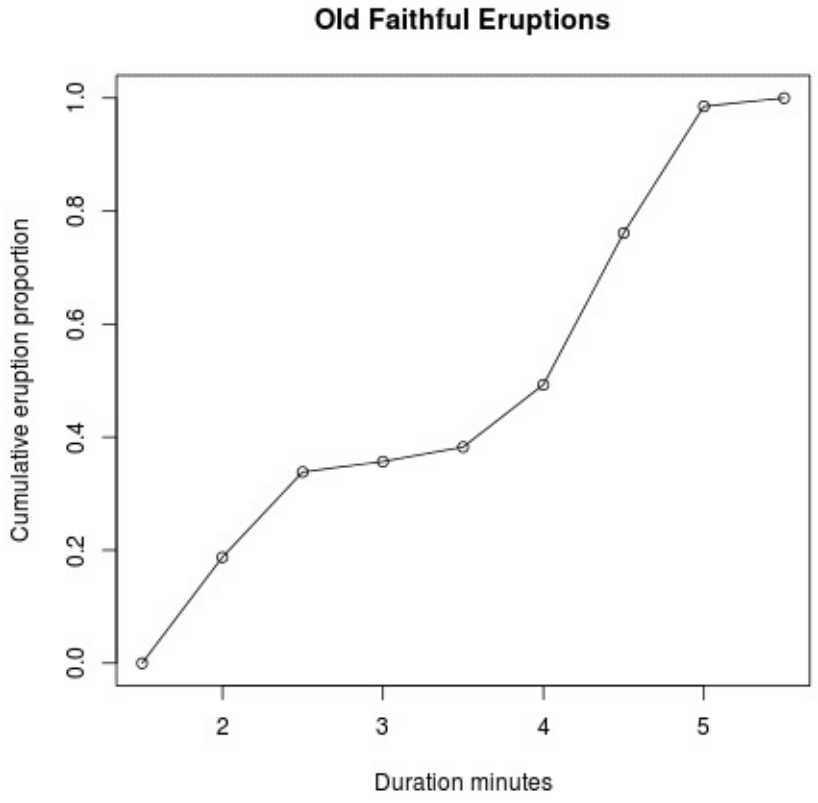
We then plot it along with the starting zero element.

```
> cumrelfreq0 =
+   c(0, duration.cumrelfreq)
> plot(breaks, cumrelfreq0,
```

```
+   main="Old Faithful Eruptions",
+   xlab="Duration minutes",
+   ylab="Cumulative eruption proportion")
> lines(breaks, cumrelfreq0)
```

Answer

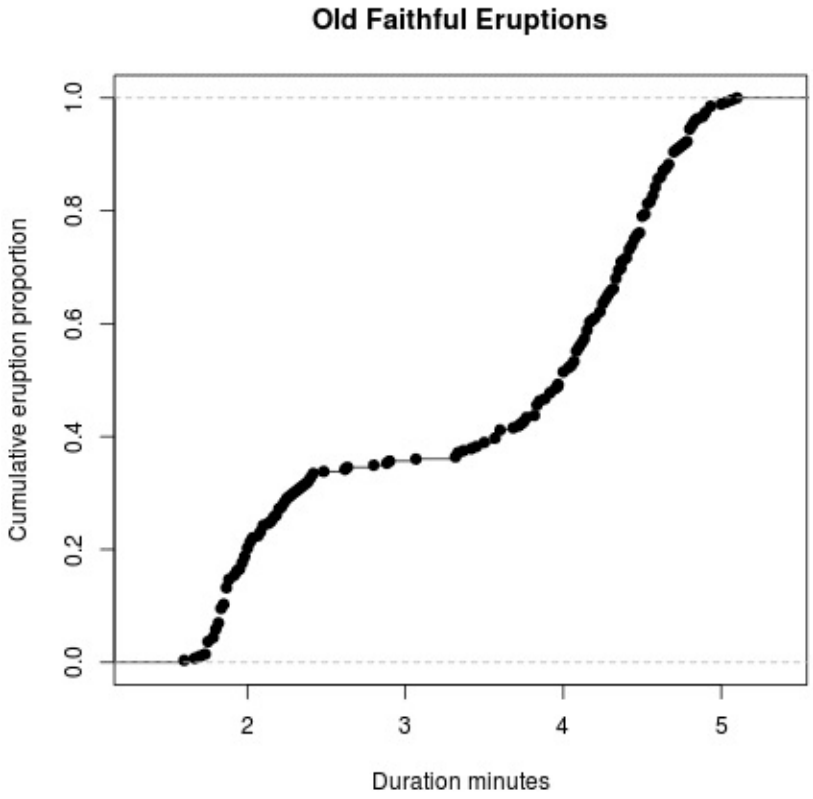
The cumulative relative frequency graph of the eruption duration is:



Alternative Solution

We create an interpolate function `Fn` with the built-in function `ecdf`. Then we plot `Fn` right away. There is no need to compute the cumulative frequency distribution *a priori*.

```
> Fn = ecdf(duration)
> plot(Fn,
+   main="Old Faithful Eruptions",
+   xlab="Duration minutes",
+   ylab="Cumulative eruption proportion")
```



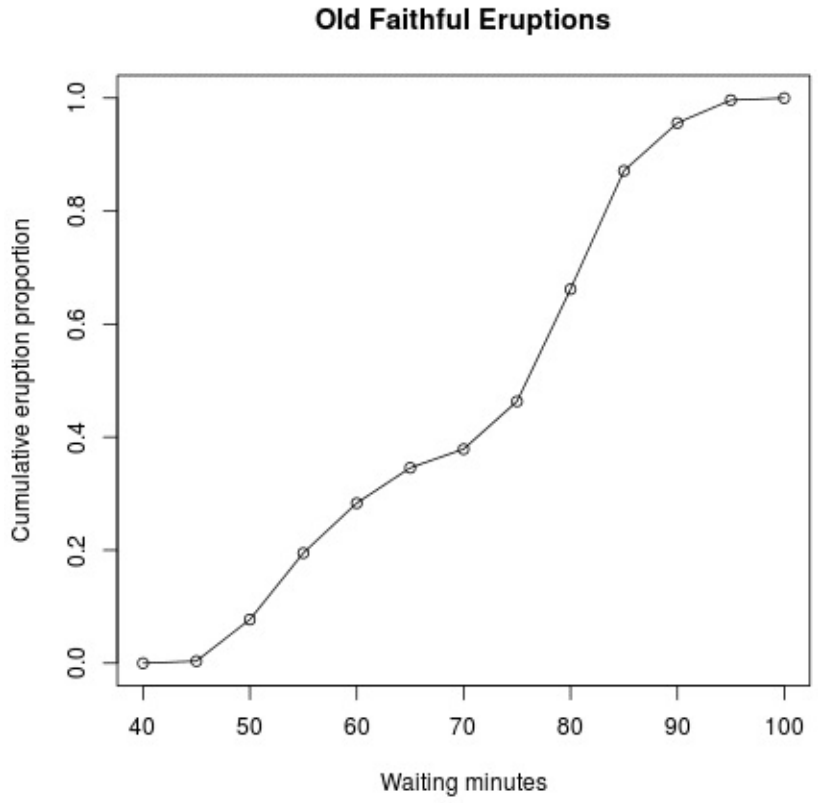
Exercise

Find the cumulative relative frequency graph of the eruption waiting periods in `faithful`.

Solution of Exercise

```
> waiting = faithful$waiting
> breaks = seq(40, 100, by=5)
> waiting.cut = cut(
+     waiting, breaks, right=FALSE)
>
> waiting.freq = table(waiting.cut)
> waiting.cumfreq =
+     cumsum(waiting.freq)
>
> waiting.cumrelfreq =
+     waiting.cumfreq/nrow(faithful)
> cumrelfreq0 =
+     c(0, waiting.cumrelfreq)
>
```

```
> plot(breaks, cumrelfreq0,
+   main="Old Faithful Eruptions",
+   xlab="Waiting minutes",
+   ylab="Cumulative eruption proportion")
> lines(breaks, cumrelfreq0)
```



7.8 Stem-and-Leaf Plot

A **stem-and-leaf** plot of a quantitative variable is a textual graph that classifies data items according to their most significant numeric digits. In addition, we often merge each alternating row with its next row in order to simplify the graph for readability.

Example

In the data set `faithful`, a stem-and-leaf plot of the `eruptions` variable identifies durations with the same two most significant digits, and queue them up in rows.

Problem

Find the stem-and-leaf plot of the eruption durations in `faithful`.

Solution

We apply the function `stem` to compute the stem-and-leaf plot of `eruptions`.

Answer

The stem-and-leaf plot of the eruption durations is

```
> duration = faithful$eruptions
> stem(duration)

  The decimal point is 1 digit(s) to the left of the |

  16 | 070355555588
  18 | 000022233333335577777777888822335777888
  20 | 00002223378800035778
  22 | 0002335578023578
  24 | 00228
  26 | 23
  28 | 080
  30 | 7
```

```
  32 | 2337
  34 | 250077
  36 | 0000823577
  38 | 2333335582225577
  40 | 000000335778888800223355557778
  42 | 03335555778800233333555577778
  44 | 022223355577800000000233333577788888
  46 | 00002333577000000023578
  48 | 00000022335800333
  50 | 0370
```

Exercise

Find the stem-and-leaf plot of the eruption waiting periods in `faithful`.

Solution of Exercise

```
> stem(faithful$waiting)

  The decimal point is 1 digit(s) to the right of the |

  4 | 3
  4 | 55566666777788899999
  5 | 000001111112222233333333444444444
  5 | 555555666677788889999999
  6 | 00000022223334444
  6 | 555667899
  7 | 00001111123333333444444
      .........
```

7.9 Scatter Plot

A **scatter plot** pairs up values of two quantitative variables in a data set and display them as geometric points inside a Cartesian diagram.

Example

In the data set `faithful`, we pair up the `eruptions` and `waiting` values in the same observation as (x, y) coordinates. Then we plot the points in the Cartesian plane. Here is a peek of the eruption data value pairs with the help of the `cbind` function.

```
> duration = faithful$eruptions
> waiting = faithful$waiting
> head(cbind(duration, waiting))
     duration waiting
[1,]    3.600      79
[2,]    1.800      54
[3,]    3.333      74
[4,]    2.283      62
[5,]    4.533      85
[6,]    2.883      55
```

Problem

Find the scatter plot of the eruption durations and waiting intervals in `faithful`. Does it reveal any relationship between the variables?

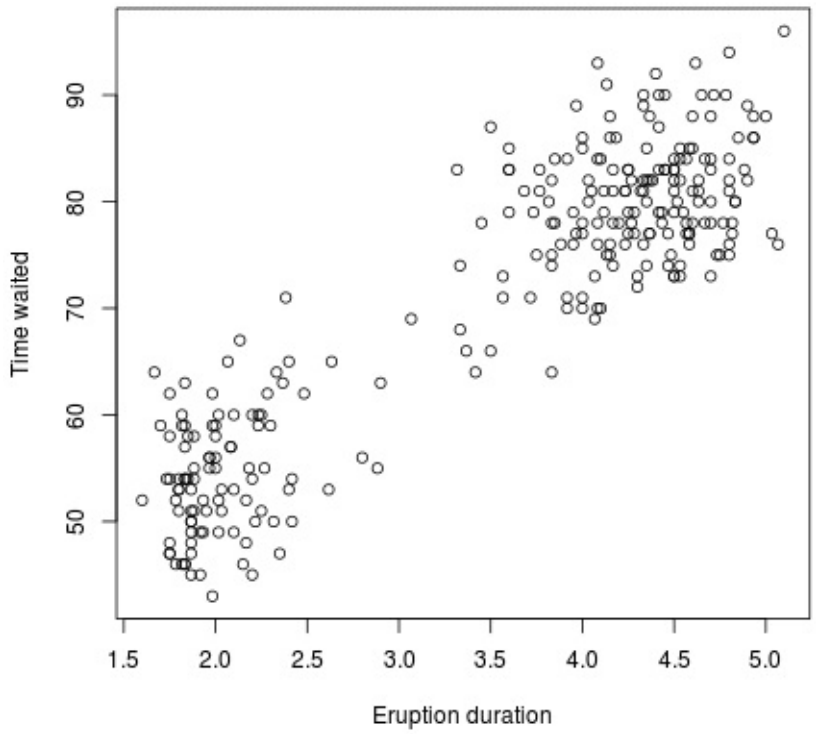
Solution

We apply the `plot` function to compute the scatter plot of `eruptions` and `waiting`.

```
> duration = faithful$eruptions
> waiting = faithful$waiting
> plot(duration, waiting,
+   xlab="Eruption duration",
+   ylab="Time waited")
```

Answer

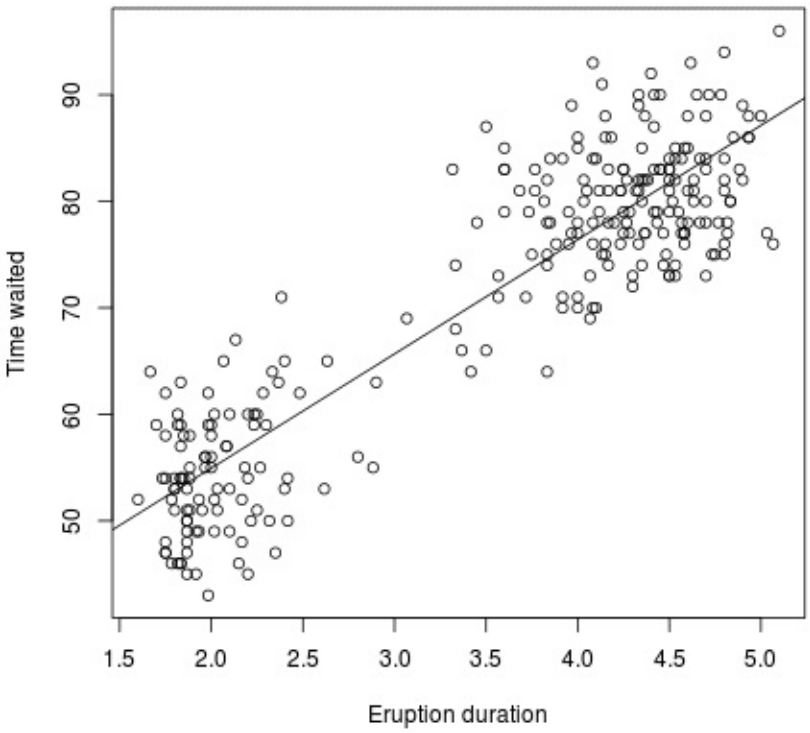
The scatter plot of the eruption durations and waiting intervals is as follows. It reveals a *positive linear relationship* between them.



Enhanced Solution

We can generate a linear regression model of the two variables with the function `lm`, and then draw a trend line with `abline`.

```
> abline(lm(waiting ~ duration))
```


Time waited
50
60
70
80
90
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
Eruption duration

Chapter 8
Numerical Measures

In this chapter, we will explain how to compute various statistical measures in R with examples. The tutorials are based on the previously discussed built-in data set `faithful`.

8.1 Mean

The **mean** of an observation variable is a numerical measure of the central location of the data values. It is the sum of its data values divided by the data count.

Hence, for a data sample of size *n*, its **sample mean** is defined as follows:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

Similarly, for a data population of size *N*, the **population mean** is:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

Problem

Find the mean eruption duration in the data set `faithful`.

Solution

We apply the `mean` function to compute the mean value of `eruptions`.

```
> duration = faithful$eruptions
> mean(duration)
[1] 3.4878
```

Answer

The mean eruption duration is about 3.5 minutes.

Exercise

Find the mean eruption waiting periods in `faithful`.

Solution of Exercise

We apply the `mean` function to the `waiting` variable and have:

```
> waiting = faithful$waiting
> mean(waiting)
[1] 70.897
```

Hence the mean waiting time between eruptions is about 71 minutes.

8.2 Median

The **median** of an observation variable is the value in the middle when the data is sorted in ascending order. It is an ordinal measure of the central location of the data values.

Problem

Find the median of eruption duration in the data set `faithful`.

Solution

We apply the `median` function to compute the median value of `eruptions`.

```
> duration = faithful$eruptions
> median(duration)
[1] 4
```

Answer

The median of eruption duration is 4 minutes.

Exercise

Find the median of the eruption waiting periods in `faithful`.

Solution of Exercise

We apply the `median` function to the `waiting` variable and have:

```
> waiting = faithful$waiting
> median(waiting)
[1] 76
```

Hence the median waiting time between eruptions is about 76 minutes.

8.3 Quartile

There are several well-defined **quartiles** of an observation variable. The **first quartile**, or **lower quartile**, is the value that cuts off the first 25% of the data when it is sorted in ascending order. The **second quartile**, or **median**, is the value that cuts off the first 50%. The **third quartile**, or **upper quartile**, is the value that cuts off the first 75%.

Problem

Find the quartiles of eruption durations in the data set `faithful`.

Solution

We apply the `quantile` function to compute the quartiles of `eruptions`.

```
> duration = faithful$eruptions
> quantile(duration)
    0%    25%    50%    75%   100%
1.6000 2.1627 4.0000 4.4543 5.1000
```

Answer

The first, second and third quartiles of eruption duration are about 2.2, 4.0 and 4.5 minutes respectively.

Exercise

Find the quartiles of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `quantile` function to the `waiting` variable and have:

```
> waiting = faithful$waiting
> quantile(waiting)
  0%  25%  50%  75% 100%
```

```
  43    58    76    82    96
```

Hence the first, second and third quartiles of eruption waiting period are 58, 76 and 82 minutes.

Note

There are several algorithms for the computation of quartiles. Details can be found in the R documentation via `help(quantile)`

8.4 Percentile

The n^{th} **percentile** of an observation variable is the value that cuts off the first n percent of the data values when it is sorted in ascending order.

Problem

Find the 32^{nd}, 57^{th} and 98^{th} percentiles of eruption durations in the data set `faithful`.

Solution

We apply the `quantile` function to compute the percentiles of `eruptions` with the desired percentage ratios.

```
> duration = faithful$eruptions
> quantile(duration, c(.32, .57, .98))
   32%    57%    98%
2.3952 4.1330 4.9330
```

Answer

The 32^{nd}, 57^{th} and 98^{th} percentiles of eruption duration are 2.3952, 4.1330 and 4.9330 minutes respectively.

Exercise

Find the 17^{th}, 43^{rd}, 67^{th} and 85^{th} percentiles of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `quantile` function to the `waiting` variable with the desired percentage ratios and have:

```
> waiting = faithful$waiting
```

```
> quantile(waiting,
+     c(.17, .43, .67, .85))
17% 43% 67% 85%
 54  73  80  84
```

Hence the 17th, 43rd, 67th and 85th percentiles of the eruption waiting period are 54, 73, 80 and 84 minutes respectively.

8.5 Range

The **range** of an observation variable is the difference of its largest and smallest data values. It is a measure of how far apart the data spreads in value.

$$Range = Largest\ Value - Smallest\ Value$$

Problem

Find the range of eruption duration in the data set `faithful`.

Solution

We apply the `max` and `min` functions to compute the largest and smallest values of `eruptions`, and then we take their difference.

```
> duration = faithful$eruptions
> max(duration) - min(duration)
[1] 3.5
```

Answer

The range of eruption duration is 3.5 minutes.

Exercise

Find the range of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `max` and `min` functions to the `waiting` variable and have:

```
> waiting = faithful$waiting
> max(waiting) - min(waiting)
[1] 53
```

Hence the range of eruption waiting period is 53 minutes.

8.6 Interquartile Range

The **interquartile range** of an observation variable is the difference between its upper and lower quartiles. It is a measure of how far apart the middle portion of data spreads in value.

$$Interquartile\ Range = Upper\ Quartile - Lower\ Quartile$$

Problem

Find the interquartile range of eruption duration in the data set `faithful`.

Solution

We apply the `IQR` function to compute the interquartile range of `eruptions`.

```
> duration = faithful$eruptions
> IQR(duration)
[1] 2.2915
```

Answer

The interquartile range of eruption duration is about 2.3 minutes.

Exercise

Find the interquartile range of eruption waiting period in `faithful`.

Solution of Exercise

We apply the `IQR` function to the `waiting` variable and have:

```
> waiting = faithful$waiting
> IQR(waiting)
[1] 24
```

Hence the interquartile range of eruption waiting period is 24 minutes.

8.7 Box Plot

The **box plot** of an observation variable is a graphical representation based on its quartiles as well as its smallest and largest values. It is a simple yet effective visual representation of data distribution.

Problem

Find the box plot of eruption duration in the data set `faithful`.

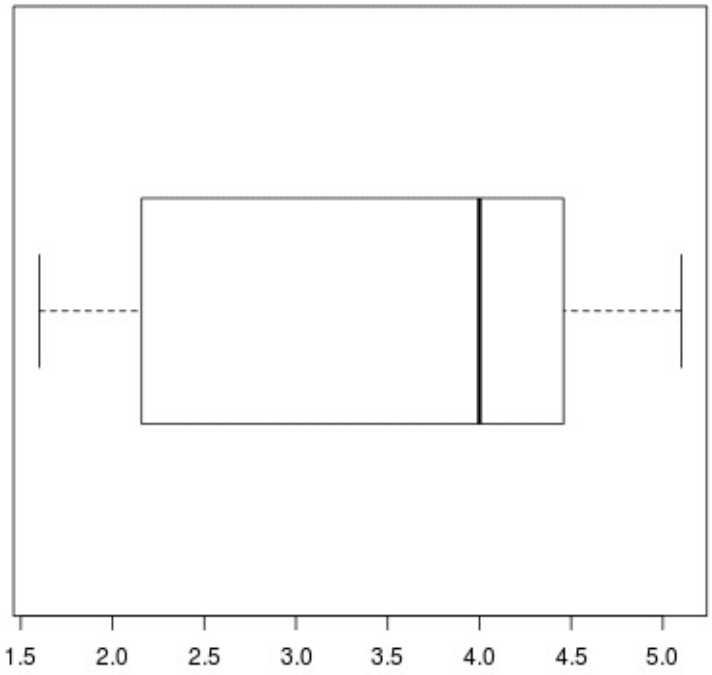
Solution

We apply the `boxplot` function to reproduce the box plot of `eruptions`.

```
> duration = faithful$eruptions
> boxplot(duration, horizontal=TRUE)
```

Answer

The box plot of eruption durations is:



Exercise

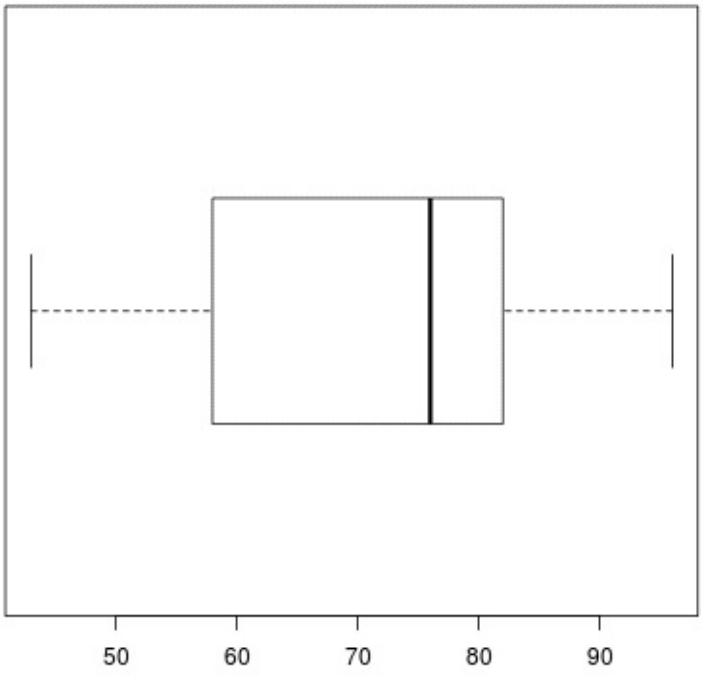
Find the box plot of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `boxplot` function to the `waiting` variable and have:

```
> waiting = faithful$waiting
> boxplot(waiting, horizontal=TRUE)
```

Hence the box plot of eruption waiting period is:


50
60
70
80
90

8.8 Variance

The `variance` is a numerical measure of how the data values is dispersed around the mean. In particular, the **sample variance** is defined as:

$$s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Similarly, the **population variance** is defined in terms of the population mean μ and population size N:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

Problem

Find the variance of eruption duration in the data set `faithful`.

Solution

We apply the `var` function to compute the variance of `eruptions`.

```
> duration = faithful$eruptions
> var(duration)
[1] 1.3027
```

Answer

The variance of eruption duration is about 1.3.

Exercise

Find the variance of the eruption waiting period in `faithful`.

Solution of Exercise

We apply the `var` function to the `waiting` variable and we have:

```
> waiting = faithful$waiting
> var(waiting)
[1] 184.82
```

Hence the variance of the eruption waiting period is about 184.

8.9 Standard Deviation

The `standard deviation` is the square root of its variance.

Problem

Find the standard deviation of eruption duration in the data set `faithful`.

Solution

We apply the `sd` function to compute the standard deviation of `eruptions`.

```
> duration = faithful$eruptions
> sd(duration)
[1] 1.1414
```

Answer

The standard deviation of eruption duration is about 1.1.

Exercise

Find the standard deviation of eruption waiting period in `faithful`.

Solution of Exercise

We apply the `sd` function to the `waiting` variable and we have:

```
> waiting = faithful$waiting
> sd(waiting)
[1] 13.595
```

Hence the standard deviation of eruption waiting period is about 13.6.

8.10 Covariance

The `covariance` of two variables *x* and *y* in a data set measures how the two are linearly related. A positive covariance would indicate a positive linear relationship between the variables, and a negative covariance would indicate the opposite.

The **sample covariance** is defined in terms of the sample means as:

$$s_{xy} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

Similarly, the **population covariance** is defined in terms of the population means μ_x and μ_y as:

$$\sigma_{xy} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_x)(y_i - \mu_y)$$

Problem

Find the covariance of eruption duration and waiting time in the data set `faithful`. Observe if there is any linear relationship between the two variables.

Solution

We apply the `cov` function to compute the covariance of `eruptions` and `waiting`.

```
> duration = faithful$eruptions
> waiting = faithful$waiting
> cov(duration, waiting)
[1] 13.978
```

Answer

The covariance of eruption duration and waiting time is about 14. It indicates a positive linear relationship between the two variables.

8.11 Covariance Coefficient

The **correlation coefficient** of two variables in a data set equals to their covariance divided by the product of their individual standard deviations. It is a normalized measurement of how the two are linearly related.

Formally, the **sample correlation coefficient** is defined by the following formula, where s_x and s_y are the sample standard deviations, and s_{xy} is the sample covariance.

$$r_{xy} = \frac{s_{xy}}{s_x s_y}$$

Similarly, the **population correlation coefficient** is defined as follows, where σ_x and σ_y are the population standard deviations, and σ_{xy} is the population covariance.

$$\rho_{xy} = \frac{\sigma_{xy}}{\sigma_x \sigma_y}$$

If the correlation coefficient is close to 1, it would indicate that the variables are positively linearly related and the scatter plot falls almost along a straight line with positive slope. For -1, it indicates that the variables are negatively linearly related and the scatter plot almost falls along a straight line with negative slope. And for zero, it would indicate a weak linear relationship between the variables.

Problem

Find the correlation coefficient of eruption duration and waiting time in the data set `faithful`. Observe if there is any linear relationship between the variables.

Solution

We apply the `cor` function to compute the correlation coefficient of `eruptions` and `waiting`.

```
> duration = faithful$eruptions
> waiting = faithful$waiting
```

```
> cor(duration, waiting)
[1] 0.90081
```

Answer

The correlation coefficient of eruption duration and waiting time is 0.90081. Since it is rather close to 1, we can conclude that the variables are positively linearly related.

8.12 Central Moment

The k^{th} **central moment** (or moment about the mean) of a data population is:

$$\mu_k = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^k$$

Similarly, the k^{th} central moment of a data sample is:

$$m_k = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^k$$

In particular, the second central moment of a population is its variance.

Problem

Find the third central moment of eruption duration in the data set `faithful`.

Solution

We apply the function `moment` from the `e1071` package. As it is not in the core R library, the package has to be installed and loaded into the R workspace.

```
> library(e1071)
> duration = faithful$eruptions
> moment(duration, order=3,
+     center=TRUE)
[1] -0.61491
```

Answer

The third central moment of eruption duration is -0.61491.

Exercise

Find the third central moment of eruption waiting period in `faithful`.

Solution of Exercise

We apply the function `moment` to the `waiting` variable and get:

```
> waiting = faithful$waiting
> moment(waiting, order=3,
+     center=TRUE)
[1] -1040.3
```

Hence the third central moment of eruption waiting period is -1040.3.

8.13 Skewness

The **skewness** of a data population is defined by the following formula, where μ_2 and μ_3 are the second and third central moments.

$$\gamma_1 = \mu_3 / \mu_2^{3/2}$$

Intuitively, the skewness is a measure of symmetry. As a rule, negative skewness indicates that the mean of the data values is less than the median, and the data distribution is *left-skewed*. Positive skewness would indicate that the mean of the data values is larger than the median, and the data distribution is *right-skewed*.

Problem

Find the skewness of eruption duration in the data set `faithful`.

Solution

We apply the function `skewness` from the `e1071` package to compute the skewness coefficient of `eruptions`. As the package is not in the core R library, it has to be installed and loaded into the R workspace.

```
> library(e1017)
> duration = faithful$eruptions
> skewness(duration)
[1] -0.41355
```

Answer

The skewness of eruption duration is -0.41355. It indicates that the eruption duration distribution is skewed towards the left.

Exercise

Find the skewness of eruption waiting period in `faithful`.

Solution of Exercise

We apply the function `skewness` to the `waiting` variable and get:

```
> waiting = faithful$waiting
> skewness(waiting)
[1] -0.41403
```

Hence the skewness of the eruption waiting period is -0.41403.

8.14 Kurtosis

The **kurtosis** of a univariate population is defined by the following formula, where μ_2 and μ_4 are respectively the second and fourth central moments.

$$\gamma_2 = \mu_4/\mu_2^2 - 3$$

Intuitively, the *kurtosis* describes the *tail shape* of the data distribution. The normal distribution has zero kurtosis, and thus the standard tail shape. It is said to be **mesokurtic**. Negative kurtosis would indicate a *thin-tailed* data distribution, and is said to be **platykurtic**. Positive kurtosis would indicate a *fat-tailed* distribution, and is said to be **leptokurtic**.

Problem

Find the kurtosis of eruption duration in the data set `faithful`.

Solution

We apply the function `kurtosis` from the `e1071` package to compute the kurtosis of `eruptions`. As the package is not in the core R library, it has to be installed and loaded into the R workspace.

```
> library(e1071)
> duration = faithful$eruptions
> kurtosis(duration)
[1] -1.5116
```

Answer

The kurtosis of eruption duration is -1.5116, which indicates that eruption duration distribution is platykurtic. This is consistent with the fact that its histogram is not bell-shaped.

Exercise

Find the kurtosis of eruption waiting period in `faithful`.

Solution of Exercise

We apply the function `kurtosis` to the `waiting` variable and get:

```
> waiting = faithful$waiting
> kurtosis(waiting)
[1] -1.1563
```

Hence the kurtosis of the eruption waiting period is -1.1563.

Note

The default algorithm of the function `kurtosis` in `e1071` is based on the formula $g_2 = m_4/s^4-3$, where m_4 and s are the fourth central moment and sample standard deviation respectively. See the R documentation for selecting other types of kurtosis algorithm.

```
> library(e1071)
> help(kurtosis)
```

Chapter 9
Probability Distributions

A **probability distribution** describes how the values of a random variable is distributed. For example, the collection of all possible outcomes of a coin toss is known to follow the binomial distribution. Whereas the means of of sufficiently large samples of a data population are known to resemble the normal distribution. Since the characteristics of these theoretical distributions are well understood, they can be used to make statistical inferences on the entire data population.

In the following tutorials, we demonstrate how to compute a few well-known probability distributions that occurs frequently in statistical study. We will reference them quite often in other sections.

9.1 Binomial Distribution

The **binomial distribution** is a discrete probability distribution. It describes the outcome of *n* identical independent trials in an experiment. Each trial is to have only two outcomes, either success or failure, and we call it a **Bernoulli trial**. If the probability of a successful trial is *p*, then the probability of having *x* successful outcomes in an experiment of *n* identical independent Bernoulli trials is

$$f(x) = \left(\begin{array}{c} n \\ x \end{array} \right) p^x (1-p)^{(n-x)} \quad where\ x = 0, 1, 2, \ldots, n$$

Problem

Suppose there are twelve multiple choice questions in an English class quiz. Each question has five possible answers, and only one of them is correct. Find the probability of having four or less correct answers if a student attempts to answer every question at random.

Solution

Since only one out of five possible answers is correct, the probability of answering a question correctly by random is 1/5=0.2. We can find the probability of having exactly 4 correct answers by random attempts as follows.

```
> dbinom(4, size=12, prob=0.2)
[1] 0.1329
```

To find the probability of having four or less correct answers by random attempts, we apply the function `dbinom` with *x*=0,…,4.

```
> dbinom(0, size=12, prob=0.2) +
+ dbinom(1, size=12, prob=0.2) +
+ dbinom(2, size=12, prob=0.2) +
+ dbinom(3, size=12, prob=0.2) +
+ dbinom(4, size=12, prob=0.2)
[1] 0.9274
```

Alternatively, we can use the cumulative probability function for binomial

distribution `pbinom`.

```
> pbinom(4, size=12, prob=0.2)
[1] 0.92744
```

Answer

The probability of four or less questions answered correctly by random in a twelve question multiple choice quiz is 92.7%.

9.2 Poisson Distribution

The **Poisson distribution** is the probability distribution of independent event occurrences in an interval. If λ is the mean occurrence per interval, then the probability of having *x* occurrences within a given interval is:

$$f(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad where\ x = 0, 1, 2, 3, \ldots$$

Problem

If there are twelve cars crossing a bridge per minute on average, find the probability of having seventeen or more cars crossing the bridge in a particular minute.

Solution

The probability of having *sixteen or less* cars crossing the bridge in a particular minute is given by the function `ppois`.

```
> ppois(16, lambda=12)   # lower tail
[1] 0.89871
```

Hence the probability of having seventeen or more cars crossing the bridge in a minute is in the *upper tail* of the probability density function.

```
> ppois(16, lambda=12, lower=FALSE)
[1] 0.10129
```

Answer

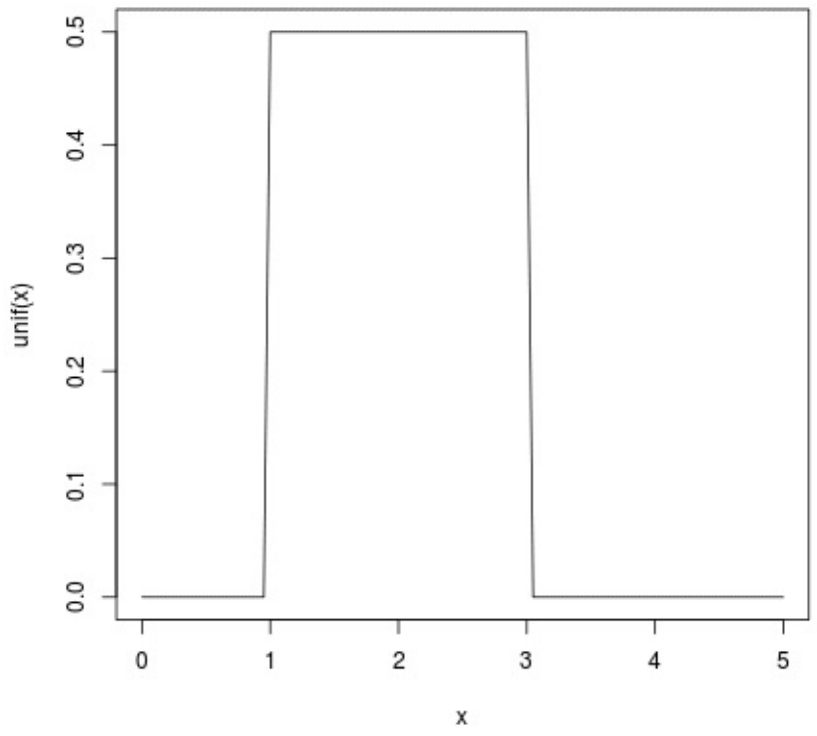
If there are twelve cars crossing a bridge per minute on average, the probability of having seventeen or more cars crossing the bridge in a particular minute is 10.1%

9.3 Continuous Uniform Distribution

The **continuous uniform distribution** is the probability distribution of random number selection from the continuous interval between *a* and *b*. Its density function is defined by the following:

$$f(x) = \begin{cases} \frac{1}{b-a} & when\ a \leq x \leq b \\ 0 & when\ x < a\ or\ x > b \end{cases}$$

Here is a graph of the continuous uniform distribution with *a*=1, *b*=3.



Problem

Select ten random numbers between one and three.

Solution

We apply the generation function `runif` of the uniform distribution to generate the random numbers required.

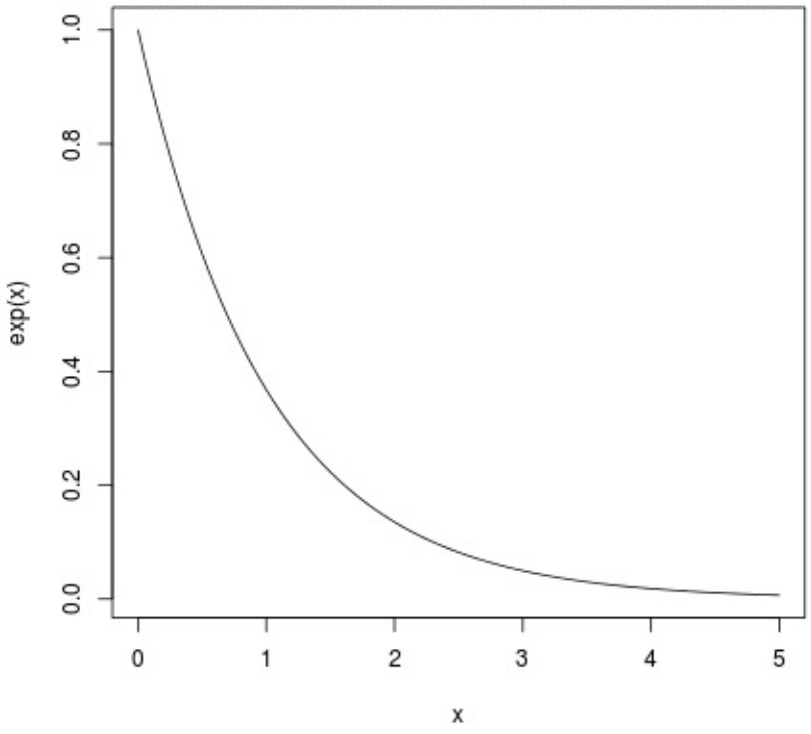
```
> runif(10, min=1, max=3)
[1] 1.6121 1.2028 1.9306 2.4233 ...
```

9.4 Exponential Distribution

The **exponential distribution** describes the arrival time of randomly recurring independent events. If μ is the mean waiting time for the next event recurrence, the probability density function is:

$$f(x) = \begin{cases} \frac{1}{\mu} e^{-x/\mu} & when\ x \geq 0 \\ 0 & when\ x < 0 \end{cases}$$

Here is a graph of the exponential distribution with μ=1.



Problem

Suppose the mean checkout time of a supermarket cashier is three minutes. Find the probability of a customer checkout being completed by the cashier in less than

two minutes.

Solution

The checkout processing rate is equals to one divided by the mean checkout completion time. Hence the processing rate is 1/3 checkouts per minute, and we can apply the function `pexp` of exponential distribution with rate=1/3.

```
> pexp(2, rate=1/3)
[1] 0.48658
```

Answer

The probability of finishing a checkout under two minutes by the cashier is 48.7%.

9.5 Normal Distribution

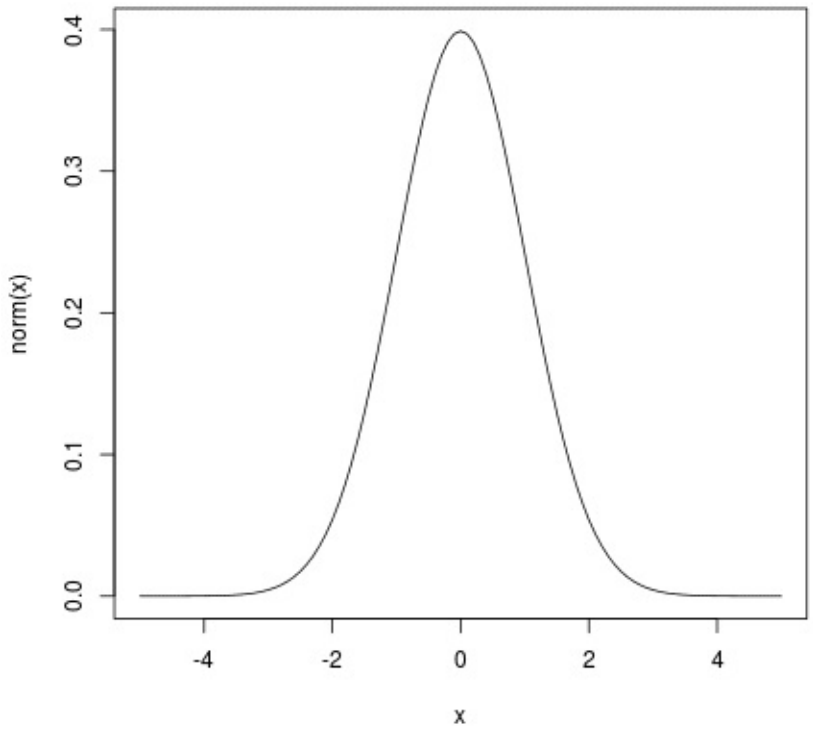
The **normal distribution** is defined by the following probability density function, where μ is the population mean, and σ^2 is the population variance.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{1}{2\sigma^2}(y-\mu)^2)$$

If a random variable *X* follows the normal distribution, then we write:

$$X \sim N(\mu, \sigma^2)$$

In particular, the normal distribution with μ=1 and σ=1 is called the **standard normal distribution**, and is denoted by *N*(0,1). The graph is a *bell curve*.



The normal distribution is important because of the **Central Limit Theorem**,

which states that the means of all possible samples of size n from a population with mean μ and variance σ^2 approaches a normal distribution with mean μ and variance σ^2/n when n approaches infinity.

Problem

Assume that the test scores of a college entrance exam fits a normal distribution. Furthermore, the mean test score is 72, and the standard deviation is 15.2. What is the percentage of students scoring 84 or more in the exam?

Solution

We apply the function `pnorm` of the normal distribution with mean 72 and standard deviation 15.2. Since we are looking for the percentage of students scoring higher than 84, we are interested in the *upper tail* of the normal distribution.

```
> pnorm(84, mean=72, sd=15.2,
+     lower.tail=FALSE)
[1] 0.21492
```

Answer

The percentage of students scoring 84 or more in the college entrance exam is 21.5%.

9.6 Chi-squared Distribution

If X_1, X_2, …, X_m are m independent random variables of the standard normal distribution, then the following quantity follows a **Chi-Squared distribution** with m degrees of freedom. Its mean is m, and its variance is $2m$.

$$V = X_1^2 + X_2^2 + \cdots + X_m^2 \sim \chi^2_{(m)}$$

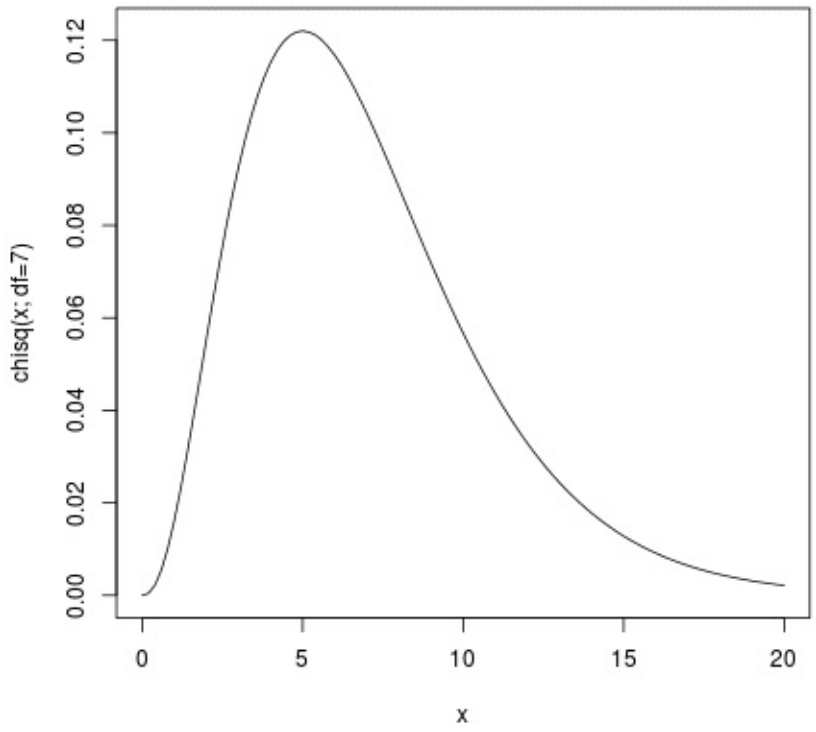
Its density function is

$$f(x) = \frac{1}{2^{m/2}\Gamma(m/2)} x^{(m/2)-1} e^{-x/2}$$

where Γ is the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$$

Here is a graph of the Chi-Squared distribution with 7 degrees of freedom.



Problem

Find the 95th percentile of the Chi-Squared distribution with 7 degrees of freedom.

Solution

We apply the quantile function `qchisq` of the Chi-Squared distribution with argument 0.95.

```
> qchisq(.95, df=7)
[1] 14.067
```

Answer

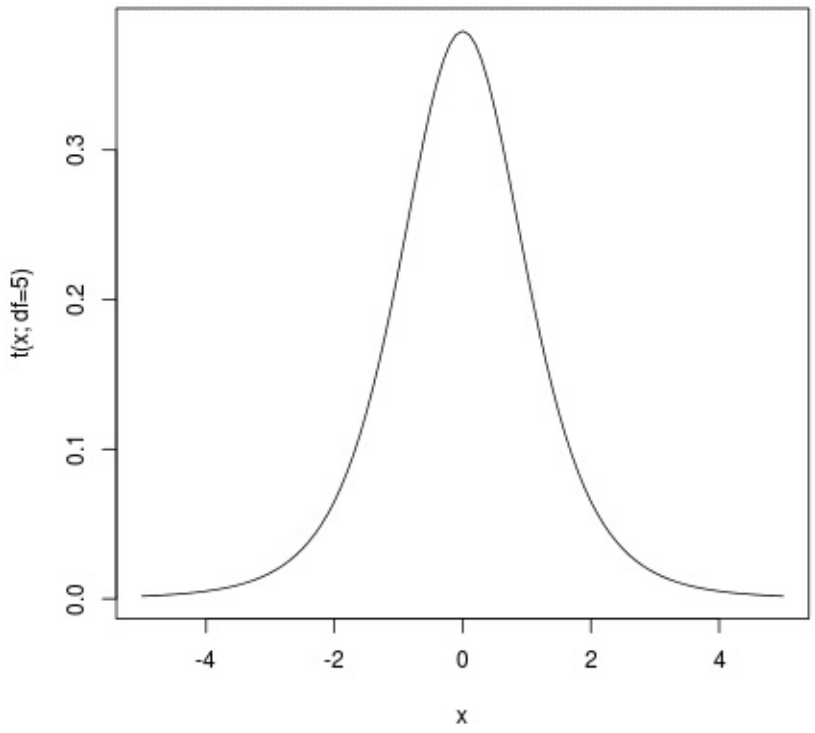
The 95th percentile of the Chi-Squared distribution with 7 degrees of freedom is 14.067.

9.7 Student t Distribution

Assume that a random variable Z has the standard normal distribution, and another random variable V has the Chi-Squared distribution with m degrees of freedom. Assume further that Z and V are independent, then the following quantity follows a **Student t distribution** with m degrees of freedom.

$$t = \frac{Z}{\sqrt{V/m}} \sim t_{(m)}$$

Here is a graph of the Student t distribution with 5 degrees of freedom, and it is a *bell curve*.



Problem

Find the 2.5^{th} and 97.5^{th} percentiles of the Student t distribution with 5 degrees of freedom.

Solution

We apply the quantile function `qt` of the Student t distribution with arguments 0.025 and 0.975.

```
> qt(c(.025, .975), df=5)
[1] -2.5706  2.5706
```

Answer

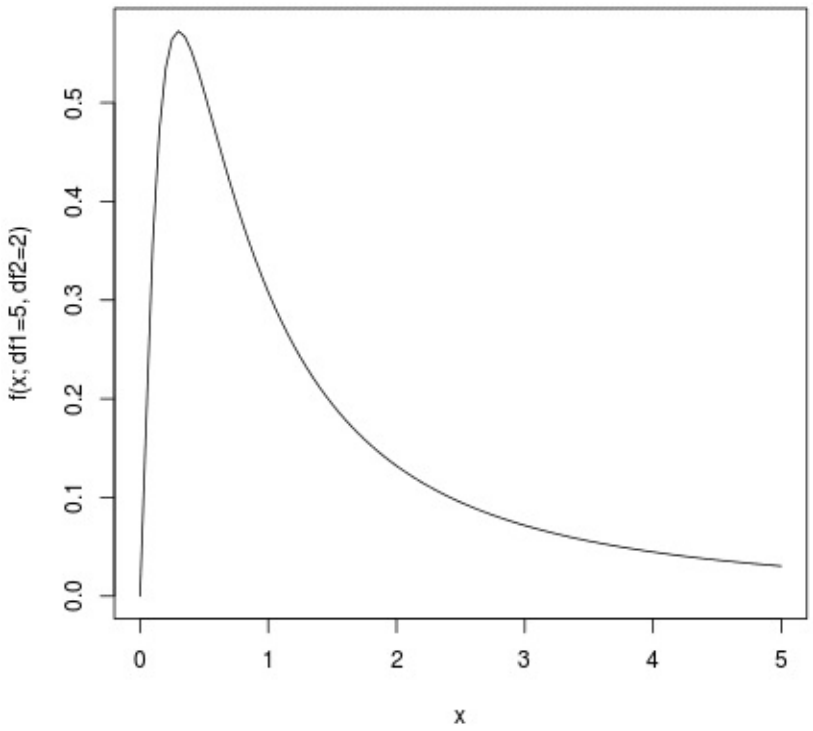
The 2.5^{th} and 97.5^{th} percentiles of the Student t distribution with 5 degrees of freedom are -2.5706 and 2.5706 respectively.

9.8 F Distribution

If V_1 and V_2 are two independent random variables having the Chi-Squared distribution with m_1 and m_2 degrees of freedom respectively, then the following quantity follows an **F distribution** with m_1 *numerator degrees of freedom* and m_2 *denominator degrees of freedom, i.e.,* (m_1, m_2) degrees of freedom.

$$F = \frac{V_1/m_1}{V_2/m_2} \sim F_{(m_1, m_2)}$$

Here is a graph of the F distribution with (5, 2) degrees of freedom.



Problem

Find the 95th percentile of the F distribution with (5, 2) degrees of freedom.

Solution

We apply the quantile function `qf` of the F distribution with argument 0.95.

```
> qf(.95, df1=5, df2=2)
[1] 19.296
```

Answer

The 95th percentile of the F distribution with (5, 2) degrees of freedom is 19.296.

9.9 Beta Distribution

For a given pair of positive numbers *a* and *b*, we define a **beta distribution** *Beta*(*a*, *b*) with the following density function

$$beta(x; a, b) = \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)} x^{a-1}(1-x)^{b-1}, \quad for\ 0 \le x \le 1$$

where Γ is the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$$

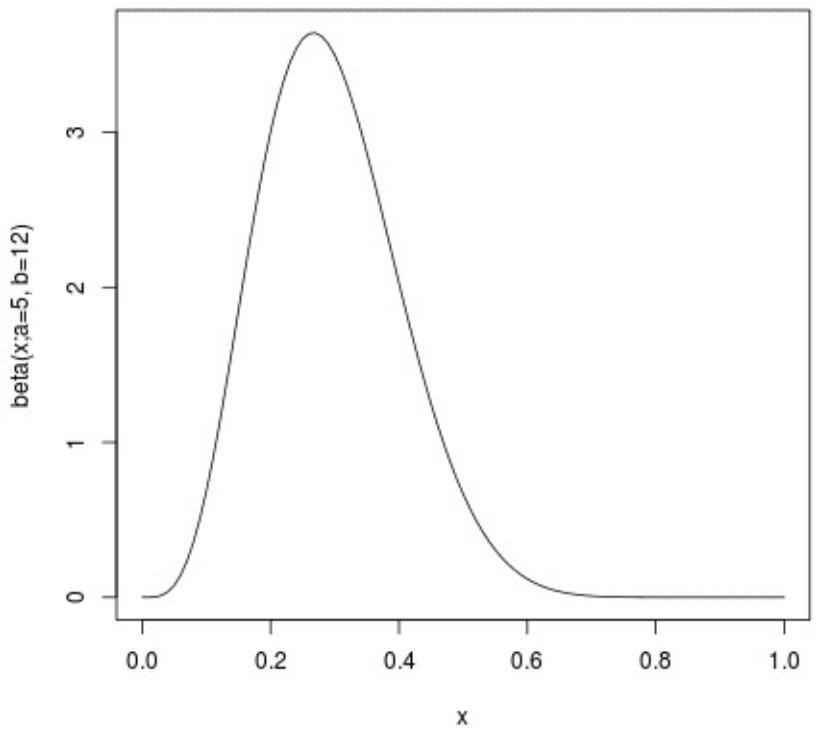
Denote the sum of *a* and *b* by *c*=*a*+*b*. Then the mean and variance of *Beta*(*a*, *b*) are

$$\mu = a/c$$

and

$$\sigma^2 = ab/(c^2(c+1))$$

For example, here is a graph of *Beta*(5, 12).



Problem

Find mean and variance of *Beta*(5, 12).

Solution

We apply the formula and find the mean and variance of the beta distribution to be 0.2941 and 0.01153 respectively.

```
> a = 5
> b = 12
> c = a+b

> mu = a/c; mu
[1] 0.2941
> s2 = a*b/(c*c*(c+1)); s2
[1] 0.01153
```

9.10 Gamma Distribution

For a given pair of positive numbers *a* and *b*, we define a **gamma distribution** *Gamma*(*a*, *b*) with the following density function

$$gamma(x; a, b) = \frac{b^a}{\Gamma(a)} x^{a-1} e^{-bx}$$

where Γ is the gamma function

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$$

We call *a* the **shape** parameter, and *b* the **rate** parameter. The mean and variance of *Gamma*(*a*, *b*) are

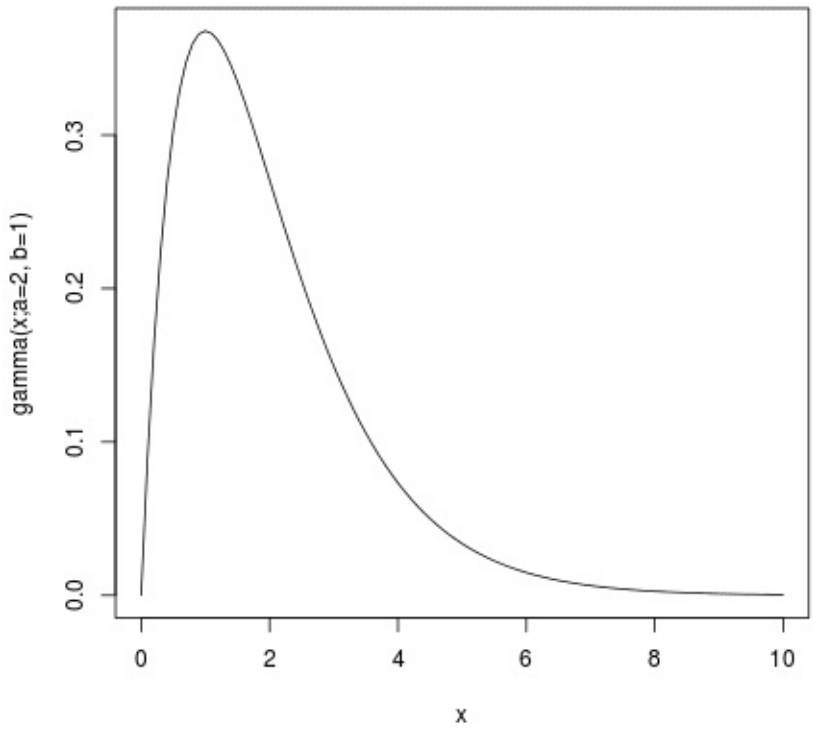
$$\mu = a/b$$

and

$$\sigma^2 = a/b^2$$

The gamma distribution is a generalization of two important distributions. In fact, *Gamma*(1, *b*) is the exponential distribution with mean 1/*b*, and *Gamma*(*m*/1, 1/2) is the Chi-squared distribution with *m* degrees of freedom.

Here is a graph of *Gamma*(2, 1).



Problem

Find the mean and variance of *Gamma*(2, 1).

Solution

We apply the formula and find the mean and variance are both equal to 2.

```
> a = 2
> b = 1

> mu = a/b; mu
[1] 2
> s2 = a/(b*b); s2
[1] 2
```

Chapter 10
Interval Estimation

It is a common requirement to efficiently estimate population parameters based on random sample data. In the tutorials of this chapter, we will discuss how to compute the population estimates. The steps are demonstrated with a built-in data set **survey**. It is the outcome of a student survey taken place in a statistics class of an Australian university.

The data set belongs to the `MASS` package, which is often implicitly pre-loaded in the R workspace. Here is a sneak preview of the data.

```
> library(MASS)
> head(survey)
     Sex Wr.Hnd NW.Hnd  ...
1 Female   18.5   18.0  ...
2   Male   19.5   20.5  ...
3   Male   18.0   13.3  ...
       ................
```

For further details of the `survey` data set, please consult the R documentation.

```
> help(survey)
```

10.1 Point Estimate of Population Mean

For any particular random sample, we can always compute its sample mean. Although most often it is not the actual population mean, it does serve as a good **point estimate**. For example, in the data set `survey`, the actual survey is conducted only on a sample of the student population. However, we can compute the sample mean and use it as an estimate of the corresponding student population mean.

Problem

Find a point estimate of the mean university student height using sample data from the data set `survey`.

Solution

For convenience, we begin with saving the survey data of student heights in the variable `height.survey`.

```
> height.survey = survey$Height
```

It turns out not all students have answered the question, and we must filter out the missing values. Hence we apply the function `mean` with the `na.rm` option set as `TRUE`.

```
> mean(height.survey, na.rm=TRUE)
[1] 172.38
```

Answer

A point estimate of the mean student height is 172.38 centimeters.

10.2 Interval Estimate of Population Mean with Known Variance

After we found a point estimate of the population mean, we would need a way to quantify its accuracy. Here, we discuss the case where the population variance σ^2 is assumed to be known.

Let us denote the 100(1-α/2) percentile of the standard normal distribution as $z_{\alpha/2}$. For random samples of sufficiently large size n, the end points of the **interval estimate** at (1-α) confidence level is given as follows:

$$\bar{x} \pm z_{\alpha/2} \frac{\sigma}{\sqrt{n}}$$

Problem

Assume the population standard deviation σ of the student height in the data set `survey` to be 9.48. Find the margin of error and interval estimate of the population mean at 95% confidence level.

Solution

We first filter out missing values in `survey$Height` with the function `na.omit`, and save it in `height.response`.

```
> height.response =
+     na.omit(survey$Height)
```

Then we compute the standard error of the mean, `sem`.

```
> n = length(height.response)
> sigma = 9.48
> sem = sigma/sqrt(n); sem
[1] 0.65575
```

Since there are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail.

Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`. We multiply it with the standard error of the mean `sem` and come up with the margin of error `E`.

```
> E = qnorm(.975)*sem; E
[1] 1.2852
```

We then add it up with the sample mean, and find the confidence interval.

```
> xbar = mean(height.response)
> xbar + c(-E, E)
[1] 171.10 173.67
```

Answer

Assuming the population standard deviation σ to be 9.48, the margin of error for the student height survey at 95% confidence level is 1.2852 centimeters. The confidence interval is between 171.10 and 173.67 centimeters.

Alternative Solution

Instead of using the textbook formula, we can apply the function `z.test` from the `TeachingDemos` package. It is not a core R package, and must be installed and loaded into the workspace beforehand.

```
> library(TeachingDemos)
> z.test(height.response, sd=sigma)

       One Sample z-test

data:  height.response
z = 262.88, n = 209.000, Std. Dev. = 9.480,
Std. Dev. of the sample mean = 0.656, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 171.10 173.67
sample estimates:
mean of height.response
                 172.38
```

10.3 Interval Estimate of Population Mean with Unknown Variance

After we found a point estimate of the population mean, we would need a way to quantify its accuracy. Here, we discuss the case where knowledge of the population variance σ^2 is *not* assumed.

Let us denote the 100(1-α/2) percentile of the Student t distribution with n-1 degrees of freedom as $t_{\alpha/2}$. For random samples of sufficiently large size n, and standard deviation s, the end points of the **interval estimate** at (1-α) confidence level is given as follows:

$$\bar{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}}$$

Problem

Without assuming the population standard deviation of the student height in the data set `survey`, find the margin of error and interval estimate of the population mean at 95% confidence level.

Solution

We first filter out missing values in `survey$Height` with the function `na.omit`, and save it in `height.response`.

```
> height.response =
+     na.omit(survey$Height)
```

Then we compute the sample standard deviation s, and the standard error estimate `SE`.

```
> n = length(height.response)
> s = sd(height.response)
> SE = s/sqrt(n); SE
[1] 0.68117
```

Since there are two tails of the Student t distribution, the 95% confidence level would imply the 97.5th percentile of the Student t distribution at the upper tail. Therefore, $t_{\alpha/2}$ is given by `qt(.975, df=n-1)`. We multiply it with the standard error estimate `SE` and get the margin of error `E`.

```
> E = qt(.975, df=n-1)*SE; E
[1] 1.3429
```

We then add it up with the sample mean, and find the confidence interval.

```
> xbar = mean(height.response)
> xbar + c(-E, E)
[1] 171.04 173.72
```

Answer

Without assumption on the population standard deviation, the margin of error for the student height survey at 95% confidence level is 1.3429 centimeters. The confidence interval is between 171.04 and 173.72 centimeters.

Alternative Solution

Instead of using the textbook formula, we can apply the function `t.test` in the built-in `stat` package.

```
> t.test(height.response)

        One Sample t-test

data:  height.response
t = 253.07, df = 208, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 171.04 173.72
sample estimates:
mean of x
   172.38
```

10.4 Sampling Size of Population Mean

The quality of a sample survey can be improved by increasing the sample size. The following formula provides the sample size needed for satisfying the requirements of estimating population mean at (1-α) confidence level and margin of error E. Here, σ^2 is the population variance, and $z_{\alpha/2}$ is the 100(1-α/2) percentile of the standard normal distribution.

$$n = \frac{(z_{\alpha/2})^2 \sigma^2}{E^2}$$

Problem

Assume the population standard deviation σ of the student height in the data set `survey` to be 9.48. Find the sample size needed to achieve a 1.2 centimeters margin of error at 95% confidence level.

Solution

Since there are two tails of the normal distribution, a 95% confidence level would the 97.5th percentile of the normal distribution at the upper tail. Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`.

```
> zstar = qnorm(.975)
> sigma = 9.48
> E = 1.2
> zstar^2 * sigma^2/ E^2
[1] 239.75
```

Answer

Based on the assumption of population standard deviation being 9.48, it needs a sample size of 240 to achieve a 1.2 centimeters margin of error at 95% confidence level.

10.5 Point Estimate of Population Proportion

Multiple choice questionnaires in a survey are often used to determine the proportion of a population with specific characteristic. For example, we may estimate the proportion of female students in the university based on the data set `survey`.

Problem

Find a point estimate of the female student proportion in the data set `survey`.

Solution

We first filter out missing values in `survey$Sex` with the function `na.omit`, and save it in `gender.response`.

```
> gender.response =
+     na.omit(survey$Sex)
> n = length(gender.response)
```

To find out the number of female students, we compare `gender.response` with the factor '`Female`', and compute the sum. Dividing it by *n* gives the female student proportion in the sample survey.

```
> k = sum(gender.response == "Female")
> pbar = k/n; pbar
[1] 0.5
```

Answer

The point estimate of the female student proportion in the data set `survey` is 50%.

10.6 Interval Estimate of Population Proportion

After we found a point sample estimate of the population proportion, we need a measure of the quality of the point estimate based on the confidence interval.

Let us denote the 100(1-α/2) percentile of the standard normal distribution as $z_{\alpha/2}$. If the sample size n and population proportion p satisfy the condition that np>=5 and $n(1-p)$>=5, then the end points of the interval estimate at (1-α) confidence level is defined in terms of the sample proportion as follows.

$$\bar{p} \pm z_{\alpha/2}\sqrt{\frac{\bar{p}(1-\bar{p})}{n}}$$

Problem

Compute the margin of error and estimate interval for the female students proportion in the data set `survey` at 95% confidence level.

Solution

We first determine the proportion point estimate. Detailed explanation can be found in our previous tutorial on Point Estimate of Population Proportion.

```
> gender.response =
+     na.omit(survey$Sex)
> n = length(gender.response)
> k = sum(gender.response == "Female")
> pbar = k/n; pbar
[1] 0.5
```

Then we estimate the standard error `SE`.

```
> SE = sqrt(pbar*(1-pbar)/n); SE
[1] 0.032547
```

Since there are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail.

Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`. Hence we multiply it with the standard error estimate `SE` and get the margin of error `E`.

```
> E = qnorm(.975)*SE; E
[1] 0.063791
```

Combining it with the sample proportion, we obtain the confidence interval.

```
> pbar + c(-E, E)
[1] 0.43621 0.56379
```

Answer

At 95% confidence level, between 43.6% and 56.3% of the university students are female, and the margin of error is 6.4%.

Alternative Solution

Instead of using the textbook formula, we can apply the function `prop.test` from the built-in `stats` package.

```
> prop.test(k, n)

        1-sample proportions test
        without continuity correction

data:  k out of n, null probability 0.5
X-squared = 0, df = 1, p-value = 1
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
 0.43672 0.56328
sample estimates:
  p
0.5
```

10.7 Sampling Size of Population Proportion

The quality of a sample survey can be improved by increasing the sample size. The following formula provides the sample size needed for the requirements of population proportion interval estimate at (1-α) confidence level, margin of error E, and *planned* proportion estimate p. Here, $z_{\alpha/2}$ is the 100(1-α/2) percentile of the standard normal distribution.

$$n = \frac{(z_{\alpha/2})^2 p(1-p)}{E^2}$$

Problem

Using a 50% planned proportion estimate, find the sample size needed to achieve 5% margin of error for the female student proportion in the data set `survey` at 95% confidence level.

Solution

Since there are two tails of the normal distribution, the 95% confidence level would imply the 97.5th percentile of the normal distribution at the upper tail. Therefore, $z_{\alpha/2}$ is given by `qnorm(.975)`.

```
> zstar = qnorm(.975)
> p = 0.5
> E = 0.05
> zstar^2 * p * (1-p) / E^2
[1] 384.15
```

Answer

With a planned proportion estimate of 50% at 95% confidence level, it needs a sample size of 385 to achieve a 5% margin of error for the survey of female student proportion.

Chapter 11
Hypothesis Testing

Researchers retain or reject hypotheses based on measurements of observed samples. The decision process is often based upon a statistical mechanism known as **hypothesis testing**, which enables researchers to estimate the probability of testing errors.

A **type I error** is the mishap of falsely rejecting a null hypothesis when the hypothesis is valid. The probability of committing a type I error is called the **significance level** of a hypothesis testing, and is denoted by the Greek letter α.

In the following tutorials, we will demonstrate the procedure of hypothesis testing in R primarily using the intuitive *critical value* approach. Then we follow up with the popular *p-value* approach as an alternative.

11.1 Lower Tail Test of Population Mean with Known Variance

The null hypothesis of the **lower tail test of the population mean** can be expressed as follows:

$$\mu \geq \mu_0$$

where μ_0 is a hypothesized lower bound of the true population mean μ.

Let us define the test statistic z in terms of the sample mean, the sample size, and the population standard deviation σ:

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

Then the null hypothesis of the lower tail test is to be *rejected* if $z <= -z_\alpha$, where z_α is the 100(1-α) percentile of the standard normal distribution.

Problem

Suppose the manufacturer claims that the mean lifetime of a light bulb is more than 10,000 hours. For a sample of 30 light bulbs, the mean lifetime turns out to be only 9,900 hours. Assume the population standard deviation to be 120 hours. At .05 significance level, can we reject the claim by the manufacturer?

Solution

The null hypothesis is that μ >= 10000. We begin with computing the test statistic.

```
> xbar = 9900      # sample mean
> mu0 = 10000      # hypothesis
> sigma = 120      # population sd
> n = 30           # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> z                # test statistic
[1] -4.5644
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
> z.alpha = qnorm(1-alpha)
> -z.alpha         # critical value
[1] -1.6449
```

Answer

The test statistic -4.5644 is less than the critical value of -1.6449. Hence, at .05 significance level, we reject the claim that the mean lifetime of the light bulbs is above 10,000 hours.

Alternative Solution

Instead of using the critical value, we apply the function `pnorm` to compute the lower tail p-value of the test statistic. As it turns out to be less than the .05 significance level, we reject the null hypothesis that μ >= 10000.

```
> pval = pnorm(z)
> pval         # lower tail p-value
[1] 2.5052e-06
```

Exercise 1

Under same conditions as the problem above, can we reject the manufacturer's claim on the lifetime of light bulbs at .01 significance level?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha)
> -z.alpha     # critical value
[1] -2.3263
```

The test statistic -4.5644 is less than the critical value of -2.3263. Hence, at .01 significance level, we reject the claim that the mean lifetime of the light bulbs is above 10,000 hours.

Alternative Solution of Exercise 1

The lower tail p-value of the test statistic, being 2.5052e-06, is less than .01. Hence we reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are the lifetime hours from a random sample of light bulbs. Assuming the population standard deviation to be 120 hours, can we reject a manufacturer's claim that the light bulbs last more than 10,000 hours at .05 significance level?

```
 9899  9991  9933  9854
 9808  9815 10024 10056
 9850  9942  9923  9937
 9861  9875  9868  9938
 9822  9981 10097  9961
 9934 10009 10115  9798
 9958  9800 10007  9787
 9721  9803
```

Solution of Exercise 2

The data is part of the latest code download, and we can use the function `scan` to load it from the file "lightbulbs.txt":

```
> x = scan("lightbulbs.txt")
```

Then we compute the lower tail p-value as before, and found it to be less than 0.05. Hence we can reject the null hypothesis that μ >= 10000.

```
> xbar = mean(x)   # sample mean
> mu0 = 10000      # hypothesis
> sigma = 120      # population sd
> n = length(x)    # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> pval = pnorm(z)
> pval
[1] 3.088e-05
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply `z.test` from the `TeachingDemos` package and have the same outcome. Since the null hypothesis is μ >= 10000, we set the `alternative` option as "less".

```
> library(TeachingDemos)
> test = z.test(x, mu=mu0,
+     stdev=sigma,
+     alternative="less")
> test$p.value
[1] 3.088e-05
```

11.2 Upper Tail Test of Population Mean with Known Variance

The null hypothesis of the **upper tail test of the population mean** can be expressed as follows:

$$\mu \leq \mu_0$$

where μ_0 is a hypothesized upper bound of the true population mean μ.

Let us define the test statistic z in terms of the sample mean, the sample size, and the population standard deviation σ:

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

Then the null hypothesis of the upper tail test is to be *rejected* if $z >= z_\alpha$, where z_α is the 100(1-α) percentile of the standard normal distribution.

Problem

Suppose the food label on a cookie bag states that there are at most 2 grams of saturated fat in a single cookie. In a sample of 35 cookies, it is found that there are 2.1 grams of saturated fat per cookie on average. Assume the population standard deviation to be 0.25 grams. At .05 significance level, can we reject the claim on food label?

Solution

The null hypothesis is that $\mu <= 2$. We begin with computing the test statistic.

```
> xbar = 2.1       # sample mean
> mu0 = 2          # hypothesis
> sigma = 0.25     # population sd
> n = 35           # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> z                # test statistic
```

```
[1] 2.3664
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
> z.alpha = qnorm(1-alpha)
> z.alpha          # critical value
[1] 1.6449
```

Answer

The test statistic 2.3664 is greater than the critical value of 1.6449. Hence, at .05 significance level, we reject the claim that there are at most 2 grams of saturated fat in a cookie.

Alternative Solution

Instead of using the critical value, we apply the function `pnorm` to compute the upper tail p-value of the test statistic. As it turns out to be less than the .05 significance level, we reject the null hypothesis that $\mu <= 2$.

```
> pval = pnorm(z, lower.tail=FALSE)
> pval         # upper tail p-value
[1] 0.0089802
```

Exercise 1

Under same conditions as the problem above, can we reject the claim on saturated fat as stated in the food label at .01 significance level?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha)
> z.alpha          # critical value
[1] 2.3263
```

The test statistic 2.3664 is greater than the critical value of 2.3263. Hence, at 0.01 significance level, we reject the claim that there are at most 2 grams of saturated fat in a cookie.

Alternative Solution of Exercise 1

The upper tail p-value of the test statistic, being 0.0089802, is less than .01. Hence we reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are the gram amount of saturated fat found in a random sample of cookies. Assuming the population standard deviation to be 0.25 grams, at .05 significance level, can we reject the claim in the food label that there are at most 2 grams of saturated fat per cookie?

```
 1.7462 1.9801 1.9418 1.7957
 2.1930 1.9586 2.2432 2.4291
 2.0638 2.0916 2.2952 2.1608
 2.3238 2.0470 2.3978 1.9862
 2.2096 2.0398 2.1565 2.1584
 2.1703 1.8295 1.8192 2.4184
 1.8511 2.2900 2.0294 2.0648
 2.0956 1.8221 1.7056 1.7600
 1.7803 1.1110 1.6458
```

Solution of Exercise 2

The data is part of the code download, and we can use the function `scan` to load it from the file "cookies.txt":

```
> x = scan("cookies.txt")
```

Then we compute the upper tail p-value as before, and found it to be greater than 0.05. Hence we do not reject the null hypothesis that $\mu <= 2$.

```
> xbar = mean(x)   # sample mean
> mu0 = 2          # hypothesis
> sigma = 0.25     # population sd
> n = length(x)    # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> pval = pnorm(z, lower.tail=FALSE)
> pval
[1] 0.33969
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply `z.test` from the `TeachingDemos` package and have the same outcome. Since the null hypothesis is $\mu <= 2$, we set the `alternative` option as "greater".

```
> library(TeachingDemos)
> test = z.test(x, mu=mu0,
+     stdev=sigma,
+     alternative="greater")
> test$p.value
[1] 0.33969
```

11.3 Two-Tailed Test of Population Mean with Known Variance

The null hypothesis of the **two-tailed test of the population mean** can be expressed as follows:

$$\mu = \mu_0$$

where μ_0 is a hypothesized value of the true population mean μ.

Let us define the test statistic z in terms of the sample mean, the sample size, and the population standard deviation σ:

$$z = \frac{\bar{x} - \mu_0}{\sigma/\sqrt{n}}$$

Then the null hypothesis of the two-tailed test is to be *rejected* if $z <= -z_{\alpha/2}$ or $z >= z_{\alpha/2}$, where $z_{\alpha/2}$ is the $100(1-\alpha/2)$ percentile of the standard normal distribution.

Problem

Suppose the mean weight of King Penguins found in an Antarctic colony last year was 15.4 kg. In a sample of 35 penguins same time this year in the same colony, the mean penguin weight is 14.6 kg. Assume the population standard deviation to be 2.5 kg. At .05 significance level, can we reject the null hypothesis that the mean penguin weight does not differ from last year?

Solution

The null hypothesis is that μ = 15.4. We begin with computing the test statistic.

```
> xbar = 14.6      # sample mean
> mu0 = 15.4       # hypothesis
> sigma = 2.5      # population sd
> n = 35           # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> z                # test statistic
```

```
[1] -1.8931
```

We then compute the critical values at .05 significance level.

```
> alpha = .05
> z.alpha = qnorm(1-alpha/2)
> c(-z.alpha, z.alpha)
[1] -1.96  1.96
```

Answer

The test statistic -1.8931 lies between the critical values -1.9600 and 1.9600. Hence, at .05 significance level, we do not reject the null hypothesis that the mean penguin weight does not differ from last year.

Alternative Solution

Instead of using the critical value, we apply the function `pnorm` to compute the two-tailed p-value of the test statistic. It doubles the *lower tail* p-value as the sample mean is *less* than the hypothesized value. Since it turns out to be greater than the .05 significance level, we do *not* reject the null hypothesis that $\mu = 15.4$.

```
> pval = 2 * pnorm(z)
> pval
[1] 0.058339
```

To avoid selecting the incorrect p-value, notice that the choice of lower tail *vs.* upper tail p-values is determined by the sign of the test statistics `z`. Hence we can automate the choice as follows, which works correctly regardless of the sign of `z`.

```
> pval = 2 * ifelse(z < 0,
+     pnorm(z),
+     pnorm(z, lower.tail=FALSE))
> pval
[1] 0.058339
```

Exercise 1

Under same conditions as the problem above, can we reject the null hypothesis at .01 significance level that the mean penguin weight stays the same as last year?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha/2)
> c(-z.alpha, z.alpha)
[1] -2.5758  2.5758
```

The test statistic -1.8931 lies between the critical values -2.5758 and 2.5758. Hence, at .01 significance level, we do not reject the null hypothesis that the mean penguin weight does not differ from last year.

Alternative Solution of Exercise 1

The two-tailed p-value of the test statistic, being 0.058339, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are body weight of King Penguins in kilograms found in a random sample within a colony. Assuming the population standard deviation to be 2.5 kg, at .05 significance level, can we reject the null hypothesis that the mean penguin weight is still 15.4 kg just like last year?

```
 12.0625 14.4009 14.0175 12.5568
 16.5302 14.1860 17.0322 18.8913
 15.2381 15.5165 17.5520 16.2080
 17.8383 15.0698 18.5780 14.4621
 16.6962 14.9984 16.1649 16.1840
 16.3026 12.8949 12.7919 18.7838
 13.1106 17.4996 14.8936 15.2481
 15.5559 12.8213 11.6561 12.1996
 12.4026  5.7097 11.0581
```

Solution of Exercise 2

The data is part of the code download, and we can use the function `scan` to load it from the file "penguins.txt":

```
> x = scan("penguins.txt")
```

Then we compute mean sample weight, and found it to be *less than* the hypothesized value of 15.4.

```
> xbar = mean(x)
> xbar              # sample mean
[1] 14.775
```

Hence we compute the two-tailed p-value as before, and found it to be greater than 0.05. Therefore we do not reject the null hypothesis that $\mu = 15.4$.

```
> mu0 = 15.4        # hypothesis
> sigma = 2.5       # population sd
> n = length(x)     # sample size
> z = (xbar-mu0)/(sigma/sqrt(n))
> pval = 2 * pnorm(z)
> pval
[1] 0.1389
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply `z.test` from the `TeachingDemos` package and have the same outcome. Since the null hypothesis is $\mu = 15.4$, we use the default `alternative` option.

```
> library(TeachingDemos)
> test = z.test(x,
+     mu=mu0, stdev=sigma)
> test$p.value
[1] 0.1389
```

11.4 Lower Tail Test of Population Mean with Unknown Variance

The null hypothesis of the **lower tail test of the population mean** can be expressed as follows:

$$\mu \geq \mu_0$$

where μ_0 is a hypothesized lower bound of the true population mean μ.

Let us define the test statistic t in terms of the sample mean, the sample size, and the sample standard deviation s:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

Then the null hypothesis of the lower tail test is to be *rejected* if $t <= -t_\alpha$, where t_α is the 100(1-α) percentile of the Student t distribution with n-1 degrees of freedom.

Problem

Suppose the manufacturer claims that the mean lifetime of a light bulb is more than 10,000 hours. For a sample of 30 light bulbs, the mean lifetime turns out to be only 9,900 hours. Assume the sample standard deviation to be 120 hours. At .05 significance level, can we reject the claim by the manufacturer?

Solution

The null hypothesis is that μ >= 10000. We begin with computing the test statistic.

```
> xbar = 9900      # sample mean
> mu0 = 10000      # hypothesis
> s = 125          # sample sd
> n = 30           # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
> t.val            # test statistic
```

```
[1] -4.3818
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
> t.alpha = qt(1-alpha, df=n-1)
> -t.alpha          # critical value
[1] -1.6991
```

Answer

The test statistic -4.3818 is less than the critical value of -1.6991. Hence, at .05 significance level, we can reject the claim that mean lifetime of the light bulbs is above 10,000 hours.

Alternative Solution

Instead of using the critical value, we apply the function `pt` to compute the lower tail p-value of the test statistic. As it turns out to be less than the .05 significance level, we reject the null hypothesis that μ >= 10000.

```
> pval = pt(t.val, df=n-1)
> pval              # lower tail
[1] 7.035e-05
```

Exercise 1

Under same conditions as the problem above, can we reject the manufacturer's claim on the lifetime of light bulbs at .01 significance level?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> n = 30
> t.alpha = qt(1-alpha, df=n-1)
> -t.alpha          # critical value
[1] -2.462
```

The test statistic -4.3818 is less than the critical value of -2.462. Hence, at .01 significance level, we can reject the claim that mean lifetime of the light bulbs is

above 10,000 hours.

Alternative Solution of Exercise 1

The lower tail p-value of the test statistic, being 7.035e-05, is less than .01. Hence we reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are the lifetime hours from a random sample of light bulbs. Without knowledge of the population standard deviation, can we reject a manufacturer's claim that the light bulbs last more than 10,000 hours at .05 significance level?

```
 9899  9991  9933  9854
 9808  9815 10024 10056
 9850  9942  9923  9937
 9861  9875  9868  9938
 9822  9981 10097  9961
 9934 10009 10115  9798
 9958  9800 10007  9787
 9721  9803
```

Solution of Exercise 2

The data is part of the code download, and we can use the function `scan` to load it from the file "lightbulbs.txt":

```
> x = scan("lightbulbs.txt")
```

Then we compute the lower tail p-value as before, and found it to be less than 0.05. Hence we can reject the null hypothesis that μ >= 10000.

```
> xbar = mean(x)   # sample mean
> mu0 = 10000      # hypothesis
> s = sd(x)        # sample sd
> n = length(x)    # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
> pval = pt(t.val, df=n-1)
> pval             # lower tail
[1] 1.5918e-05
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply the function `t.test` and have the same outcome. Since the null hypothesis is μ >= 10000, we set the `alternative` option as "less".

```
> test = t.test(x, mu=mu0,
+     alternative="less")
> test$p.value
[1] 1.5918e-05
```

11.5 Upper Tail Test of Population Mean with Unknown Variance

The null hypothesis of the **upper tail test of the population mean** can be expressed as follows:

$$\mu \leq \mu_0$$

where μ_0 is a hypothesized upper bound of the true population mean μ.

Let us define the test statistic t in terms of the sample mean, the sample size, and the sample standard deviation s:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

Then the null hypothesis of the upper tail test is to be *rejected* if $t >= t_\alpha$, where t_α is the 100(1-α) percentile of the Student t distribution with n-1 degrees of freedom.

Problem

Suppose the food label on a cookie bag states that there are at most 2 grams of saturated fat in a single cookie. In a sample of 35 cookies, it is found that there are 2.1 grams of saturated fat per cookie on average. Assume the sample standard deviation to be 0.3 grams. At .05 significance level, can we reject the claim on food label?

Solution

The null hypothesis is that $\mu <= 2$. We begin with computing the test statistic.

```
> xbar = 2.1       # sample mean
> mu0 = 2          # hypothesis
> s = 0.3          # sample sd
> n = 35           # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
```

```
> t.val            # test statistic
[1] 1.9720
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
> t.alpha = qt(1-alpha, df=n-1)
> t.alpha          # critical value
[1] 1.6909
```

Answer

The test statistic 1.9720 is greater than the critical value of 1.6991. Hence, at .05 significance level, we can reject the claim that there are at most 2 grams of saturated fat in a cookie.

Alternative Solution

Instead of using the critical value, we apply the function `pt` to compute the upper tail p-value of the test statistic. As it turns out to be less than the .05 significance level, we reject the null hypothesis that $\mu <= 2$.

```
> pval = pt(t.val, df=n-1,
+     lower.tail=FALSE)
> pval             # upper tail
[1] 0.028393
```

Exercise 1

Under same conditions as the problem above, can we reject the claim on saturated fat as stated in the food label at .01 significance level?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> n = 35
> t.alpha = qt(1-alpha, df=n-1)
> t.alpha          # critical value
[1] 2.4411
```

The test statistic 1.9720 is less than the critical value of 2.4411. Hence, at .01 significance level, we do not reject the claim that there are at most 2 grams of saturated fat in a cookie.

Alternative Solution of Exercise 1

The upper tail p-value of the test statistic, being 0.028393, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are the gram amount of saturated fat found in a random sample of cookies. Without knowledge of the population standard deviation, at .05 significance level, can we reject the claim in the food label that there are at most 2 grams of saturated fat per cookie?

```
 1.7462 1.9801 1.9418 1.7957
 2.1930 1.9586 2.2432 2.4291
 2.0638 2.0916 2.2952 2.1608
 2.3238 2.0470 2.3978 1.9862
 2.2096 2.0398 2.1565 2.1584
 2.1703 1.8295 1.8192 2.4184
 1.8511 2.2900 2.0294 2.0648
 2.0956 1.8221 1.7056 1.7600
 1.7803 1.1110 1.6458
```

Solution of Exercise 2

The data is part of the code download, and we can use the function `scan` to load it from the file "cookies.txt":

```
> x = scan("cookies.txt")
```

Then we compute the upper tail p-value as before, and found it to be greater than 0.05. Hence we do not reject the null hypothesis that $\mu <= 2$.

```
> xbar = mean(x)   # sample mean
> mu0 = 2          # hypothesis
> s = sd(x)        # sample sd
> n = length(x)    # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
> pval = pt(t.val, df=n-1,
+     lower.tail=FALSE)
```

```
> pval              # upper tail
[1] 0.3497
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply the function `t.test` and have the same outcome. Since the null hypothesis is $\mu <= 2$, we set the `alternative` option as "greater".

```
> test = t.test(x, mu=mu0,
+     alternative="greater")
> test$p.value
[1] 0.3497
```

11.6 Two-Tailed Test of Population Mean with Unknown Variance

The null hypothesis of the **two-tailed test of the population mean** can be expressed as follows:

$$\mu = \mu_0$$

where μ_0 is a hypothesized value of the true population mean μ.

Let us define the test statistic t in terms of the sample mean, the sample size, and the sample standard deviation s:

$$t = \frac{\bar{x} - \mu_0}{s/\sqrt{n}}$$

Then the null hypothesis of the two-tailed test is to be *rejected* if $t <= -t_{\alpha/2}$ or $t >= t_{\alpha/2}$, where $t_{\alpha/2}$ is the 100(1-α) percentile of the Student t distribution with n-1 degrees of freedom.

Problem

Suppose the mean weight of King Penguins found in an Antarctic colony last year was 15.4 kg. In a sample of 35 penguins same time this year in the same colony, the mean penguin weight is 14.6 kg. Assume the sample standard deviation to be 2.5 kg. At .05 significance level, can we reject the null hypothesis that the mean penguin weight does not differ from last year?

Solution

The null hypothesis is that μ = 15.4. We begin with computing the test statistic.

```
> xbar = 14.6      # sample mean
> mu0 = 15.4       # hypothesis
> s = 2.5          # sample sd
> n = 35           # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
```

```
> t.val            # test statistic
[1] -1.8931
```

We then compute the critical values at .05 significance level.

```
> alpha = .05
> ta = qt(1-alpha/2, df=n-1)
> c(-ta, ta)
[1] -2.0322  2.0322
```

Answer

The test statistic -1.8931 lies between the critical values -2.0322, and 2.0322. Hence, at .05 significance level, we do not reject the null hypothesis that the mean penguin weight does not differ from last year.

Alternative Solution

Instead of using the critical value, we apply the function `pt` to compute the two-tailed p-value of the test statistic. It doubles the *lower tail* p-value as the sample mean is *less* than the hypothesized value. Since it turns out to be greater than the .05 significance level, we do *not* reject the null hypothesis that $\mu = 15.4$.

```
> pval = 2 * pt(t.val, df=n-1)
> pval
[1] 0.066876
```

To avoid selecting the incorrect p-value, notice that the choice of lower tail *vs.* upper tail p-values is determined by the sign of the test statistics `t.val`. Hence we can automate the choice as follows, which works correctly regardless of the sign of `t.val`.

```
> pval = 2 * ifelse(t.val < 0,
+     pt(t.val, df=n-1),
+     pt(t.val, df=n-1, lower=FALSE))
> pval
[1] 0.066876
```

Exercise 1

Under same conditions as the problem above, can we reject the null hypothesis at .01 significance level that the mean penguin weight stays the same as last year?

Solution of Exercise 1

We compute the critical value at .01 significance level as:

```
> alpha = .01
> n = 35
> t.alpha = qt(1-alpha/2, df=n-1)
> c(-t.alpha, t.alpha)
[1] -2.7284  2.7284
```

The test statistic -1.8931 lies between the critical values -2.7284, and 2.7284. Hence, at .01 significance level, we do not reject the null hypothesis that the mean penguin weight does not differ from last year.

Alternative Solution of Exercise 1

The two-tailed p-value of the test statistic, being 0.066876, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

Exercise 2

Suppose the following are body weight of King Penguins in kilograms found in a random sample within a colony. Without knowledge of the population standard deviation, at .05 significance level, can we reject the null hypothesis that the mean penguin weight is still 15.4 kg just like last year?

```
 12.0625 14.4009 14.0175 12.5568
 16.5302 14.1860 17.0322 18.8913
 15.2381 15.5165 17.5520 16.2080
 17.8383 15.0698 18.5780 14.4621
 16.6962 14.9984 16.1649 16.1840
 16.3026 12.8949 12.7919 18.7838
 13.1106 17.4996 14.8936 15.2481
 15.5559 12.8213 11.6561 12.1996
 12.4026  5.7097 11.0581
```

Solution of Exercise 2

The data is part of the code download, and we can use the function `scan` to load it from the file "penguins.txt":

```
> x = scan("penguins.txt")
```

Then we compute mean sample weight, and found it to be *less than* the hypothesized value of 15.4.

```
> xbar = mean(x)
> xbar              # sample mean
[1] 14.775
```

Hence we compute the two-tailed p-value as before, and found it to be greater than 0.05. Therefore we do not reject the null hypothesis that $\mu = 15.4$.

```
> mu0 = 15.4        # hypothesis
> s = sd(x)         # sample sd
> n = length(x)     # sample size
> t.val = (xbar-mu0)/(s/sqrt(n))
> pval = 2 * pt(t.val, df=n-1)
> pval              # two-tailed
[1] 0.17227
```

Alternative Solution of Exercise 2

Instead of using the textbook formula, we can apply the function `t.test` and have the same outcome. Since the null hypothesis is $\mu = 15.4$, we use the default `alternative` option.

```
> test = t.test(x, mu=mu0)
> test$p.value
[1] 0.17227
```

11.7 Lower Tail Test of Population Proportion

The null hypothesis of the **lower tail test about population proportion** can be expressed as follows:

$$p \geq p_0$$

where p_0 is a hypothesized lower bound of the true population proportion p.

Let us define the test statistic z in terms of the sample proportion and sample size:

$$z = \frac{\bar{p} - p_0}{\sqrt{p_0(1-p_0)/n}}$$

Then the null hypothesis of the lower tail test is to be *rejected* if $z <= -z_\alpha$, where z_α is the 100(1-α) percentile of the standard normal distribution.

Problem

Suppose 60% of citizens voted in the last election. 85 out of 148 people in a telephone survey said that they voted in current election. At 0.5 significance level, can we reject the null hypothesis that the proportion of voters in the population is above 60% this year?

Solution

The null hypothesis is that $p >= 0.6$. We begin with computing the test statistic.

```
> pbar = 85/148    # sample prop
> p0 = .6          # hypothesis
> n = 148          # sample size
> z = (pbar-p0)/sqrt(p0*(1-p0)/n)
> z                # test statistic
[1] -0.6376
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
```

```
> z.alpha = qnorm(1-alpha)
> -z.alpha         # critical value
[1] -1.6449
```

Answer

The test statistic -0.6376 is *not* less than the critical value of -1.6449. Hence, at .05 significance level, we do *not* reject the null hypothesis that the proportion of voters in the population is above 60% this year.

Alternative Solution 1

Instead of using the critical value, we apply the function `pnorm` to compute the lower tail p-value of the test statistic. As it turns out to be greater than the .05 significance level, we do not reject the null hypothesis that $p >= 0.6$.

```
> pval = pnorm(z)
> pval             # lower tail
[1] 0.26187
```

Alternative Solution 2

We can apply the function `prop.test` to compute the p-value directly. The Yates continuity correction is disabled for pedagogical reasons.

```
> prop.test(85, 148, p=.6,
+     alt="less", correct=FALSE)

        1-sample proportions test
        without continuity correction

data:  85 out of 148, null probability 0.6
X-squared = 0.4065, df = 1, p-value = 0.2619
alternative hypothesis: true p is less than 0.6
95 percent confidence interval:
 0.0000 0.63925
sample estimates:
      p
0.57432
```

Exercise

Under same conditions as the problem above, can we reject the null hypothesis at

.01 significance level that the voting percentage is above 60% this year?

Solution of Exercise

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha)
> -z.alpha         # critical value
[1] -2.3263
```

The test statistic -0.6376 is *not* less than the critical value of -2.3263. Hence, at .01 significance level, we do *not* reject the null hypothesis that voting percentage is above 60% this year.

Alternative Solution of Exercise

The lower tail p-value of the test statistic, being 0.26187, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

11.8 Upper Tail Test of Population Proportion

The null hypothesis of the **upper tail test about population proportion** can be expressed as follows:

$$p \leq p_0$$

where p_0 is a hypothesized upper bound of the true population proportion p.

Let us define the test statistic z in terms of the sample proportion and sample size:

$$z = \frac{\bar{p} - p_0}{\sqrt{p_0(1-p_0)/n}}$$

Then the null hypothesis of the upper tail test is to be *rejected* if $z >= z_\alpha$, where z_α is the 100(1-α) percentile of the standard normal distribution.

Problem

Suppose that 12% of apples harvested in an orchard last year was rotten. 30 out of 214 apples in a harvest sample this year turns out to be rotten as well. At .05 significance level, can we reject the null hypothesis that the proportion of rotten apples in harvest stays below 12% this year?

Solution

The null hypothesis is that $p <= 0.12$. We begin with computing the test statistic.

```
> pbar = 30/214     # sample prop
> p0 = .12          # hypothesis
> n = 214           # sample size
> z = (pbar-p0)/sqrt(p0*(1-p0)/n)
> z                 # test statistic
[1] 0.90875
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
```

```
> z.alpha = qnorm(1-alpha)
> z.alpha          # critical value
[1] 1.6449
```

Answer

The test statistic 0.90875 is *not* greater than the critical value of 1.6449. Hence, at .05 significance level, we do *not* reject the null hypothesis that the proportion of rotten apples in harvest stays below 12% this year.

Alternative Solution 1

Instead of using the critical value, we apply the function `pnorm` to compute the upper tail p-value of the test statistic. As it turns out to be greater than the .05 significance level, we do not reject the null hypothesis that $p <= 0.12$.

```
> pval = pnorm(z, lower.tail=FALSE)
> pval             # upper tail
[1] 0.18174
```

Alternative Solution 2

We can apply the function `prop.test` to compute the p-value directly. The Yates continuity correction is disabled for pedagogical reasons.

```
> prop.test(30, 214, p=.12,
+     alt="greater", correct=FALSE)

        1-sample proportions test
        without continuity correction

data:  30 out of 214, null probability 0.12
X-squared = 0.8258, df = 1, p-value = 0.1817
alternative hypothesis: true p is greater than 0.12
95 percent confidence interval:
 0.10563 1.00000
sample estimates:
      p
0.14019
```

Exercise

Under same conditions as the problem above, can we reject the null hypothesis at

.01 significance level that the proportion of rotten apples stays below 12% this year?

Solution of Exercise

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha)
> z.alpha          # critical value
[1] 2.3263
```

The test statistic 0.90875 is *not* greater than the critical value of 2.3263. Hence, at .01 significance level, we do *not* reject the null hypothesis that the proportion of rotten apples in harvest stays below 12% this year.

Alternative Solution of Exercise

The upper tail p-value of the test statistic, being 0.18174, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

11.9 Two-Tailed Test of Population Proportion

The null hypothesis of the **two-tailed test about population proportion** can be expressed as follows:

$$p = p_0$$

where p_0 is a hypothesized value of the true population proportion p.

Let us define the test statistic z in terms of the sample proportion and sample size:

$$z = \frac{\bar{p} - p_0}{\sqrt{p_0(1-p_0)/n}}$$

Then the null hypothesis of the two-tailed test is to be *rejected* if $z <= -z_{\alpha/2}$ or $z >= z_{\alpha/2}$, where $z_{\alpha/2}$ is the 100(1-α/2) percentile of the standard normal distribution.

Problem

Suppose a coin toss turns up 12 heads out of 20 trials. At .05 significance level, can one reject the null hypothesis that the coin toss is fair?

Solution

The null hypothesis is that p = 0.5. We begin with computing the test statistic.

```
> pbar = 12/20     # sample prop
> p0 = .5          # hypothesis
> n = 20           # sample size
> z = (pbar-p0)/sqrt(p0*(1-p0)/n)
> z                # test statistic
[1] 0.89443
```

We then compute the critical value at .05 significance level.

```
> alpha = .05
> z.alpha = qnorm(1-alpha/2)
> c(-z.alpha, z.alpha)
```

```
[1] -1.96  1.96
```

Answer

The test statistic 0.89443 lies between the critical values -1.9600 and 1.9600. Hence, at .05 significance level, we do *not* reject the null hypothesis that the coin toss is fair.

Alternative Solution 1

Instead of using the critical value, we apply the function `pnorm` to compute the two-tailed p-value of the test statistic. It doubles the *upper tail* p-value as the sample proportion is *greater* than the hypothesized value. Since it turns out to be greater than the .05 significance level, we do not reject the null hypothesis that p = 0.5.

```
> pval = 2 * pnorm(z,
+     lower.tail=FALSE)
> pval
[1] 0.37109
```

Alternative Solution 2

We can apply the function `prop.test` to compute the p-value directly. The Yates continuity correction is disabled for pedagogical reasons.

```
> prop.test(12, 20, p=0.5,
+     correct=FALSE)

        1-sample proportions test
        without continuity correction

data:  12 out of 20, null probability 0.5
X-squared = 0.8, df = 1, p-value = 0.3711
alternative hypothesis: true p is not equal to 0.5
95 percent confidence interval:
  0.38658 0.78119
sample estimates:
  p
0.6
```

Exercise

Under same conditions as the problem above, can we reject the null hypothesis at .01 significance level that the coin is fair?

Solution of Exercise

We compute the critical value at .01 significance level as:

```
> alpha = .01
> z.alpha = qnorm(1-alpha/2)
> c(-z.alpha, z.alpha)
[1] -2.5758  2.5758
```

The test statistic 0.89443 lies between the critical values -2.5758 and 2.5758. Hence, at .01 significance level, we do *not* reject the null hypothesis that the coin toss is fair.

Alternative Solution of Exercise

The two-tailed p-value of the test statistic, being 0.37109, is greater than .01. Hence we do not reject the null hypothesis at .01 significance level.

Chapter 12
Type II Errors

In hypothesis testing, a **type II error** is due to a failure of rejecting an invalid null hypothesis. The probability of avoiding a type II error is called the power of the hypothesis test, and is denoted by the quantity $1-\beta$.

In the following tutorials, we demonstrate how to compute the power of a hypothesis test based on scenarios from our previous discussions on hypothesis testing. The approach is based on a parametric estimate of the region where the null hypothesis would not be rejected. The probability of a type II error is then derived based on a hypothetical population value.

12.1 Lower Tail Test of Population Mean with Known Variance

In a lower tail test of the population mean, the null hypothesis claims that the true population mean μ is greater than a given hypothetical value μ_0.

$$\mu \geq \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact less than μ_0.

Assume that the population has a known variance σ^2. Due to the Central Limit Theorem, the means of all possible samples of sufficiently large size *n* resembles the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will *not* be rejected, and then obtain an estimate of the probability of type II error.

We demonstrate the procedure with the following:

Problem

Suppose the manufacturer claims that the mean lifetime of a light bulb is over 10,000 hours. Assume the actual mean light bulb lifetime to be 9,950 hours and the population standard deviation to be 120 hours. At .05 significance level, what is the probability of having a type II error for a sample of 30 light bulbs?

Solution

We begin with computing the standard deviation of the mean, `sem`.

```
> n = 30       # sample size
> sigma = 120  # population sd
> sem = sigma/sqrt(n); sem
[1] 21.909
```

We next compute the lower bound of sample means for which the null hypothesis μ >= 10000 would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 10000  # hypothesis
> q = qnorm(alpha,
+     mean=mu0, sd=sem); q
[1] 9964
```

Therefore, so long as the sample mean is greater than 9964 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 9950, we can compute the probability of a sample mean being above 9964, and thus found the probability of type II error.

```
> mu = 9950    # population mean
> pnorm(q, mean=mu, sd=sem,
+       lower.tail=FALSE)
[1] 0.26196
```

Answer

If the light bulbs sample size is 30, the actual mean light bulb lifetime is 9,950 hours and the population standard deviation is 120 hours, then the probability of type II error for testing the null hypothesis μ >= 10000 at .05 significance level is 26.2%, and the power of the hypothesis test is 73.8%.

Exercise

Under same assumptions as above, if the actual mean light bulb lifetime is 9,965 hours, what is the probability of type II error at .05 significance level? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, the lower bound of sample means for which the null hypothesis would not be rejected is `q = 9964`. Therefore, if the population mean is 9,965 hours, the probability of a sample mean being above `q` is:

```
> mu = 9965    # population mean
> pnorm(q, mean=mu, sd=sem,
+       lower.tail=FALSE)
```

```
[1] 0.51887
```

Hence, assuming $\mu = 9965$, the probability for type II error is 51.9%, and the power of the hypothesis test is 48.1%.

12.2 Upper Tail Test of Population Mean with Known Variance

In a upper tail test of the population mean, the null hypothesis claims that the true population mean μ is less than a given hypothetical value μ_0.

$$\mu \leq \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact greater than μ_0.

Assume that the population has a known variance σ^2. Due to the Central Limit Theorem, the means of all possible samples of sufficiently large size *n* resembles the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will *not* be rejected, and then obtain an estimate of the probability of type II error.

We demonstrate the procedure with the following:

Problem

Suppose the food label on a cookie bag states that there are at most 2 grams of saturated fat in a single cookie. Assume the actual mean amount of saturated fat per cookie to be 2.09 grams, and the population standard deviation to be 0.25 grams. At .05 significance level, what is the probability of having a type II error for a sample of 35 cookies?

Solution

We begin with computing the standard deviation of the mean, `sem`.

```
> n = 35       # sample size
> sigma = 0.25 # population sd
> sem = sigma/sqrt(n); sem
[1] 0.042258
```

We next compute the upper bound of sample means for which the null hypothesis μ <= 2 would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 2      # hypothesis
> q = qnorm(alpha, mean=mu0, sd=sem,
+       lower.tail=FALSE)
> q
[1] 2.0695
```

Therefore, so long as the sample mean is less than 2.0695 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 2.09, we can compute the probability of a sample mean being below 2.0695, and thus found the probability of type II error.

```
> mu = 2.09    # population mean
> pnorm(q, mean=mu, sd=sem)
[1] 0.31386
```

Answer

If the cookies sample size is 35, the actual mean amount of saturated fat per cookie is 2.09 grams and the population standard deviation is 0.25 grams, then the probability of type II error for testing the null hypothesis μ <= 2 at .05 significance level is 31.4%, and the power of the hypothesis test is 68.6%.

Exercise

Under same assumptions as above, if the actual mean amount of saturated fat per cookie is 2.075 grams, what is the probability of type II errors? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, the upper bound of sample means for which the null hypothesis would not be rejected is `q = 2.0695`. Therefore, if the actual mean amount of saturated fat per cookie is 2.075 grams, the probability of a sample mean being below `q` is:

```
> mu = 2.075   # population mean
> pnorm(q, mean=mu, sd=sem)
[1] 0.44829
```

Hence, assuming $\mu = 2.075$, the probability for type II error is 44.8%, and the power of the hypothesis test is 55.2%.

12.3 Two-Tailed Test of Population Mean with Known Variance

In a two-tailed test of the population mean, the null hypothesis claims that the true population mean μ is equal to a given hypothetical value μ_0.

$$\mu = \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact different from μ_0.

Assume that the population has a known variance σ^2. Due to the Central Limit Theorem, the means of all possible samples of sufficiently large size n resembles the normal distribution. Hence we can compute the range of sample means for which the null hypothesis will *not* be rejected, and then obtain an estimate of the probability of type II error.

We demonstrate the procedure with the following:

Problem

Suppose the mean weight of King Penguins found in an Antarctic colony last year was 15.4 kg. Assume the actual mean population weight to be 15.1 kg, and the population standard deviation to be 2.5 kg. At .05 significance level, what is the probability of having a type II error for a sample of 35 penguins?

Solution

We begin with computing the standard deviation of the mean, `sem`.

```
> n = 35       # sample size
> sigma = 2.5  # population sd
> sem = sigma/sqrt(n); sem
[1] 0.42258
```

We next compute the lower and upper bounds of sample means for which the null hypothesis $\mu = 15.4$ would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 15.4   # hypothetical mean
> I = c(alpha/2, 1-alpha/2)
> q = qnorm(I, mean=mu0, sd=sem); q
[1] 14.572 16.228
```

Therefore, so long as the sample mean is between 14.572 and 16.228 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 15.1, we can compute the lower tail probabilities at both end points.

```
> mu = 15.1    # population mean
> p = pnorm(q, mean=mu, sd=sem); p
[1] 0.10564 0.99621
```

Finally, the probability of type II error is the probability between the two end points.

```
> diff(p)      # p[2]-p[1]
[1] 0.89056
```

Answer

If the penguin sample size is 35, the actual mean population weight is 15.1 kg and the population standard deviation is 2.5 kg, then the probability of type II error for testing the null hypothesis $\mu = 15.4$ at .05 significance level is 89.1%, and the power of the hypothesis test is 10.9%.

Exercise

Under same assumptions as above, if actual mean population weight is 14.9 kg, what is the probability of type II errors? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, so long as the sample mean is between 14.572 and 16.228 in a hypothesis test, the null hypothesis will not be rejected. Therefore, if the actual mean is 14.9, we can compute the lower tail probabilities

at both end points.

```
> mu = 14.9    # population mean
> p = pnorm(q, mean=mu, sd=sem); p
[1] 0.21865 0.99916
```

Finally, the probability of type II error is the probability between the two end points.

```
> diff(p)      # p[2]-p[1]
[1] 0.78051
```

Hence, assuming $\mu = 14.9$, the probability for type II error is 78.1%, and the power of the hypothesis test is 21.9%.

12.4 Lower Tail Test of Population Mean with Unknown Variance

In a lower tail test of the population mean, the null hypothesis claims that the true population mean μ is greater than a given hypothetical value μ_0.

$$\mu \geq \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact less than μ_0.

Let s^2 be the sample variance. For sufficiently large n, the following statistics of all possible samples of size n resembles a Student t distribution with n-1 degrees of freedom.

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

This allows us to compute the range of sample means for which the null hypothesis will *not* be rejected, and to obtain the probability of type II error. We demonstrate the procedure with the following:

Problem

Suppose the manufacturer claims that the mean lifetime of a light bulb is over 10,000 hours. Assume that in a random sample of 30 light bulbs, the standard deviation of their lifetime is 125 hours. If the actual mean lifetime is 9,950 hours, what is the probability of type II error for a hypothesis test at .05 significance level?

Solution

We begin with computing the standard error estimate `SE`.

```
> n = 30       # sample size
```

```
> s = 125      # sample sd
> SE = s/sqrt(n); SE
[1] 22.822
```

We next compute the lower bound of sample means for which the null hypothesis μ >= 10000 would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 10000  # hypothesis
> q = mu0 + qt(alpha, df=n-1)*SE; q
[1] 9961.2
```

Therefore, so long as the sample mean is greater than 9961.2 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 9950, we can compute the probability of a sample mean being above 9961.2, and thus found the probability of type II error.

```
> mu = 9950    # population mean
> pt((q - mu)/SE, df=n-1,
+     lower.tail=FALSE)
[1] 0.31329
```

Answer

If the light bulbs sample size is 30, the sample standard variance is 125 hours and the actual mean lifetime is 9,950 hours, then the probability of type II error for testing the null hypothesis μ >= 10000 at .05 significance level is 31.3%, and the power of the hypothesis test is 68.7%.

Exercise

Under same assumptions as above, if the actual mean lifetime is 9,965 hours, what is the probability of type II error at .05 significance level? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, the lower bound of sample means for which the null hypothesis would not be rejected is `q = 9961.2`. Therefore, if the population mean is 9,965 hours, the probability of a sample mean being above `q` is:

```
> mu = 9965    # population mean
> pt((q - mu)/SE, df=n-1,
+     lower.tail=FALSE)
[1] 0.56515
```

Hence, assuming $\mu = 9965$, the probability for type II error is 56.5%, and the power of the hypothesis test is 43.5%.

12.5 Upper Tail Test of Population Mean with Unknown Variance

In a upper tail test of the population mean, the null hypothesis claims that the true population mean μ is less than a given hypothetical value μ_0.

$$\mu \leq \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact greater than μ_0.

Let s^2 be the sample variance. For sufficiently large n, the following statistics of all possible samples of size n resembles a Student t distribution with n-1 degrees of freedom.

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

This allows us to compute the range of sample means for which the null hypothesis will *not* be rejected, and to obtain the probability of type II error. We demonstrate the procedure with the following:

Problem

Suppose the food label on a cookie bag states that there are at most 2 grams of saturated fat in a single cookie. Assume that in a random sample of 35 cookies, the standard deviation of saturated fat is 0.3 grams. If actual mean amount of saturated fat per cookie is 2.09 grams, what is the probability of type II error for a hypothesis test at .05 significance level?

Solution

We begin with computing the standard error estimate `SE`.

```
> n = 35       # sample size
```

```
> s = 0.3      # sample sd
> SE = s/sqrt(n); SE
[1] 0.050709
```

We next compute the upper bound of sample means for which the null hypothesis μ <= 2 would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 2      # hypothesis
> q = mu0 + qt(alpha, df=n-1,
+     lower.tail=FALSE)*SE
> q
[1] 2.0857
```

Therefore, so long as the sample mean is less than 2.0857 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 2.09, we can compute the probability of a sample mean being below 2.0857, and thus found the probability of type II error.

```
> mu = 2.09    # population mean
> pt((q - mu)/SE, df=n-1)
[1] 0.46681
```

Answer

If the cookies sample size is 35, the sample standard deviation of saturated fat per cookie is 0.3 grams and the actual mean amount of saturated fat per cookie is 2.09 grams, then the probability of type II error for testing the null hypothesis μ <= 2 at .05 significance level is 46.7%, and the power of the hypothesis test is 53.3%.

Exercise

Under same assumptions as above, if the actual mean saturated fat per cookie is 2.075 grams, what is the probability of type II errors? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, the upper bound of sample means for which the null hypothesis would not be rejected is `q = 2.0857`. Therefore, if the actual mean amount of saturated fat per cookie is 2.075 grams, the probability of a sample mean being below `q` is:

```
> mu = 2.075   # population mean
> pt((q - mu)/SE, df=n-1)
[1] 0.58328
```

Hence, assuming $\mu = 2.075$, the probability for type II error is 58.3%, and the power of the hypothesis test is 41.7%.

12.6 Two-Tailed Test of Population Mean with Unknown Variance

In a two-tailed test of the population mean, the null hypothesis claims that the true population mean μ is equal to a given hypothetical value μ_0.

$$\mu = \mu_0$$

A **type II error** occurs if the hypothesis test based on a random sample fails to reject the null hypothesis even when the true population mean μ is in fact different from μ_0.

Let s^2 be the sample variance. For sufficiently large n, the following statistics of all possible samples of size n resembles a Student t distribution with n-1 degrees of freedom.

$$\frac{\bar{x} - \mu}{s/\sqrt{n}}$$

This allows us to compute the range of sample means for which the null hypothesis will *not* be rejected, and to obtain the probability of type II error. We demonstrate the procedure with the following:

Problem

Suppose the mean weight of King Penguins found in an Antarctic colony last year was 15.4 kg. Assume that in a random sample of 35 penguins, the standard deviation of the weight is 2.5 kg. If actual mean penguin weight is 15.1 kg, what is the probability of type II error for a hypothesis test at .05 significance level?

Solution

We begin with computing the standard error estimate `SE`.

```
> n = 35       # sample size
> s = 2.5      # sample sd
```

```
> SE = s/sqrt(n); SE
[1] 0.42258
```

We next compute the lower and upper bounds of sample means for which the null hypothesis $\mu = 15.4$ would not be rejected.

```
> alpha = .05  # significance level
> mu0 = 15.4   # hypothesis
> I = c(alpha/2, 1-alpha/2)
> q = mu0 + qt(I, df=n-1)*SE; q
[1] 14.541 16.259
```

Therefore, so long as the sample mean is between 14.541 and 16.259 in a hypothesis test, the null hypothesis will not be rejected. Since we assume that the actual population mean is 15.1, we can compute the lower tail probabilities at both end points.

```
> mu = 15.1    # population mean
> p = pt((q - mu)/SE, df=n-1); p
[1] 0.097445 0.995168
```

Finally, the probability of type II error is the probability between the two end points.

```
> diff(p)      # p[2]-p[1]
[1] 0.89772
```

Answer

If the penguin sample size is 35, the sample standard deviation of penguin weight is 2.5 kg and the actual mean population weight is 15.1 kg, then the probability of type II error for testing the null hypothesis $\mu = 15.4$ at .05 significance level is 89.8%, and the power of the hypothesis test is 10.2%.

Exercise

Under same assumptions as above, if actual mean population weight is 14.9 kg, what is the probability of type II errors? What is the power of the hypothesis test?

Solution of Exercise

According to the previous computation, so long as the sample mean is between

14.541 and 16.259 in a hypothesis test, the null hypothesis will not be rejected. Therefore, if the actual mean is 14.9, we can compute the lower tail probabilities at both end points.

```
> mu = 14.9    # population mean
> p = pt((q - mu)/SE, df=n-1); p
[1] 0.20090 0.99857
```

Finally, the probability of type II error is the probability between the two end points.

```
> diff(p)      # p[2]-p[1]
[1] 0.79767
```

Hence, assuming $\mu = 14.9$, the probability for type II error is 79.8%, and the power of the hypothesis test is 20.2%.

Chapter 13
Inference About Two Populations

It is often necessary to compare two populations based on their data samples. In the following tutorials, we discuss how to estimate the difference of means and difference of proportions between two populations that fit the normal distribution.

13.1 Population Mean Between Two Matched Samples

Two data samples are said to be **matched** if they come from repeated observations of the same subject. Here, we assume that the data populations follow the normal distribution. Using the **paired t-test**, we can obtain an interval estimate of the difference of population means.

Example

In the built-in data set named **immer**, we can find barley yields of six selected locations in each year of 1931 and 1932. The yield data are presented in the data frame columns `Y1` and `Y2`.

```
> library(MASS)
> head(immer)
  Loc Var    Y1    Y2
1  UF   M  81.0  80.7
2  UF   S 105.4  82.3
    .....
```

Problem

Assuming that the data in `immer` follows the normal distribution, find the 95% confidence interval estimate of the difference between the mean barley yield in years 1931 and 1932.

Solution

We apply the function `t.test` to compute the difference in means of the matched samples. As it is a paired test, we set the argument `paired` as `TRUE`.

```
> t.test(immer$Y1, immer$Y2,
+     paired=TRUE)

        Paired t-test

data:  immer$Y1 and immer$Y2
t = 3.324, df = 29, p-value = 0.002413
```

```
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
  6.122 25.705
sample estimates:
mean of the differences
                 15.913
```

Answer

For years 1931 and 1932, the 95% confidence interval of the difference in means of the barley yield in the data set `immer` is between 6.122 and 25.705.

Exercise

Using the barley yield record in the data set `immer`, demonstrate how to estimate the difference between the means of matched samples using your textbook formula.

Solution of Exercise

Let X to be the difference of the two matched samples. Then the interval estimate of the population mean of X is:

$$\bar{x} \pm t_{\alpha/2} \frac{s}{\sqrt{n}}$$

where $t_{\alpha/2}$ is the 100(1-α/2) percentile of the Student t distribution with n-1 degrees of freedom, s is the sample standard deviation of X, and n is the sample size.

Therefore we evaluate as follows:

```
> X = immer$Y1 - immer$Y2
> n = length(X)
>
> alpha = .05
> t.alpha = qt(1-alpha/2, df=n-1)
> t.alpha
[1] 2.0452
>
> E = t.alpha * sqrt(var(X)/n)
> mean(X) + c(-E, E)
[1]  6.122 25.705
```

The computation thus shows identical result as `t.test`.

13.2 Population Mean Between Two Independent Samples

Two data samples are called **independent** if they come from unrelated populations and the samples do not affect each other. Here, we assume that the data populations follow the normal distribution. Using the **unpaired t-test**, we can obtain an interval estimate of the difference between two population means.

Example

In the data set `mtcars`, the data frame column `mpg` contains gas mileage statistics of various 1974 U.S. automobiles.

```
> mtcars$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 ...
```

Meanwhile, another data column in `mtcars`, named `am`, indicates the transmission type of the automobile model (0 = automatic, 1 = manual).

```
> mtcars$am
 [1] 1 1 1 0 0 0 0 0 ...
```

We can regard the gas mileages for manual and automatic transmissions as two independent data populations.

Problem

Assuming that the data in `mtcats` follows the normal distribution, find the 95% confidence interval estimate of the difference between the mean gas mileages of manual and automatic transmissions.

Solution

As discussed in our previous tutorial on data frame row slice, the gas mileage for automatic transmission can be expressed as follows:

```
> L = mtcars$am == 0
> mpg.auto = mtcars[L,]$mpg
> mpg.auto
 [1] 21.4 18.7 18.1 14.3 24.4 ...
```

By applying the negation of the vector `L`, we can find the gas mileage for manual transmission.

```
> mpg.manual = mtcars[!L,]$mpg
> mpg.manual
 [1] 21.0 21.0 22.8 32.4 30.4 ...
```

We can now apply the function `t.test` to compute the difference in means of the two sample data.

```
> t.test(mpg.auto, mpg.manual)

     Welch Two Sample t-test

data:  mpg.auto and mpg.manual
t = -3.7671, df = 18.332, p-value = 0.001374
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -11.2802  -3.2097
sample estimates:
mean of x mean of y
   17.147    24.392
```

Answer

In the data set `mtcars`, the mean gas mileage of automatic transmission is 17.15 mpg, and the manual transmission is 24.39 mpg. The 95% confidence interval of the difference in mean gas mileage is between 3.21 and 11.28 mpg.

Alternative Solution

We can model the response variable `mtcars$mpg` by the predictor `mtcars$am`, and apply the function `t.test` to estimate the difference in population means.

```
> t.test(mpg ~ am, data=mtcars)

     Welch Two Sample t-test

data:  mpg by am
t = -3.7671, df = 18.332, p-value = 0.001374
```

```
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -11.2802  -3.2097
sample estimates:
mean in group 0 mean in group 1
         17.147          24.392
```

Note

Some textbooks truncate down the degree of freedom to an integer, and the result would differ from `t.test`.

Exercise

Using the gas mileage data in `mtcars`, demonstrate how to estimate the difference between the means of independent samples using your textbook formula.

Solution of Exercise

The interval estimate of the difference in population means is:

$$\bar{x}_1 - \bar{x}_2 \pm t_{\alpha/2}\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

where $t_{\alpha/2}$ is the 100(1-α/2) percentile of the Student t distribution with *df* degrees of freedom, s_1^2 and s_2^2 are the sample variances, n_1 and n_2 are the sample sizes.

The degree of freedom for the Student t distribution in the formula above is:

$$df = \left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2 / \left\{\frac{1}{n_1 - 1}\left(\frac{s_1^2}{n_1}\right)^2 + \frac{1}{n_2 - 1}\left(\frac{s_2^2}{n_2}\right)^2\right\}$$

Step 1: Find the degree of freedom *df*. We begin with denoting the quantity s_1^2/n_1 by q_1, and the quantity s_2^2/n_2 by q_2.

```
> n1 = length(mpg.auto)
> n2 = length(mpg.manual)
>
> q1 = var(mpg.auto)/n1
> q2 = var(mpg.manual)/n2
> u = q1+q2
```

```
>
> v1 = q1*q1/(n1-1)
> v2 = q2*q2/(n2-1)
> df = u*u/(v1+v2); df
[1] 18.332
```

Step 2: Find the margin of error `E` for the 95% confidence interval.

```
> alpha = .05
> t.alpha = qt(1-alpha/2, df=df)
> E = t.alpha * sqrt(q1+q2); E
[1] 4.0353
```

Step 3: Add the margin of error to the point estimate of difference in population means:

```
> xbar1 = mean(mpg.auto)
> xbar2 = mean(mpg.manual)
> xbar1 - xbar2 + c(-E, E)
[1] -11.2802  -3.2097
```

Therefore, the 95% confidence interval of the difference in mean gas mileages is between 3.21 and 11.28 mpg.

13.3 Comparison of Two Population Proportions

A survey conducted in two distinct populations will produce different results. It is often necessary to compare the survey response proportion between the two populations. Here, we assume that the data populations follow the normal distribution.

Example

In the built-in data set named `quine`, children from an Australian town are classified by ethnic background, gender, age, learning status and the number of days absent from school.

```
> library(MASS)
> head(quine)
  Eth Sex Age Lrn Days
1   A   M  F0  SL    2
2   A   M  F0  SL   11
    .....
```

More specifically, within the data set, the column `Eth` indicates whether the student is Aboriginal or Not ("A" or "N"), while the column `Sex` indicates Male or Female ("M" or "F").

In R, we can tally the student ethnicity against the gender with the function `table`. As the result shows, within the Aboriginal student population, 38 students are female. Whereas within the Non-Aboriginal student population, 42 are female.

```
> table(quine$Eth, quine$Sex)

    F  M
A 38 31
N 42 35
```

Problem

Assuming that the data in `quine` follows the normal distribution, find the 95% confidence interval estimate of the difference between the female proportion of

Aboriginal students and the female proportion of Non-Aboriginal students, each within their own ethnic group.

Solution

We apply the function `prop.test` to compute the difference in female proportions. The Yates's continuity correction is disabled for pedagogical reasons.

```
> prop.test(
+     table(quine$Eth, quine$Sex),
+     correct=FALSE)

     2-sample test for
     equality of proportions
     without continuity correction

data:  table(quine$Eth, quine$Sex)
X-squared = 0.0041, df = 1, p-value = 0.949
alternative hypothesis: two.sided
95 percent confidence interval:
 -0.15642  0.16696
sample estimates:
 prop 1  prop 2
0.55072 0.54545
```

Answer

The 95% confidence interval estimate of the difference between the female proportion of Aboriginal students and the female proportion of Non-Aboriginal students is between -15.6% and 16.7%.

Exercise

Using the female student data in `quine`, demonstrate how to estimate the difference between two population proportions using your textbook formula.

Solution of Exercise

The interval estimate of the difference between two population proportions is:

$$\bar{p}_1 - \bar{p}_2 \pm z_{\alpha/2}\sqrt{\frac{\bar{p}_1(1-\bar{p}_1)}{n_1} + \frac{\bar{p}_2(1-\bar{p}_2)}{n_2}}$$

where $z_{\alpha/2}$ is the 100(1-α/2) percentile of the standard normal distribution, and n_1 and n_2 are the sample sizes.

Step 1: Recall the contingency table as follows.

```
> table(quine$Eth, quine$Sex)

    F  M
A 38 31
N 42 35
```

Step 2: Find the proportion of female students within the Aboriginal student population:

```
> n1 = 38+31; n1
[1] 69
> p1 = 38/n1; p1
[1] 0.55072
```

Step 3: Find the proportion of female students within the Non-aboriginal student population:

```
> n2 = 42+35; n2
[1] 77
> p2 = 42/n2; p2
[1] 0.54545
```

Step 4: Find the 95% confidence interval estimate:

```
> q1 = p1*(1-p1)/n1
> q2 = p2*(1-p2)/n2
>
> alpha = 0.05
> z.alpha = qnorm(1-alpha/2)
> E = z.alpha*sqrt(q1+q2)
>
> p1 - p2 + c(-E, E)
[1] -0.15642  0.16696
```

Therefore, the 95% confidence interval estimate of the difference between the two female population proportions is between -15.6% and 16.7%.

Chapter 14
Goodness of Fit

Many statistical quantities derived from data samples are found to follow the Chi-squared distribution. In the following tutorials, we will demonstrate how to use it to test whether a particular population fits certain theoretical probability distribution.

14.1 Multinomial Goodness of Fit

A data population is called **multinomial** if it is categorical, and has been classified into a collection of discrete non-overlapping classes.

The null hypothesis for **goodness of fit test for multinomial distribution** is that the observed frequency f_i is equal to an expected frequency count e_i in each category. The hypothesis to be rejected if the p-value of the following **Chi-squared test statistic** is less than a given significance level α.

$$\chi^2 = \sum_i \frac{(f_i - e_i)^2}{e_i}$$

Example

In the built-in data set `survey`, the column `Smoke` contains survey response about student smoking habit. As there are exactly four proper response in the survey: "Heavy", "Regul" (regularly), "Occas" (occasionally) and "Never", the `Smoke` data column is multinomial. This can be confirmed with the function `levels`:

```
> library(MASS)
> levels(survey$Smoke)
[1] "Heavy" "Never" "Occas" "Regul"
```

As discussed in our previous tutorial on frequency distribution of qualitative data, we can find the frequency distribution of the `Smoke` variable with the `table` function.

```
> smoke.freq = table(survey$Smoke)
> smoke.freq

Heavy Never Occas Regul
   11   189    19    17
```

Problem

Suppose the campus smoking statistic is as below. Determine whether it is

supported by the sample data in `survey` at .05 significance level.

```
Heavy   Never   Occas   Regul
4.5%    79.5%   8.5%    7.5%
```

Solution

We save the campus smoking statistic in the vector `smoke.prob`. Then we apply the function `chisq.test` and perform the Chi-Squared test.

```
> smoke.prob = +
      c(.045, .795, .085, .075)
> chisq.test(smoke.freq,
+     p=smoke.prob)

     Chi-squared test for
     given probabilities

data:  smoke.freq
X-squared = 0.1074, df = 3, p-value = 0.991
```

Answer

As the p-value 0.991 is greater than the .05 significance level, we do not reject the null hypothesis that the sample data in `survey` supports the campus-wide smoking statistic.

Exercise

Using the smoking habit data in `survey`, demonstrate how to conduct the Chi-squared goodness of fit test by computing the p-value with the textbook formula.

Solution of Exercise

Step 1: Save the observed frequency in `f`:

```
> f = table(survey$Smoke)
```

Step 2: Find the expected frequency count `e` by multiplying the assumed probability with the sample size:

```
> e = smoke.prob*length(survey$Smoke)
```

```
> e
[1]  10.665 188.415  20.145  17.775
```

Step 3: Find the Chi-squared test statistic:

```
> d = f-e
> chi = sum(d*d/e); chi
[1] 0.11121
```

Step 4: Find the p-value with the function `pchisq`:

```
> df = length(f)-1
> pchisq(chi, df=df, lower=FALSE)
[1] 0.99046
```

We note that the p-value 0.99046 as found is almost identical to the one from `chisq.test`, and would yield the same conclusion on the null hypothesis.

14.2 Chi-squared Test of Independence

Two random variables *x* and *y* are said to be **independent** if the probability distribution of one variable is not affected by the presence of another.

Assume f_{ij} is the observed frequency count of events belonging to both *i*-th category of *x* and *j*-th category of *y*. Also assume e_{ij} to be the corresponding expected frequency count if *x* and *y* are independent. The null hypothesis of the independence assumption is to be rejected if the p-value of the following Chi-squared test statistic is less than a given significance level α.

$$\chi^2 = \sum_{i,\ j} \frac{(f_{ij} - e_{ij})^2}{e_{ij}}$$

Example

In the built-in data set `survey`, the `Smoke` column records the students smoking habit, while the `Exer` column records their exercise level. The allowed values in `Smoke` are "Heavy", "Regul" (regularly), "Occas" (occasionally) and "Never". As for `Exer`, they are "Freq" (frequently), "Some" and "None".

We can tally the students smoking habit against the exercise level with the function `table`. The result is called the **contingency table** of the two variables.

```
> library(MASS)
> tbl = table(
+     survey$Smoke, survey$Exer)
> tbl          # contingency table

        Freq None Some
  Heavy    7    1    3
  Never   87   18   84
  Occas   12    3    4
  Regul    9    1    7
```

Problem

Test the hypothesis whether the students smoking habit is independent of their exercise level at .05 significance level.

Solution

We apply the function `chisq.test` to the contingency table `tbl` above, and found the p-value to be 0.4828.

```
> chisq.test(tbl)

     Pearson's Chi-squared test

data:  table(survey$Smoke, survey$Exer)
X-squared = 5.4885, df = 6, p-value = 0.4828

Warning message:
In chisq.test(table(survey$Smoke, survey$Exer)) :
  Chi-squared approximation may be incorrect
```

Answer

As the p-value 0.4828 is greater than the .05 significance level, we do not reject the null hypothesis that the smoking habit is independent of the exercise level of the students.

Enhanced Solution

The warning message found in the solution above is due to the small cell values in the contingency table. To avoid such warning, we can combine the second and third columns of `tbl`, and save it in a new table named `ctbl`. Then we apply the function `chisq.test` against the new table `ctbl` instead.

```
> ctbl = cbind(
+   tbl[,"Freq"],
+   tbl[,"None"] + tbl[,"Some"])

> ctbl
      [,1] [,2]
Heavy    7    4
Never   87  102
Occas   12    7
Regul    9    8

> chisq.test(ctbl)
```

```
     Pearson's Chi-squared test

data:  ctbl
X-squared = 3.2328, df = 3, p-value = 0.3571
```

Exercise

Using the student data of smoking and exercise in `survey`, demonstrate how to conduct the Chi-squared independence test by computing the p-value with the textbook formula.

Solution of Exercise

Step 1: Save the contingency table in a variable `f`.

```
> f = table(survey$Smoke,
+     survey$Exer); f

        Freq None Some
  Heavy    7    1    3
  Never   87   18   84
  Occas   12    3    4
  Regul    9    1    7
```

Step 2: Find the sum of each row using the function `rowSums`.

```
> rowsum = rowSums(f); rowsum
Heavy Never Occas Regul
   11   189    19    17
```

Alternatively, we can use the general purpose function `apply` and set the margin option as 1 for the same result.

```
> rowsum = apply(f, 1, sum); rowsum
```

Step 3: Find the sum of each column with the function `colSums`.

```
> colsum = colSums(f); colsum
Freq None Some
 115   23   98
```

Similarly, we can use the general purpose function `apply` and set the margin option as 2 for the same result.

```
> colsum = apply(f, 2, sum); colsum
```

Step 4: If we treat `rowsum` as a 4x1 matrix, and `colsum` as a 3x1 matrix, we can find their outer product by performing the matrix multiplication of `rowsum` with the *transpose* of `colsum`.

```
> p = rowsum %*% t(colsum)
```

Step 5: Divide the outer product `p` by the sample size and get the expected frequency count `e`.

```
> e = p / nrow(survey); e
        Freq    None    Some
[1,]  5.3376  1.0675  4.5485
[2,] 91.7089 18.3418 78.1519
[3,]  9.2194  1.8439  7.8565
[4,]  8.2489  1.6498  7.0295
```

Step 6: Find the Chi-squared test statistic:

```
> d = f-e
> chi = sum(d*d/e); chi
[1] 5.516
```

Step 7: Find the p-value with the function `pchisq`:

```
> df = (nrow(f)-1)*(ncol(f)-1); df
[1] 6
> pchisq(chi, df=df, lower=FALSE)
[1] 0.47952
```

We note that the p-value 0.47952 as found is very close to the p-value from `chisq.test`, and would yield the same conclusion as before.

Chapter 15
Analysis of Variance

In an experimental study, various treatments are applied to test subjects and the response data is gathered for analysis. A critical tool for carrying out such analysis is the **Analysis of Variance** (ANOVA). It enables a researcher to differentiate treatment results based on easily computed statistical quantities from the treatment outcome.

The statistical process is derived from estimates of the population variance via two separate approaches. The first approach is based on the variance of the sample means, and the second one is based on the mean of the sample variances. Under the ANOVA assumptions as stated below, the ratio of the two statistical estimates follows the F distribution. Hence we can test the null hypothesis on the equality of various response data from different treatments via estimates of critical regions.

The assumptions of ANOVA are:

- The treatment responses are independent of each other.
- The response data follow the normal distribution.
- The variances of the response data are identical.

In the following tutorials, we demonstrate how to perform ANOVA on a few basic experimental designs.

15.1 Completely Randomized Design

In a **completely randomized design**, there is only one primary factor under consideration in the experiment. The test subjects are assigned to treatment levels of the primary factor at random.

Example

A fast food franchise is test marketing 3 new menu items. To find out if they the same popularity, 18 franchisee restaurants are randomly chosen for participation in the study. In accordance with the completely randomized design, 6 of the restaurants are randomly chosen to test market the first new menu item, another 6 for the second menu item, and the remaining 6 for the last menu item.

Problem

Suppose the following table represents the sales figures of the 3 new menu items in the 18 restaurants after a week of test marketing. At .05 level of significance, test whether the mean sales figures of the 3 new menu items are all equal.

Item1	Item2	Item3
22	52	16
42	33	24
44	8	19
52	47	18
45	43	34
37	32	39

Solution

Step 1: Copy the sales figure above into a table file named "fastfood-1.txt" with a text editor.

Step 2: Load the file into a data frame `df1` with the function `read.table`. Since the first line in the file contains the column names, we enable the `header` option.

```
> df1 = read.table(
```

```
+     "fastfood-1.txt",
+     header=TRUE)
> df1
  Item1 Item2 Item3
1    22    52    16
2    42    33    24
3    44     8    19
4    52    47    18
5    45    43    34
6    37    32    39
```

Step 3: Concatenate the data rows of `df1` into a single vector `r`. Note the use of the function `t` for matrix transpose.

```
> r = c(t(as.matrix(df1))) # response
> r
 [1] 22 52 16 42 33 ....
```

Step 4: Assign new variables for the treatment levels and number of observations.

```
> f = c("Item1", "Item2", "Item3")
> k = 3      # number of treatments
> n = 6      # data per treatment
```

Step 5: Create a vector of treatment factors that corresponds to members of `r` in step 3.

```
> tm = gl(k, 1, n*k, factor(f))
> tm
 [1] Item1 Item2 Item3 Item1 ...
```

Step 6: Apply the function `aov` to a formula that describes the response `r` by the treatment factor `tm`.

```
> av = aov(r ~ tm)
```

Step 7: Print out the ANOVA table with the function `summary`.

```
> summary(av)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2    745     373    2.54   0.11
Residuals   15   2200     147
```

Answer

Since the p-value of 0.11 is greater than the .05 significance level, we do not reject the null hypothesis that the mean sales figures of the new menu items are all equal.

Exercise

Create the response data in step 3 above along *vertical* columns instead of horizontal rows. Adjust the factor levels in step 5 accordingly.

Solution of Exercise

We load the data into a data frame as before:

```
> df1 = read.table(
+     "fastfood-1.txt",
+     header=TRUE)
```

Then we concatenate the column vectors by *skipping* the use of the function `t`.

```
> r = c(as.matrix(df1))  # response
> r
 [1] 22 42 44 52 45 ...
>
```

Now we create the factor values `tm` by setting arguments in the function `gl` according to how the response data is arranged.

```
> f = c("Item1", "Item2", "Item3")
> k = 3      # treatment levels
> n = 6      # data per treatment
> tm = gl(k, n, n*k, factor(f))
> tm
 [1] Item1 Item1 Item1 Item1 ...
```

Finally, we apply the function `aov` and print out the summary.

```
> av = aov(r ~ tm)
> summary(av)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2    745     373    2.54   0.11
Residuals   15   2200     147
```

15.2 Randomized Block Design

In a **randomized block design**, there is only one primary factor under consideration in the experiment. Similar test subjects are grouped into blocks. Each block is tested against all treatment levels of the primary factor at random order. The intention is to eliminate possible extraneous influence by other factors.

Example

A fast food franchise is test marketing 3 new menu items. To find out if they have the same popularity, 6 franchisee restaurants are randomly chosen for participation in the study. In accordance with the randomized block design, each restaurant will be test marketing all 3 new menu items. Furthermore, a restaurant will test market only one menu item per week, and it takes 3 weeks to test market all menu items. The testing order of the menu items for each restaurant is randomly assigned as well.

Problem

Suppose each row in the following table represents the sales figures of the 3 new menu in a restaurant after a week of test marketing. At .05 level of significance, test whether the mean sales figures of the 3 new menu items are all equal.

```
Item1 Item2 Item3
   31    27    24
   31    28    31
   45    29    46
   21    18    48
   42    36    46
   32    17    40
```

Solution

Step 1: Copy the sales figure above into a table file named "fastfood-2.txt" with a text editor.

Step 2: Load the file into a data frame `df2` with the function `read.table`. Since

the first line in the file contains the column names, we enable the `header` option.

```
> df2 = read.table(
+     "fastfood-2.txt",
+     header=TRUE)
> df2
  Item1 Item2 Item3
1    31    27    24
2    31    28    31
3    45    29    46
4    21    18    48
5    42    36    46
6    32    17    40
```

Step 3: Concatenate the data rows of `df2` into a single vector `r`. Note the use of the function `t` for matrix transpose.

```
> r = c(t(as.matrix(df2)))
> r
 [1] 31 27 24 31 28 ...
```

Step 4: Assign new variables for the treatment levels and number of control blocks.

```
> f = c("Item1", "Item2", "Item3")
> k = 3      # treatment levels
> n = 6      # data per treatment
```

Step 5: Create a vector of treatment factors that corresponds to members of `r` in step 3.

```
> tm = gl(k, 1, n*k, factor(f))
> tm
 [1] Item1 Item2 Item3 Item1 ...
```

Step 6: Similarly, create a vector of blocking factors for members of the response data `r`.

```
> blk = gl(n, k, k*n)
> blk
 [1] 1 1 1 2 2 2 3 3 3 4 4 4 ...
Levels: 1 2 3 4 5 6
```

Step 7: Apply the function `aov` to a formula that describes the response `r` by both the treatment factor `tm` and the block control `blk`.

```
> av = aov(r ~ tm + blk)
```

Step 8: Print out the ANOVA table with the function `summary`.

```
> summary(av)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2    539     269    4.96  0.032 *
blk          5    560     112    2.06  0.155
Residuals   10    543      54
```

Answer

Since the p-value of 0.032 is less than the .05 significance level, we reject the null hypothesis that the mean sales figures of the new menu items are all equal.

Exercise

Create the response data in step 3 above along *vertical* columns instead of horizontal rows. Adjust the factor levels in steps 5 and 6 accordingly.

Solution of Exercise

We load the data into a data frame as before:

```
> df2 = read.table(
+     "fastfood-2.txt",
+     header=TRUE)
```

Then we concatenate the column vectors by *skipping* the use of the function `t`.

```
> r = c(as.matrix(df2))
> r
 [1] 31 31 45 21 42 ...
```

Now we create the factor values `tm` and `blk` by setting arguments in the function `gl` according to how the response data is arranged.

```
> f = c("Item1", "Item2", "Item3")
> k = 3      # treatment levels
> n = 6      # data per treatment
>
> tm = gl(k, n, n*k, factor(f))
> tm
 [1] Item1 Item1 Item1 Item1 ...
```

```
>
> blk = gl(n, 1, k*n)
> blk          # blocking factor
 [1] 1 2 3 4 5 6 1 2 3 4 5 6 1 2 3 4 5 6
Levels: 1 2 3 4 5 6
```

Finally, we apply the function `aov` and print out the summary.

```
> av = aov(r ~ tm + blk)
> summary(av)
            Df Sum Sq Mean Sq F value Pr(>F)
tm           2    539   269.4    4.96  0.032 *
blk          5    560   112.0    2.06  0.155
Residuals   10    543    54.3
```

15.3 Factorial Design

In a **factorial design**, there are more than one factors under consideration in the experiment. The test subjects are assigned to treatment levels of every factor combinations at random.

Example

A fast food franchise is test marketing 3 new menu items in both East and West Coasts of continental United States. To find out if they have the same popularity, 12 franchisee restaurants from each coast are randomly chosen for participation in the study. In accordance with the factorial design, within the 12 restaurants from East Coast, 4 are randomly chosen to test market the first new menu item, another 4 for the second menu item, and the remaining 4 for the last menu item. The 12 restaurants from the West Coast are arranged likewise.

Problem

Suppose the following tables represent the sales figures of the 3 new menu items after a week of test marketing. Each row in the upper table represents the sales figures of 3 different East Coast restaurants. The lower half represents West Coast restaurants. At .05 level of significance, test whether the mean sales figures of the new menu items are all equal. Decide also whether the mean sales figures of the two coastal regions differ.

```
East Coast:
==========
   Item1 Item2 Item3
E1    25    39    36
E2    36    42    24
E3    31    39    28
E4    26    35    29

West Coast:
==========
   Item1 Item2 Item3
W1    51    43    42
W2    47    39    36
```

```
W3    47    53    32
W4    52    46    33
```

Solution

Step 1: Save the sales figure into a file named "fastfood-3.csv" in CSV format as follows:

```
Item1,Item2,Item3
E1,25,39,36
E2,36,42,24
E3,31,39,28
E4,26,35,29
W1,51,43,42
W2,47,39,36
W3,47,53,32
W4,52,46,33
```

Step 2: Load the file into a data frame `df3` with the function `read.csv`.

```
> df3 = read.csv("fastfood-3.csv")
```

Step 3: Concatenate the data rows of `df3` into a single vector `r`. Note the use of the function `t` for matrix transpose.

```
> r = c(t(as.matrix(df3)))
> r
 [1] 25 39 36 36 42 ...
```

Step 4: Assign new variables for the treatment levels and number of observations.

```
> f1 = c("Item1", "Item2", "Item3")
> f2 = c("East", "West")
> k1 = length(f1)
> k2 = length(f2)
> n = 4        # data per treatment
```

Step 5: Create a vector that corresponds to the first treatment level of the response data `r` in step 3.

```
> tm1 = gl(k1, 1, n*k1*k2,
+     factor(f1))
> tm1
 [1] Item1 Item2 Item3 Item1 ...
```

Step 6: Similarly, create a vector that corresponds to the second treatment level of the response data `r` in step 3.

```
> tm2 = gl(k2, n*k1, n*k1*k2,
+     factor(f2))
> tm2
 [1] East East East East East ...
```

Step 7: Apply the function `aov` to a formula that describes the response `r` by the two treatment factors `tm1` and `tm2` with interaction.

```
> av = aov(r ~ tm1 * tm2)
```

Step 8: Print out the ANOVA table with the function `summary`.

```
> summary(av)
             Df Sum Sq Mean Sq F value  Pr(>F)
tm1           2    385     193    9.55  0.0015 **
tm2           1    715     715   35.48 1.2e-05 ***
tm1:tm2       2    234     117    5.81  0.0113 *
```

Answer

Since the p-value of 0.0015 for the menu items is less than the .05 significance level, we reject the null hypothesis that the mean sales figures of the new menu items are all equal. Moreover, the p-value of 1.2e-05 for the east-west coasts comparison is also less than the .05 significance level. It shows there is a difference in overall sales figures between the coasts. Finally, the last p-value of 0.0113 (< 0.05) indicates that there is a possible interaction between the menu item and coast location factors, *i.e.*, customers from different coastal regions probably have different tastes.

Exercise

Create the response data in step 3 above along vertical columns instead of horizontal rows. Adjust the factor levels in steps 5 and 6 accordingly.

Solution of Exercise

We load the data into a data frame as before:

```
> df3 = read.csv("fastfood-3.csv")
```

Then we concatenate the column vectors by *skipping* the use of the function `t`.

```
> r = c(as.matrix(df3))
> r
 [1] 25 36 31 26 51 ...
```

Now we create the factor values `tm1` and `tm2` by setting arguments in the function `gl` according to how the response data is arranged.

```
> f1 = c("Item1", "Item2", "Item3")
> f2 = c("East", "West")
> k1 = length(f1)
> k2 = length(f2)
> n = 4      # data per treatment
>
> tm1 = gl(k1, n*k2, n*k1*k2,
+     factor(f1))
> tm1
 [1] Item1 Item1 Item1 Item1 ...
>
> tm2 = gl(k2, n, n*k1*k2,
+     factor(f2))
> tm2
 [1] East East East East West ...
```

Finally, we apply the function `aov` and print out the summary.

```
> av = aov(r ~ tm1 * tm2)
> summary(av)
        Df Sum Sq Mean Sq F value  Pr(>F)
tm1      2    385     193    9.55  0.0015 **
tm2      1    715     715   35.48 1.2e-05 ***
tm1:tm2  2    234     117    5.81  0.0113 *
```

Chapter 16
Non-parametric Methods

A statistical method is called **non-parametric** if it makes no assumptions on the population distribution or sample size.

This is in contrast with most parametric methods in elementary statistics that assume the data to be quantitative, the population to be normally distributed, and the sample size to be sufficiently large.

In general, conclusions drawn from non-parametric methods are not as powerful as the parametric ones. However, as non-parametric methods make fewer assumptions, they are more flexible, more robust, and applicable to qualitative data.

16.1 Sign Test

A **sign test** is used to decide whether a population with a binomial distribution has equal chances of success and failure.

Example

A soft drink company has invented a new drink, and would like to find out if it will be as popular as the existing favorite drink. For this purpose, its research department arranges 18 participants for taste testing. Each participant tries both drinks in random order before giving his or her opinion.

Problem

It turns out that 5 of the participants like the new drink better, and the rest prefer the old one. At .05 significance level, can we reject the notion that the two drinks are equally popular?

Solution

The null hypothesis is that the drinks are equally popular. Here we apply the function `binom.test`. As the p-value turns out to be 0.096525, and is greater than the .05 significance level, we do not reject the null hypothesis.

```
> binom.test(5, 18)

     Exact binomial test

data:  5 and 18
number of successes = 5, number of trials = 18,
p-value = 0.09625
alternative hypothesis:
  true probability of success is not equal to 0.5
95 percent confidence interval:
 0.09695 0.53480
sample estimates:
probability of success
               0.27778
```

Answer

At .05 significance level, we do not reject the notion that the two drinks are equally popular.

16.2 Wilcoxon Signed-Rank Test

Two data samples are said to be **matched** if they come from repeated observations of the same subject. Using the **Wilcoxon Signed-Rank Test**, we can decide whether the corresponding data distributions are identical without assuming them to follow the normal distribution.

Example

In the built-in data set named **immer**, the barley yield in years 1931 and 1932 of a few selected fields are recorded. The yield data are presented in the data frame columns `Y1` and `Y2`.

```
> library(MASS)
> head(immer)
  Loc Var    Y1    Y2
1  UF   M  81.0  80.7
2  UF   S 105.4  82.3
    .....
```

Problem

Without assuming the data to have normal distribution, test at .05 significance level if the barley yields of 1931 and 1932 in data set `immer` have identical data distributions.

Solution

The null hypothesis is that the barley yields of the two sample years are identically distributed. To test the hypothesis, we apply the function `wilcox.test` to compare the matched samples. For the paired test, we set the argument `paired` as `TRUE`. As the p-value turns out to be 0.005318, and is less than the .05 significance level, we reject the null hypothesis.

```
> wilcox.test(immer$Y1, immer$Y2,
+     paired=TRUE)
```

```
        Wilcoxon signed rank test
        with continuity correction

data:  immer$Y1 and immer$Y2
V = 368.5, p-value = 0.005318
alternative hypothesis:
  true location shift is not equal to 0

Warning message:
In wilcox.test.default(
  immer$Y1, immer$Y2, paired = TRUE) :
  cannot compute exact p-value with ties
```

Answer

At .05 significance level, we conclude that the barley yields of 1931 and 1932 from the data set `immer` have different data distributions.

16.3 Mann-Whitney-Wilcoxon Test

Two data samples are called **independent** if they come from distinct populations and the samples do not affect each other. Using the **Mann-Whitney-Wilcoxon Test**, we can decide whether the data distributions are identical *without* assuming them to follow the normal distribution.

Example

In the data frame column `mpg` of the data set `mtcars`, there are gas mileage data of various 1974 U.S. automobiles.

```
> mtcars$mpg
 [1] 21.0 21.0 22.8 21.4 18.7 ...
```

Meanwhile, another data column in `mtcars`, named `am`, indicates the transmission type of the automobile model (0 = automatic, 1 = manual). In other words, it is the differentiating factor of the transmission type.

```
> mtcars$am
 [1] 1 1 1 0 0 0 0 0 ...
```

In particular, the gas mileage data for manual and automatic transmissions are independent.

Problem

Without assuming the data to have normal distribution, decide at .05 significance level if the gas mileage data of manual and automatic transmissions in `mtcars` have identical data distributions.

Solution

The null hypothesis is that the gas mileage data of manual and automatic transmissions have identical data distributions. To test the hypothesis, we apply the function `wilcox.test` to compare the independent samples. As the p-value

turns out to be 0.001817, and is less than the .05 significance level, we reject the null hypothesis.

```
> wilcox.test(mpg ~ am, data=mtcars)

        Wilcoxon rank sum test
        with continuity correction

data:  mpg by am
W = 42, p-value = 0.001871
alternative hypothesis:
  true location shift is not equal to 0

Warning message:
In wilcox.test.default(
  x = c(21.4, 18.7, 18.1, 14.3, 24.4, 22.8,  :
  cannot compute exact p-value with ties
```

Answer

At .05 significance level, we conclude that the gas mileage data of manual and automatic transmissions in `mtcars` have different data distributions.

16.4 Kruskal-Wallis Test

A collection of data samples are called **independent** if they come from unrelated populations and the samples do not affect each other. Using the **Kruskal-Wallis Test**, we can decide whether the data distributions are identical *without* assuming them to follow the normal distribution.

Example

In the built-in data set named `airquality`, the daily air quality measurements in New York, May to September 1973, are recorded. The ozone density are presented in the data frame column `Ozone`.

```
> head(airquality)
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
    .....
```

Problem

Without assuming the data to be normally distributed, test at .05 significance level if the monthly ozone density in New York has identical data distributions during the months from May 1973 to September 1973.

Solution

The null hypothesis is that the monthly ozone density have identical data distributions. To test the hypothesis, we apply the function `kruskal.test` to compare the independent monthly data. The p-value turns out to be nearly zero (6.901e-06). Hence we reject the null hypothesis.

```
> kruskal.test(Ozone ~ Month,
+     data = airquality)

        Kruskal-Wallis rank sum test

data:  Ozone by Month
```

```
Kruskal-Wallis chi-squared = 29.267, df = 4,
  p-value = 6.901e-06
```

Answer

At .05 significance level, we conclude that the monthly ozone density in New York from May to September 1973 have different data distributions.

Chapter 17
Simple Linear Regression

A **simple linear regression model** describes a dependent variable *y* by an independent variables *x* using a linear equation. In the linear regression model below, the numbers α and β are called **parameters**, and ε is called the **error term**.

$$y = \alpha + \beta x + \epsilon$$

For example, in the data set `faithful`, there are two random variables named `waiting` and `eruptions`. Each row of the data set is an observation of a geyser eruption described by the `(waiting, eruptions)` value pair. The value in the `waiting` variable is the length of time waited until an eruption, and the value in `eruptions` is duration of the eruption. Assigning `waiting` as the independent variable, and `eruptions` as the dependent variable, the linear regression model becomes:

$$Eruptions = \alpha + \beta * Waiting + \epsilon$$

17.1 Estimated Simple Regression Equation

If we choose the parameters α and β in the simple linear regression model so as to minimize the sum of squares of the error term ε, we will have the so called **estimated simple regression equation**. It enables us to compute **fitted values** of y based on values of x:

$$\hat{y} = a + bx$$

Problem

Assuming the simple linear regression model for the data set `faithful`, estimate the next eruption duration if the waiting time since the last eruption has been 80 minutes.

Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
```

Then we extract the parameters of the estimated regression equation with a function named `coefficients`.

```
> coeffs =
+     coefficients(eruption.lm)
> coeffs
(Intercept)     waiting
  -1.874016    0.075628
```

We now fit the eruption duration using the estimated regression equation.

```
> waiting = 80
> duration = coeffs[1] +
```

```
+     coeffs[2]*waiting
> duration
(Intercept)
     4.1762
```

Answer

Based on the simple linear regression model, if the waiting time since the last eruption has been 80 minutes, we expect the eruption to last about 4 minutes.

Alternative Solution

Instead of using the estimated simple regression equation, we will use the generic function `predict`.

Firstly, we wrap the `waiting` parameter value inside a data frame `newdata`.

```
> newdata = data.frame(waiting=80)
```

Then we apply the function `predict` to the linear regression model object `eruption.lm` along with the parameter `newdata`. The result is the fitted eruption duration.

```
> predict(eruption.lm, newdata)
     1
4.1762
```

17.2 Coefficient of Determination

The **coefficient of determination** of a simple linear regression model is the quotient between variance of fitted values and variance of observed values of the dependent variable. It is a measure of the proportion of variance in *y* that is accountable by the fitted values.

$$\begin{aligned} r^2 &= \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2} \\ where\ y_i &= observed\ values, \\ \hat{y}_i &= fitted\ values, \\ and\ \bar{y} &= mean\ value \end{aligned}$$

Problem

Find the coefficient of determination for the simple linear regression model of the data set `faithful`.

Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
```

Then we extract the coefficient of determination from the `r.squared` attribute of its summary.

```
> summary(eruption.lm)$r.squared
[1] 0.81146
```

Answer

The coefficient of determination of the simple linear regression model for the data set `faithful` is 0.81146.

Note

Further detail of the `r.squared` attribute can be found in the R documentation.

```
> help(summary.lm)
```

17.3 Significance Test for Linear Regression

Assume that the error term ε in the linear regression model is independent of the x variable, is normally distributed, has zero mean, and has constant variance. We can decide whether there is any **significant relationship** between x and y by testing the null hypothesis that $\beta = 0$.

Problem

Decide whether there is a significant relationship between the variables in the linear regression model of the data set `faithful` at .05 significance level.

Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
```

Then we print out the F-statistics of the significance test.

```
> summary(eruption.lm)

Call:
lm(formula = eruptions ~ waiting, data = faithful)

Residuals:
    Min      1Q  Median      3Q     Max
-1.2992 -0.3769  0.0351  0.3491  1.1933

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) -1.87402    0.16014   -11.7   <2e-16 ***
waiting      0.07563    0.00222    34.1   <2e-16 ***
---
Residual standard error: 0.497 on 270 degrees of freedom
Multiple R-squared: 0.811, Adjusted R-squared: 0.811
```

```
F-statistic: 1.16e+03 on 1 and 270 DF,
p-value: <2e-16
```

Answer

As the p-value of the `waiting` variable is nearly zero, we reject the null hypothesis that $\beta = 0$. Hence there is a significant relationship between the variables in the linear regression model of the data set `faithful`.

Note

Further detail of the function `summary` for linear regression model can be found in the R documentation.

```
> help(summary.lm)
```

17.4 Confidence Interval of Linear Regression

Assume that the error term ε in the linear regression model is independent of the x variable, is normally distributed, has zero mean, and has constant variance. For a given value of x, an interval estimate of the mean of the dependent variable y is called a **confidence interval**.

Problem

For the data set `faithful`, construct a 95% confidence interval of the eruption duration if the waiting time is 80 minutes.

Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`.

```
> attach(faithful)
> eruption.lm =
+     lm(eruptions ~ waiting)
```

Then we create a data frame containing the waiting time parameter.

```
> newdata = data.frame(waiting=80)
```

We now apply the function `predict` by setting the predictor variable in the argument `newdata`. We also set the interval type as "confidence", and use the default 0.95 confidence level.

```
> predict(eruption.lm, newdata,
+       interval="confidence")
     fit    lwr    upr
1 4.1762 4.1048 4.2476
> detach(faithful)     # clean up
```

Answer

The 95% confidence interval of the eruption duration for the waiting time of 80 minutes is between 4.1048 and 4.2476 minutes.

Note

Further detail of the function `predict` for linear regression model can be found in the R documentation.

```
> help(predict.lm)
```

17.5 Prediction Interval of Linear Regression

Assume that the error term ε in the linear regression model is independent of the x variable, is normally distributed, has zero mean, and has constant variance. For a given value of x, an interval estimate of the dependent variable y is called a **prediction interval**.

Problem

For the data set `faithful`, construct a 95% prediction interval of the eruption duration if the waiting time is 80 minutes.

Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`.

```
> attach(faithful)
> eruption.lm =
+     lm(eruptions ~ waiting)
```

Then we create a data frame containing the waiting time parameter.

```
> newdata = data.frame(waiting=80)
```

We now apply the function `predict` by setting the predictor variable in the argument `newdata`. We also set the interval type as "predict", and use the default 0.95 confidence level.

```
> predict(eruption.lm, newdata,
+       interval="predict")
     fit    lwr    upr
1 4.1762 3.1961 5.1564
> detach(faithful)     # clean up
```

Answer

The 95% prediction interval of the eruption duration for the waiting time of 80 minutes is between 3.1961 and 5.1564 minutes.

Note

Further detail of the function `predict` for linear regression model can be found in the R documentation.

```
> help(predict.lm)
```

17.6 Residual Plot

The **residual** data of a simple linear regression model is the difference between the observed values and the fitted values of the dependent variable.

$$Residual = y - \hat{y}$$

Problem

Plot the residual of the simple linear regression model of the data set `faithful` against the independent variable `waiting`.

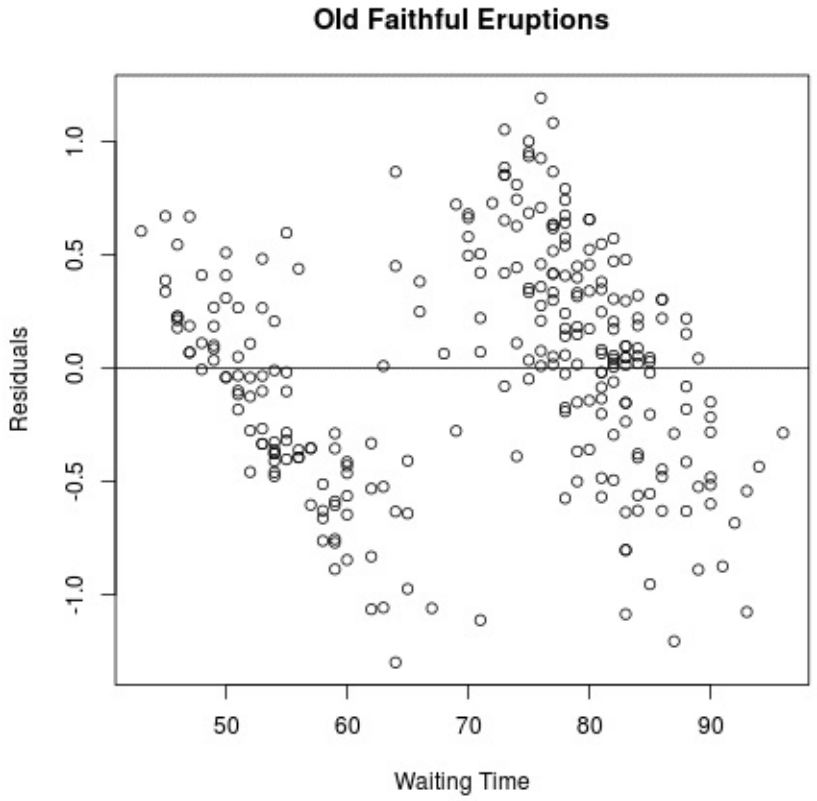
Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`. Then we compute the residual with the function `resid`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
> eruption.res = resid(eruption.lm)
```

We now plot the residual values against the independent variable `waiting`.

```
> plot(faithful$waiting,
+     eruption.res,
+     ylab="Residuals",
+     xlab="Waiting Time",
+     main="Old Faithful Eruptions")
> abline(0, 0)     # the horizon
```



Note

Further detail of the function `resid` can be found in the R documentation.

```
> help(resid)
```

17.7 Standardized Residual

The **standardized residual** data of a simple linear regression model is its residual divided by the standard deviation.

$$Standardized\ Residual = \frac{Residual}{Standard\ Deviation\ of\ Residual}$$

Problem

Plot the standardized residual of the simple linear regression model of the data set `faithful` against the independent variable `waiting`.

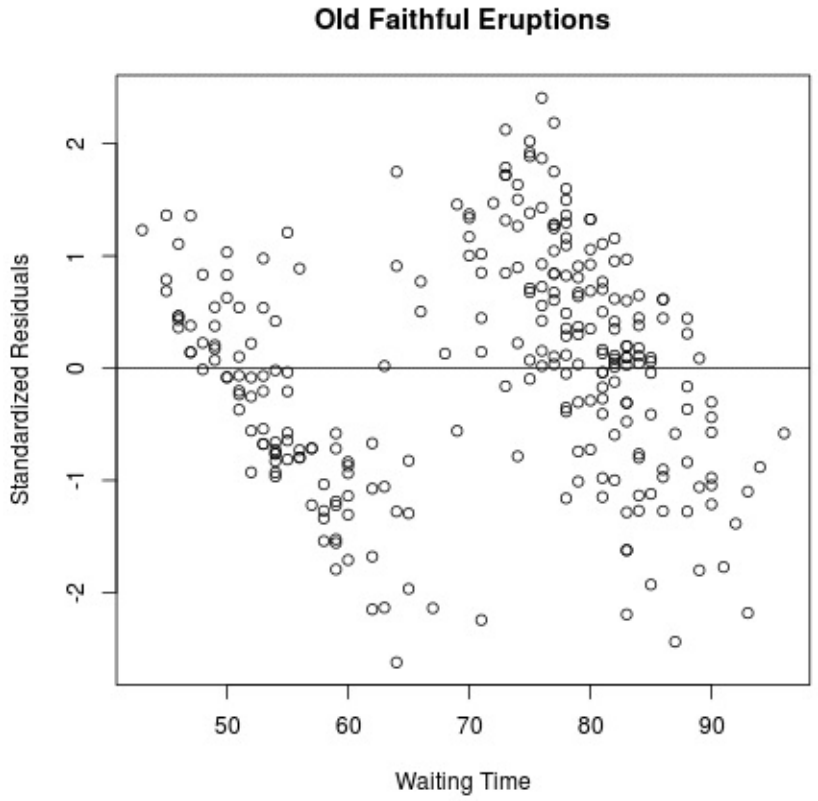
Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`. Then we compute the standardized residual with the function `rstandard`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
> eruption.stdres =
+     rstandard(eruption.lm)
```

We now plot the standardized residual against the independent variable `waiting`.

```
> plot(faithful$waiting,
+     eruption.stdres,
+     ylab="Standardized Residuals",
+     xlab="Waiting Time",
+     main="Old Faithful Eruptions")
> abline(0, 0)     # the horizon
```



Note

Further detail of the function `rstandard` can be found in the R documentation.

```
> help(rstandard)
```

17.8 Normal Probability Plot of Residuals

The **normal probability plot** is a graphical tool for comparing a data set against the normal distribution. We can apply it to the standardized residual of the linear regression model and therefore verify if the error term ε is actually normally distributed.

Problem

Create the normal probability plot for the standardized residual of the data set `faithful`.

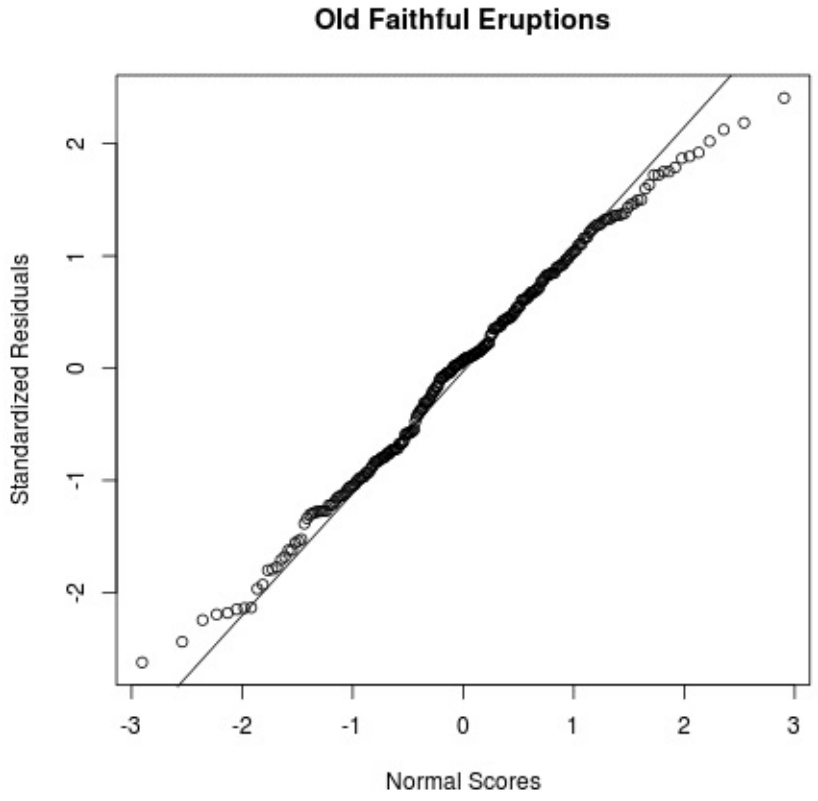
Solution

We apply the function `lm` to a formula that describes the dependent variable `eruptions` by the independent variable `waiting`, and save the new linear regression model in a variable `eruption.lm`. Then we compute the standardized residual with the function `rstandard`.

```
> eruption.lm = lm(
+     eruptions ~ waiting,
+     data=faithful)
> eruption.stdres =
+     rstandard(eruption.lm)
```

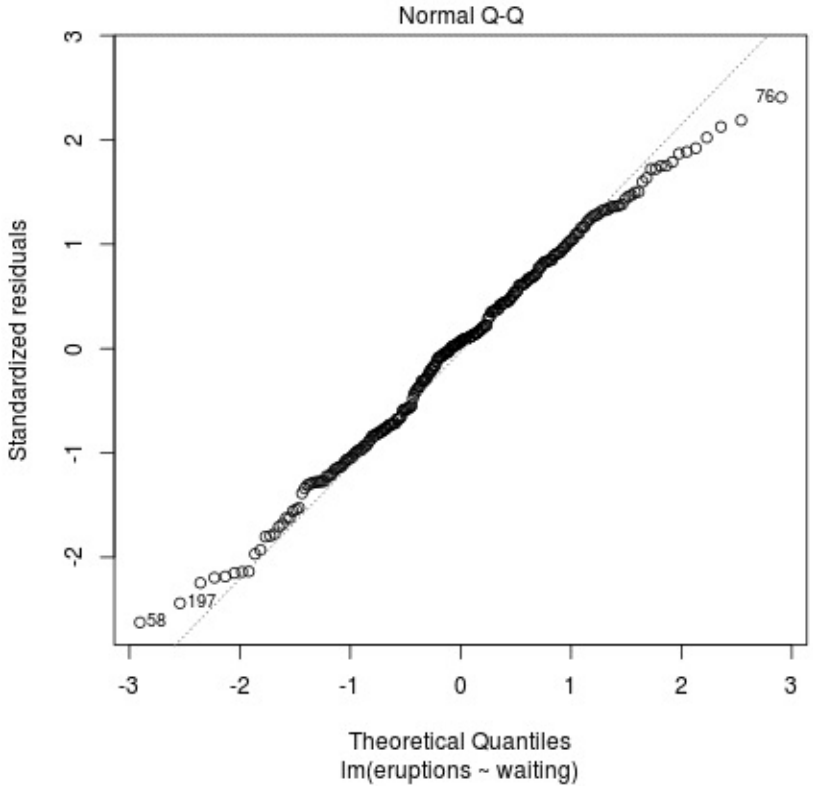
We now create the normal probability plot with the function `qqnorm`, and add a `qqline` as baseline.

```
> qqnorm(eruption.stdres,
+     ylab="Standardized Residuals",
+     xlab="Normal Scores",
+     main="Old Faithful Eruptions")
> qqline(eruption.stdres)
```



Alternatively, we can achieve the same by selecting the second plot of the linear regression model:

```
> plot(eruption.lm, which=2)
```



Answer

Since the normal probability plot is fairly close to the baseline, the error term of the simple linear regression model seems to resemble the normal distribution.

Note

Further detail of the functions `qqnorm` and `qqline` can be found in the R documentation.

```
> help(qqnorm)
```

Chapter 18
Multiple Linear Regression

A **multiple linear regression** (MLR) model describes a dependent variable y by a set of independent variables $x_1, x_2, \ldots, x_p$ ($p > 1$) in a linear relationship. In the MLR below, the numbers α and β_k ($k = 1, 2, \ldots, p$) are called **parameters**, and ε is called the **error term**.

$$y = \alpha + \sum_k \beta_k x_k + \epsilon$$

For example, in the built-in data set `stackloss` made from observations of a chemical plant operation, if we assign the column `stack.loss` as the dependent variable, and assign `Air.Flow` (cooling air flow), `Water.Temp` (inlet water temperature) and `Acid.Conc.` (acid concentration) as independent variables, the multiple linear regression model is:

$$Stack.Loss = \alpha + \beta_1 * Air.Flow + \beta_2 * Water.Temp + \beta_3 * Acid.Conc. + \epsilon$$

Further detail of the data set `stackloss` can be found in the R documentation.

```
> help(stackloss)
```

18.1 Estimated Multiple Regression Equation

If we choose the parameters α and β_k (k = 1, 2, …, p) in the multiple linear regression model so as to minimize the sum of squares of the error term ε, we will have the so called **estimated multiple regression equation**. It allows us to compute the **fitted values** of y based on a set of independent variables x_k (k = 1, 2, …, p).

$$\hat{y} = a + \sum_k b_k x_k$$

Problem

Assuming the multiple linear regression model for the data set `stackloss`, predict the stack loss if the air flow is 72, water temperature is 20, and acid concentration is 85.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.,
+ data=stackloss)
```

Then we create a data frame `newdata` containing the parameters.

```
> newdata = data.frame(Air.Flow=72,
+     Water.Temp=20,
+     Acid.Conc.=85)
```

Lastly, we apply the function `predict` to `stackloss.lm` and `newdata`.

```
> predict(stackloss.lm, newdata)
     1
24.582
```

Answer

Based on the multiple linear regression model and the given parameters, the predicted stack loss is 24.582.

18.2 Multiple Coefficient of Determination

The **coefficient of determination** of a multiple linear regression model is the quotient between variance of fitted values and variance of observed values of the dependent variable. It is a measure of the proportion of variance in *y* that is accountable by the fitted values.

$$R^2 = \frac{\sum(\hat{y}_i - \bar{y})^2}{\sum(y_i - \bar{y})^2}$$

$$\begin{aligned} where\ y_i &= observed\ values, \\ \hat{y}_i &= fitted\ values, \\ and\ \bar{y} &= mean\ value \end{aligned}$$

Problem

Find the coefficient of determination for the multiple linear regression model of the data set `stackloss`.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.,
+ data=stackloss)
```

Then we extract the coefficient of determination from the attribute `r.squared` of its summary.

```
> summary(stackloss.lm)$r.squared
[1] 0.91358
```

Answer

The coefficient of determination of the multiple linear regression model for the data set `stackloss.lm` is 0.91358.

Note

Further detail of the attribute `r.squared` can be found in the R documentation.

```
> help(summary.lm)
```

18.3 Adjusted Coefficient of Determination

The **adjusted coefficient of determination** of a multiple linear regression model is defined in terms of the coefficient of determination as follows, where *n* is the number of observations in the data set, and *p* is the number of independent variables. The adjustment is intended to counter influence by *p*.

$$R^2_{adj} = 1 - (1 - R^2)\frac{n-1}{n-p-1}$$

Problem

Find the adjusted coefficient of determination for the multiple linear regression model of the data set `stackloss`.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.,
+ data=stackloss)
```

Then we extract the adjusted coefficient of determination from the attribute `adj.r.squared` of its summary.

```
> summary(stackloss.lm)$adj.r.squared
[1]  0.89833
```

Answer

The adjusted coefficient of determination of the multiple linear regression model for the data set `stackloss.lm` is 0.89833.

Note

Further detail of the attribute `adj.r.squared` can be found in the R documentation.

```
> help(summary.lm)
```

18.4 Significance Test for MLR

Assume that the error term ε in the multiple linear regression (MLR) model is independent of x_k (k = 1, 2, …, p), is normally distributed, has zero mean, and has constant variance. We can decide whether there is any significant relationship between the dependent variable y and the independent variables x_k (k = 1, 2, …, p).

Problem

Decide which of the independent variables in the multiple linear regression model of the data set `stackloss` are statistically significant at .05 significance level.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.,
+ data=stackloss)
```

Then the p-values of the independent variables can be found in the summary.

```
> summary(stackloss.lm)

Call:
lm(formula = stack.loss ~ Air.Flow + Water.Temp
         + Acid.Conc., data = stackloss)

Residuals:
   Min     1Q Median     3Q    Max
-7.238 -1.712 -0.455  2.361  5.698

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  -39.920     11.896   -3.36   0.0038 **
Air.Flow       0.716      0.135    5.31  5.8e-05 ***
```

```
Water.Temp     1.295      0.368    3.52   0.0026 **
Acid.Conc.    -0.152      0.156   -0.97   0.3440
---
Residual standard error: 3.24 on 17 degrees of freedom
Multiple R-squared: 0.914,
Adjusted R-squared: 0.898
F-statistic: 59.9 on 3 and 17 DF,  p-value: 3.02e-09
```

Answer

As the p-values of `Air.Flow` and `Water.Temp` are less than 0.05, they are both statistically significant in the multiple linear regression model of `stackloss.lm`. However, as the p-value of `Acid.Conc` (0.3440) is greater than 0.05, it is not statistically significant.

Note

Further detail of function `summary` can be found in the R documentation.

```
> help(summary.lm)
```

18.5 Confidence Interval of MLR

Assume that the error term ε in the multiple linear regression (MLR) model is independent of x_k (k = 1, 2, …, p), is normally distributed, has zero mean, and has constant variance. For given values of x_k (k = 1, 2, …, p), an interval estimate of the mean of the dependent variable y is called a **confidence interval**.

Problem

For the data set `stackloss`, construct a 95% confidence interval of the stack loss if the air flow is 72, water temperature is 20 and acid concentration is 85.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> attach(stackloss)
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.)
```

Then we create a data frame `newdata` containing the parameters.

```
> newdata = data.frame(Air.Flow=72,
+     Water.Temp=20,
+     Acid.Conc.=85)
```

We now apply the function `predict` and set the predictor values in the argument `newdata`. We also set the interval type as `"confidence"`, and use the default 0.95 confidence level.

```
> predict(stackloss.lm, newdata,
+     interval="confidence")
     fit    lwr    upr
1 24.582 20.218 28.945
> detach(stackloss)    # clean up
```

Answer

The 95% confidence interval of the stack loss with the given parameters is between 20.218 and 28.945.

Note

Further detail of function `predict` can be found in the R documentation.

```
> help(predict.lm)
```

18.6 Prediction Interval of MLR

Assume that the error term ε in the multiple linear regression (MLR) model is independent of x_k (k = 1, 2, …, p), is normally distributed, has zero mean, and has constant variance. For given values of x_k (k = 1, 2, …, p), an interval estimate of the dependent variable y is called a **prediction interval**.

Problem

For the data set `stackloss`, construct a 95% prediction interval of the stack loss if the air flow is 72, water temperature is 20 and acid concentration is 85.

Solution

We apply the function `lm` to a formula that describes the variable `stack.loss` by `Air.Flow`, `Water.Temp` and `Acid.Conc.` And we save the linear regression model in `stackloss.lm`.

```
> attach(stackloss)
> stackloss.lm = lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.)
```

Then we create a data frame `newdata` containing the parameters.

```
> newdata = data.frame(
+     Air.Flow=72,
+     Water.Temp=20,
+     Acid.Conc.=85)
```

We now apply the function `predict` and set the predictor values in the argument `newdata`. We also set the interval type as `"predict"`, and use the default 0.95 confidence level.

```
> predict(stackloss.lm, newdata,
+     interval="predict")
     fit    lwr    upr
1 24.582 16.466 32.697
> detach(stackloss)    # clean up
```

Answer

The 95% prediction interval of the stack loss with the given parameters is between 16.466 and 32.697.

Note

Further detail of function `predict` can be found in the R documentation.

```
> help(predict.lm)
```

Chapter 19
Logistic Regression

We use the **logistic regression equation** to predict the probability of a dependent variable taking the dichotomy values 0 or 1. Suppose x_1, x_2, …, x_p are the independent variables, α and β_k (k = 1, 2, …, p) are the **parameters**, and $E(y)$ is the expected value of the dependent variable y, then the logistic regression equation is:

$$E(y) = 1/(1 + e^{-(\alpha + \sum_k \beta_k x_k)})$$

For example, in the built-in data set `mtcars`, the data column `am` represents the transmission type of the automobile model (0 = automatic, 1 = manual). With the logistic regression equation, we can model the probability of a manual transmission in a vehicle based on its engine horsepower and weight data.

$$P(Manual\ Transmission) = 1/(1 + e^{-(\alpha + \beta_1 * Horsepower + \beta_2 * Weight)})$$

19.1 Estimated Logistic Regression Equation

Using the generalized linear model, an **estimated logistic regression equation** can be formulated as below. The coefficients a and b_k (k = 1, 2, …, p) are determined according to a maximum likelihood approach, and it allows us to estimate the probability of the dependent variable y taking on the value 1 for given values of x_k (k = 1, 2, …, p).

$$Estimate\ of\ P(y = 1 \mid x_1, ...x_p) = 1/(1 + e^{-(a+\sum_k b_k x_k)})$$

Problem

By use of the logistic regression equation of vehicle transmission in the data set `mtcars`, estimate the probability of a vehicle being fitted with a manual transmission if it has a 120hp engine and weights 2800 lbs.

Solution

We apply the function `glm` to a formula that describes the transmission type `am` by the horsepower `hp` and weight `wt`. This creates a generalized linear model (GLM) in the binomial family.

```
> am.glm = glm(
+     formula=am ~ hp + wt,
+     data=mtcars,
+     family=binomial)
```

We then wrap the test parameters inside a data frame `newdata`.

```
> newdata =
+     data.frame(hp=120, wt=2.8)
```

Now we apply the function `predict` to the generalized linear model `am.glm` along with `newdata`. We will have to select *response* prediction type in order to obtain the predicted probability.

```
> predict(am.glm, newdata,
```

```
+     type="response")
      1
0.64181
```

Answer

For an automobile with 120hp engine and 2800 lbs weight, the probability of it being fitted with a manual transmission is about 64%.

Note

Further detail of the function `predict` for generalized linear model can be found in the R documentation.

```
> help(predict.glm)
```

19.2 Significance Test for Logistic Regression

We can decide whether there is any significant relationship between the dependent variable y and the independent variables x_k (k = 1, 2, …, p) in the logistic regression equation. In particular, if any of the null hypothesis that $\beta_k = 0$ (k = 1, 2, …, p) is valid, then x_k is statistically insignificant in the logistic regression model.

Problem

At .05 significance level, decide if any of the independent variables in the logistic regression model of vehicle transmission in the data set `mtcars` is statistically significant.

Solution

We apply the function `glm` to a formula that describes the transmission type `am` by the horsepower `hp` and weight `wt`. This creates a generalized linear model (GLM) in the binomial family.

```
> am.glm = glm(
+     formula=am ~ hp + wt,
+     data=mtcars,
+     family=binomial)
```

We then print out the summary of the generalized linear model and check for the p-values of the `hp` and `wt` variables.

```
> summary(am.glm)

Call:
glm(formula = am ~ hp + wt, family = binomial, data = mtcars)

Deviance Residuals:
    Min       1Q   Median       3Q      Max
-2.2537  -0.1568  -0.0168   0.1543   1.3449

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
```

```
(Intercept)  18.8663     7.4436    2.53   0.0113 *
hp            0.0363     0.0177    2.04   0.0409 *
wt           -8.0835     3.0687   -2.63   0.0084 **
```

Answer

As the p-values of the `hp` and `wt` variables are less than 0.05, both `hp` and `wt` are significant at 0.05 level in the logistic regression model.

Note

Further detail of the function `summary` for the generalized linear model can be found in the R documentation.

```
> help(summary.glm)
```

Part III
Bayesian Statistics Using OpenBUGS

In previous chapters, we have treated population parameters as fixed values, and provided point estimates and confidence intervals for them. We saw probability as frequency measure of an interval containing a given population parameter under repeated sampling. We call this approach the *frequentist statistics*. If there is a collection of possible candidate values to select from, a frequentist statistician would choose the one that maximizes a pre-defined likelihood function.

An alternative approach is the *Bayesian statistics*. It treats population parameters as random variables. Probability becomes a measure of our belief in the present state of knowledge. Bayesian statistics dictates keeping track of all possible parameter values, and thus requires much heavier computation machineries. Nevertheless, as we will prove in subsequent chapters, the Bayesian approach is much more flexible. With new tools like OpenBUGS, a Bayesian practitioner only need to build a model, and the system will take care of the rest. Tackling new problems means building new prototype models, instead of relying on yet another R command.

In the next few chapters, we will discuss introductory Bayesian statistics, and briefly discuss simple cases with available closed analytic solutions. Then we move on to discuss Bayesian ANOVA and regression models using OpenBUGS. At conclusion, we demonstrate how to build more advanced models, and use CODA to study Markov Chain Monte Carlo (MCMC) convergence.

Chapter 20
Bayesian Finite Inference

According to the Bayesian theory, probability is a measure of our belief in the likelihood of an event. For instance, if we flip a coin into the air, we would expect equal chance of catching head or tail. Therefore, for a fair coin, the probability of either outcome is 0.5. Mathematically, we write

$$P(head \mid fair\ coin) = 0.5$$

In general, for an event X, we denote its probability of occurrence by $P(X)$. For two events X and Y, we denote the probability of their joint occurrence by $P(X, Y)$.

We define the **conditional probability** of an event X given occurrence of another event Y as

$$P(X|Y) = \frac{P(X,Y)}{P(Y)}$$

By symmetry of the definition, we have

$$P(X|Y)P(Y) = P(X,Y) = P(Y|X)P(X)$$

From this, we obtain the **Bayes' Theorem**

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)} \propto P(Y|X)P(X)$$

In Bayesian terminology, we say that $P(X)$ is the **prior probability**, $P(Y|X)$ is the **likelihood**, and $P(X|Y)$ is the **posterior probability**. Hence *the posterior is proportional to the likelihood times the prior*, and we write

$$Posterior \propto Likelihood \times Prior$$

20.1 A Classical Example

A classical example of the Bayes' Theorem is the interpretation of a medical test result, which we can characterize by its **sensitivity** (proportion of true positives) and **specificity** (proportion of true negatives). For example, if a medical screening test for a disease is 99% sensitive and 97% specific, then it will correctly report the disease 99% of the time, and correctly rule out the disease 97% of the time. With additional information about the proportion of general population being infected, we can find out the probability of a person actually having the disease upon a positive test result.

Problem

Suppose a patient has been tested positive by a medical screening procedure that is 99% sensitive and 97% specific, and the infection rate of the general population is 1%, what is the probability of the patient actually being infected?

Solution

Denote the event of a disease infection by *X*, and a positive test result by *Y*. We further denote absence of infection by ~*X* and a negative test result by ~*Y*.

Since the test is 99% sensitive and 97% specific, we have $P(Y|X)=0.99$, and $P(\sim Y|\sim X)=0.97$. As only 1% of the population is infected, the prior probability is $P(X)=0.01$. Our task is to find $P(X|Y)$.

First of all, we find the probability of a positive test result $P(Y)$. It maybe a *true positive*, whose probability is $P(Y|X)P(X)$, or it maybe a *false positive*, whose probability is $P(Y|\sim X)P(\sim X)$. Hence,

$$\begin{aligned} P(Y) &= P(Y|X)P(X) + P(Y|\sim X)P(\sim X) \\ &= P(Y|X)P(X) + (1 - P(\sim Y|\sim X)) \times (1 - P(X)) \\ &= 0.99 \times 0.01 + (1 - 0.97) \times (1 - 0.01) \\ &= 0.0396 \end{aligned}$$

Then, by Bayes' Theorem, we find the posterior probability $P(X|Y)$.

$$\begin{aligned} P(X|Y) &= P(Y|X) \times P(X)/P(Y) \\ &= (0.99 \times 0.01)/0.0396 \\ &= 0.25 \end{aligned}$$

Exercise

If the patient takes the same test again and the result is still positive, what is the probability of the patient being infected?

Solution

After the first positive test result, the prior probability $P(X)$ becomes 25%. Repeat the same computation as before and we find $P(X|Y)$=91.67%.

20.2 Finite Distribution with Uniform Prior

A key ingredient of Bayesian inference is the prior probability, which encapsulates our prior belief in the likelihood of events. In absence of specific information, we usually select a so-called *non-informative* prior.

Example

Consider a mixed bag of apples and oranges. Suppose there are six fruits in the bag and they are concealed from view. If we know nothing about the actual number of apples in the bag, then we would assume the chances of having zero, one, two, …, or six apples in the bag are all the same. Therefore we select the discrete uniform distribution as our non-informative prior.

Problem

Suppose the first fruit we draw from the bag is an apple. Using uniform prior, find posterior probability estimate of the original number of apples in the bag. If we draw the next fruit from the bag without replacing the first fruit, what is the probability that it is also an apple?

Solution

We denote the number of fruits in the bag by `N`, and the number of apples by `x`. The possible values of `x` are 0, 1, 2, …, `N`. Therefore we have

```
> N <- 6    # number of fruits
> x <- 0:N  # number of apples
> x
[1] 0 1 2 3 4 5 6
```

With uniform prior, all values in `x` are equally likely. We denote this probability distribution by a vector called `prior`. Since there are seven members in `x`, each member has the same probability of 1/7.

```
> len <- length(x)
> prior <- rep(1/len, len)
```

To improve readability, we annotate the vector `prior` by members of `x`. When we print out `prior` with the method `cbind`, the first column will have corresponding values of `x` for cross reference.

```
> names(prior) <- x
> cbind(prior)

  prior
0 0.1429
1 0.1429
2 0.1429
3 0.1429
4 0.1429
5 0.1429
6 0.1429
```

If there are `x` apples in the bag, then the likelihood of drawing an apple from the bag is `x/N`, which we denote by another vector called `y`.

```
> y <- x/N    # likelihood
> cbind(prior, likelihood=y)

   prior likelihood
0 0.1429     0.0000
1 0.1429     0.1667
2 0.1429     0.3333
3 0.1429     0.5000
4 0.1429     0.6667
5 0.1429     0.8333
6 0.1429     1.0000
```

Now we multiply the `prior` and the likelihood `y` together, and save it in a vector called `LP`. Then we normalize `LP` by dividing with its sum, and have the posterior probability `PP`.

```
> LP <- prior * y  # prior*likelihood
> PP <- LP/sum(LP) # posterior
> cbind(prior, likelihood=y, LP, PP)

   prior likelihood      LP      PP
0 0.1429     0.0000 0.00000 0.00000
1 0.1429     0.1667 0.02381 0.04762
2 0.1429     0.3333 0.04762 0.09524
3 0.1429     0.5000 0.07143 0.14286
4 0.1429     0.6667 0.09524 0.19048
5 0.1429     0.8333 0.11905 0.23810
6 0.1429     1.0000 0.14286 0.28571
```

If we compare the `prior` and `PP` columns in the table above, we see that the probability estimates of less than 3 apples are getting smaller in `PP`, while the probability estimates of 4 or more apples are getting larger in `PP`. This is because of the first apple we draw from the bag, which suggests that there maybe more apples than oranges in the original bag.

After we draw an apple from the bag, there are only `x-1` apples left in the bag of `N-1` fruits. The following shows the probability of drawing an apple next from the bag is about 67%.

```
> sum(PP*(x-1)/(N-1))
[1] 0.6667
```

Exercise

Consider a bag of six apples and oranges as above. If the first fruit we draw from the bag is an orange, what is the probability of drawing an apple next?

Solution

Using uniform prior, the probability of drawing an orange from the original bag is `1-x/N`, where `x` is the number of apples in the bag. Computation as before shows that the probability of drawing an apple next is 33%.

Alternatively, we see that drawing an orange is just the complement event of drawing an apple. Hence the probability is 1-67%=33%.

20.3 Finite Distribution with Non-Uniform Prior

In the previous tutorial, we estimated the probability distribution of apples in a bag by drawing a fruit. We now use the previously found posterior probability estimate as the new prior probability for further investigation.

Problem

After drawing an apple from a mixed bag of six apples and oranges, we find the next fruit we draw from the bag is still an apple. What is the new probability estimate of the original number of apples in the bag? What is the chance that the third fruit we draw from the bag is also an apple?

Solution

Recall the posterior probability `PP` from the previous tutorial. We denote it as the new prior probability `prior1`.

```
> prior1 <- PP
> cbind(prior1)
   prior1
0 0.00000
1 0.04762
2 0.09524
3 0.14286
4 0.19048
5 0.23810
6 0.28571
```

Since there are `x-1` apples in the bag after drawing the first apple, the likelihood of drawing the next apple from the bag is `(x-1)/(N-1)`, which we denote by `y1`. Its first negative entry below can be safely ignored as the corresponding prior probability is zero.

```
> y1 <- (x-1)/(N-1)
> cbind(prior1, y1)

   prior1   y1
0 0.00000 -0.2
```

```
1 0.04762  0.0
2 0.09524  0.2
3 0.14286  0.4
4 0.19048  0.6
5 0.23810  0.8
6 0.28571  1.0
```

Now we multiply the `prior1` and the likelihood `y1` together, and save it in a variable `LP1`. Then we normalize `LP1` and find the new posterior probability `PP1`.

```
> LP1 <- prior1 * y1
> PP1 <- LP1/sum(LP1)

> cbind(prior1, y1, LP1, PP1)
   prior1   y1     LP1     PP1
0 0.00000 -0.2 0.00000 0.00000
1 0.04762  0.0 0.00000 0.00000
2 0.09524  0.2 0.01905 0.02857
3 0.14286  0.4 0.05714 0.08571
4 0.19048  0.6 0.11429 0.17143
5 0.23810  0.8 0.19048 0.28571
6 0.28571  1.0 0.28571 0.42857
```

After drawing the second apple from the bag, there are only `x-2` apples left in the bag of `N-2` fruits. Hence the probability of drawing the third apple from the bag is

```
> sum(PP1*(x-2)/(N-2))
[1] 0.75
```

Alternative Solution

We can start from scratch with the uniform prior before drawing any fruits from the bag. Assuming there are `x` apples in the bag, the likelihood of drawing two apples from the bag at the start is

```
> y2 <- x*(x-1)/(N*(N-1))
> cbind(prior, y2)

   prior      y2
0 0.1429 0.00000
1 0.1429 0.00000
2 0.1429 0.06667
3 0.1429 0.20000
4 0.1429 0.40000
5 0.1429 0.66667
6 0.1429 1.00000
```

Now we multiply the `prior` and the likelihood `y2` together, and save it in a variable `LP2`. Then we normalize `LP2` and find the posterior probability `PP2`. Notice that it is identical to the `PP1` previously found.

```
> LP2 <- prior * y2
> PP2 <- LP2/sum(LP2)
>
> cbind(prior,
+     likelihood=y2, LP2, PP2)

   prior likelihood      LP2     PP2
0 0.1429    0.00000 0.000000 0.00000
1 0.1429    0.00000 0.000000 0.00000
2 0.1429    0.06667 0.009524 0.02857
3 0.1429    0.20000 0.028571 0.08571
4 0.1429    0.40000 0.057143 0.17143
5 0.1429    0.66667 0.095238 0.28571
6 0.1429    1.00000 0.142857 0.42857
```

After drawing two apples from the bag, there are only `x-2` remaining apples in the bag of `N-2` fruits. Hence the probability of drawing next apple from the bag is:

```
> sum(PP2*(x-2)/(N-2))
[1] 0.75
```

Exercise

Consider the same bag of six apples and oranges. If the first fruit we draw from the bag is an orange, and the second fruit we draw is an apple, what is the probability that the third fruit we draw is also an apple?

Solution

If the first fruit we draw from the bag is an orange, then the probability of subsequently drawing an apple from the bag is `x/(N-1)`. Using the posterior probability of the previous exercise as the new prior, it shows that the probability of the third fruit being an apple is 50%.

Alternatively, seeing that the apple and orange effects cancel out each other, the third fruit has equal chance of being an apple or orange.

Chapter 21
Bayesian Binomial Inference

Population proportion is arguably the most visible form of statistics. We see them daily in marketing survey, political polls, and science news. A standard mathematical description of population proportion is the binomial model, which gives the number of successes in a finite sequence of identical independent Bernoulli trials. Denote the probability of success of each trial by *p*, then the binomial probability of having *y* successes is:

$$P(y \mid p) = \left(\begin{array}{c} N \\ y \end{array} \right) p^y (1-p)^{N-y} \quad where\ y = 0, 1, 2, ..., N$$

By Bayes' Theorem, if *P*(*p*) is the prior of *p*, then we can find its posterior probability as follows.

$$\begin{aligned} P(p \mid y) &\propto P(y \mid p) \times P(p) \\ &\propto p^y (1-p)^{N-y} P(p) \end{aligned}$$

21.1 Binomial Inference with Conjugate Prior

In order to perform Bayesian inference for a population proportion, we have to decide on a prior of the binomial probability. A popular choice is the beta distribution *Beta*(*a*, *b*), whose special case of *Beta*(1,1) is the continuous uniform distribution between 0 and 1. We will see why the choice is popular shortly.

Using *Beta*(*a*, *b*) as the prior of a binomial probability *p*, we have

$$\begin{aligned} P(p) &= beta(p; a, b) \\ &\propto p^{a-1}(1-p)^{b-1} \end{aligned}$$

Hence, with *y* successes, the posterior is

$$\begin{aligned} P(p \mid y) &\propto p^{y}(1-p)^{N-y}P(p) \\ &\propto p^{y}(1-p)^{N-y}p^{a-1}(1-p)^{b-1} \\ &= p^{a+y-1}(1-p)^{b+N-y-1} \end{aligned}$$

Normalizing the expression gives

$$P(p \mid y) = beta(p;\ a+y,\ b+N-y)$$

Hence the posterior is just another beta distribution *Beta*(*a*+*y*, *b*+*N*-*y*), and we call the beta distribution to be a *conjugate distribution* of the binomial likelihood.

Based on the formula above, to find the first conjugate beta parameter, we would increase the first parameter by the number of successes *y*. As for the second conjugate beta parameter, we would increase the second parameter by the number of failures *N*-*y*.

Problem

The data set `survey` contains sample smoker statistics among university students. Denote the number of student smokers in the survey by *y*, the total number of students in the survey by *N*, and the proportion of smokers in the general student

population by *p*. We regard *y* as the outcome of a binomial experiment of size *N* and success probability *p*. Using uniform prior, find the mean and standard deviation of the posterior of *p*.

Solution

We first summarize the student smoker statistics as follows. It turns out there are 47 smokers out of 236 students in the survey.

```
> library(MASS)
> tbl <- table(survey$Smoke); tbl

Heavy Never Occas Regul
   11   189    19    17

> N <- as.numeric(sum(tbl)); N
[1] 236

> y <- N - as.numeric(tbl["Never"])
> y
[1] 47
```

Using the uniform prior *beta*(1,1) for *p*, the conjugate parameters of the posterior are:

```
> a <- 1+y; a
[1] 48

> b <- 1+(N-y); b
[1] 190
```

We now apply the formula for beta distribution and find the posterior mean and standard deviation of *p* to be 0.2017 and 0.02596 respectively.

```
> c <- a+b
> mu <- a/c; mu
[1] 0.2017

> sigma <- sqrt(a*b/(c*c*(c+1)))
> sigma
[1] 0.02596
```

Exercise

Assume the prior of student smoker proportion p in the data set `survey` to be *Beta*(85,120). Find the posterior mean and standard deviation of p.

Solution

With the the prior being *Beta*(85,120), we can compute the conjugate posterior parameters as before, and find the posterior mean and standard deviation of p to be 0.2945 and 0.02176 respectively.

21.2 Binomial Inference with Prior Parameters

For a beta distribution *Beta*(*a*,*b*), if we we denote the sum of the beta parameters by *c*=*a*+*b*, then the mean and variance are

$$\begin{aligned} \mu &= a/c \\ \sigma^2 &= ab/(c^2(c+1)) \end{aligned}$$

Solving the equations, we find that

$$\begin{aligned} c &= \mu(1-\mu)/\sigma^2 - 1 \\ a &= \mu c \\ b &= (1-\mu)c \end{aligned}$$

Therefore, with knowledge of the mean and variance, we can determine the parameters of a beta distribution.

Problem

The data set `survey` contains sample smoker statistics among university students. Denote the proportion of smokers in the general student population by *p*. Suppose that past historical surveys indicates the prior of *p* to be a beta distribution with mean 0.35 and standard deviation 0.025. Find the mean and and standard deviation of the posterior of *p*.

Solution

We find that there are 236 students in the survey, and 47 of them smoke.

```
> library(MASS)
> tbl <- table(survey$Smoke); tbl

Heavy Never Occas Regul
   11   189    19    17

> N <- as.numeric(sum(tbl)); N
[1] 236
> y <- N - as.numeric(tbl["Never"])
```

```
> y
[1] 47
```

With given values of the mean `mu` and standard deviation `sigma`, we can find the prior beta parameters `a` and `b`.

```
> mu <- 0.35
> sigma <- 0.025

> c <- mu*(1-mu)/(sigma*sigma) - 1
> a <- mu*c; a
[1] 127
> b <- (1-mu)*c; b
[1] 235.9
```

We now compute the posterior conjugate parameters `a1` and `b1`.

```
> a1 <- a+y; a1
[1] 174

> b1 <- b+(N-y); b1
[1] 424.9
```

And we find the posterior mean and standard deviation of `p` to be 0.2906 and 0.01854 respectively.

```
> c1 <- a1+b1
> mu1 <- a1/c1; mu1
[1] 0.2906

> sigma1 <-
+     sqrt(a1*b1/(c1*c1*(c1+1)))
> sigma1
[1] 0.01854
```

Exercise

Assume the prior of student smoker population proportion *p* to be a beta distribution with mean 0.15 and standard deviation 0.03. Estimate the posterior mean and and standard deviation of *p* based on the data set `survey`.

Solution

We can find the prior parameters *a* and *b* based on the given values of mean and

standard deviation. Similar calculation as before shows that the posterior mean and standard deviation of p are 0.1808 and 0.0198 respectively.

21.3 Binomial Inference Using OpenBUGS

We will use the same student survey example in this chapter as our first demonstration of OpenBUGS. Although the example is elementary, it does contain all the essential steps.

The central concept of OpenBUGS is the BUGS model. As we will prove, it is not always necessary to create a BUGS model from scratch. Instead, we can build our models incrementally from simple ones. Hence our first task is to create our own library of basic BUGS models that we can reuse later.

The BUGS language bears a strong resemblance to R. We will introduce more BUGS syntax as we move along. If lack of patience, there is full detail in the WinBUGS online manual.

We begin with introducing the "~" operator, which describes the probability distribution of a random variable. For example, the following indicates that a random variable `y` follows a binomial distribution with probability of success `p` and size `N`.

```
y ~ dbin(p, N)
```

Note there are instructions to install OpenBUGS and the R2OpenBUGS extension in the Appendix section.

Problem

The data set `survey` contains sample smoker statistics among university students. Denote the proportion of smokers in the general student population by *p*. With uniform prior, find the mean and and standard deviation of the posterior of *p* using OpenBUGS.

Solution

The first step is to create a BUGS model. As we portrait the number of student smokers `y` as the outcome of a binomial experiment of size `N` and success

probability `p`, we write the following in BUGS.

```
    y ~ dbin(p, N)
```

We assume the prior distribution of `p` to be uniform between 0 and 1, which is equivalent to *Beta*(1,1).

```
    p ~ dbeta(1, 1)
```

In summary, we have the following BUGS model. The sole purpose of the `model` function below is to serve as a packaging wrapper for OpenBUGS. The function itself is meaningless to R.

```
model <- function() {
    # Prior
    p ~ dbeta(1, 1)

    # Likelihood
    y ~ dbin(p, N)
}
```

To transfer the model to OpenBUGS, we load the `R2OpenBUGS` extension and write the model to a temporary location using the method `write.model`. We denote the model file location by `model.file`.

```
> library(R2OpenBUGS)
> model.file <- file.path(tempdir(),
+     "model.txt")
> write.model(model, model.file)
```

Then we have to decide data parameters of the BUGS model. We find that there are 236 students in the survey, and 47 of them smoke, which we denote by `N` and `y` respectively.

```
> library(MASS)
> tbl <- table(survey$Smoke)
> N <- as.numeric(sum(tbl)); N
[1] 236

> y <- N - as.numeric(tbl["Never"])
> y
[1] 47
```

We then identify the data variables in a list called `data`.

```
> data <- list("N", "y")
```

And we identify the variable to be monitored in a vector called `params`.

```
> params <- c("p")
```

Lastly, we need to select some initial parameters for the simulation. A rule of thumb is to choose values as close to the expected result as possible. In this case, we initialize `p` to be 0.5. Notice how we wrap the initial values inside a list that is to be returned by a function.

```
> inits <- function() {
+     list(p=0.5) }
```

Then we invoke OpenBUGS with the namesake method `bugs` and save the result in a variable `out`. We select 10,000 iterations per simulation chain.

```
> out <- bugs(data, inits, params,
+    model.file, n.iter=10000)
```

Upon completion of the `bugs` method, we should check the `Rhat` component of the output. If we find any value higher than the 1.1 threshold, we should increase the `n.iter` argument in the `bugs` method and try again. However, even if all values in `Rhat` are less than 1.1, it still does not guarantee convergence. We shall demonstrate later how to perform further analysis using CODA.

```
> all(out$summary[,"Rhat"] < 1.1)
[1] TRUE
```

Finally, we can retrieve the posterior mean and standard deviation of `p` from the output.

```
> out$mean["p"]
$p
[1] 0.2015

> out$sd["p"]
$p
[1] 0.02575
```

It is informative to print out the simulation result in full detail at this point.

```
> print(out, digits=5)
```

Source Listing

```
library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)

library(MASS)
tbl <- table(survey$Smoke)
N <- as.numeric(sum(tbl)); N
y <- N - as.numeric(tbl["Never"]); y

data <- list("N", "y")
params <- c("p")
inits <- function() { list(p=0.5) }
out <- bugs(data, inits, params,
    model.file, n.iter=10000)

all(out$summary[,"Rhat"] < 1.1)

out$mean["p"]
out$sd["p"]
```

Exercise

Assume the prior distribution of student smoker proportion `p` to be *Beta*(85,120). Estimate the posterior mean and standard deviation using OpenBUGS.

Solution

We modify the prior parameters in our BUGS model for *Beta*(85,120) as follows.

```
model <- function() {
    # Prior
    p ~ dbeta(85, 120)

    # Likelihood
    y ~ dbin(p, N)
}
```

And find the results of the simulation.

```
> out$mean["p"]
$p
[1] 0.2992
```

```
> out$sd["p"]
$p
[1] 0.02178
```

Chapter 22
Bayesian Poisson Inference

Although the binomial model is effective for inference of population proportions, it does require information of the population size *N*. In a situation when *N* is large but unknown, and the probability of success *p* is small, we can replace it with a Poisson model, which does not require the knowledge of *N*. More specifically, we can replace the binomial distribution by a Poisson distribution with mean parameter λ equal to $N \times p$.

Recall the Poisson probability of having *y* event occurrences.

$$P(y \mid \lambda) = \frac{\lambda^y e^{-\lambda}}{y!} \quad where\ y = 0, 1, 2, 3, \ldots$$

Applying Bayes' Theorem, if $P(\lambda)$ is the prior probability of λ, then its posterior probability is

$$\begin{aligned} P(\lambda \mid y) &\propto P(y \mid \lambda) \times P(\lambda) \\ &\propto \lambda^y e^{-\lambda} P(\lambda) \end{aligned}$$

22.1 Poisson Inference with Conjugate Prior

Analogous to Bayesian binomial inference, a conjugate prior distribution for Poisson inference is the gamma distribution. In particular, we can denote a prior of λ as follows.

$$\begin{aligned} P(\lambda) &= gamma(\lambda; a, b) \\ &\propto \lambda^{a-1} e^{-b\lambda} \end{aligned}$$

Hence, with y event occurrences, we have

$$\begin{aligned} P(\lambda \mid y) &\propto \lambda^{y} e^{-\lambda} P(\lambda) \\ &\propto \lambda^{y} e^{-\lambda} \lambda^{a-1} e^{-b\lambda} \\ &= \lambda^{a+y-1} e^{-(b+1)\lambda} \end{aligned}$$

Normalizing the expression gives

$$P(\lambda \mid y) = gamma(\lambda;\ a + y,\ b + 1)$$

Repeating the calculation for a sequence of n identical independent Poisson variables y_i (i=1,…,n), we have

$$P(\lambda \mid y_1, \ldots, y_n) = gamma(\lambda;\ a + \sum_i y_i,\ b + n)$$

Hence the posterior is also a gamma distribution, and is called a *conjugate distribution* of the Poisson likelihood. Based on the formula above, to find the first conjugate gamma parameter of a sample Poisson sequence, we will increase the first parameter by the sample sum. As for the second conjugate gamma parameter, we will increase the second parameter by the sample size.

Problem

The data set `discoveries` contains annual number of *great* scientific inventions from 1860 to 1959. Assume the data follows a Poisson distribution with mean parameter λ. Using uniform prior, estimate the posterior mean, median, and standard deviation of λ based on the data set `discoveries`.

Solution

For convenience, we save the content of `discoveries` in a variable `y`.

```
> y <- as.numeric(discoveries)
```

We assume the prior of λ to be the uniform distribution, which is equivalent to *Gamma*(1,0). According to the formula above for conjugate Poisson distribution, the first conjugate gamma parameter is `1+sum(y)`, and the second conjugate gamma parameter is `0+length(y)`.

```
> a <- 1+sum(y); a
[1] 311

> b <- 0+length(y); b
[1] 100
```

Now we apply the formula for mean and standard deviation of gamma distribution.

```
> mu <- a/b; mu
[1] 3.11

> sigma <- sqrt(a)/b; sigma
[1] 0.1764
```

To find the median, we use the method `qgamma` to find the 50% quantile.

```
> qgamma(0.5, a, b)
[1] 3.107
```

Exercise

Assume the prior of the mean annual number of great scientific inventions for the data set `discoveries` to be *Gamma*(250, 120). Estimate the posterior mean, median, and standard deviation of the mean annual number of inventions.

Solution

Repeat the calculation for the prior *Gamma*(250, 120) as before, we find the mean, standard deviation and 50% quantile of λ to be 2.545, 0.1076 and 2.544 respectively.

22.2 Poisson Inference with Prior Parameters

As we have mentioned, the mean and variance of a gamma distribution *Gamma*(*a*, *b*) are

$$\begin{aligned} \mu &= a/b \\ \sigma^2 &= a/b^2 \end{aligned}$$

Solving the equations for the gamma parameters *a* and *b*, we find that

$$\begin{aligned} b &= \mu/\sigma^2 \\ a &= \mu b = \mu^2/\sigma^2 \end{aligned}$$

Therefore, with knowledge of the mean and variance, we can determine the gamma distribution.

Problem

Assume the annual number of great scientific inventions in the data set `discoveries` to be Poisson with mean λ. Assume also that the prior of λ is a gamma distribution with mean 2.15 and standard deviation 0.15. Find the posterior mean, median, and standard deviation of λ based on the data set `discoveries`.

Solution

For convenience, we save the content of `discoveries` in a variable `y`.

```
> y <- as.numeric(discoveries)
```

Assuming the prior distribution of λ to be *Gamma*(*a*, *b*), we can find the prior parameters from the given mean and standard deviation values.

```
> mu <- 2.15
> sigma <- 0.15

> b <- mu/(sigma*sigma); b
```

```
[1] 95.56

> a <- mu*b; a
[1] 205.4
```

We can now compute the posterior conjugate parameters `a1` and `b1`. The first conjugate gamma parameter is `a+sum(y)`, and the second conjugate gamma parameter is `b+length(y)`.

```
> a1 <- a+sum(y); a1
[1] 515.4

> b1 <- b+length(y); b1
[1] 195.6
```

Then we find the posterior mean and standard deviation.

```
> mu1 <- a1/b1; mu1
[1] 2.636

> sigma1 <- sqrt(a1)/b1; sigma1
[1] 0.1161
```

To find the posterior median, we use the method `qgamma` and find the 50% quantile.

```
> qgamma(0.5, a1, b1)
[1] 2.634
```

Exercise

Assume the prior of the mean annual number of great scientific inventions to be a gamma distribution with mean 3.82 and standard deviation 0.17. Estimate the posterior mean, median, and standard deviation based on the data set `discoveries`.

Solution

With the prior mean of λ being 3.82 and standard deviation being 0.17, same computation as before shows that the posterior estimates of mean, median, and standard deviation of λ based on the data set `discoveries` are 3.51, 0.123 and 3.508 respectively.

22.3 Poisson Inference Using OpenBUGS

During previous discussion of Bayesian binomial inference, we have seen how to model a single random variable in BUGS. To model a sequence of random variables, we can use the `for` loop. For example, suppose `y` is a vector of independent random variables, and each of its member follows an identical Poisson distribution with mean parameter `lambda`, then we can express `y` in a `for` loop as follows.

```
for (i in 1:m) {
    y[i] ~ dpois(lambda)
}
```

Problem

Assume the prior of the mean annual number of great scientific inventions in the data set `discoveries` to be a uniform distribution. Use OpenBUGS to estimate the posterior mean, median, and standard deviation of the mean annual number of inventions.

Solution

For convenience, we save the content of `discoveries` in a variable `y`. Since the data set is in time series format, we have to cast it into numeric type.

```
> y <- as.numeric(discoveries)
```

Assuming that every element in `y` fits a Poisson distribution with parameter `lambda`, we can write the following.

```
for (i in 1:m) {
    y[i] ~ dpois(lambda)
}
```

Since the `lambda` variable can have arbitrarily positive values, we should use the continuous uniform distribution over the positive real line as its prior. However, as it is not a proper distribution, we can only use a uniform distribution between

zero and a large positive number instead.

```
    lambda ~ dunif(0, 1e10)
```

Therefore, we write our BUGS model as:

```
model <- function() {
    # Prior
    lambda ~ dunif(0, 1e10)

    # Likelihood
    for (i in 1:m) {
        y[i] ~ dpois(lambda)
    }
}
```

In order to load the model into OpenBUGS, we write the model to a temporary file.

```
> library(R2OpenBUGS)
> model.file <- file.path(tempdir(),
+    "model.txt")
> write.model(model, model.file)
```

Then we enter the data values as follows. We select the initial simulation value of `lambda` to be 1.

```
> data <- list(y=y, m=length(y))
> params <- c("lambda")
> inits <- function() {
+     list(lambda=1) }
```

After the simulation, all values of the `Rhat` component are below 1.1, which suggests convergence.

```
> out <- bugs(data, inits, params,
+    model.file, n.iter=10000)
> all(out$summary[,"Rhat"] < 1.1)
[1] TRUE
```

Finally, we can retrieve the posterior mean, median and standard deviation of `lambda` from the output.

```
> out$mean["lambda"]
$lambda
[1] 3.109
```

```
> out$median["lambda"]
$lambda
[1] 3.104

> out$sd["lambda"]
$lambda
[1] 0.1774
```

As always, it is informative to print out the simulation result in full detail.

```
> print(out, digits=5)
```

Source Listing

```
library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)

y <- as.numeric(discoveries)
data <- list(y=y, m=length(y))
params <- c("lambda")
inits <- function() { list(lambda=1) }

out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$mean["lambda"]
out$median["lambda"]
out$sd["lambda"]
```

Exercise

Assume the prior of the mean annual number of great scientific inventions in the data set `discoveries` to be *Gamma*(250,120). Use OpenBUGS to estimate the posterior mean, median, and standard deviation of the mean annual number of inventions.

Solution

We just need to modify the prior parameters in the BUGS model.

```
model <- function() {
```

```
    # Prior
    lambda ~ dgamma(250, 120)

    # Likelihood
    for (i in 1:m){
        y[i] ~ dpois(lambda)
    }
}
```

The result is as follows.

```
> out$mean["lambda"]
$lambda
[1] 2.545

> out$median["lambda"]
$lambda
[1] 2.542

> out$sd["lambda"]
$lambda
[1] 0.1084
```

Chapter 23
Bayesian Inference for Normal Mean

The normal distribution plays a central role in probability theory. For example, the Central Limit Theorem states that sample means of identical independent random variables resembles a normal distribution when the sample size is sufficiently large.

Assume a random variable y fits a normal distribution $N(\mu, \sigma^2)$ with mean μ and variance σ^2. With abuse of notation, we denote its probability density as

$$P(y \mid \mu) = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{1}{2\sigma^2}(y-\mu)^2)$$

By Bayes' Theorem, we can then find the posterior probability density of μ.

$$\begin{aligned} P(\mu \mid y) &\propto P(y \mid \mu) \times P(\mu) \\ &\propto exp(-\frac{1}{2\sigma^2}(y-\mu)^2) \times P(\mu) \end{aligned}$$

23.1 Inference for Normal Mean with Uniform Prior

A simple prior density for the mean of a normal distribution $N(\mu, \sigma^2)$ is the continuous uniform distribution. Ignoring all terms unrelated to μ as proportional constant, the posterior of μ becomes

$$\begin{aligned} P(\mu \mid y) &\propto exp(-\frac{1}{2\sigma^2}(y-\mu)^2) \times 1 \\ &\propto exp(\frac{1}{2\sigma^2}(2\mu y - \mu^2)) \end{aligned}$$

Repeating the calculation for a sequence of identical independent normally distributed variables y_i (i=1,…,n), we have

$$\begin{aligned} P(\mu \mid y_1, \ldots, y_n) &\propto exp(\frac{1}{2\sigma^2}(2\mu \sum_i y_i - n\mu^2)) \\ &\propto exp(-\frac{n}{2\sigma^2}(\mu - \frac{1}{n}\sum_i y_i)^2) \end{aligned}$$

Hence the posterior probability of μ for the uniform prior is another normal distribution, whose mean is the sample mean, and its variance is the population variance divided by the sample size, *i.e.*, σ^2/n.

Problem

The data set `survey` contains the height measurements of a class of university statistics students. We assume that the student heights fit a normal distribution $N(\mu, 10^2)$. Using uniform prior, estimate the posterior mean and standard deviation of μ.

Solution

We first apply the method `na.omit` to filter out missing values in the survey data, and save it in a variable `y`.

```
> library(MASS)
> y <- na.omit(survey$Height)
> n <- length(y)
```

Using uniform prior, the posterior mean of μ is the sample mean. Hence we have

```
> mean(y)
[1] 172.4
```

As for the posterior standard deviation of μ, it is the population standard deviation divided by `sqrt(n)`.

```
> 10/sqrt(n)
[1] 0.6917
```

Exercise

The data set `survey` also contains the pulse rates of the students. Assume the pulse rates fit $N(\mu, 12^2)$. Using uniform prior, estimate the posterior mean and standard deviation of μ.

Solution

With the data in the `Pulse` component of `survey`, using uniform prior, we find that the posterior mean and standard deviation of μ are 74.15 and 0.866 respectively.

23.2 Inference for Normal Mean with Conjugate Prior

If we have any prior knowledge about the mean of a normal distribution $N(\mu, \sigma^2)$, we should incorporate it into a prior probability. For example, if we know the prior mean and variance of μ, we can select another normal distribution with the given information, say $N(m, s^2)$, to be the prior of μ. Hence,

$$\begin{aligned} P(\mu) &= \frac{1}{s\sqrt{2\pi}} exp(-\frac{1}{2s^2}(\mu - m)^2) \\ &\propto exp(-\frac{1}{2s^2}(\mu - m)^2) \end{aligned}$$

We define the **precision** of a normal distribution to be the reciprocal of the variance. In particular, we define the following precision parameters for $N(\mu, \sigma^2)$ and $N(m, s^2)$.

$$\begin{aligned} \tau &= 1/\sigma^2 \\ t &= 1/s^2 \end{aligned}$$

Therefore we have

$$\begin{aligned} P(\mu \mid y) &\propto exp(-\frac{1}{2\sigma^2}(y - \mu)^2) \times P(\mu) \\ &\propto exp(-\frac{\tau}{2}(\mu - y)^2) \times exp(-\frac{t}{2}(\mu - m)^2) \\ &\propto exp(-\frac{1}{2}(t + \tau)\mu^2 + (tm + \tau y)\mu) \end{aligned}$$

Repeating the calculation for a sequence of identical independent normally distributed random variables, we have

$$P(\mu \mid y_1, \ldots, y_n) \propto exp(-\frac{1}{2}(t + n\tau)\mu^2 + (tm + \tau \sum_i y_i)\mu)$$

Now define the following new variables m' and t'.

$$
\begin{aligned}
m' &= \frac{tm + \tau \sum_i y_i}{t + n\tau} \\
t' &= t + n\tau
\end{aligned}
$$

And we get

$$
P(\mu \mid y_1, \ldots, y_n) \propto exp(-\frac{1}{2} t'(\mu - m')^2)
$$

Therefore the posterior of μ is also a normal distribution, whose mean is m' and precision is t'. We thus call the normal distribution to be *self-conjugate*.

Problem

The data set `survey` contains height measurements of a class of university statistics students. We assume the student heights fit $N(\mu, 10^2)$. As for the prior distribution of μ, based on past surveys, we decide to use $N(160, 1.2^2)$. Estimate the posterior mean and standard deviation of μ.

Solution

As before, we save the height measurements in a variable `y`, and denote its length by `n`.

```
> library(MASS)
> y <- na.omit(survey$Height)
> n <- length(y)
```

Denote the precision of `y` by `tau`, and the prior precision of μ by `t.prior`. Based on assumptions on the variance of `y` and the prior of μ, we have:

```
> tau <- 1/(10*10)
> t.prior <- 1/(1.2*1.2)
```

According to the formula above, we can find the posterior precision of μ, which we denote by `t.post`. The reciprocal of its square root then gives us the posterior standard deviation `s.post`.

```
> t.post <- t.prior + n*tau
> s.post <- sqrt(1/t.post); s.post
[1] 0.5993
```

Denote the prior mean of μ by `m.prior`. Applying the formula again, we find the posterior mean `m.post`.

```
> m.prior <- 160
> m.post <- (1/t.post) *
+     (m.prior*t.prior + sum(y)*tau)
> m.post
[1] 169.3
```

Exercise

Consider the pulse rate of the students in the data set `survey`. Assume the pulse data fit a normal distribution $N(\mu, 12^2)$. As for the prior of μ, we decide to use a normal distribution $N(68, 1.4^2)$. Estimate the posterior mean and standard deviation of μ.

Solution

Repeating the computation as before with the pulse data shows that the posterior mean and standard deviation of μ are 72.45 and 0.7365 respectively.

23.3 Inference for Normal Mean with Known Variance

A key distinction between BUGS and the R language is the parametrization of the normal distribution. More specifically, in the BUGS language, we parametrize the normal distribution by its mean and precision values. For example, the following BUGS expression indicates that a random variable `x` is normally distributed with population mean `mu` and precision `tau`.

```
    x ~ dnorm(mu, tau)
```

To select a non-informative prior for the mean of a normal distribution, an obvious choice is the uniform distribution. It has the nice characteristic of being invariant under translation. However, the uniform distribution is improper over the real line, and we have to truncate it between two numbers.

```
    mu ~ dunif(-1e10, 1e10)
```

Nevertheless, it is a safe bet so long as we are certain that the population mean never exceeds the limits, which is indeed the case in our next example.

We should mention that another popular prior for the mean of normal distribution is a vague conjugate prior such as `dnorm(0, 0.001)`, which we will use later.

Problem

The data set `survey` contains the height measurements of a class of university statistics students. We assume that the student heights fit a normal distribution $N(\mu, 10^2)$. Using uniform prior, estimate the mean and standard deviation of μ in OpenBUGS.

Solution

We denote the student heights by `y`, which is a sequence of `n` identical independent normally distributed variables. With the truncated uniform prior, we have

```
model <- function() {
    # Prior
```

```
    mu ~ dunif(-1e10, 1e10)

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu, tau)
    }
}
```

Then we define the data parameters with the given values.

```
> y <- as.numeric(
    na.omit(survey$Height))
> n <- length(y)

> tau <- 1/(10*10)
> data <- list("y", "n", "tau")
> params <- c("mu")
> inits <- function() { list(mu = 0) }
```

After the simulation, we can find the posterior mean and standard deviation of μ from the output.

```
> out$mean["mu"]
$mu
[1] 172.4

> out$sd["mu"]
$mu
[1] 0.6957
```

Source Listing

```
library(MASS)
y <- as.numeric(
    na.omit(survey$Height))
n <- length(y)

tau <- 1/(10*10)
data <- list("y", "n", "tau")
params <- c("mu")
inits <- function() { list(mu = 0) }

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
```

```
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$mean["mu"]
out$sd["mu"]
```

Exercise

Consider the pulse rate of the students in the data set `survey`. Assume the pulse data fits $N(\mu, 12^2)$. Using uniform prior, estimate the mean and standard deviation of μ in OpenBUGS.

Solution

We make the following modification for the pulse data, and find the mean and standard deviation of μ to be 74.15 and 0.8705 respectively.

```
y <- as.numeric(na.omit(survey$Pulse))
n <- length(y)
tau <- 1/(12*12)
```

23.4 Inference for Normal Mean with Unknown Variance

All examples in this chapter so far assume prior knowledge of the population variance, which is rather impractical. With OpenBUGS, as a reward for the hard works we have spent in learning the new tools, we can now remove the restriction with little effort.

In fact, since the population variance is the inverse of the precision parameter τ, it suffices to find a posterior estimate of τ. Hence we need to select a prior of τ, and would like it to be *scale invariant*. If $u=\log(\tau)$, then u should be translational invariant. Observe that

$$du = \frac{1}{\tau} d\tau$$

Therefore, if u has uniform density, then the probability density of τ should be $1/\tau$, which unfortunately is improper. Nevertheless, we will be content with a close approximation using the following gamma distribution.

```
tau ~ dgamma(0.001, 0.001)
```

To derive the standard deviation `sigma` from `tau`, we can relate the two with the assignment operator "`<-`" in a **logical expression**.

```
sigma <- sqrt(1/tau)
```

Problem

The data set `survey` contains the height measurements of a class of university statistics students. Assuming the student heights fit a normal distribution $N(\mu, \sigma^2)$, find point estimates for both μ and σ.

Solution

We denote the student height measurements by `y`, and derive the standard deviation

sigma as follows.

```
model <- function() {
    # Prior
    mu ~ dunif(-1e10, 1e10)
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu, tau)
    }

    # Derived
    sigma <- sqrt(1/tau)
}
```

We then define y and n for our data parameters, and set mu and sigma in params for monitoring the output. Note that we have to select a positive initial value for tau.

```
> y <- as.numeric(
+     na.omit(survey$Height))
> n <- length(y)

> data <- list("y", "n")
> params <- c("mu", "sigma")
> inits <- function() {
+     list(mu=0, tau=1)
+ }
```

After the simulation, we can retrieve the posterior estimates of the parameters from the output. The unlist method is here to improve the display format.

```
> unlist(out$mean[params])
     mu   sigma
172.378   9.883

> unlist(out$sd[params])
    mu  sigma
0.6867 0.4874
```

Source Listing

```
library(MASS)
y <- as.numeric(
    na.omit(survey$Height))
```

```
n <- length(y)

data <- list("y", "n")
params <- c("mu", "sigma")
inits <- function() {
    list(mu=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

unlist(out$mean[params])
unlist(out$sd[params])
```

Exercise

Consider the pulse rate of the students in the data set `survey`. Assume the data fits a normal distribution $N(\mu, \sigma^2)$. Find posterior estimates of both μ and σ.

Solution

We modify the parameters for the pulse data and find the result as follows.

```
> unlist(out$mean[params])
   mu sigma
74.15 11.73

> unlist(out$sd[params])
    mu  sigma
0.8489 0.6036
```

Chapter 24
Bayesian Inference for Two Populations

We have seen how to compare two normally distributed populations using frequentist statistics. To perform similar Bayesian inference with OpenBUGS, we will create models that explicitly describe the problem, instead of relying on an opaque test. It will be more flexible, and more reusable for future development.

Analogous to confidence interval in frequentist statistics, we have the concept of **credible interval** in Bayesian statistics. It is an interval estimate of the posterior probability. For example, a 95% credible interval of a population mean is an interval of 95% probability under the posterior distribution.

However, we will avoid explicit significance test. Although it is simple to use, significance test is controversial in Bayesian statistics. Rigorous framework of Bayesian hypothesis testing do exist, but they are beyond our scope.

24.1 Inference for Two Matched Samples

When we have two matched samples from repeated measurements in the same experiment, say `y1` and `y2`, we can pair up the data and calculate their difference.

```
    y <- y1 -y2
```

Therefore, comparing the population means of two matched samples is the same as finding the population mean of their difference, and we can apply one of our previous models for this purpose.

Problem

The data set `immer` contains the yield data of six barley fields in years 1931 and 1932. The 1931 yield is in `Y1`, and the 1932 yield is in `Y2`. Assuming the data to be normally distributed, find a 95% credible interval of the difference in population means between `Y1` and `Y2`.

Solution

We denote the two barley yield data as `y1` and `y2`. With matched samples, we can denote their difference as `y`.

```
> library(MASS)
> y1 <- immer$Y1
> y2 <- immer$Y2

> y <- y1 - y2
> n <- length(y)
```

We then investigate the expected value of `y` using our previous model for the mean of a normal distribution.

```
model <- function() {
    # Priors
    mu ~ dunif(-1e10, 1e10)
    tau ~ dgamma(0.001, 0.001)
```

```
    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu, tau)
    }
}
```

Then we define similar data and initial parameters.

```
> data <- list("y", "n")
> params <- c("mu")
> inits <- function() {
+     list(mu=0, tau=1)
+ }
```

After the simulation, we can retrieve the result from the output summary.

```
> unlist(out$mean[params])
   mu
15.88
```

And we can find the 2.5% and 97.5% quantiles in the output summary as well. Note that the 95% credible interval of `mu` excludes the zero value, which suggests a non-trivial difference between the means of the two data populations.

```
> out$summary[params,
+     c("2.5%", "97.5%")]
  2.5%  97.5%
 6.076 25.550
```

Source Listing

```
library(MASS)
y1 <- immer$Y1
y2 <- immer$Y2

y <- y1 - y2
n <- length(y)

data <- list("y", "n")
params <- c("mu")
inits <- function() {
    list(mu=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
```

```
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

unlist(out$mean[params])

out$summary[params,
    c("2.5%", "97.5%")]
```

Note 1

The 95% credible interval is similar to the 95% confidence interval found in the paired `t.test`.

```
> t.test(y1, y2, pair=TRUE)

    Paired t-test

data:  y1 and y2
t = 3.324, df = 29, p-value = 0.002413
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
  6.122 25.705
sample estimates:
mean of the differences
                  15.91
```

Note 2

To find a 99% credible interval, we can compute the 0.5% and 99.5% quantiles of the corresponding `sims.list` component in the simulation output.

```
> alpha <- 0.01
> I <- c(alpha/2, 1-alpha/2)
> quantile(out$sims.list$mu, I)
  0.5%  99.5%
 2.515 29.050
```

24.2 Inference with Equal Variances

We have seen how to use a `for` loop to model a sequence of random variables in Bayesian Poisson inference. For two independent data samples, we would need two separate `for` loops. Assume that the data samples are normally distributed with a common population variance. We can write the `for` loops as follows.

```
for (i in 1:n1) {
    y1[i] ~ dnorm(mu1, tau)
}

for (j in 1:n2) {
    y2[j] ~ dnorm(mu2, tau)
}
```

Problem

The data set `ToothGrowth` contains the result of a food induced tooth growth experiment on guinea pigs, which are divided into two separate groups for treatments in orange juice or Vitamin C. Assume the test results to be normally distributed with identical population variance. Find a 95% credible interval of the mean difference in tooth growth between the two groups.

Solution

We create a logical filter `L` that determines if the treatment is in orange juice (OJ) or not. It separates the treatment results into `y1` and `y2`.

```
> L <- ToothGrowth$supp == "OJ"
> len <- ToothGrowth$len
> y1 <- len[L]
> y2 <- len[!L]
```

Specify the respective means of `y1` and `y2` by `mu1` and `mu2`, and denote their difference by `delta`. Assuming common population variance, we have the following BUGS model for the two data samples.

```
model <- function() {
```

```
    # Priors
    mu1 ~ dunif(-1e10, 1e10)
    mu2 ~ dunif(-1e10, 1e10)
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n1) {
        y1[i] ~ dnorm(mu1, tau)
    }

    for (j in 1:n2) {
        y2[j] ~ dnorm(mu2, tau)
    }

    # Difference in means
    delta <- mu1 - mu2
}
```

Then we define the data and initial parameters for both samples.

```
> n1 <- length(y1)
> n2 <- length(y2)

> data <- list("y1", "y2", "n1", "n2")
> params <- c("mu1", "mu2", "delta")
> inits <- function() {
+     list(mu1=0, mu2=0, tau=1)
+ }
```

After the simulation, we can retrieve the population means and their difference from the output summary.

```
> unlist(out$mean[params])
   mu1    mu2  delta
20.663 16.961  3.701
```

And we can find the 2.5% and 97.5% quantiles as follows. Note that the 95% credible interval of `delta` fails to exclude the zero value.

```
> out$summary[params,
+     c("2.5%", "97.5%")]
         2.5%  97.5%
mu1   17.8900 23.410
mu2   14.2497 19.710
delta -0.1908  7.568
```

Source Listing

```
L <- ToothGrowth$supp == "OJ"
len <- ToothGrowth$len
y1 <- len[L]
y2 <- len[!L]

n1 <- length(y1)
n2 <- length(y2)

data <- list("y1", "y2", "n1", "n2")
params <- c("mu1", "mu2", "delta")
inits <- function() {
    list(mu1=0, mu2=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

unlist(out$mean[params])

out$summary[params,
    c("2.5%", "97.5%")]
```

Note 1

We can apply `var.test` to verify common variance.

```
> var.test(y1, y2)

    F test to compare two variances

data:  y1 and y2
F = 0.6386, num df = 29, denom df = 29,
p-value = 0.2331
alternative hypothesis: true ratio of variances is not equal to 1
95 percent confidence interval:
 0.3039 1.3417
sample estimates:
ratio of variances
            0.6386
```

Note 2

The point estimates of `mu1` and `mu2` is similar to the results in the unpaired `t.test`. So is the 95% credible interval of `delta`.

```
> t.test(y1, y2)

    Welch Two Sample t-test

data:  y1 and y2
t = 1.915, df = 55.31, p-value = 0.06063
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -0.171  7.571
sample estimates:
mean of x mean of y
    20.66     16.96
```

24.3 Inference with Unequal Variances

In the previous section, we have used separate `for` loops to compare two independent data samples with common population variance. In general, we need separate precision parameters `tau1` and `tau2` for the loops.

```
    for (i in 1:n1) {
        y1[i] ~ dnorm(mu1, tau1)
    }

    for (j in 1:n2) {
        y2[j] ~ dnorm(mu2, tau2)
    }
```

Problem

The data set `beavers` contains body temperature measurements of two pairs of female beavers. Assume the data to be normally distributed, find a 95% credible interval of the difference in their mean body temperatures.

Solution

With separate precision parameters for each data loop, we have the following.

```
model <- function() {
    # Priors
    mu1 ~ dunif(-1e10, 1e10)
    tau1 ~ dgamma(0.001, 0.001)

    mu2 ~ dunif(-1e10, 1e10)
    tau2 ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n1) {
        y1[i] ~ dnorm(mu1, tau1)
    }

    for (j in 1:n2) {
        y2[j] ~ dnorm(mu2, tau2)
    }
```

```
    # Derived
    delta <- mu1 - mu2
}
```

Then we define the data and initial parameters with the given data.

```
> y1 <- beaver1$temp
> y2 <- beaver2$temp

> n1 <- length(y1)
> n2 <- length(y2)

> data <- list("y1", "y2", "n1", "n2")
> params <- c("mu1", "mu2", "delta")
> inits <- function() {
+ list(mu1=0, mu2=0, tau1=1, tau2=1)
+ }
```

After the simulation, we can retrieve the population means and their difference from the output summary.

```
> unlist(out$mean[params])
    mu1     mu2   delta
36.8621 37.5964 -0.7343
```

Similarly we can find the 2.5% and 97.5% quantiles. Note that the 95% credible interval of `delta` excludes the zero value, which suggests non-trivial difference in body temperatures of the two groups.

```
> out$summary[params,
+     c("2.5%", "97.5%")]
         2.5%  97.5%
mu1   36.8300 36.900
mu2   37.5100 37.690
delta -0.8302 -0.637
```

Source Listing

```
y1 <- beaver1$temp
y2 <- beaver2$temp

n1 <- length(y1)
n2 <- length(y2)

data <- list("y1", "y2", "n1", "n2")
```

```
params <- c("mu1", "mu2", "delta")
inits <- function() {
  list(mu1=0, mu2=0, tau1=1, tau2=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

unlist(out$mean[params])
```

Note

The point estimates of `mu1` and `mu2` are similar to the results of the unpaired `t.test`. So is the 95% credible interval of `delta`.

```
> t.test(y1, y2)

    Welch Two Sample t-test

data:  y1 and y2
t = -15.23, df = 131.1, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -0.8299 -0.6391
sample estimates:
mean of x mean of y
    36.86     37.60
```

Chapter 25
Bayesian Analysis of Variance

The Analysis of Variance (ANOVA) is an important tool for analyzing experimental results. In the first few sections of this chapter, we will demonstrate how to perform Bayesian ANOVA in OpenBUGS under the classical assumptions: independent response treatments, normally distributed response data, and homogeneous group variance. Lastly, we will show how to remove the assumption of variance homogeneity, a.k.a. *homoscedasticity*.

25.1 Bayesian Completely Randomized Design

In a completely randomized design, there is only a single primary factor under consideration in the experiment. The test subjects are assigned to treatment levels at random.

Our first step is to designate the first treatment level in the primary factor to be the *baseline treatment level*, and denote the expected value of its response as `alpha`. We then denote the *contrast effects* with other treatment levels by a vector `beta`. For example, `beta[2]` is the difference in expected values between the response of the second treatment level and the baseline treatment level, whereas `beta[3]` is the difference in expected values between the response of the third treatment level and the baseline treatment level, and so forth. The first element `beta[1]` is always zero.

Assume `y` to be a vector of response variables. Denote the expected value of `y[i]` by `mu[i]`. Since `y[i]` is normally distributed with common variance, we can write

```
y[i] ~ dnorm(mu[i], tau)
```

We denote the treatment levels of the response data by a vector `tm`. For example, if `tm[i]=2`, then `y[i]` belongs to the second treatment level. Since the contrast effect of the second treatment group is `beta[2]`, the expected value of `y[i]` is `alpha+beta[2]`. Therefore we have the following in general.

```
mu[i] <- alpha + beta[tm[i]]
```

Enumerating all members of `y`, we have the following.

```
for (i in 1:N) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha + beta[tm[i]]
}
```

Example

A fast food franchise is test marketing 3 new menu items. To find out if they the same popularity, 18 franchisee restaurants are randomly chosen for participation in the study. In accordance with the completely randomized design, 6 of the restaurants are randomly chosen to test market the first new menu item, another 6 for the second menu item, and the remaining 6 for the last menu item.

Problem

Suppose the following table represents the sales figures of the 3 new menu items in the 18 restaurants after a week of test marketing. Find 95% credible intervals of differences in mean sales figures.

```
 Item1 Item2 Item3
    22    52    16
    42    33    24
    44     8    19
    52    47    18
    45    43    34
    37    32    39
```

Solution

The sales figures is part of the code download. We load the data in "fastfood-1.txt" with the method `read.table`, and find 18 items in 3 separate column groups, each of which has 6 row items.

```
> df <- read.table("fastfood-1.txt",
+     header=TRUE)
> ngroup <- ncol(df); ngroup
[1] 3
> nsample <- nrow(df); nsample
[1] 6
> N <- ngroup*nsample; N
[1] 18
```

We then create a vector of treatment labels `tm` that matches the column position of elements in `df`. Hence the first group of six elements in `tm` matches the first column of `df`, the second group of six elements in `tm` matches the second column of `df`, and so forth.

```
> tm <- gl(ngroup, nsample, N); tm
 [1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 ...
Levels: 1 2 3
```

It is helpful to print out the design matrix at this point. The first column of the following design matrix matches the baseline effect. The second and third columns match contrast effects.

```
> model.matrix(~tm)
   (Intercept) tm2 tm3
1            1   0   0
2            1   0   0
3            1   0   0
4            1   0   0
5            1   0   0
6            1   0   0
7            1   1   0
8            1   1   0
9            1   1   0
        ......
```

Since the design matrix compares only the first baseline treatment level with the rest, we need to define an extra contrast parameter `eff` to compare the second and third treatment levels separately.

To summarize, based on the ANOVA assumptions and the design matrix, we have the following BUGS model. Note that `beta[1]` is hardcoded to be zero.

```
model <- function() {
    # Priors
    alpha ~ dunif(-1e10, 1e10)

    beta[1] <- 0
    for (k in 2:ngroup) {
        beta[k] ~ dunif(-1e10, 1e10)
    }

    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:N) {
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta[tm[i]]
    }

    # Derived
    eff <- beta[2] - beta[3]
}
```

While we define `beta[1]` to be zero in the BUGS model, we set it as `NA` during

the initialization.

```
inits <- function() {
    beta <- numeric(ngroup)
    beta[1] <- NA

    list(alpha=0, beta=beta, tau=1)
}
```

Then we set the data parameters. We have to cast the factor variable `tm` into numeric type for OpenBUGS.

```
y <- as.numeric(as.matrix(df))
data <- list(
    y = y,
    tm = as.numeric(tm),
    ngroup = ngroup,
    N = N
    )
params <- c("alpha", "beta", "eff")
```

After the simulation, we can retrieve the 95% credible intervals of the contrast parameters `beta[2]`, `beta[3]`, and `eff`. Only `beta[3]` manages to exclude the zero value.

```
> out$summary[,  c("2.5%", "97.5%")]
            2.5%    97.5%
alpha     30.160  51.0302
beta[2]  -19.760   9.5546
beta[3]  -30.550  -0.7822
eff       -4.064  25.5602
deviance 138.100 150.6000
```

Source Listing

```
df <- read.table("fastfood-1.txt",
    header=TRUE); df

ngroup <- ncol(df); ngroup
nsample <- nrow(df); nsample
N <- ngroup*nsample; N

tm <- gl(ngroup, nsample, N); tm
model.matrix(~tm)

y <- as.numeric(as.matrix(df))
```

```
data <- list(
    y = y,
    tm = as.numeric(tm),
    ngroup = ngroup,
    N = N
    )
params <- c("alpha", "beta", "eff")

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$summary[,  c("2.5%", "97.5%")]
```

Note

We apply the `anova` method and find the result to be a little bit off.

```
> fastfood.lm <- lm(y ~ tm)
> anova(fastfood.lm)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
tm         2    745     373    2.54   0.11
Residuals 15   2200     147
```

However, the following shows almost identical parameter estimates with the classical linear model. As added bonus, we managed to estimate the contrast between the second and third treatment levels without extra effort using OpenBUGS.

```
> cbind(unlist(out$mean[params]))
         [,1]
alpha  40.353
beta1  -4.475
beta2 -15.369
eff    10.895

> cbind(coefficients(fastfood.lm))
              [,1]
(Intercept)  40.33
tm2          -4.50
```

tm3 -15.33

25.2 Bayesian Randomized Block Design

In a randomized block design, we still have only one primary factor in the experiment. Yet we group similar test subjects into blocks that are independent of the primary treatment.

We designate the first treatment on the first subject block to be the baseline level, and denote the expected value of its response by `alpha`. We then denote the contrast effects with other treatments by a vector `tm.beta`, and the contrast effects with other block assignments by `blk.beta`.

If we denote the treatment levels by a vector `tm`, and block assignments by a vector `blk`, then similar argument as Bayesian Completely Randomized Design gives the following.

```
for (i in 1:N) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha +
                tm.beta[tm[i]] +
                blk.beta[blk[i]]
}
```

Example

A fast food franchise is test marketing 3 new menu items. To find out if they have the same popularity, 6 franchisee restaurants are randomly chosen for participation in the study. In accordance with the randomized block design, each restaurant will be test marketing all 3 new menu items. Furthermore, a restaurant will test market only one menu item per week, and it takes 3 weeks to test market all menu items. The testing order of the menu items for each restaurant is randomly assigned as well.

Problem

Suppose each row in the following table represents the sales figures of the 3 new menu items in a restaurant after a week of test marketing. Find 95% credible intervals of differences in mean sales figures among the menu items.

```
 Item1 Item2 Item3
    31    27    24
    31    28    31
    45    29    46
    21    18    48
    42    36    46
    32    17    40
```

Solution

The data is part of the code download, and we can find it in the file "fastfood-2.txt". It contains 18 items in 3 separate column groups, each of which with 6 row items.

```
> df <- read.table("fastfood-2.txt",
+         header=TRUE)
> ngroup <- ncol(df); ngroup
[1] 3
> nsample <- nrow(df); nsample
[1] 6
> N <- ngroup*nsample; N
[1] 18
```

We then create a vector of treatment labels `tm` that matches column positions of elements in `df`. We also create a vector of block labels `blk` that matches row positions of elements in `df`.

```
> tm <- gl(ngroup, nsample, N); tm
 [1] 1 1 1 1 1 1 2 2 2 2 2 2 3 3 ...
Levels: 1 2 3

> blk <- gl(nsample, 1, N); blk
 [1] 1 2 3 4 5 6 1 2 3 4 5 6 1 2 ...
Levels: 1 2 3 4 5 6
```

It is helpful to print out the design matrix at this point. The first column of the following design matrix matches the baseline effect, while the rest match contrast effects of other treatment groups and blocking factors.

```
> model.matrix(~tm + blk)
   (Intercept) tm2 tm3 blk2 blk3  ...
1            1   0   0    0    0  ...
2            1   0   0    1    0  ...
3            1   0   0    0    1  ...
       ........
```

Since the design matrix compares only the baseline treatment level with the second and third treatment levels, we need to define an extra contrast parameter `tm.eff` to compare the second and third treatment levels.

To summarize, based on the ANOVA assumptions and the design matrix, we have the following BUGS model. Note that `tm.beta[1]` and `blk.beta[1]` are hardcoded to be zero.

```
model <- function() {
    # Priors
    alpha ~ dunif(-1e10, 1e10)

    tm.beta[1] <- 0
    for (k in 2:ngroup) {
        tm.beta[k] ~
             dunif(-1e10, 1e10)
    }

    blk.beta[1] <- 0
    for (k in 2:nsample) {
        blk.beta[k] ~
             dunif(-1e10, 1e10)
    }

    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:N) {
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha +
                    tm.beta[tm[i]] +
                    blk.beta[blk[i]]
    }

    # Derived
    tm.eff <- tm.beta[2] - tm.beta[3]
}
```

During the initialization, we set the first elements of `tm.beta` and `blk.beta` as `NA` to allow for explicit values within the model.

```
inits <- function() {
    tm.beta <- numeric(ngroup)
    tm.beta[1] <- NA
    blk.beta <- numeric(nsample)
    blk.beta[1] <- NA
```

```
    list(alpha = 0,
        tm.beta = tm.beta,
        blk.beta = blk.beta,
        tau = 1
        )
}
```

Then we set the data parameters after casting the treatment and block factors into the numeric type.

```
y <- as.numeric(as.matrix(df))
data <- list(
    y = y,
    tm = as.numeric(tm),
    blk = as.numeric(blk),
    ngroup = ngroup,
    nsample = nsample,
    N = N)
params <- c("alpha",
    "tm.beta", "tm.eff", "blk.beta")
```

After the simulation, we can retrieve the 95% credible intervals of the contrast parameters. Note that `tm.eff` is the contrast between the second and third menu items, and its 95% credible interval excludes the zero value.

```
> out$summary[,  c("2.5%", "97.5%")]
                2.5%   97.5%
alpha        17.1800  38.850
tm.beta[2]  -17.6003   1.624
tm.beta[3]   -3.9101  15.080
tm.eff      -22.9400  -4.153
blk.beta[2] -10.6002  16.100
blk.beta[3]  -0.4757  26.060
blk.beta[4] -11.6500  15.080
blk.beta[5]   0.6675  27.190
blk.beta[6] -10.8502  15.820
deviance    116.1000 139.000
```

Source Listing

```
df <- read.table("fastfood-2.txt",
        header=TRUE); df

ngroup <- ncol(df); ngroup
nsample <- nrow(df); nsample
N <- ngroup*nsample; N
```

```
tm <- gl(ngroup, nsample, N); tm
blk <- gl(nsample, 1, N); blk
model.matrix(~tm + blk)

y <- as.numeric(as.matrix(df))
data <- list(
    y = y,
    tm = as.numeric(tm),
    blk = as.numeric(blk),
    ngroup = ngroup,
    nsample = nsample,
    N = N)
params <- c("alpha",
    "tm.beta", "tm.eff", "blk.beta")

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$summary[,  c("2.5%", "97.5%")]
```

Note

We can apply the `anova` method for comparison.

```
> fastfood.lm <- lm(y ~ tm + blk)
> anova(fastfood.lm)
Analysis of Variance Table

Response: y
          Df Sum Sq Mean Sq F value Pr(>F)
tm         2    539   269.4    4.96  0.032 *
blk        5    560   112.0    2.06  0.155
Residuals 10    543    54.3
```

Nevertheless, the following shows that the parameters have similar point estimates from either approach. We also have `tm.eff` as added bonus from OpenBUGS.

```
> cbind(unlist(out$mean[params]))
             [,1]
alpha      27.996
tm.beta1   -7.826
```

```
tm.beta2    5.558
tm.eff    -13.384
blk.beta1   2.696
blk.beta2  12.789
blk.beta3   1.765
blk.beta4  14.062
blk.beta5   2.443

> cbind(coefficients(fastfood.lm))
             [,1]
(Intercept) 28.111
tm2         -7.833
tm3          5.500
blk2         2.667
blk3        12.667
blk4         1.667
blk5        14.000
blk6         2.333
```

25.3 Bayesian Factorial Design

In a factorial design, there are more than one treatment factors under consideration in the experiment, and there maybe interaction effects. Here, we consider an example with only two treatment factors.

We designate the combined effect from first levels of both treatment factors to be the baseline effect, and denote it by `alpha`. We then denote the contrast effects with the first treatment factor by `t1.beta`. and contrast effects with the second treatment factor by `t2.beta`. We also define an extra interaction factor, and denote its contrast effects by `t3.beta`. We will show how to compute the interaction factor shortly.

We further denote the treatment factor levels of each response by `t1` and `t2`, and the interaction factor level by `t3`. The total effect is a combination of the baseline effect with all contrast effects. Therefore,

```
for (i in 1:N) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha +
                t1.beta[t1[i]] +
                t2.beta[t2[i]] +
                t3.beta[t3[i]]
}
```

Example

A fast food franchise is test marketing 3 new menu items in both East and West Coasts of continental United States. To find out if they have the same popularity, 12 franchisee restaurants from each coast are randomly chosen for participation in the study. In accordance with the factorial design, within the 12 restaurants from East Coast, 4 are randomly chosen to test market the first new menu item, another 4 for the second menu item, and the remaining 4 for the last menu item. The 12 restaurants from the West Coast are arranged likewise.

Problem

Suppose the following tables represent the sales figures of the 3 new menu items after a week of test marketing. The upper half of the table represents the sales figures of East Coast restaurants. The lower half represents West Coast restaurants. Find 95% credible intervals of differences in mean sales figures among the menu items and between the coasts.

```
East Coast:
==========
   Item1 Item2 Item3
E1    25    39    36
E2    36    42    24
E3    31    39    28
E4    26    35    29

West Coast:
==========
   Item1 Item2 Item3
W1    51    43    42
W2    47    39    36
W3    47    53    32
W4    52    46    33
```

Solution

The data is part of the code download, and we can find it in the file "fastfood-3.csv". There are three columns of data, with each column evenly divided into upper and lower halves.

```
> df <- read.csv("fastfood-3.csv"); df
   Item1 Item2 Item3
E1    25    39    36
E2    36    42    24
E3    31    39    28
E4    26    35    29
W1    51    43    42
W2    47    39    36
W3    47    53    32
W4    52    46    33
```

We then define the names of the treatment levels. There are 4 observations per treatment levels pair.

```
> f1 <- c("Item1", "Item2", "Item3")
> f2 <- c("East", "West")
> n1 <- length(f1)
```

```
> n2 <- length(f2)
>
> nsample <- 4
> N <- n1*n2*nsample; N
[1] 24
```

We create a factor label vector `tm1` that matches the first treatment factor of the column-wise elements in `df`. Therefore the first group of eight elements in `tm1` denotes restaurants with the first menu item, the second group of eight elements denotes restaurants with the second menu item, and so forth.

```
> tm1 <- gl(n1, nsample*n2, N); tm1
 [1] 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2
[17] 3 3 3 3 3 3 3 3
Levels: 1 2 3
```

Similarly, we create another factor label vector `tm2` that matches the second treatment factor of the column-wise elements in `df`. Hence the first half of the each group of eight elements in `tm2` denotes the East Coast restaurants, and the second half of same group denotes the West Coast restaurants.

```
> tm2 <- gl(n2, nsample, N); tm2
 [1] 1 1 1 1 2 2 2 2 1 1 1 1 2 2 2 2
[17] 1 1 1 1 2 2 2 2
Levels: 1 2
```

It is helpful to print out the design matrix at this point. The first column in the following design matrix matches the baseline effect, and the rest are contrast effects. For example, `tm13` denotes the contrast due to the third treatment level in `tm1`, and `tm13:tm22` denotes the contrast due to the interaction between the third treatment level in `tm1` and the second treatment level in `tm2`.

```
> model.matrix(~tm1 * tm2)
   (Intercept) tm12 tm13 tm22 tm12:tm22 tm13:tm22
1            1    0    0    0    ...
2            1    0    0    0    ...
3            1    0    0    0    ...
       ........
```

Based on the design matrix, we can find `t3` for the interaction factor levels of the response data as follows.

```
> t1 <- as.numeric(tm1)
> t2 <- as.numeric(tm2)
```

```
> t3 <- ifelse(t1==1 | t2==1, 1,
+         (t2-2)*(n1-1)+t1)
> t3
 [1] 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2
[17] 1 1 1 1 3 3 3 3
```

We can now define the following BUGS model using the ANOVA assumptions and the design matrix. Note how we hardcode `t1.beta[1]`, `t2.beta[1]`, and `t3.beta[1]` to be zero.

```
model <- function() {
    # Priors
    alpha ~ dunif(-1e10, 1e10)

    t1.beta[1] <- 0
    for (k in 2:n1) {
        t1.beta[k] ~
             dunif(-1e10, 1e10)
    }

    t2.beta[1] <- 0
    for (k in 2:n2) {
        t2.beta[k] ~
             dunif(-1e10, 1e10)
    }

    t3.beta[1] <- 0
    for (k in 2:n3) {
        t3.beta[k] ~
             dunif(-1e10, 1e10)
    }

    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:N) {
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha +
                    t1.beta[t1[i]] +
                    t2.beta[t2[i]] +
                    t3.beta[t3[i]]
    }

}
```

We define the initialization as follows. Note the use of `NA` as first elements in the contrast effects, as it allows us to define explicit values in the model.

```
inits <- function() {
    t1.beta <- numeric(n1)
    t1.beta[1] <- NA
    t2.beta <- numeric(n2)
    t2.beta[1] <- NA
    t3.beta <- numeric(n3)
    t3.beta[1] <- NA

    list(alpha = 0,
        t1.beta = t1.beta,
        t2.beta = t2.beta,
        t3.beta = t3.beta,
        tau = 1
        )
}
```

Then we define the data parameters. Since there is no need to compute values, we just enter the variable names.

```
y <- as.numeric(as.matrix(df))
data <- list("y", "t1", "t2", "t3",
            "n1", "n2", "n3", "N")
params <- c("alpha", "t1.beta",
            "t2.beta", "t3.beta")
```

After the simulation, we can retrieve the 95% credible intervals of the contrast effects.

```
> out$summary[,  c("2.5%", "97.5%")]
              2.5%   97.5%
alpha       24.690  34.150
t1.beta[2]   2.613  15.990
t1.beta[3]  -6.898   6.446
t2.beta[2]  12.960  26.480
t3.beta[2] -22.720  -3.524
t3.beta[3] -22.770  -3.531
deviance   135.300 152.300
```

We observe that `t1.beta[2]` measures the contrast effect between the first and second menu items, and its 95% credible interval excludes the zero value. Furthermore, `t2.beta[2]` measures the difference in sales between the coasts, and its 95% credible interval excludes the zero value as well.

Source Listing

```
df <- read.csv("fastfood-3.csv"); df
```

```
f1 <- c("Item1", "Item2", "Item3")
f2 <- c("East", "West")
n1 <- length(f1)
n2 <- length(f2)

nsample <- 4
N <- n1*n2*nsample; N

tm1 <- gl(n1, nsample*n2, N); tm1
tm2 <- gl(n2, nsample, N); tm2

model.matrix(~tm1 * tm2)

t1 <- as.numeric(tm1)
t2 <- as.numeric(tm2)

t3 <- ifelse(t1==1 | t2==1, 1,
        (t2-2)*(n1-1)+t1)
t3

n3 <- (n1-1)*(n2-1)+1
n3

y <- as.numeric(as.matrix(df))
data <- list("y", "t1", "t2", "t3",
            "n1", "n2", "n3", "N")
params <- c("alpha", "t1.beta",
            "t2.beta", "t3.beta")

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$summary[,  c("2.5%", "97.5%")]
```

Note

We apply the usual `anova` method and find similar conclusions.

```
> fastfood.lm <- lm(y ~ tm1 * tm2)
> anova(fastfood.lm)
Analysis of Variance Table
```

```
Response: y
          Df Sum Sq Mean Sq F value  Pr(>F)
tm1        2    385     193    9.55  0.0015 **
tm2        1    715     715   35.48 1.2e-05 ***
tm1:tm2    2    234     117    5.81  0.0113 *
Residuals 18    363      20
```

The following shows that the main and contrast parameters are similar in either approach.

```
> cbind(unlist(out$mean[params]))
             [,1]
alpha     29.4131
t1.beta1   9.3430
t1.beta2  -0.1823
t2.beta   19.8511
t3.beta1 -13.3733
t3.beta2 -13.3212

> cbind(coefficients(fastfood.lm))
              [,1]
(Intercept)  29.50
tm12          9.25
tm13         -0.25
tm22         19.75
tm12:tm22   -13.25
tm13:tm22   -13.25
```

25.4 Heterogeneous Group Variance

So far in this chapter, we have assumed homogeneous population variance among treatment groups. With OpenBUGS, we can easily remove the assumption by assigning individual precision parameter for each treatment group.

For example, we can modify the model in the Completely Randomized Design, and define individual precision parameter `prec[k]` for each treatment group as follows.

```
    for (i in 1:N) {
        y[i] ~ dnorm(mu[i], tau[i])
        mu[i] <- alpha + beta[tm[i]]
        tau[i] <- prec[tm[i]]
    }
```

As for the standard deviation `sigma`, we can derive it from the precision variable `prec`.

```
    for (k in 1:ngroup) {
        prec[k] ~ dgamma(0.001, 0.001)
        sigma[k] <- sqrt(1/prec[k])
    }
```

Problem

Without assumption of homogeneous treatment group variance, perform ANOVA for our previous fastfood example of Completely Randomized Design. Find also 95% credible intervals of standard deviation of sales figure for each menu item.

Solution

Assigning individual precision parameter for each treatment group, the BUGS model becomes

```
model <- function() {
    # Priors
    alpha ~ dunif(-1e10, 1e10)
```

```
    beta[1] <- 0
    for (k in 2:ngroup) {
        beta[k] ~ dunif(-1e10, 1e10)
    }

    for (k in 1:ngroup) {
        prec[k] ~ dgamma(0.001, 0.001)
        sigma[k] <- sqrt(1/prec[k])
    }

    # Likelihood
    for (i in 1:N) {
        y[i] ~ dnorm(mu[i], tau[i])
        mu[i] <- alpha + beta[tm[i]]
        tau[i] <- prec[tm[i]]
    }

    # Derived
    eff <- beta[2] - beta[3]
}
```

We have to initialize the extra `prec` parameter.

```
inits <- function() {
    beta <- numeric(ngroup)
    beta[1] <- NA

    prec <- rep(1, ngroup)
    list(alpha=0, beta=beta, prec=prec)
}
```

We also need to add `sigma` to the monitor list.

```
params <- c("alpha", "beta",
    "eff", "sigma")
```

After the simulation, we can retrieve the 95% credible intervals. Since the credible intervals of elements in `sigma` have a fairly large common overlapping region, it strongly suggests homogeneous group variance.

```
> out$summary[,  c("2.5%", "97.5%")]
            2.5%    97.5%
alpha     29.250  50.3805
beta[2]  -23.941  14.9702
beta[3]  -29.720  -0.3576
eff       -8.625  30.0300
```

```
sigma[1]   6.353  24.4102
sigma[2]   9.773  38.9302
sigma[3]   5.919  23.4710
deviance 137.100 152.9000
```

Source Listing

```
df <- read.table("fastfood-1.txt",
    header=TRUE); df

ngroup <- ncol(df); ngroup
nsample <- nrow(df); nsample
N <- ngroup*nsample; N

tm <- gl(ngroup, nsample, N); tm
model.matrix(~tm)

y <- as.numeric(as.matrix(df))
data <- list(
    y = y,
    tm = as.numeric(tm),
    ngroup = ngroup,
    N = N
    )
params <- c("alpha", "beta",
    "eff", "sigma")

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$summary[,  c("2.5%", "97.5%")]
```

Chapter 26
Bayesian Simple Linear Regression

A simple linear regression model uses a linear equation to describe a dependent variable *y* with an independent variable *x* and an error term. In the following, we call α the intercept coefficient, β the slope coefficient, and *x* the **covariate**.

$$y = \alpha + \beta x + \epsilon$$

We will demonstrate simple linear regression using OpenBUGS with the data set `faithful`, which contains durations and waiting times of the Old Faithful geyser in the U.S. Yellowstone National Park. In this case, the linear model becomes

$$Eruptions = \alpha + \beta * Waiting + \epsilon$$

26.1 Coefficients of Simple Linear Regression

In a simple linear regression model, for a given value of x, the response data y is a random variable with expected value $E(y)$. We can break up the model into two parts. The first equation below denotes a *stochastic component* that expresses y as a normally distributed random variable, and the second equation denotes a *deterministic component* that expresses the expected value $E(y)$ linearly in terms of x.

$$\begin{aligned} y &= E(y) + \epsilon \\ E(y) &= \alpha + \beta x \end{aligned}$$

Hence, for a sequence of observations, we can write the following.

```
for (i in 1:n) {
    y[i] ~ dnorm(mu[i], tau)
    mu[i] <- alpha + beta*x[i]
}
```

Problem

Create a simple linear regression model for the data set `faithful`. Then find point estimates and 95% credible intervals of the intercept coefficient α and slope coefficient β.

Solution

We define the BUGS model as follows. Here, we use the vague prior `dnorm(0, 0.001)` for `alpha` and `beta`.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n) {
```

```
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*x[i]
    }
}
```

To prepare the data parameters, we extract the `waiting` component of `faithful` into a standalone variable. Since it is important *to center the covariate for simple linear regression*, we define a variable `x` as the difference of `waiting` and its mean. Note that this will affect the value of the intercept coefficient `alpha`. We also extract the `eruption` component into another variable *y*.

```
waiting <- faithful$waiting
x.m <- mean(waiting)

x <- waiting - x.m
y <- faithful$eruptions
```

Then we setup the data and initial parameters.

```
n <- length(x)
data <- list("x", "y", "n")
params <- c("alpha", "beta", "mu")
inits <- function() {
    list(alpha=0, beta=0, tau=1)
}
```

After the simulation, we can retrieve point estimates of the coefficients.

```
> cbind(unlist(out$mean[
+     c("alpha", "beta")]))

         [,1]
alpha 3.48693
beta  0.07565
```

The 95% credible intervals of the model coefficients are

```
> out$summary[c("alpha", "beta"),
+     c("2.5%", "97.5%")]

         2.5%   97.5%
alpha 3.42700 3.54600
beta  0.07118 0.08007
```

Source Listing

```
waiting <- faithful$waiting
x.m <- mean(waiting)

x <- waiting - x.m
y <- faithful$eruptions
n <- length(waiting)

data <- list("x", "y", "n")
params <- c("alpha", "beta", "mu")
inits <- function() {
    list(alpha=0, beta=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=5000)
all(out$summary[,"Rhat"] < 1.1)

# fitting the model
cbind(unlist(out$mean[
    c("alpha", "beta")]))

# credible intervals
out$summary[c("alpha", "beta"),
    c("2.5%", "97.5%")]
```

Note 1

The point estimates of the coefficients are almost identical to the classical linear parameters.

```
> faithful.lm <- lm(y ~ x)
> cbind(coefficients(faithful.lm))
                [,1]
(Intercept) 3.48778
x           0.07563
```

Note 2

We can improve robustness by using a Student t distribution with 2 degrees of freedom in place of the usual normal distribution. It will have better tolerance for outliers.

```
# Likelihood
for (i in 1:n) {
    y[i] ~ dt(mu[i], tau, 2)
    mu[i] <- alpha + beta*x[i]
}
```

26.2 Prediction of Simple Linear Regression

Now that we have found the coefficients of a simple linear regression model, we can extend the code to make predictions. It turns out all we need to do is to insert `NA` placeholders in the response data, and OpenBUGS will fill in the missing values.

Problem

Use the simple linear regression model of the data set `faithful` to estimate the eruption duration if the waiting time is 80 minutes. Find 95% credible intervals of the predicted mean and duration values.

Solution

We will reuse the same model as in previous discussion.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*x[i]
    }
}
```

In order to predict the duration, we must adjust the waiting time according the same offset as `x`. We denote the new predictor by `x0`.

```
x0 <- 80 - x.m
```

Then we define the data parameters. Note how we precede `x` with the adjusted predictor `x0`, and precede `y` with a `NA` placeholder.

```
n <- length(x)
```

```
data <- list(
    x=c(x0, x),
    y=c(NA, y),
    n=n+1)
params <- c("mu", "y")
inits <- function() {
    list(alpha=0, beta=0, tau=1)
}
```

After the simulation, we can retrieve a point estimate of the predicted duration value.

```
> out$mean$y
[1] 4.172
```

We then retrieve the 95% credible intervals of the predicted mean duration and predicted duration values.

```
> out$summary[c("mu[1]", "y[1]"),
+     c("2.5%", "97.5%")]

       2.5% 97.5%
mu[1] 4.106 4.248
y[1]  3.173 5.147
```

Source Listing

```
waiting <- faithful$waiting
x.m <- mean(waiting)

x <- waiting - x.m
y <- faithful$eruptions

x0 <- 80 - x.m
n <- length(x)
data <- list(
    x=c(x0, x),
    y=c(NA, y),
    n=n+1)
params <- c("mu", "y")
inits <- function() {
    list(alpha=0, beta=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
```

```
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=5000)
all(out$summary[,"Rhat"] < 1.1)

# prediction
out$mean$y
cbind(c(mu=out$mean$mu[1],
        y=out$mean$y))

# credible intervals
out$summary[c("mu[1]", "y[1]"),
    c("2.5%", "97.5%")]
```

Note

The result almost completely coincides with the classical linear model.

```
> faithful.lm <- lm(y ~ x)
> newdata <- data.frame(x=x0)

> predict(faithful.lm, newdata,
+     interval="confidence")
    fit   lwr   upr
1 4.176 4.105 4.248

> predict(faithful.lm, newdata,
+     interval="predict")
    fit   lwr   upr
1 4.176 3.196 5.156
```

26.3 Bayesian Standardized Residual

The standardized residual is the difference between observed and fitted values as normalized by the standard deviation of the error. It is an important measure of model fitness. For Bayesian regression, we can evaluate it directly within a BUGS model as follows.

```
    stdres[i] <- (y[i]-mu[i])/sigma
```

Problem

Create a simple linear regression model of the data set `faithful`. Then use the standardized residual and normal probability plots to evaluate model fitness.

Solution

We can insert the standardized residual `stdres` into our previous BUGS model.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)

    tau ~ dgamma(0.001, 0.001)
    sigma <- 1/sqrt(tau)

    for (i in 1:n) {
        # Likelihood
        y[i] ~ dnorm(mu[i], tau)
        mu[i] <- alpha + beta*x[i]

        # Derived
        stdres[i] <-
              (y[i]-mu[i])/sigma
    }
}
```

Then we prepare the data variables `x` and `y` in much the same way. We just need to add `stdres` to the monitor list.

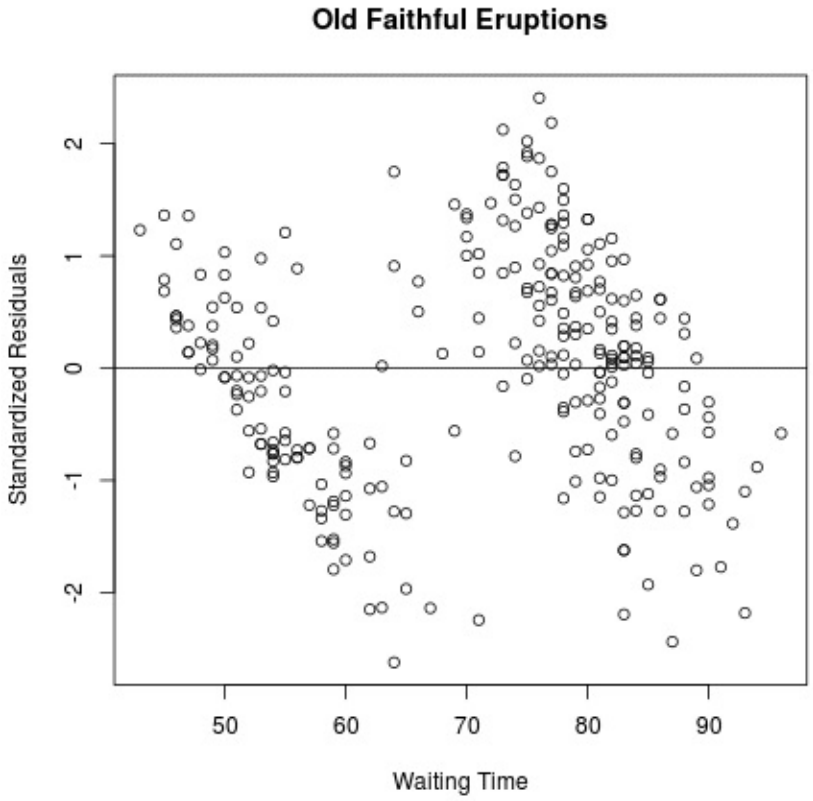
```
waiting <- faithful$waiting
x.m <- mean(waiting)

x <- waiting - x.m
y <- faithful$eruptions

n <- length(x)
data <- list("x", "y", "n")
params <- c("alpha", "beta",
    "mu", "stdres")
```

After the simulation, we can save the point estimates of the standardized residual in a vector `eruption.stdres`.

```
eruption.stdres <- out$mean$stdres
```

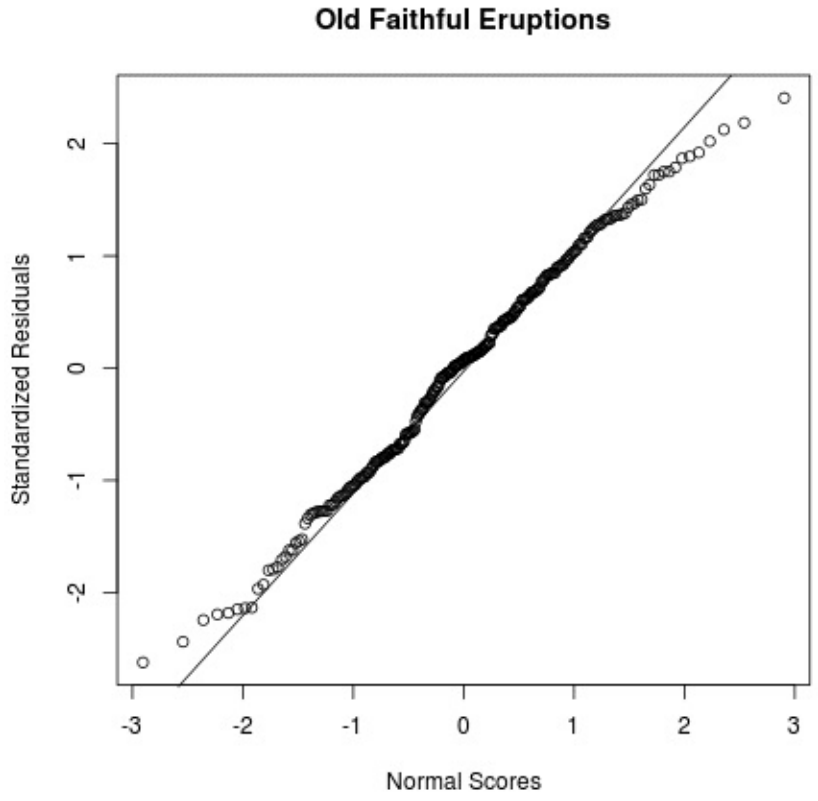
Then we can create the standardized residual plot. It shows the residual is fairly evenly distributed around the horizontal baseline.

```
> plot(faithful$waiting,
+     eruption.stdres,
+     ylab="Standardized Residuals",
+     xlab="Waiting Time",
+     main="Old Faithful Eruptions")
> abline(0, 0)     # the horizon
```



Lastly, we can create the normal probability plot. It suggests that the assumption of normally distributed error is mostly confirmed.

```
> qqnorm(eruption.stdres,
+     ylab="Standardized Residuals",
+     xlab="Normal Scores",
+     main="Old Faithful Eruptions")
> qqline(eruption.stdres)
```



Source Listing

```
waiting <- faithful$waiting
x.m <- mean(waiting)

x <- waiting - x.m
y <- faithful$eruptions

n <- length(x)
data <- list("x", "y", "n")
params <- c("alpha", "beta",
    "mu", "stdres")
inits <- function() {
    list(alpha=0, beta=0, tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=5000)
all(out$summary[,"Rhat"] < 1.1)
```

```
eruption.stdres <- out$mean$stdres

plot(faithful$waiting,
    eruption.stdres,
    ylab="Standardized Residuals",
    xlab="Waiting Time",
    main="Old Faithful Eruptions")
abline(0, 0)            # the horizon

qqnorm(eruption.stdres,
    ylab="Standardized Residuals",
    xlab="Normal Scores",
    main="Old Faithful Eruptions")
qqline(eruption.stdres)
```

Chapter 27
Bayesian Multiple Linear Regression

A multiple linear regression model describes a dependent variable *y* by a set of independent variables x_1, x_2, …, x_p ($p > 1$) in a linear relationship including an error term. We call the explanatory variables x_k ($k = 1, 2, \ldots, p$) to be *covariates*, and α, β_k ($k = 1, 2, \ldots, p$) are the regression coefficients.

$$y = \alpha + \sum_k \beta_k x_k + \epsilon$$

We will be using the data set `stackloss` for our discussion. Its multiple linear regression model is

$$Stack.Loss = \alpha + \beta_1 * Air.Flow + \beta_2 * Water.Temp + \beta_3 * Acid.Conc. + \epsilon$$

27.1 Coefficients of Multiple Linear Regression

In much the same way as simple linear regression, we can break up the multiple linear regression model into two components. In the following, the first equation is the *stochastic component* that denotes y as a normally distributed random variable around its expected value $E(y)$, and the second equation is the *deterministic component* that expresses $E(y)$ in a linear relationship with x_k (k = 1, 2, …, p). Note that the last term in the second equation is an inner product between two vectors $\{\beta_k\}_{k=1,\ldots,p}$ and $\{x_k\}_{k=1,\ldots,p}$.

$$\begin{aligned} y &= E(y) + \epsilon \\ E(y) &= \alpha + \sum_k \beta_k x_k \end{aligned}$$

Hence, we can write the following for a sequence of observations. We use the `inprod` function in BUGS to find the inner product of the `beta` coefficients with row vectors in `x` for each observation.

```
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], tau)

        mu[i] <- alpha +
            inprod(beta[], x[i, ])
    }
```

Problem

Create a multiple linear regression model for the data set `stackloss`. Then find point estimates and 95% credible intervals of the regression coefficients.

Solution

Based on the linear regression model, we define the BUGS model as follows.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    for (k in 1:p) {
```

```
        beta[k] ~ dnorm(0, 0.001)
    }
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], tau)

        mu[i] <- alpha +
            inprod(beta[], x[i, ])
    }
}
```

To create the data parameters, we extract a matrix `x` from the first three components of `stackloss`, and extract a response vector `y` from the last component.

```
x <- as.matrix(subset(stackloss,
    select=c(
         "Air.Flow",
         "Water.Temp",
         "Acid.Conc.")
    ))
y <- stackloss$stack.loss
```

It is important *to center and scale the covariates for a multiple linear regression*. In order to do so, we save the column means of `x` in a vector `x.m`, and column standard deviations in a vector `x.sd`. Incidentally, this will later affect the values of the regression coefficients.

```
x.m <- apply(x, 2, mean)
x.sd <- apply(x, 2, sd)
```

Now we can center and scale the columns of `x` using the method `sweep`.

```
x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/")
```

And we setup the data and initial parameters.

```
n <- nrow(x)
p <- ncol(x)

data <- list("x", "y", "n", "p")
params <- c("alpha", "beta", "mu")
inits <- function() {
```

```
    list(alpha=0, beta=numeric(p),
         tau=1)
}
```

After the simulation, we can retrieve point estimates of the linear regression coefficients.

```
> cbind(unlist(
+     out$mean[c("alpha", "beta")]))
         [,1]
alpha 17.5151
beta1  6.5180
beta2  4.1199
beta3 -0.7978
```

The 95% credible intervals are

```
> out$summary[1:4,
+     c("2.5%", "97.5%")]

          2.5%   97.5%
alpha   15.990 19.0300
beta[1]  3.941  9.0960
beta[2]  1.703  6.5410
beta[3] -2.575  0.9492
```

Source Listing

```
x <- as.matrix(subset(stackloss,
    select=c(
         "Air.Flow",
         "Water.Temp",
         "Acid.Conc.")
    ))
y <- stackloss$stack.loss

x.m <- apply(x, 2, mean); x.m
x.sd <- apply(x, 2, sd); x.sd

x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/"); x

n <- nrow(x)
p <- ncol(x)

data <- list("x", "y", "n", "p")
params <- c("alpha", "beta", "mu")
```

```
inits <- function() {
    list(alpha=0, beta=numeric(p),
         tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
     "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

# fitting
cbind(unlist(
     out$mean[c("alpha", "beta")]))

# credible intervals
out$summary[1:4,
    c("2.5%", "97.5%")]
```

Note 1

The point estimates of the regression coefficients are almost identical to the classical linear parameters.

```
> stackloss.lm <- lm(y ~ x)
> cbind(coefficients(stackloss.lm))
                [,1]
(Intercept) 17.5238
xAir.Flow    6.5612
xWater.Temp  4.0941
xAcid.Conc. -0.8152
```

Note 2

We can improve tolerance of outliers by using the Student t distribution with p+1 degrees of freedom in place of the usual normal distribution. At the time of writing, OpenBUGS does not support expression evaluation in function arguments, and we have to replace p+1 with a standalone variable.

```
    # Likelihood
    pp <- p+1
    for (i in 1:n) {
        y[i] ~ dt(mu[i], tau, pp)
        mu[i] <- alpha + beta*x[i]
```

```
}
```

27.2 Prediction of Multiple Linear Regression

Just as in Bayesian simple linear regression, to make predictions in Bayesian multiple linear regression, we just need to insert `NA` placeholders in the response data. We illustrate the procedure with the following.

Problem

Using the multiple linear regression model of the data set `stackloss`, predict the stack loss when the air flow is 72, water temperature is 20, and acid concentration is 85. Find also 95% credible intervals of the predicted mean and values.

Solution

We reuse the same model as in previous session.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    for (k in 1:p) {
        beta[k] ~ dnorm(0, 0.001)
    }
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], tau)

        mu[i] <- alpha +
            inprod(beta[], x[i, ])
    }
}
```

The prediction parameters are `c(72,20,85)`, which we need to adjust using the same scale as `x`. We denote it by a new variable `x0`.

```
x0 <- (c(72, 20, 85)-x.m)/x.sd
```

Then we define the data parameters by preceding `x` with `x0`, and `y` with `NA`. Since

`x0` is a vector, we use `rbind` to place it on top of `x`.

```
data <- list(
    x=rbind(x0, x),
    y=c(NA, y),
    p=p,
    n=n+1
    )
params <- c("mu", "y")
inits <- function() {
    list(alpha=0, beta=numeric(p),
         tau=1)
}
```

After the simulation, we can retrieve a point estimate of the predicted stack loss.

```
> out$mean$y
[1] 24.53
```

We can also retrieve the 95% credible intervals of the predicted mean and value.

```
> out$summary[c("mu[1]", "y[1]"),
+     c("2.5%", "97.5%")]
       2.5% 97.5%
mu[1] 20.17 28.84
y[1]  16.45 32.82
```

Source Listing

```
x <- as.matrix(subset(stackloss,
    select=c(
         "Air.Flow",
         "Water.Temp",
         "Acid.Conc.")
    ))
y <- stackloss$stack.loss

x.m <- apply(x, 2, mean); x.m
x.sd <- apply(x, 2, sd); x.sd

x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/"); x

n <- nrow(x)
p <- ncol(x)

x0 <- (c(72, 20, 85)-x.m)/x.sd
```

```
data <- list(
    x=rbind(x0, x),
    y=c(NA, y),
    p=p,
    n=n+1
    )
params <- c("mu", "y")
inits <- function() {
    list(alpha=0, beta=numeric(p),
         tau=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

# prediction
out$mean$y
cbind(c(mu=out$mean$mu[1],
         y=out$mean$y))

# credible intervals
out$summary[c("mu[1]", "y[1]"),
    c("2.5%", "97.5%")]
```

Note

The result is consistent with the classical linear model.

```
> stackloss.lm <- lm(stack.loss ~
+ Air.Flow + Water.Temp + Acid.Conc.,
+ data=stackloss)
> newdata <- data.frame(Air.Flow=72,
+   Water.Temp=20, Acid.Conc.=85)

> predict(stackloss.lm, newdata)
    1
24.58

> predict(stackloss.lm, newdata,
+     interval="confidence")
    fit   lwr   upr
1 24.58 20.22 28.94
```

```
> predict(stackloss.lm, newdata,
+     interval="predict")
    fit   lwr  upr
1 24.58 16.47 32.7
```

Chapter 28
Bayesian Logistic Regression

A logistic regression model describes the probability of a dichotomy variable y in terms of a set of independent variables x_1, x_2, …, x_p. If $E(y)$ is the expected value of y, then the logistic regression equation is

$$E(y) = 1/(1 + e^{-(\alpha + \sum_k \beta_k x_k)})$$

Using the logit function, we can rewrite the equation as follows.

$$logit(E(y)) = \alpha + \sum_k \beta_k x_k$$

28.1 Bernoulli Regression

A binomial experiment consists of a sequence of independent trials with dichotomy outcomes, which can be either success or failure. Each of such trial is called a Bernoulli trial. In BUGS, we describe Bernoulli outcomes with the distribution function `dbern`.

In analogous with the Bayesian multiple linear regression, we have the following Bayesian logistic regression model, where `y` is a vector of observed outcomes, and `prob` is the probability of success. As we have done previously, we use the `inprod` function to find the inner product of the `beta` coefficients with row vectors in `x` for each observation.

```
    for (i in 1:n) {
        y[i] ~ dbern(prob[i])
        logit(prob[i]) <- alpha +
            inprod(beta[], x[i, ])
    }
```

Problem

The data set `mtcars` consists of a few 1974 vehicles data. It includes the transmission type (`am`, 0 = automatic), weight (`wt`, lbs/1000), and engine horsepower (`hp`) information. Establish a logistic regression model of the vehicle transmission type based on its weight and engine horsepower. Find point estimates and 95% credible intervals of the regression coefficients. If a vehicle has a 120hp engine and weights 2800 lbs, give a point estimate and 95% credible interval of the probability that the vehicle is fitted with manual transmission.

Solution

We view the presence of manual transmission in a vehicle as the outcome of a Bernoulli trial. Therefore we have the following.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
```

```
    for (k in 1:p) {
        beta[k] ~ dnorm(0, 0.001)
    }

    # Likelihood
    for (i in 1:n) {
        y[i] ~ dbern(prob[i])
        logit(prob[i]) <- alpha +
            inprod(beta[], x[i, ])
    }
}
```

We then extract a matrix `x` from the `hp` and `wt` components of `mtcars`, and a vector `y` from the `am` component.

```
x <- as.matrix(subset(mtcars,
        select=c("hp", "wt")))
y <- mtcars$am
```

We center and scale columns of `x` with the column means and standard deviations.

```
x.m <- apply(x, 2, mean)
x.sd <- apply(x, 2, sd)

x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/")
```

Finally, we use the adjusted predictor `x0` for the data parameters.

```
n <- nrow(x)
p <- ncol(x)

x0 <- (c(120, 2.8)-x.m)/x.sd
data <- list(
    x=rbind(x0, x),
    y=c(NA, y),
    p=p,
    n=n+1)
params <- c("alpha", "beta",
    "prob", "y")
inits <- function() {
    list(alpha=0, beta=numeric(p))
}
```

After the simulation, we can retrieve the point estimate and 95% credible intervals of the regression coefficients.

```
> cbind(unlist(out$mean)[1:3])
         [,1]
alpha  -2.547
beta1   3.677
beta2 -11.228

> out$summary[1:3,
+     c("2.5%", "97.5%")]
           2.5%   97.5%
alpha    -5.590 -0.3145
beta[1]   1.219  7.2700
beta[2] -19.920 -5.0739
```

We can also retrieve corresponding statistics for the probability of manual transmission in the given vehicle.

```
> cbind(unlist(out$mean$prob[1]))
      [,1]
[1,] 0.651

> out$summary["prob[1]",
+     c("2.5%", "97.5%")]
  2.5%  97.5%
0.2182 0.9658
```

Source Listing

```
x <- as.matrix(subset(mtcars,
        select=c("hp", "wt")))
y <- mtcars$am

x.m <- apply(x, 2, mean); x.m
x.sd <- apply(x, 2, sd); x.sd

x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/"); x

n <- nrow(x)
p <- ncol(x)

x0 <- (c(120, 2.8)-x.m)/x.sd
data <- list(
    x=rbind(x0, x),
    y=c(NA, y),
    p=p,
    n=n+1)
params <- c("alpha", "beta",
     "prob", "y")
```

```
inits <- function() {
    list(alpha=0, beta=numeric(p))
}

library(R2OpenBUGS)
model.file <- file.path(
    tempdir(), "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

# regression coefficients
cbind(unlist(out$mean)[1:3])

out$summary[1:3,
    c("2.5%", "97.5%")]

# predicted probability
cbind(unlist(out$mean$prob[1]))

out$summary["prob[1]",
    c("2.5%", "97.5%")]
```

Note

The estimated probability is fairly similar to the result of the classical `glm` function.

```
> am.glm <- glm(formula=am ~ hp+wt,
+     data=mtcars, family=binomial)
> newdata <-
+     data.frame(hp=120, wt=2.8)
> predict(am.glm, newdata,
+     type="response")
     1
0.6418
```

28.2 Binomial Regression

In our previous discussion of the logistic model, we require observations of individual Bernoulli trials, which maybe infeasible. Instead, we may only have aggregated data summary to work with.

Example

Consider an experiment of fungicide treatment on oak trees. The effectiveness of the fungicide depends on the log concentration levels of the fungicide solution applied in the treatments.

For instance, the first row of the summary table below shows that when the log concentration is -1.2198, 11 out of 15 treated oak trees are infected. However, if we raise the log concentration to 0.7695 as in the last row of the table, only one of 16 treated oak tree is infected.

```
       x  y  n
-1.2198 11 15
-0.5371  8 20
-0.3064  3 12
 0.1578  2 18
 0.7695  1 16
```

We can view the number of infected oak trees as a binomial random variable, and describe it with the binomial function `dbin` in BUGS. The logistic regression thus becomes

```
for (i in 1:len) {
    y[i] ~ dbin(prob[i], n[i])
    logit(prob[i]) <-
        inprod(beta[], x[i, ])
}
```

Problem

Establish a logistic regression model for the fungicide experiment. Find a point estimate and 95% credible interval of infection probability if we apply the

fungicide at -0.40 log concentration level to 24 oak trees.

Solution

We see the number of infected oak trees as a binomial random variable. Since the log concentration is the only covariate, we have the following simple logistic regression model.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)

    # Likelihood
    for (i in 1:len) {
        y[i] ~ dbin(prob[i], n[i])
        logit(prob[i]) <-
            alpha + beta*(x[i])
    }
}
```

We save the given data as follows.

```
x <- c(-1.2198, -0.5371, -0.3064,
         0.1578, 0.7695)
y <- c(11, 8, 3, 2, 1)
n <- c(15, 20, 12, 18, 16)
```

We then precede `x` with the prediction parameter -0.40, and adjust it with the mean of covariates for data input. Similarly, we precede `y` with an `NA` for placeholder.

```
x.m <- mean(x)
data <- list(
    x=c(-0.40, x)-x.m,
    y=c(NA, y),
    n=c(24, n),
    len=length(x)+1)
params <- c("alpha", "beta",
     "prob", "y")
inits <- function() {
    list(alpha=0, beta=1)
}
```

After the simulation, we can retrieve the point estimate and credible intervals of infection probability.

```
> out$mean$prob[1]
[1] 0.32

> out$summary["prob[1]",
+     c("2.5%", "97.5%")]
  2.5%  97.5%
0.2065 0.4460
```

Source Listing

```
x <- c(-1.2198, -0.5371, -0.3064,
         0.1578, 0.7695)
y <- c(11, 8, 3, 2, 1)
n <- c(15, 20, 12, 18, 16)

x.m <- mean(x)
data <- list(
    x=c(-0.40, x)-x.m,
    y=c(NA, y),
    n=c(24, n),
    len=length(x)+1)
params <- c("alpha", "beta",
     "prob", "y")
inits <- function() {
    list(alpha=0, beta=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

out$mean$prob[1]
out$summary["prob[1]",
    c("2.5%", "97.5%")]
```

28.3 Reverse Logistic Regression

A useful application of logistic regression is to find out the needed covariate value for a desired probability outcome. For example, in the fungicide experiment in last section, it would be useful to find out just the log concentration necessary to protect 90% of the oak trees.

For this purpose, we denote the desired probability of success by `prob0`, Then we can relate it with the needed covariate value `x0` in a simple logistic regression model as follows.

```
    # Derived
    x0 <- (logit(prob0) - alpha)/beta
```

Problem

Recall the fungicide experiment from last section. Give a point estimate and 95% credible interval of fungicide log concentration necessary to protect 90% of the oak trees from infection.

Solution

We modify the previous BUGS model so as to allow the extra parameters `prob0` and `x0`.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)

    # Likelihood
    for (i in 1:len) {
        y[i] ~ dbin(prob[i], n[i])
        logit(prob[i]) <-
            alpha + beta*(x[i])
    }

    # Derived
    x0 <- (logit(prob0) - alpha)/beta
```

```
}
```

Since we would like to have 90% of the oak tree protected, we set the allowed infection rate to be 10%. Hence we can setup the data parameters as follows.

```
x.m <- mean(x)
data <- list(
    prob0=0.10,
    x=x-x.m,
    y=y,
    n=n,
    len=length(x))
params <- c("alpha", "beta",
     "prob", "x0")
inits <- function() {
    list(alpha=0, beta=1)
}
```

After the simulation, we can retrieve the point estimate and credible intervals of `x0`. We have to restore the offset `x.m` at the end.

```
> x.m + out$mean$x0
[1] 0.3106

> x.m + out$summary["x0",
+     c("2.5%", "97.5%")]
    2.5%    97.5%
-0.09334  0.94482
```

Source Listing

```
x <- c(-1.2198, -0.5371, -0.3064,
         0.1578, 0.7695)
y <- c(11, 8, 3, 2, 1)
n <- c(15, 20, 12, 18, 16)

x.m <- mean(x)
data <- list(
    prob0=0.10,
    x=x-x.m,
    y=y,
    n=n,
    len=length(x))
params <- c("alpha", "beta",
     "prob", "x0")
inits <- function() {
    list(alpha=0, beta=1)
```

```
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
    model.file, n.iter=10000)
all(out$summary[,"Rhat"] < 1.1)

x.m + out$mean$x0

x.m + out$summary["x0",
    c("2.5%", "97.5%")]
```

Chapter 29
Hierarchical Models

In our discussion of Bayesian inferences and regressions, we have successfully developed a series of basic BUGS models. We will now demonstrate how to use the basic models to build more advanced ones, and use CODA to properly confirm MCMC simulation convergence.

29.1 Ridge Regression

In a multiple linear regression model, if one of the covariates is correlated with other covariates, the outcome may exhibit irregularities. A technique to prevent such effect is the **ridge regression**, which penalizes large estimates by including the slope parameters in the total regression residual.

The implementation of the ridge regression in OpenBUGS is simple. All we have to do is to model the slope parameters `beta` as a random effect with zero expected value. More specifically, we fit the prior of `beta` with a normal distribution $N(0, \sigma^2)$ as follows.

```
    for (k in 1:p) {
        # ridge regression
        beta[k] ~ dnorm(0, phi)
    }
```

Problem

Implement ridge regression for the previous Bayesian multiple linear regression model of the data set `stackloss`. Then find point estimates and 95% credible intervals of the regression coefficients.

Solution

Using a normal distribution with zero mean as a prior for the `beta` parameters, we have the following.

```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    for (k in 1:p) {
        # ridge regression
        beta[k] ~ dnorm(0, phi)
    }
    tau ~ dgamma(0.001, 0.001)
    phi ~ dgamma(0.001, 0.001)

    # Likelihood
```

```
    for (i in 1:n) {
        y[i] ~ dnorm(mu[i], tau)

        mu[i] <- alpha +
            inprod(beta[], x[i, ])
    }

}
```

The data parameters setup is the same as before. We just need to initialize the extra precision parameter `phi`.

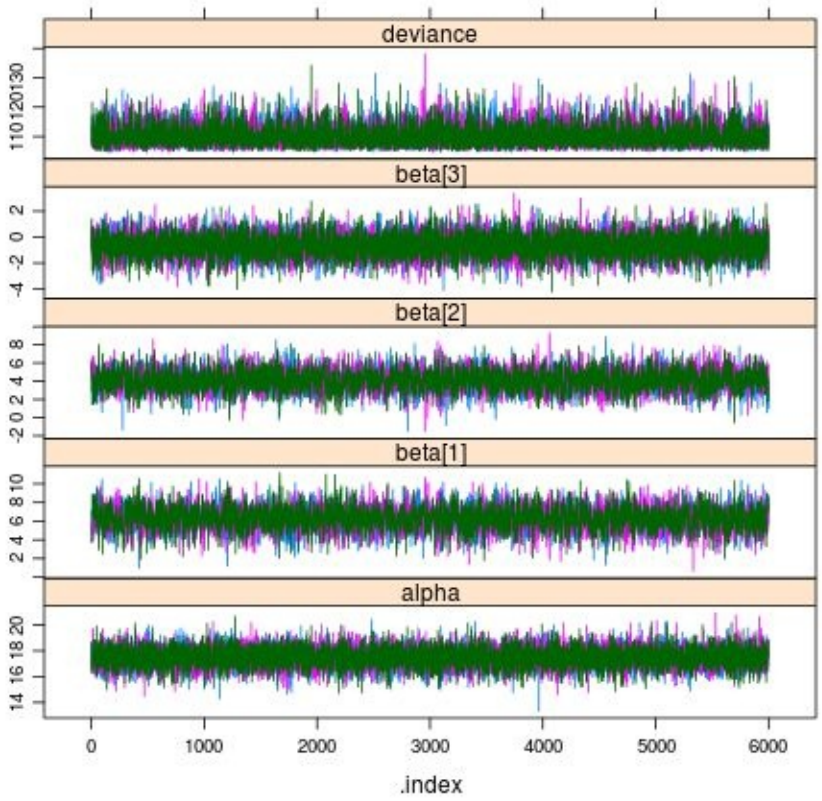
```
inits <- function() {
    list(alpha=0, beta=numeric(p),
         tau=1, phi=1)
}
```

Since we will be using CODA for convergence analysis, we have to enable the `codaPkg` option in the `bugs` method. Then we can extract the simulation output using `read.bugs`.

```
out <- bugs(data, inits, params,
     model.file, codaPkg=TRUE,
     n.iter=12000)
out.coda <- read.bugs(out)
```

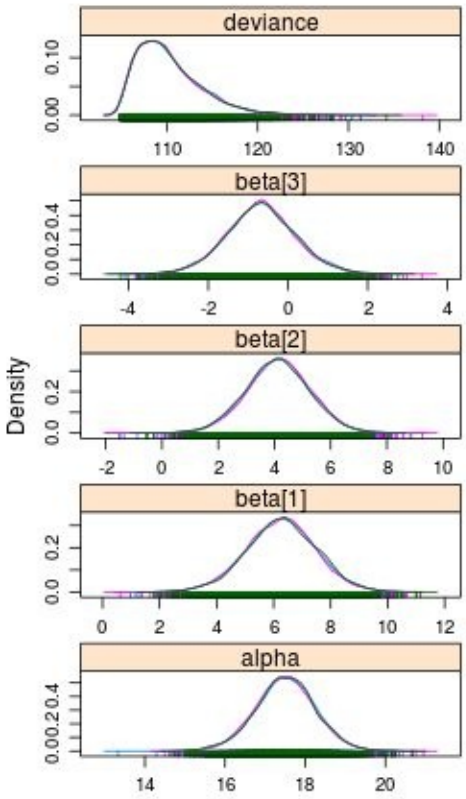
To check for convergence, we create an `xyplot` from the `coda` extension. It gives a trace of the simulation values during the iteration cycles. With lack of obvious upward or downward trends, the trace plot below suggests simulation convergence.

```
> library(coda)
> xyplot(out.coda)
```



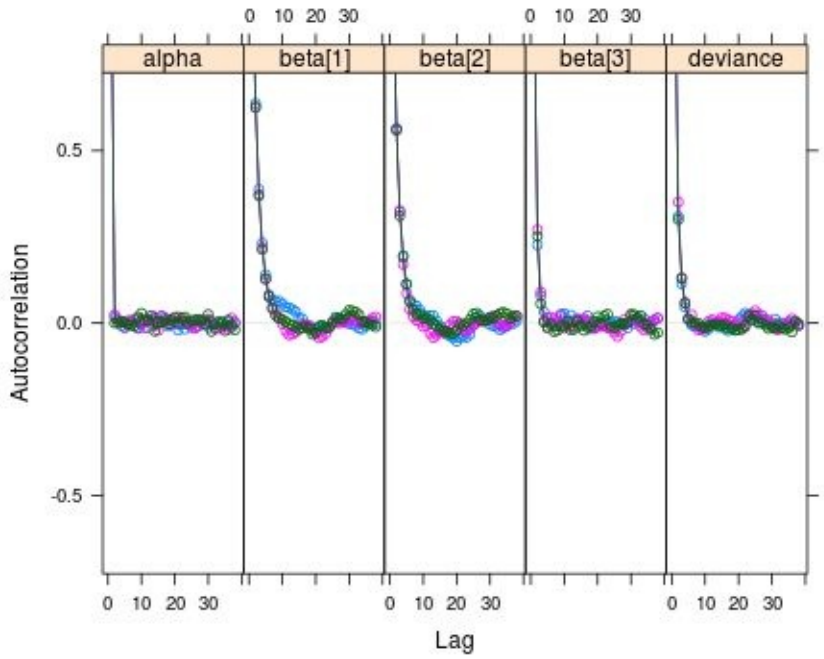
Next, we should see if the density plots have well-defined peaks, which are indeed the case here.

```
> densityplot(out.coda)
```



In addition, we should use `acfplot` to see if the auto-correlation of the time series converge to zero.

```
> acfplot(out.coda)
```

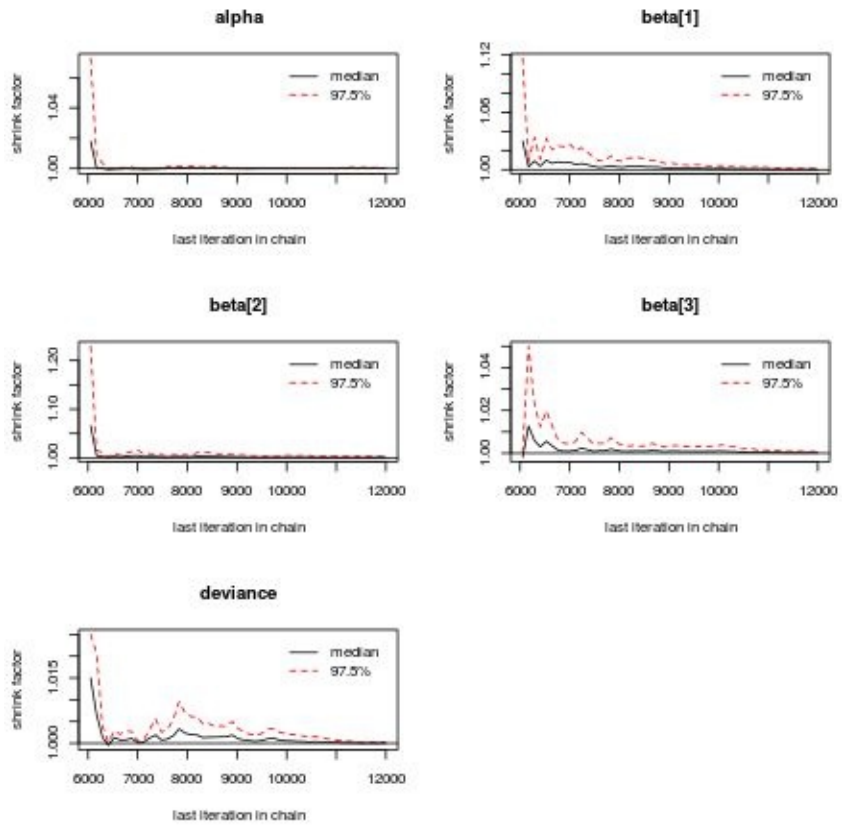


Lastly, we should perform the Gelman-Rubin convergence diagnostic with `gelman.diag`. The shrink factors should be below 1.05.

```
> gelman.diag(out.coda)
Potential scale reduction factors:

         Point est. Upper C.I.
alpha             1          1
beta[1]           1          1
beta[2]           1          1
beta[3]           1          1
deviance          1          1

Multivariate psrf

1
```

Actually, it is prudent to create the Gelman-Rubin-Brooks plot for visual confirmation of the shrink factor convergence as well.

```
> gelman.plot(out.coda)
```



As the convergence is reassured, we can print out the point estimates and 95% credible intervals of the regression coefficients.

```
> out.summary <- summary(out.coda,
+     q=c(0.025, 0.975))

> out.summary$stat[1:4,
+     "Mean", drop=FALSE]
           Mean
alpha     17.5073
beta[1]    6.2665
beta[2]    4.1354
beta[3]   -0.6683

> out.summary$q[1:4, ]
           2.5%   97.5%
alpha     16.000  18.980
beta[1]    3.749   8.717
beta[2]    1.858   6.388
beta[3]   -2.393   1.123
```

For further documentation, we can use the following.

```
> library(help=coda)
> help(summary.mcmc)
> help(xyplot.mcmc)
```

Source Listing

```
x <- as.matrix(subset(stackloss,
    select=c(
         "Air.Flow",
         "Water.Temp",
         "Acid.Conc.")
    ))
y <- stackloss$stack.loss

x.m <- apply(x, 2, mean); x.m
x.sd <- apply(x, 2, sd); x.sd

x <- sweep(x, 2, x.m)
x <- sweep(x, 2, x.sd, F="/"); x

n <- nrow(x)
p <- ncol(x)

data <- list("x", "y", "n", "p")
params <- c("alpha", "beta")
inits <- function() {
    list(alpha=0, beta=numeric(p),
    tau=1, phi=1)
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
    "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
     model.file, codaPkg=TRUE,
     n.iter=12000)
out.coda <- read.bugs(out)

library(coda)
xyplot(out.coda)
densityplot(out.coda)
acfplot(out.coda)

gelman.diag(out.coda)
gelman.plot(out.coda)

out.summary <- summary(out.coda,
    q=c(0.025, 0.975))
```

```
out.summary$stat[1:4,
    "Mean", drop=FALSE]
out.summary$q[1:4, ]
```

29.2 Logit Probability

In a logistic regression model, we assume the logit probability to be linearly dependent on some regression covariates. However, in absence of covariates, we can assume the logit probability to be just a random effect. Denote the logit probability variables by a vector `x`, and we can modify the logistic regression model as follows.

```
    for (i in 1:k) {
        y[i] ~ dbin(prob[i], n[i])
        logit(prob[i]) <- x[i]
        x[i] ~ dnorm(mu, tau)
    }
```

Problem

The Surgical example of OpenBUGS contains mortality data of baby heart surgeries in 12 hospitals. Assume the logit probability of surgical death to be a normally distributed random effect. Find a point estimate and 95% credible interval of the expected surgical mortality rate.

Solution

We can relate the logit of the expected mortality rate `pbar` with the mean of the random effect `mu` via a logical expression.

```
    # Derived
    logit(pbar) <- mu
```

The model then becomes

```
model <- function() {
    # Priors
    mu ~ dnorm(0, 0.001)
    tau ~ dgamma(0.001, 0.001)

    # Likelihood
    for (i in 1:k) {
        y[i] ~ dbin(prob[i], n[i])
```

```
        logit(prob[i]) <- x[i]
        x[i] ~ dnorm(mu, tau)
    }

    # Derived
    logit(pbar) <- mu
}
```

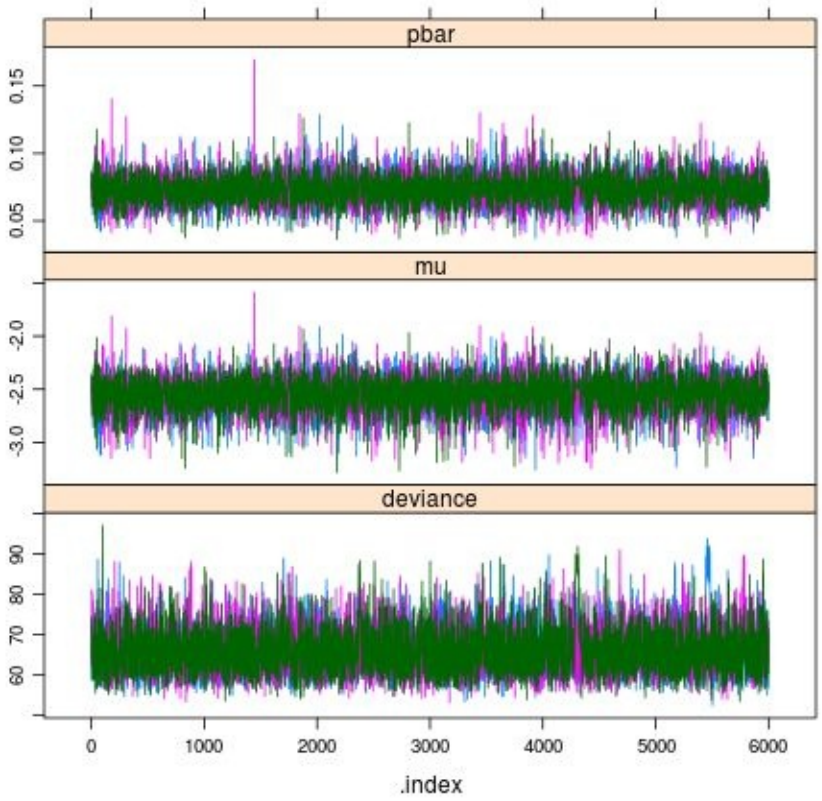
Denote the number of surgical deaths by `y`, and the number of operations by `n`. From the data in the Surgical example of OpenBUGS, we have

```
y <- c( 0, 18,  8, 46,  8, 13,
        9, 31, 14,  8, 29, 24)
n <- c( 47, 148, 119, 810, 211, 196,
       148, 215, 207,  97, 256, 360)
```

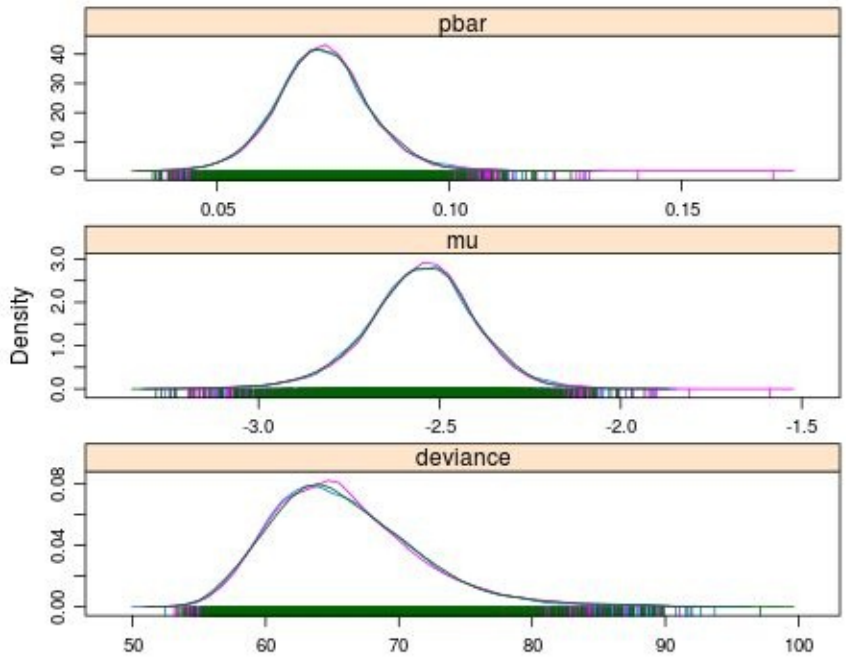
Then we define the data and initialization parameters.

```
k <- 12
data <- list("y", "n", "k")
params <- c("mu", "pbar")
inits <- function() {
    list(mu=0, tau=1, x=numeric(k))
}
```

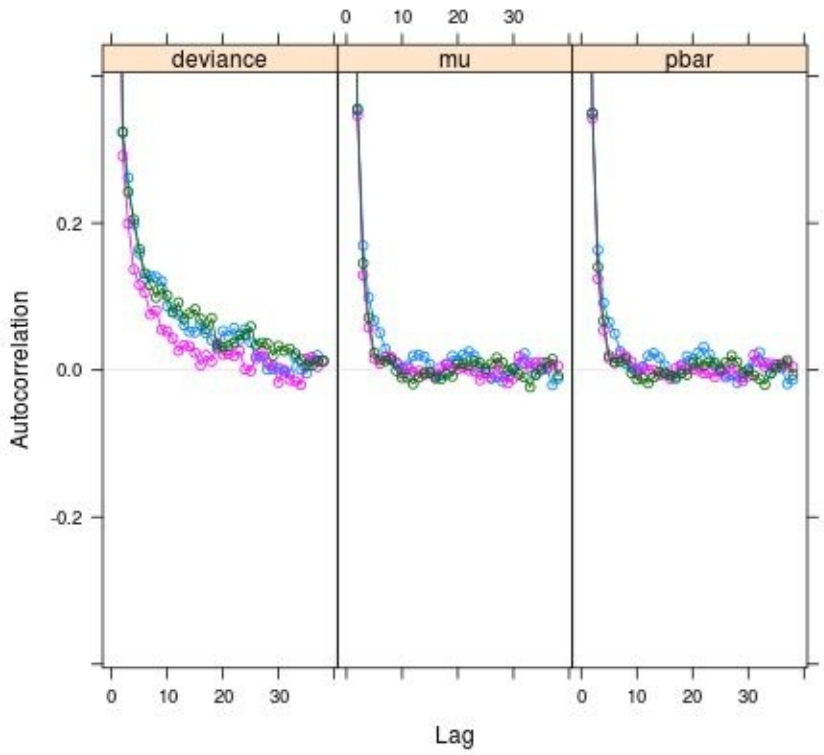
After the simulation, the `xyplot` shows no obvious upward or downward trends in the trace plot.



And we see that the following density plots have well-defined peaks.



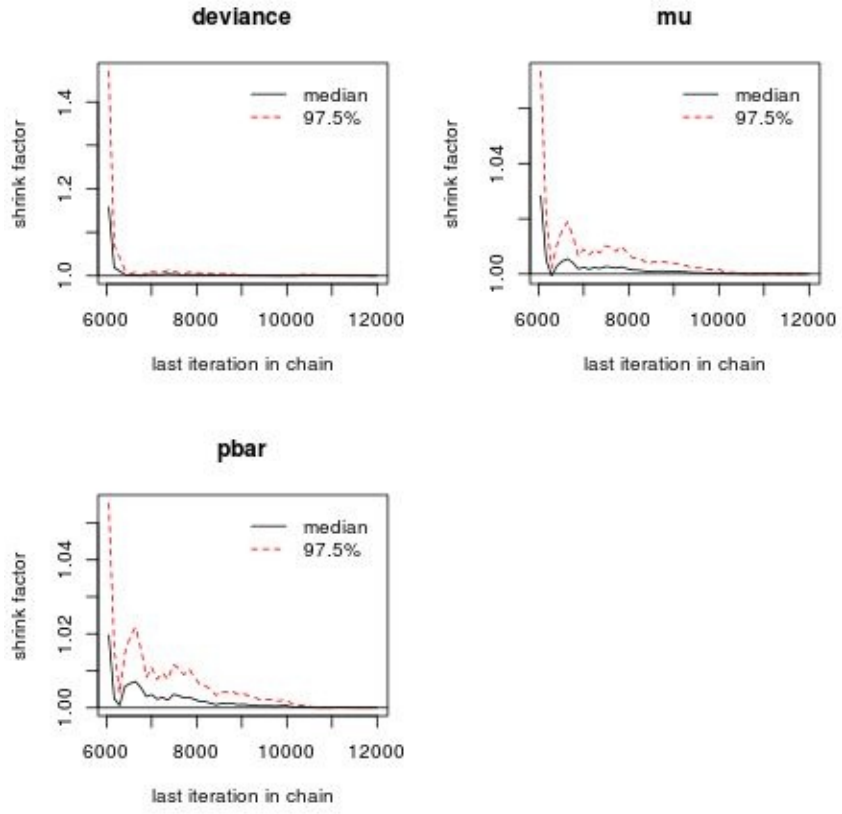
In addition, the `acfplot` shows that the auto-correlation converges to zero.



As for the Gelman-Rubin diagnostic, it shows that the shrink factors are all below 1.05.

```
> gelman.diag(out.coda)
Potential scale reduction factors:

         Point est. Upper C.I.
deviance          1          1
mu                1          1
pbar              1          1

Multivariate psrf

1
```

And the Gelman-Rubin-Brooks plot as below shows that the shrink factors all converge to 1.



Assured convergence, we retrieve the expected mortality rate and its 95% credible interval.

```
> out.summary <- summary(out.coda,
+     q=c(0.025, 0.975))
> out.summary$stat["pbar",
+     "Mean", drop=FALSE]
       Mean
pbar 0.07292

> out.summary$q["pbar", ]
   2.5%   97.5%
0.05332 0.09401
```

Source Listing

```
y <- c( 0, 18,  8, 46,  8, 13,
        9, 31, 14,  8, 29, 24)
n <- c( 47, 148, 119, 810, 211, 196,
       148, 215, 207,  97, 256, 360)

k <- 12
```

```
data <- list("y", "n", "k")
params <- c("mu", "pbar")
inits <- function() {
    list(mu=0, tau=1, x=numeric(k))
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
     "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
     model.file, codaPkg=TRUE,
     n.iter=12000)
out.coda <- read.bugs(out)

library(coda)
xyplot(out.coda)
densityplot(out.coda)
acfplot(out.coda)

gelman.diag(out.coda)
gelman.plot(out.coda)

out.summary <- summary(out.coda,
    q=c(0.025, 0.975))
out.summary$stat["pbar",
    "Mean", drop=FALSE]
out.summary$q["pbar", ]
```

29.3 Measurement Intervals

In a regression model, we always assume the covariates to have precise values, such as the log concentrations in the fungicide experiment of the previous chapter. This may not be the case if we have only the information of approximate measurement intervals.

For instance, in the Air example of OpenBUGS, we only know that the NO2 (nitrogen dioxide) concentration level falls into 3 categories: less than 20 ppb, 20 to 40 ppb or over 40 ppb. The task is to decide if we can relate the number of children respiratory illness to the NO2 concentration levels.

Denote the NO2 concentration covariates by `x`, and the number of children respiratory illness by `y`, then we have the following binomial logistic model.

```
for (i in 1:k) {
    # logistic regression
    y[i] ~ dbin(prob[i], n[i])
    logit(prob[i]) <-
         alpha + beta*x[i]
}
```

However, since we only have the knowledge of the measurement intervals, the NO2 concentration covariates are not precise values, but are random variables. Past study of the air quality measurement shows that a plausible model of the NO2 covariates is the following simple linear model.

```
for (i in 1:k) {
    # linear regression
    x[i] ~ dnorm(mu[i], 0.01234)
    mu[i] <- 4.48 + 0.76*z[i]
}
```

Hence we can cascade the logistic and linear models into the following Air Quality model.

```
for (i in 1:k) {
    # logistic regression
    y[i] ~ dbin(prob[i], n[i])
```

```
        logit(prob[i]) <-
             alpha + beta*x[i]

        # linear regression
        x[i] ~ dnorm(mu[i], 0.01234)
        mu[i] <- 4.48 + 0.76*z[i]
    }
```

Problem

In the Air Quality model, find point estimates and 95% credible intervals of the logistic regression parameters `alpha` and `beta`.

Solution

Assigning the usual vague priors, we have the following.

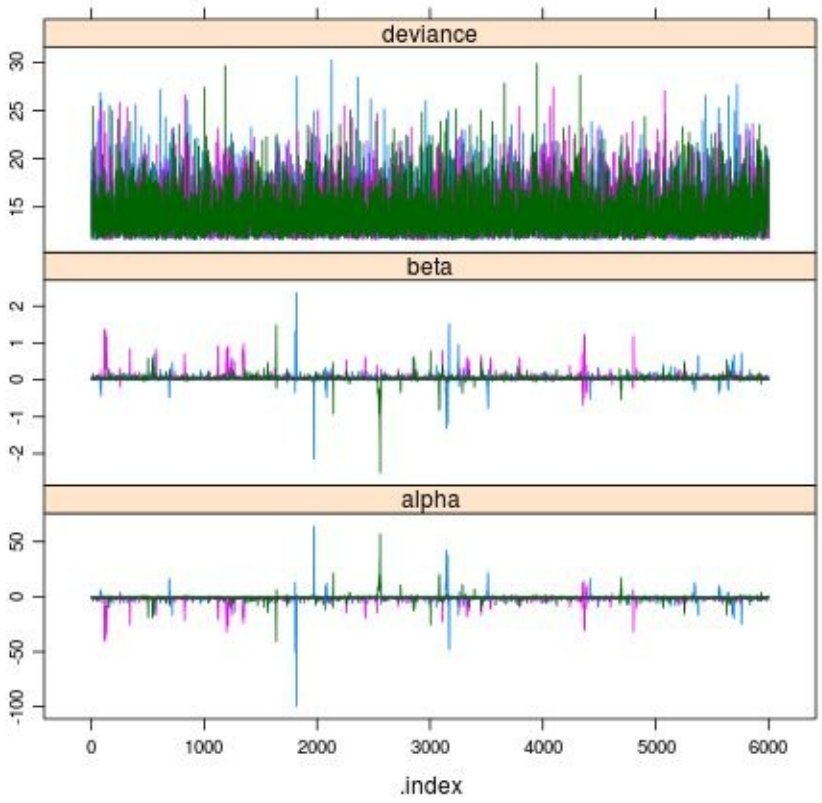
```
model <- function() {
    # Priors
    alpha ~ dnorm(0, 0.001)
    beta ~ dnorm(0, 0.001)

    for (i in 1:k) {
        # logistic regression
        y[i] ~ dbin(prob[i], n[i])
        logit(prob[i]) <-
             alpha + beta*x[i]

        # linear regression
        x[i] ~ dnorm(mu[i], 0.01234)
        mu[i] <- 4.48 + 0.76*z[i]

    }
}
```

With the given data in the Air example of OpenBUGS, we can define the data and initialization parameters.

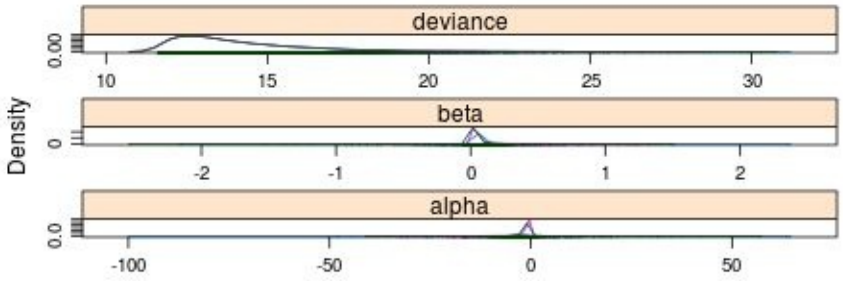
```
z <- c(10, 30, 50)
y <- c(21, 20, 15)
n <- c(48, 34, 21)

k <- length(n)
data <- list("z", "y", "n", "k")
params <- c("alpha", "beta")
inits <- function() {
```

```
    list(alpha=0, beta=0, x=numeric(k))
}
```

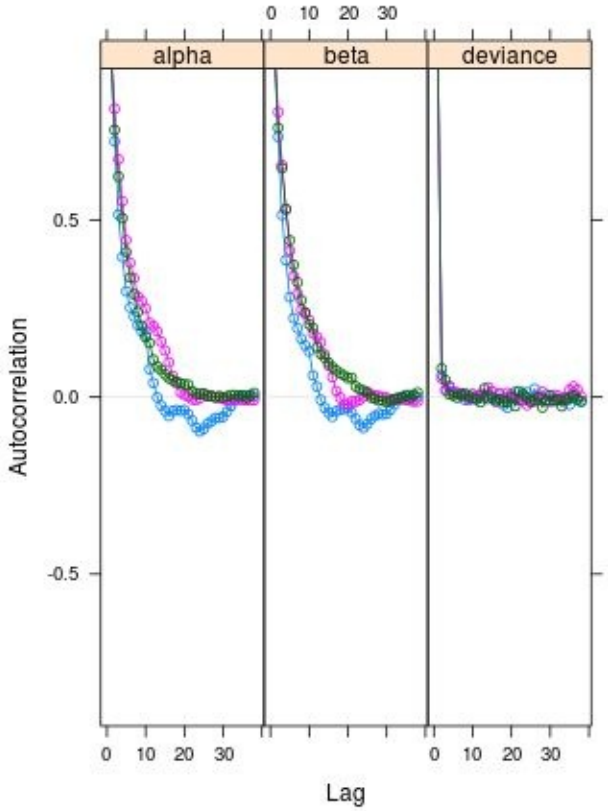
After the simulation, we find no obvious trends in the `xyplot`.



And the density plots below have well-defined peaks.



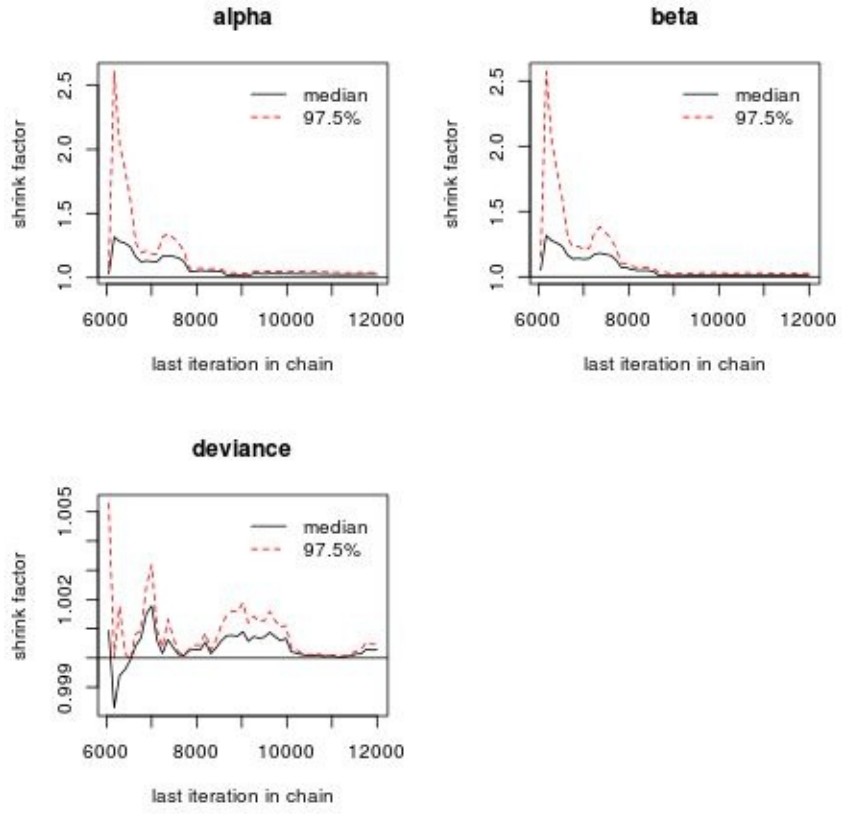
In addition, the `acfplot` shows that the auto-correlation converges to zero.



And the following shows that the shrink factors are all below 1.05 as they should be.

```
> gelman.diag(out.coda)
Potential scale reduction factors:

          Point est. Upper C.I.
alpha           1.03       1.04
beta            1.01       1.02
deviance        1.00       1.00

Multivariate psrf

1
```

Finally, the Gelman-Rubin-Brooks plot confirms the shrink factors convergence.



Assured convergence, we retrieve point estimates of the logistic regression parameters and the 95% credible intervals. Note that the latter fails to exclude the null value.

```
> out.summary <- summary(out.coda,
+     q=c(0.025, 0.975))
> out.summary$stat[params,
+     "Mean", drop=FALSE]
          Mean
alpha -1.07826
beta   0.05132

> out.summary$q[params, ]
           2.5%  97.5%
alpha -5.709050 0.4338
beta  -0.006202 0.2161
```

Source Listing

```
z <- c(10, 30, 50)
y <- c(21, 20, 15)
n <- c(48, 34, 21)
```

```
k <- length(n)
data <- list("z", "y", "n", "k")
params <- c("alpha", "beta")
inits <- function() {
    list(alpha=0, beta=0,
          x=numeric(k))
}

library(R2OpenBUGS)
model.file <- file.path(tempdir(),
     "model.txt")
write.model(model, model.file)
out <- bugs(data, inits, params,
     model.file, codaPkg=TRUE,
     n.iter=12000)
out.coda <- read.bugs(out)

library(coda)
xyplot(out.coda)
densityplot(out.coda)
acfplot(out.coda)

gelman.diag(out.coda)
gelman.plot(out.coda)

out.summary <- summary(out.coda,
    q=c(0.025, 0.975))
out.summary$stat[params,
    "Mean", drop=FALSE]
out.summary$q[params, ]
```

Part IV
GPU Computing with R

Statistics is computationally intensive. Routine statistical tasks such as data extraction, graphical summary, and technical interpretation all require pervasive use of modern computing machinery. Obviously, these tasks can benefit greatly from a parallel computing environment where extensive calculations can be performed simultaneously.

Recent advances in consumer computer hardware makes parallel computing capability widely available to most users. In fact, the mundane video graphics cards in many PCs nowadays support parallel computing operations besides the routine graphical functions. Applications that make effective use of the so-called **graphics processing units** (GPU) have reported significant performance gains.

Although *r-tutor.com* is dedicated to elementary statistics with R, it is evident that parallel computing will be of tremendous importance in the near future, and it is imperative for students to be acquainted with the new technology as soon as possible. Thus we are opening a new series of articles on the subject. Students will be able to tackle problems previously deemed impractical due to resource constraints.

We begin with selecting a GPU computing platform. One of the most affordable options available is NVIDIA's CUDA. Even with its most inexpensive entry level equipment, there are dozens of processing cores for parallel computing. Therefore, our GPU computing tutorials will be based on CUDA for now. Incidentally, the CUDA programming interface is vector oriented, and fits perfectly with the R language paradigm.

We will not deal with CUDA directly or its advanced C/C++ interface. Instead, we will rely on **`rpud`** and other R packages for studying GPU computing. We will compare the performance of GPU functions with their regular R counterparts and verify the performance advantage. The GPU package `rpud` and its add-on `rpudplus` can be found in r-tutor.com.

As a disclaimer, *r-tutor.com* is not affiliated with NVIDIA in any way. I am open to discussions about other GPU computing platforms. Nevertheless, included are tutorials on how to install the CUDA Toolkit on Fedora and Ubuntu Linux. You may modify the instruction for your favorite environment. Please verify your CUDA hardware compatibility before proceed.

Chapter 30
Statistical Computing on GPU

The NVIDIA GPU is based on the SIMD (Single Instruction, Multiple Data) architecture. Its performance is sensitive to memory alignment and coalesced memory access pattern. Hence a major challenge of CUDA programmers is to understand GPU hardware characteristics and constraints. They must carefully fine tune existing algorithms or invent entirely new software methodology for maximal speed gain.

The following discussions represent our attempts to perform statistical computing on the GPU. It is an ongoing effort, and more results will come along in the near future.

30.1 Distance Matrix

The measure of **distance** is an important tool in statistical analysis, as it quantifies **dissimilarity** within sample data for numerical computation. A popular choice of distance metric is the Euclidean distance, which is the square root of the sum of squares of attribute differences. In particular, for two data points *x* and *y* with *n* numerical attributes, the Euclidean distance between them is:

$$d(x,\ y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

For example, the data frame `mtcars` consists of measurements from a collection of 32 automobiles. Since there are 11 measurement attributes for each automobile, the data set can be seen as a collection of 32 sample vectors in an 11 dimensional space. To find out the dissimilarity between two automobiles, say Honda Civic and Camaro Z28, we can calculate the distance between them with the function `dist`.

```
> x <- mtcars["Honda Civic",]
> y <- mtcars["Camaro Z28",]
> dist(rbind(x, y))
           Honda Civic
Camaro Z28      335.89
```

Likewise, we can compute the distance between Camaro Z28 and Pontiac Firebird:

```
> z <- mtcars["Pontiac Firebird",]
> dist(rbind(y, z))
                 Camaro Z28
Pontiac Firebird     86.267
```

As the distance between Camaro Z28 and Pontiac Firebird (86.267) is smaller than the distance between Camaro Z28 and Honda Civic (335.89), we conclude that Camaro Z28 is more similar to Pontiac Firebird than to Honda Civic.

If we apply the same distance computation between all possible pairs of

automobiles in `mtcars`, and arrange the result into a 32x32 symmetric matrix, with the element at the *i*-th row and *j*-th column being the distance between the *i*-th and *j*-th automobiles in the data set, we will have the so-called **distance matrix**. It can be computed for `mtcars` as:

```
> dist(as.matrix(mtcars))
```

In general, for a data sample of size *M*, the distance matrix is an *M*×*M* symmetric matrix with *M*×(*M*-1)/2 distinct elements. Hence for a data sample of size 4,500, its distance matrix has about ten million distinct elements. Occasionally, depending on your application, a sample size of 4,500 may still be too small.

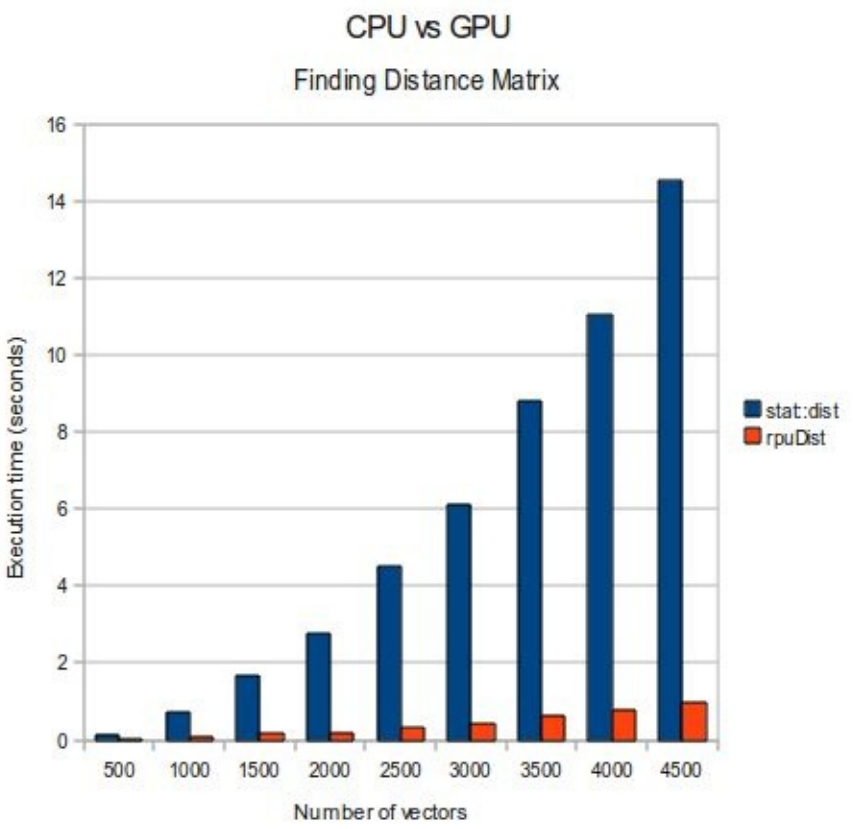
The following measures the time spent on finding the distance matrix of a collection of 4,500 random vectors in a 120 dimension space. On a desktop computer with AMD Phenom II X4 CPU, it takes about 14 seconds to finish.

```
> test.data <-
+     function(dim, num, seed=17) {
+       set.seed(seed)
+       matrix(rnorm(dim * num),
+         nrow=num)
+ }
> m <- test.data(120, 4500)
> system.time(dist(m))
   user  system elapsed
 14.343   0.185  14.533
```

Now load the `rpud` package, and compute the same distance matrix using the function `rpuDist`. For an NVIDIA GTX 460 GPU, the execution time is about 1 second. Furthermore, with the `rpudplus` add-on, you can compute distance matrices of larger data sets that fit inside the system RAM available to R.

```
> library(rpud)
> system.time(rpuDist(m))
   user  system elapsed
  0.674   0.305   0.980
```

Here is a chart comparing the performance between the CPU and GPU.



Exercise 1

Run the performance test with more vectors in higher dimensions.

Solution of Exercise 1

We exercise the performance test on 6,000 random vectors in 240 dimensions. It takes over a minute on the Phenom X4 CPU, but less than 2 seconds on the GTX 460 GPU.

```
> m <- test.data(240, 6000)
> system.time(dist(m))
   user  system elapsed
 65.390   0.350  65.747
> system.time(rpuDist(m))
   user  system elapsed
  1.280   0.570   1.862
```

Exercise 2

Find the distance matrix for other types of distances, such as maximum, Manhattan, Canberra, binary and Minkowski.

Solution of Exercise 2

We can create the distance matrix of other types by setting the "method" option. For example, for the Canberra distance, we simply set the `method` option to be `"canberra"`.

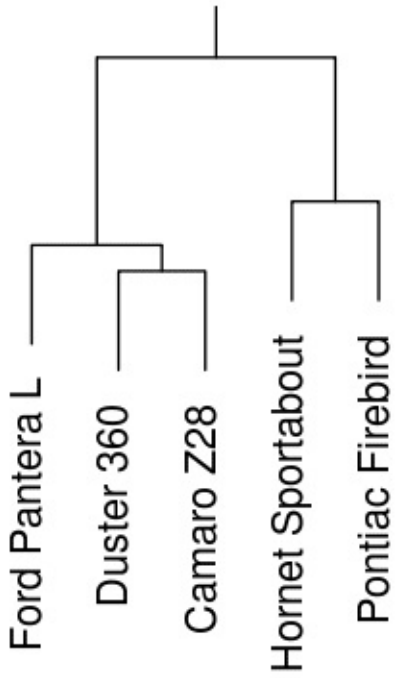
```
> m <- test.data(120, 2400)
> system.time(
+     rpuDist(m, method="canberra"))
   user  system elapsed
  0.230   0.060   0.289
```

30.2 Hierarchical Cluster Analysis

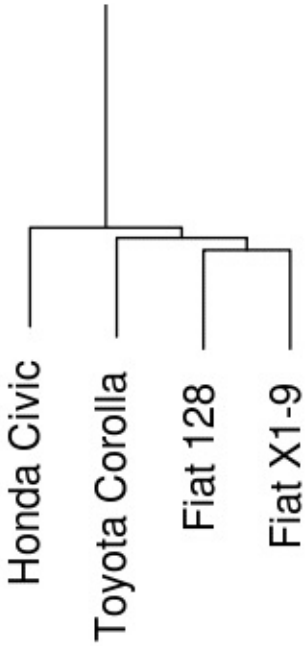
With the distance matrix found in previous tutorial, we can use various techniques of cluster analysis for relationship discovery. For example, in the data set `mtcars`, we can create a distance matrix with the function `hclust`, and plot a dendrogram that displays a hierarchical relationship among the vehicles.

```
> d <- dist(as.matrix(mtcars))
> hc <- hclust(d)
> plot(hc)
```

Careful inspection of the dendrogram shows that 1974 Pontiac Firebird and Camaro Z28 are classified as close relatives as expected.



Similarly, the dendrogram shows that the 1974 Honda Civic and Toyota Corolla are close to each other.



In general, there are many choices of cluster analysis methodology. The function `hclust` in R uses the **complete linkage** method for **hierarchical clustering** by default. This particular clustering method defines the cluster distance between two clusters to be the maximum distance between their individual components. At every stage of the clustering process, the two nearest clusters are merged into a new cluster. The process is repeated until the whole data set is agglomerated into one single cluster. For a data set with 4,000 elements, it takes `hclust` about 2 minutes to finish the task on an AMD Phenom II X4 CPU.

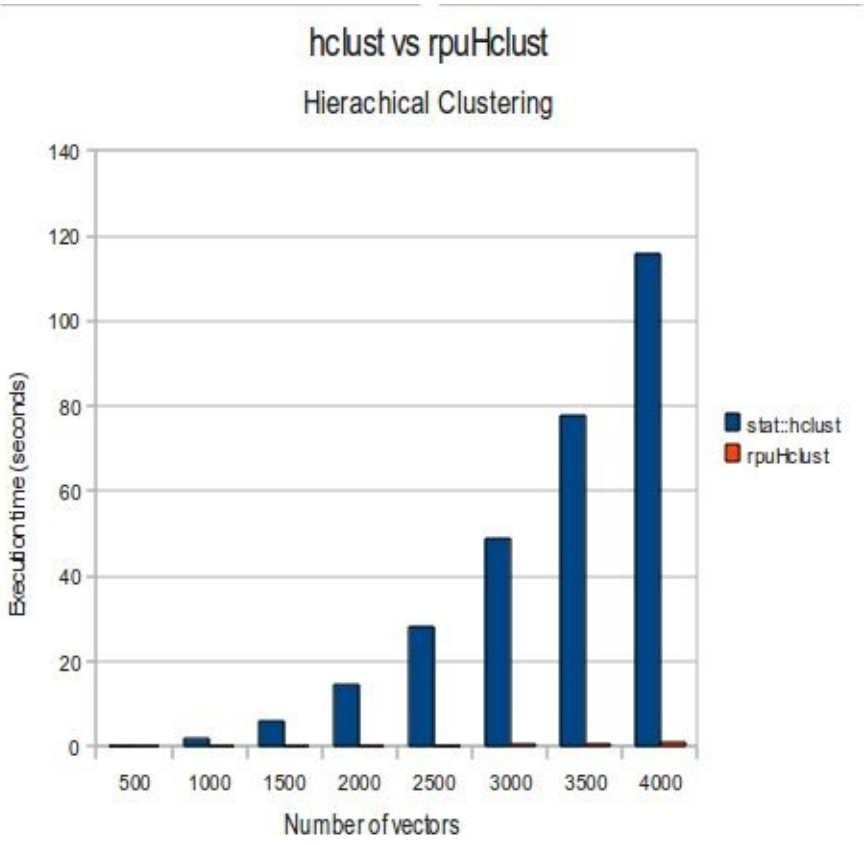
```
> test.data <-
+       function(dim, num, seed=17) {
+          set.seed(seed)
+          matrix(rnorm(dim * num),
+         nrow=num)
+ }
> m <- test.data(120, 4500)
>
> library(rpud)
> d <- rpuDist(m)
>
> system.time(hclust(d))
```

```
   user  system elapsed
115.765   0.087 115.914
```

With clever code optimization, the function `rpuHclust` in `rpud` equipped with the `rpudplus` add-on performs much better. It creates identical cluster analysis output just like the original `hclust` in R but in much higher speed.

```
> system.time(rpuHclust(d))
   user  system elapsed
  0.792   0.104   0.896
```

Nevertheless, the faster algorithm in `rpudplus` is still mostly CPU bound. The heavy cost of memory access turns out to be too excessive for pure GPU computing. Here is a chart that compares the performance of `hclust` against `rpuHclust` of `rpudplus`:



Exercise 1

Run the performance test with more vectors in higher dimensions.

Solution of Exercise 1

We exercise the performance test with 6,000 random vectors in 240 dimensions. It takes over six and a half minutes for the function `hclust` in stock R, but less than 3 seconds with `rpudplus`.

```
> m <- test.data(240, 6000)
> d <- rpuDist(m)
> system.time(hclust(d))
   user  system elapsed
 390.60    0.22  390.95
> system.time(rpuHclust(d))
   user  system elapsed
  2.570   0.220   2.783
>
```

Exercise 2

Compute hierarchical clustering with other linkage methods, such as single, median, average, centroid, Ward's and McQuitty's.

Solution of Exercise 2

We can perform cluster analysis of other linkage types by setting the "method" option. For example, for the Ward method, we simply set the `method` option to be `"ward"`.

```
> m <- test.data(120, 2400)
> d <- rpuDist(m)
> system.time(
+     rpuHclust(d, method="ward"))
   user  system elapsed
  0.280   0.020   0.304
```

30.3 Kendall Rank Coefficient

The correlation coefficient is a measurement of association between two random variables. While its numerical calculation is straightforward, it is not readily applicable to non-parametric statistics.

For example, in the data set `survey`, the exercise level `Exer` and smoking habit `Smoke` are qualitative attributes. To find their correlation coefficient, we would have to assign artificial numeric values to the qualitative data, which is not very elegant to say the least.

A more robust approach is to compare the rank orders between the variables. For example, suppose student A exercises more than student B in the data set `survey`. We can check whether it is also the case that student A smokes more than student B. If so, then A and B would have the same relative rank orders, and we say that A and B are **concordant pairs** with respect to the random variables `Exer` and `Smoke`. If student A turns out to smoke less than student B, then we say that A and B are **discordant pairs**.

Intuitively, it is clear that if the number of concordant pairs is much larger than the number of discordant pairs, then the random variables are positively correlated. If the number of concordant pairs is much less than discordant pairs, then the variables are negatively correlated. Finally, if the number of concordant pairs is about the same as discordant pairs, then the variables are weakly correlated. As a result, the **Kendall rank correlation coefficient** between the two random variables with *n* observations is defined as:

$$\tau = \frac{(number\ of\ concordant\ pairs) - (number\ of\ discordant\ pairs)}{n(n-1)/2}$$

To find the Kendall coefficient between `Exer` and `Smoke`, we will first create a matrix `m` consisting only of the `Exer` and `Smoke` columns. Then we apply the function `cor` with the `"kendall"` option. In order to filter out missing survey values, we set the `use` option as `"pairwise.complete.obs"`. The computation shows that the Kendall coefficient between `Exer` and `Smoke` is 0.083547, which is pretty close to zero. Hence the student exercise level and smoking habit are

weakly correlated variables.

```
> library(MASS)
> smoke <-
+     as.numeric(factor(survey$Smoke,
+     levels=c("Never","Occas",
+       "Regul","Heavy")))
> exer <-
+     as.numeric(factor(survey$Exer,
+     levels=c("None","Some",
+       "Freq")))

> m <- cbind(exer, smoke)
> cor(m, method="kendall",
+     use="pairwise")
          exer    smoke
exer  1.000000 0.083547
smoke 0.083547 1.000000
```

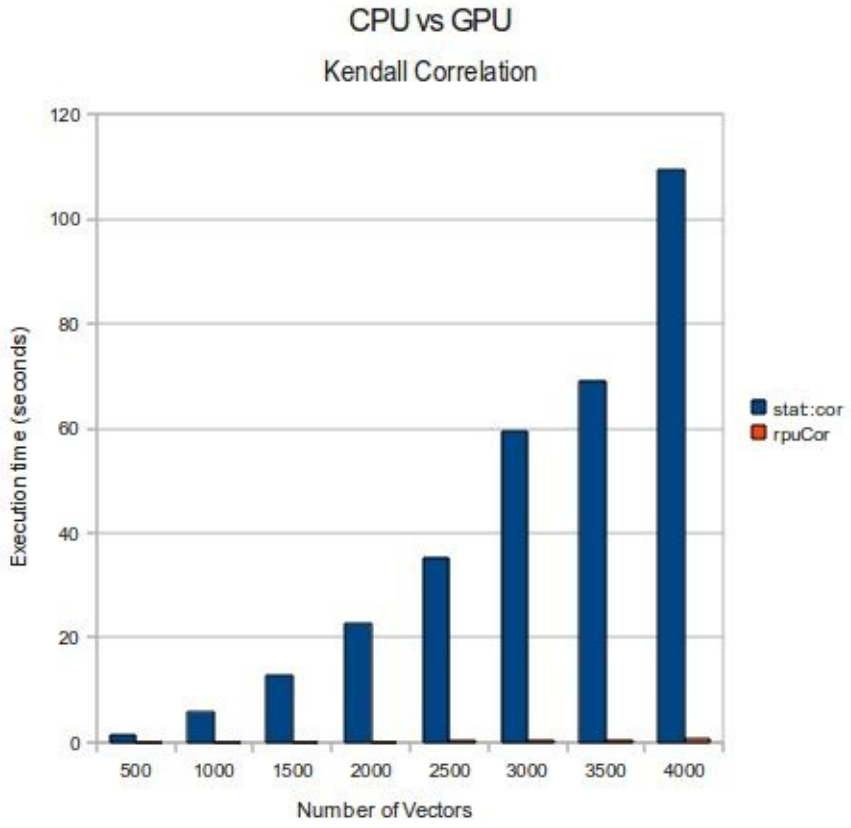
Computation of the Kendall coefficient is very time consuming. For a random data set with 24 variables and 4,000 observations, it takes about 2 minutes to calculate the Kendall correlation coefficients on an AMD Phenom II X4 CPU.

```
> test.data <-
+     function(dim, num, seed=17) {
+       set.seed(seed)
+       matrix(rnorm(dim * num),
+         nrow=num)
+ }
>
> m <- test.data(dim=24, num=4000)
> system.time(
+     cor(m, method="kendall"))
   user  system elapsed
109.344   0.004 109.374
```

The computation time is drastically reduced for an NVIDIA GTX 460 GPU. If we load the `rpud` package with the `rpudplus` add-on, and compute the same Kendall correlation coefficients using the function `rpucor`, the execution time takes only about 0.5 seconds.

```
> library(rpud)
> system.time(
+     rpucor(m, method="kendall"))
   user  system elapsed
  0.529   0.001   0.531
```

Here is a chart that compares the performance between AMD Phenom II X4 and NVIDIA GTX 460 for finding the Kendall rank coefficient.



Exercise 1

Run the performance test with more variables and observations.

Solution of Exercise 1

We exercise the performance test on 4,500 random vectors in 36 dimensions. It takes over 4 minutes on the Phenom X4 CPU, but less than 2 seconds on the GTX 460 GPU.

```
> m <- test.data(dim=36, num=4500)
> system.time(
+     cor(m, method="kendall"))
   user  system elapsed
 257.28    0.02  257.34
>
> library(rpud)
```

```
> system.time(
+     rpucor(m, method="kendall"))
   user  system elapsed
  1.760   0.010   1.774
```

Exercise 2

Find the Kendall correlation coefficients of the data set `survey` with various options for handling missing values, such as "everything" (default), "all.obs", "complete.obs", "na.or.complete", and "pairwise.complete.obs".

Solution of Exercise 2

We can find the Kendall coefficients with various missing value handling options by selecting the `use` argument. For example, if we set the `use` argument as `"all.obs"`, then any missing value in the data would trigger an error.

```
> m <- cbind(exer=survey$Exer,
+    smoke=survey$Smoke)
> rpucor(m, method="kendall",
+     use="all.obs")
Error in rpucor(m, method = "kendall", use = "all.obs") :
  missing observations in rpucor
```

Further details of the `use` argument can be found in the R documentation for the function `cor`.

```
> help(cor)
```

30.4 Significance Test for Kendall's Tau-b

A variation of the definition of Kendall correlation coefficient is necessary in order to deal with data samples with tied ranks. It known as the **Kendall's tau-b coefficient** and is more effective in determining whether two non-parametric data samples with ties are correlated.

Formally, the Kendall's tau-b is defined as follows. It replaces the denominator of the original definition with the product of square roots of data pair counts not tied in the target features.

$$\tau_B = \frac{(number\ of\ concordant\ pairs) - (number\ of\ discordant\ pairs)}{\sqrt{N_1} \times \sqrt{N_2}}$$

$$\begin{aligned} where\ N_1 &= number\ of\ data\ pairs\ not\ tied\ in\ a\ target\ feature \\ N_2 &= number\ of\ data\ pairs\ not\ tied\ in\ another\ target\ feature \end{aligned}$$

In the context of our previous example based on the data set `survey`, N_1 would be the number of student pairs with different smoking habits, whereas N_2 would be the number of student pairs with different exercise practice levels. We say the two variables `Exer` and `Smoke` in the data set are **uncorrelated** if their correlation coefficient is zero.

As before, the function `cor` shows that the Kendall correlation coefficient between `Exer` and `Smoke` is 0.083547.

```
> smoke <-
+     as.numeric(factor(survey$Smoke,
+     levels=c("Never","Occas",
+       "Regul","Heavy")))
> exer <-
+     as.numeric(factor(survey$Exer,
+     levels=c("None","Some",
+       "Freq")))

> m <- cbind(exer, smoke)
> cor(m, method="kendall",
+     use="pairwise")
           exer    smoke
```

```
exer  1.000000 0.083547
smoke 0.083547 1.000000
```

In order to decide whether the variables are uncorrelated, we test the null hypothesis that $\tau_B = 0$. The *alternative hypothesis* is that the variables are correlated, and τ_B is non-zero.

To test the null hypothesis, we apply the function `cor.test`, and found a p-value of 0.1709. Hence we do not reject the null hypothesis that variables are uncorrelated at 0.05 significance level.

```
> cor.test(exer, smoke,
+     method="kendall")

    Kendall's rank correlation tau

data:  exer and smoke
z = 1.3694, p-value = 0.1709
alternative hypothesis: true tau is not equal to 0
sample estimates:
     tau
0.083547
```

The same can be achieved with the latest version of `rpudplus`. We begin with the function `rpucor`, which now uses Kendall's tau-b to compute the correlation coefficient, and comes up with the same correlation coefficient as the function `cor.test`.

```
> library(rpud)
> rpucor(m, method="kendall",
+     use="pairwise")
          exer    smoke
exer  1.000000 0.083547
smoke 0.083547 1.000000
attr(,"method")
[1] "kendall"
attr(,"use")
[1] "pairwise.complete.obs"
```

Alternatively, we can now use the function `rpucor.test` to extract the Kendall's tau-b from the `estimate` component of the result.

```
> rt <- rpucor.test(m,
+     method="kendall",
+     use="pairwise")
> rt$estimate
```

```
          exer    smoke
exer  1.000000 0.083547
smoke 0.083547 1.000000
```

And we can find the p-value of the correlation estimate in the `p.value` component.

```
> rt$p.value
         exer   smoke
exer  0.00000 0.17088
smoke 0.17088 0.00000
```

To decide whether to reject the null hypothesis that the variables are uncorrelated, we compare the p-values against 0.05. Since the comparison result at `["exer", "smoke"]` is `FALSE`, we do not reject the null hypothesis that `Exer` and `Smoke` are uncorrelated at 0.05 significance level.

```
> rt$p.value < 0.05
       exer smoke
exer   TRUE FALSE
smoke FALSE  TRUE
```

To evaluate performance, we create random sample data drawn from 16 ordered symbols, and found the function `cor` takes more than one and a half minutes to compute the Kendall's coefficient for 4000 observations in 24 features on an AMD Phenom II X4 CPU. At the time of writing, the function `cor.test` does not yet support multivariate data for comparison.

```
> test.data <-
+     function(dim, num, seed=17) {
+       set.seed(seed)
+       matrix(sample(1:16, dim*num,
+         replace=TRUE), nrow=num)
+ }
> m <- test.data(24, 4000)
>
> system.time(cor(m,
+     method="kendall"))
   user  system elapsed
 95.610   0.010  95.614
```

The same task would take merely 0.5 seconds for `rpucor` on a NVIDIA GTX 460 GPU.

```
> system.time(rpucor(m,
```

```
+     method="kendall"))
   user  system elapsed
  0.570   0.000   0.569
```

And it takes similarly short amount of time for `rpucor.test`.

```
> system.time(rpucor.test(m,
+     method="kendall"))
   user  system elapsed
  0.520   0.000   0.516
```

Exercise 1

Determine which pairs of variables in the data set `USJudgeRatings` are correlated based on the Kendall's method at 0.05 significance level.

Solution of Exercise 1

We apply the function `rpucor.test` to the data set `USJudgeRatings` and compare the `p.value` component of the returned value against 0.05. The result shows that we cannot reject the hypothesis that the contact level ([CONT]) between the lawyer and the judge is uncorrelated with any other variables at 0.05 significance level.

```
> library(rpud)
> rpucor.test(
+     USJudgeRatings
+     )$p.value < 0.05
      CONT  INTG  DMNR  DILG ...
CONT  TRUE FALSE FALSE FALSE ...
INTG FALSE  TRUE  TRUE  TRUE ...
       ....
```

Exercise 2

Determine which pairs of variables in the data set `ToothGrowth` are correlated based on the Kendall's method at 0.05 significance level. (Hint: transform its dichotomy variable `supp` into numeric type before proceed.)

Solution of Exercise 2

Using the method `transform`, we replace the `supp` column by numeric values.

Then we apply the function `rpucor.test`, and shows that the length of teeth growth is correlated with the dosage level regardless of the supplement type.

```
> library(rpud)
> m <- transform(ToothGrowth,
+     supp=as.numeric(supp))
> pval <- rpucor.test(m,
+     method="kendall")$p.value
> pval < 0.05
       len  supp  dose
len   TRUE FALSE  TRUE
supp FALSE  TRUE FALSE
dose  TRUE FALSE  TRUE
```

30.5 Support Vector Machine, Part I

Most elementary statistical inference algorithms assume that the data can be modeled by linear parameters with a normally distributed error component. According to Vladimir Vapnik in *Statistical Learning Theory* (Vapnik 1998), the assumption is inappropriate for modern large scale problems, and his invention of the **Support Vector Machine** (SVM) makes such assumption unnecessary. There are many implementations of the algorithm, and a popular one is the LIBSVM (C.-C. Chang *et al.*), which can be invoked in R via the `e1071` package.

For demonstration purpose, we will train a regression model based on the California housing prices data from the 1990 Census. The data set is called `cadata`, and can be downloaded from the LIBSVM site.

Before training the SVM model, we pre-scale the downloaded data in a terminal using a standalone tool `rpusvm-scale` in `RPUSVM`, which is available from r-tutor.com. This will create a new data set `cadata.scaled`. As a good practice, we also save the scale parameters in a secondary file `cadata.save` for later use.

```
$ rpusvm-scale -x "-1:1" -y "-1:1" \
  -s cadata.save cadata cadata.scaled
```

Now we can load `cadata.scaled` in R with the function `read.svm.data` in the `rpudplus` add-on. Since the response values in the data set are not in factor levels, we have to set the argument `fac` as `FALSE`. We also save the `x` and `y` components as standalone variables for convenience.

```
> library(rpud)
> cadata.scaled <- read.svm.data(
+     "cadata.scaled", fac=FALSE)
> x <- cadata.scaled$x
> y <- cadata.scaled$y
```

Then we train an SVM regression model using the function `svm` in `e1071`. As the data has been pre-scaled, we disable the `scale` option. The data set has about 20,000 observations, and the training takes over a minute on an AMD Phenom II X4 system.

```
> library(e1071)
> system.time(cadata.libsvm <-
```

```
+     e1071::svm(x, y,
+       type="eps-regression",
+       scale=FALSE))
   user  system elapsed
 64.630   0.010  64.659
```

We can do likewise with the function `rpusvm` of the `rpudplus` add-on. The same training now takes only 6 seconds on an NVIDIA GTX 460 GPU:

```
> system.time(cadata.rpusvm <-
+     rpusvm(x, y,
+       type="eps-regression",
+       scale=FALSE))
.........**.
   user  system elapsed
  6.170   0.020   6.184
```

The models trained by the two packages are numerically equivalent, as is evidenced by their respective mean square errors. For the LIBSVM model from `e1071`, the mean square error is about 0.0696.

```
> res.libsvm <-
+     cadata.libsvm$residuals
> sum(res.libsvm*res.libsvm) /
+     length(res.libsvm)
[1] 0.069568
```

This is almost identical to the mean square error from the function `rpusvm` in `rpudplus`:

```
> res.rpusvm <-
+     cadata.rpusvm$residuals
> sum(res.rpusvm*res.rpusvm) /
+     length(res.rpusvm)
[1] 0.069566
```

Sometimes it is more effective to invoke LIBSVM directly in a terminal. Using OpenMP to parallelize LIBSVM v3.1 on an AMD Phenom II X4 CPU, training a regression model of `cadata` takes about 28 seconds.

```
$ time svm-train -s 3 -m 1000 \
      cadata.scaled cadata.libsvm
........

optimization finished, #iter = 9677
nu = 0.590304
```

```
obj = -2232.720805, rho = -0.299943
nSV = 12216, nBSV = 12156

real    0m28.633s
user    0m28.190s
sys     0m0.390s
```

Using a standalone Linux tool in RPUSVM, we can invoke the same code of `rpusvm` in a terminal. The training takes about 5 seconds on a GTX 460 GPU:

```
$ time rpusvm-train -s 3 \
      cadata.scaled cadata.rpusvm

rpusvm-train 0.1.2
http://www.r-tutor.com
Copyright (C) 2011-2012 Chi Yau. All Rights Reserved.
This software is free for academic use only. There is absolutely

GeForce GTX 460 GPU

.........**.

Finished optimization in 9498 iterations
nu = 0.590179
obj = -2232.72, rho = -0.300649
nSV = 12218, nBSV = 12157
Total nSV = 12218

real    0m5.100s
user    0m4.940s
sys     0m0.150s
```

Finally, we compare their prediction speeds on `cadata`. Parallelized LIBSVM takes about 11 seconds on Phenom II X4:

```
$ time svm-predict cadata.scaled \
      cadata.libsvm cadata.out

Mean squared error = 8500.66 (regression)
Squared correlation coefficient = 0.000325578 (regression)

real    0m11.176s
user    0m44.540s
sys     0m0.010s
```

The same task takes RPUSVM under 2 seconds on GTX 460:

```
$ time rpusvm-predict cadata.scaled \
      cadata.rpusvm cadata.out

rpusvm-predict 0.1.2
http://www.r-tutor.com
Copyright (C) 2011-2012 Chi Yau. All Rights Reserved.
This software is free for academic use only. There is absolutely

GeForce GTX 460 GPU

Mean squared error = 0.0695664
Pearson correlation coefficient = 0.698953

real    0m1.631s
user    0m1.440s
sys     0m0.170s
```

Note 1

Suppose we would like to perform prediction on a data file stored in LIBSVM format, say `test.dat`. We must first pre-scale it with the scale parameter file `cadata.save` that we created earlier in preparation of training `cadata`.

```
$ rpusvm-scale -r cadata.save \
      test.dat test.scaled
```

Then we load it in R with the `read.svm.data` method in `rpudplus` and apply the function `predict` as usual. Just make sure to manually restore the result to the original y-scale before use.

```
> test.scaled <- read.svm.data(
+     "test.scaled", fac=FALSE)
> pred <- predict(
+     cadata.rpusvm, test.scaled$x)
> head(pred)
      1         2         3    ...
 0.592945  0.782728  0.557078  ...
```

These data scaling juggernauts can be avoided with the latest `rpudplus` and `RPUSVM`. See our next tutorial for details.

Note 2

A much faster algorithm for large scale document classification *without* the use of a GPU is LIBLINEAR (R.-E. Fan *et al.*). It can process millions of records in

seconds.

Exercise 1

Train SVM models of larger data sets using `rpusvm`.

Solution of Exercise 1

We will use the data set `a9a` from the LIBSVM site. The data set is pre-scaled, and has about 32,000 observations. Training the C-classification model takes `e1017` more than 3 minutes on AMD Phenom II X4:

```
> a9a <- read.svm.data("a9a")
> x <- a9a$x; y <- a9a$y

> system.time(a9a.libsvm <-
+     e1071::svm(x, y, scale=FALSE))
   user  system elapsed
 201.99    0.02  202.08
```

The same task takes `rpusvm` about 10 seconds on the GTX 460 GPU.

```
> system.time(a9a.rpusvm <-
+     rpusvm(x, y, scale=FALSE))
.......*.
   user  system elapsed
 10.650   0.010  10.658
```

Exercise 2

Find probability estimates of the regression model of `cadata` by enabling the `probability` option in `rpusvm`.

Solution of Exercise 2

By enabling the `probability` option in `rpusvm`, we find that the scale parameter of the probability model of `cadata` with Laplace distribution is about 0.18.

```
> system.time(cadata.rpusvm <-
+     rpusvm(x, y,
+       type="eps-regression",
+       scale=FALSE,
+       probability=TRUE))
```

```
.......**.
.......**.
.......*.*.
.......*.*.
.......*.*.
.........**.
   user  system elapsed
 23.780   0.030  23.808

> cadata.rpusvm$sigma
[1] 0.18041
```

Exercise 3

Perform cross-validation for the regression model of `cadata` by enabling the `cross` option in `rpusvm`.

Solution of Exercise 3

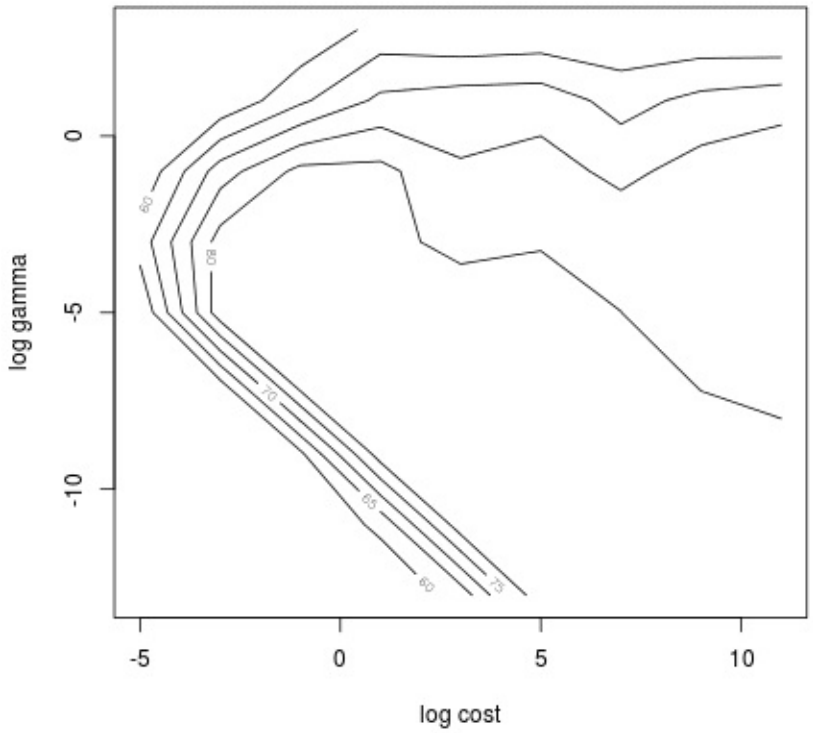
We perform a five-fold cross-validation on `cadata` by setting the argument `cross` to be 5. The total mean square error found in `tot.MSE` is 0.070506.

```
> system.time(cadata.rpusvm <-
+     rpusvm(x, y,
+       type="eps-regression",
+       scale=FALSE, cross=5))
.........**.
........**.
.......*.*.
.......*.
.......*.*.
.......**.
   user  system elapsed
 23.990   0.030  24.026
>
> cadata.rpusvm$tot.MSE
[1] 0.070506
```

Exercise 4

Search for optimal SVM kernel and parameters for the regression model of `cadata` using `rpusvm` based on similar procedures explained in the text *A Practical Guide to Support Vector Classification* (C.-W. Hsu *et al.*). In particular, create a similar contour map as below for selecting smaller regions for

further optimization.



Solution of Exercise 4

Examples on how to search for optimal SVM parameters can be found in Part III of our SVM tutorials. It includes both C-classification and epsilon regression SVM training. The code allows easy adaptation for other kernel types.

30.6 Support Vector Machine, Part II

In our last tutorial on SVM training with GPU, we mentioned a necessary step to pre-scale the data with `rpusvm-scale`, and to reverse scaling the prediction outcome. This cumbersome procedure is now simplified with the latest `RPUSVM`.

For example, we can work directly with the `cadata` from the LIBSVM site. Just load it into the R workspace with `read.svm.data` and apply the function `rpusvm` right away. The overhead of the implicit data scaling turns out to be rather negligible.

```
> library(rpud)
> cadata <- read.svm.data(
+     "cadata", fac=FALSE)
> x <- cadata$x
> y <- cadata$y
> system.time(cadata.rpusvm <-
+     rpusvm(x, y,
+       type="eps-regression"))
........**.
   user  system elapsed
  6.510   0.020   6.539
```

We can inspect the range of each attribute of `cadata` in the SVM model `cdata.rpusvm`. In particular, for a data set with *N* attributes, the `x.bound` component of `cdata.rpusvm` is a *2*x*N* matrix, with each column containing the lower and upper bounds of the corresponding attribute. Likewise, the `y.bound` component contains the lower and upper bounds of the response variable.

```
> cadata.rpusvm$x.bound
        [,1] [,2]  [,3] [,4]  ...
[1,]  0.4999    1     2    1  ...
[2,] 15.0001   52 39320 6445  ...

> cadata.rpusvm$y.bound
       [,1]
[1,]  14999
[2,] 500001
```

Using the `residuals` component of the SVM model, we can compute the mean square error:

```
> cadata.res <-
+     cadata.rpusvm$residuals
> sum(cadata.res*cadata.res) /
+     length(cadata.res)
[1] 4.091e+09
```

As for prediction, we can apply the function `predict` on a test data set without further post-processing:

```
> test.dat <- read.svm.data(
+     "test.dat", fac=FALSE)
> pred <- predict(
+     cadata.rpusvm, test.dat$x)
> head(pred)
  1      2      3      4    ...
401234 447244 392549 330003 ...
```

If we decide to train an SVM in a terminal, we can apply the standalone `rpusvm` tool by adding extra scale parameters:

```
$ time rpusvm-train \
    -x "-1:1" -y "-1:1" -s 3 \
    cadata cadata.rpusvm

rpusvm-train 0.1.2
http://www.r-tutor.com
Copyright (C) 2011-2012 Chi Yau. All Rights Reserved.
This software is free for academic use only. There is absolutely

GeForce GTX 460 GPU

.........**.

Finished optimization in 9583 iterations
nu = 0.590296
obj = -2232.72, rho = -0.300248
nSV = 12221, nBSV = 12160
Total nSV = 12221

real    0m5.171s
user    0m5.020s
sys     0m0.130s
```

A glance of the SVM model `cadata.rpusvm` shows that the scale parameters are gathered in the header section.

```
$ head -n 20 cadata.rpusvm
```

```
SvmType: smo-epsilon-regression
KernelType: radial
Gamma: 0.125
X-scale: -1 1
  0: 0.4999 15.0001
  1: 1 52
  2: 2 39320
  3: 1 6445
  4: 3 35682
  5: 1 6082
  6: 32.54 41.95
  7: -124.35 -114.31
Y-scale: -1 1
   14999 500001
NrClass: 2
TotalSV: 12221
Rho: -0.300248
```

Finally, we can use the model for prediction without pre-scaling the test data or post-scaling the outcome.

```
$ time rpusvm-predict cadata \
      cadata.rpusvm cadata.out

rpusvm-predict 0.1.2
http://www.r-tutor.com
Copyright (C) 2011-2012 Chi Yau. All Rights Reserved.
This software is free for academic use only. There is absolutely

GeForce GTX 460 GPU

Mean squared error = 4.09101e+09
Pearson correlation coefficient = 0.698961

real    0m1.691s
user    0m1.480s
sys     0m0.170s
```

30.7 Support Vector Machine, Part III

With the SVM tools we discussed in the previous tutorials, we can now proceed to explain how to select optimal parameters of an SVM model. The algorithm is actually a grid search similar to the one found in the LIBSVM python tool `grid.py`. You can download the R source of this discussion from r-tutor.com.

C-classification

We first deal with the case for C-classification training. For simplicity, we demonstrate with the LIBSVM sample data set `heart_scale`. The data set has been pre-scaled, and we can load it directly into the R workspace with the function `read.svm.data` in `rpudplus`. We also follow our habit of isolating the `x` and `y` components into standalone variables.

```
> library(rpud)
> hs <- read.svm.data("heart_scale")
> x <- hs$x
> y <- hs$y
```

We then select a sequence of integers from -5 to 11 as exponents of candidate cost parameters in the SVM model. It corresponds to cost parameter ranges from 2^{-5} = 0.03125 to 2^{11} = 2048.

```
> c.log <- seq(-5, 11, by=2)
```

For illustrative purpose, we use the Gaussian RBF kernel here. We select another sequence of integers from -13 to 3 as exponents for candidate gamma parameters. It corresponds to gamma parameter ranges from 2^{-13} = 0.00012207 to 2^{3} = 8.

```
> g.log <- seq(-13, 3, by=2)
```

Then we create a matrix called `accuracy` to keep track of the progress, and we will determine its maximum element later.

```
> nc <- length(c.log)
> ng <- length(g.log)
> accuracy <-
+     matrix(nrow=nc, ncol=ng)
```

Now we run a double for-loop and perform a grid search based on the cost and gamma parameters.

```
> for (i in 1:nc) {
+      for (j in 1:ng) {
+           c <- 2^c.log[i]
+           g <- 2^g.log[j]
+         # run C-SVM for (c, g)
+      }
+ }
```

Within the double for-loop, we perform a 5-fold cross-validation, and save the result in the `accuracy` matrix.

```
>    svm.model <- rpusvm(x, y,
+         gamma=g, cost=c,
+         scale=FALSE, cross=5)
>    accuracy[i, j] <-
+         svm.model$tot.accuracy
```

At the end of the loop, we can determine the maximum accuracy value found.

```
> max.accuracy <- max(accuracy)
> max.accuracy
[1] 84.815
```

We need to find the cost and gamma parameters that produce the maximum accuracy value. For this, we deduce the row and column indexes via modular arithmetic based on the sequential position of the maximum accuracy value within the matrix.

```
> max.index <- which(
+     accuracy == max.accuracy
+     ) - 1

> log.cost  <-
+     c.log[max.index %% nc + 1]
> optimal.cost  <- 2^log.cost

> log.gamma <-
+     g.log[max.index %/% nc + 1]
> optimal.gamma <- 2^log.gamma
```

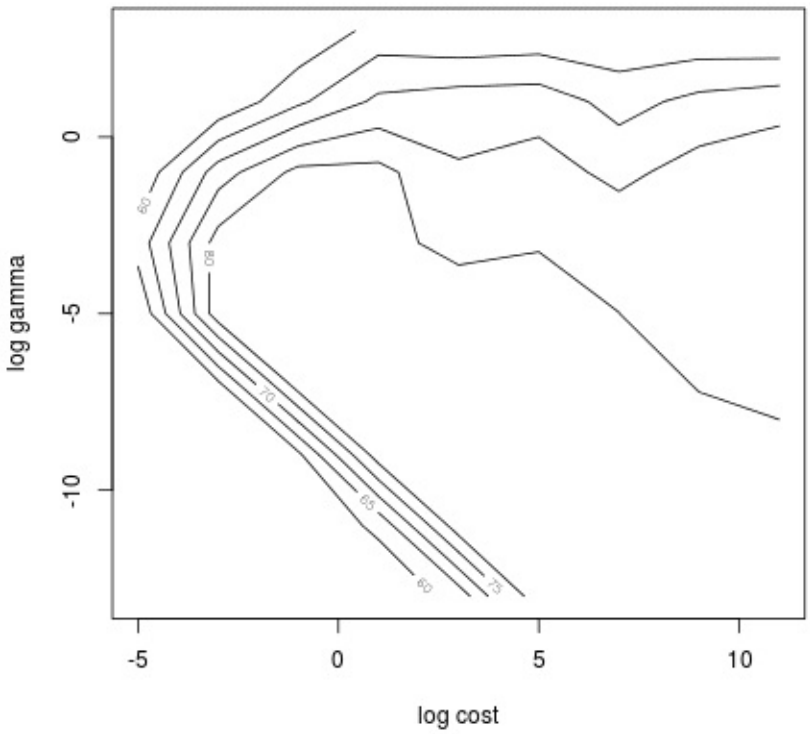
And we print out the result with `cbind`:

```
> cbind(log.cost, log.gamma,
```

```
+     optimal.cost, optimal.gamma)
     log.cost log.gamma optimal.cost
[1,]        7        -9          128
[2,]        1        -5            2
     optimal.gamma
[1,]     0.0019531
[2,]     0.0312500
```

Finally, we can create a contour map that describes the optimal region for accuracy. This enables us to narrow down smaller regions for further optimization.

```
> plot(0, 0, type = "n",
+     xlim = range(c.log),
+     ylim = range(g.log),
+     xlab = "log cost",
+     ylab = "log gamma")

> contour(x=c.log, y=g.log,
+     accuracy, lty = "solid",
+     add = TRUE,
+     vfont = c(
+       "sans serif", "plain"))
```



The following is the R source code for searching optimal parameters in the C-classification RBF SVM model of `heart_scale`:

```
library(rpud)

hs <- read.svm.data("heart_scale")
x <- hs$x
y <- hs$y

c.log <- seq(-5, 11, by=2)
g.log <- seq(-13, 3, by=2)

nc <- length(c.log)
ng <- length(g.log)
accuracy <-
    matrix(nrow=nc, ncol=ng)

for (i in 1:nc) {
    for (j in 1:ng) {
        c <- 2^c.log[i]
        g <- 2^g.log[j]
        svm.model <- rpusvm(x, y,
            gamma=g, cost=c,
            scale=FALSE, cross=5)
        accuracy[i, j] =
             svm.model$tot.accuracy
        cat("accuracy(", c,
             ",", g, ") =",
            accuracy[i, j], "%\n")
    }
}

max.accuracy <- max(accuracy)
max.accuracy
max.index <- which(
     accuracy == max.accuracy) - 1

log.cost <-
     c.log[max.index %% nc + 1]
optimal.cost  <- 2^log.cost

log.gamma <-
     g.log[max.index %/% nc + 1]
optimal.gamma <- 2^log.gamma

cbind(log.cost, log.gamma,
      optimal.cost, optimal.gamma)

# contour map
plot(0, 0, type = "n",
      xlim = range(c.log),
      ylim = range(g.log),
```

```
      xlab = "log cost",
      ylab = "log gamma")

contour(x=c.log, y=g.log,
      accuracy, lty = "solid",
      add = TRUE,
      vfont = c(
        "sans serif", "plain"))
```

Epsilon Regression

The algorithm for regression training is slightly different, as we are aiming for minimal mean square error in this case. We also need to load our data set `heart_scale` slightly differently, and set the `fac` parameter as `FALSE` so as to avoid loading the response data as factors.

```
> library(rpud)
> hs <- read.svm.data(
+     "heart_scale", fac=FALSE)
> x <- hs$x
> y <- hs$y
```

We will create same integer sequences as before, along with a matrix `mse` for keeping track of the mean square error.

```
> c.log <- seq(-5, 11, by=2)
> g.log <- seq(-13, 3, by=2)

> nc <- length(c.log)
> ng <- length(g.log)
> mse <-
+     matrix(nrow=nc, ncol=ng)
```

Now we run a double for-loop and perform a grid search based on the cost and gamma parameters.

```
> for (i in 1:nc) {
+      for (j in 1:ng) {
+           c <- 2^c.log[i]
+           g <- 2^g.log[j]
+       # run epsilon-SVR for (c, g)
+      }
+ }
```

Within the double for-loop, we perform a 5-fold cross-validation, and save the result in the `mse` matrix.

```
>  svm.model <- rpusvm(x, y,
+       type="eps-regression",
+       gamma=g, cost=c,
+       scale=FALSE, cross=5)
>  mse[i, j] = svm.model$tot.MSE
```

At the end of the loop, we can determine the minimum mean square error.

```
> min.mse <- min(mse); mse
[1] 0.52383
```

Then we find the cost and gamma parameters that reproduce the minimum mean square error.

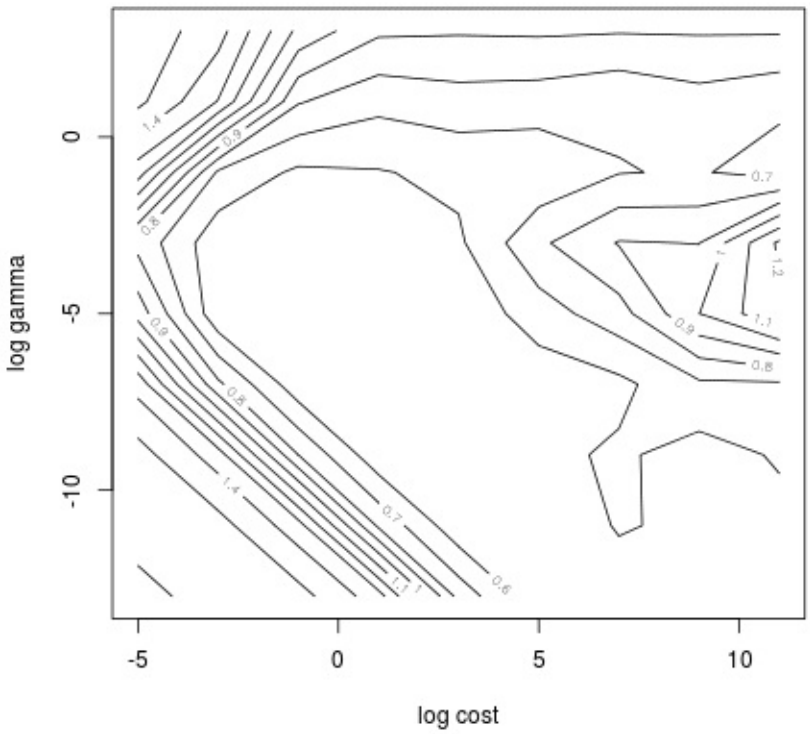
```
> min.index <-
+     which(mse == min.mse) - 1
> log.cost  <-
+     c.log[min.index  %% nc + 1]
> optimal.cost  <- 2^log.cost

> log.gamma <-
+     g.log[min.index %/% nc + 1]
> optimal.gamma <- 2^log.gamma
```

And we print out the result with `cbind`:

```
> cbind(log.cost, log.gamma,
+     optimal.cost, optimal.gamma)
     log.cost log.gamma optimal.cost
[1,]       -3        -3        0.125
     optimal.gamma
[1,]         0.125
```

Finally, we can create a contour map that describes the optimal region for mean square error. This enables us to narrow down smaller regions for further optimization.

```
> plot(0, 0, type = "n",
+     xlim = range(c.log),
+     ylim = range(g.log),
+     xlab = "log cost",
+     ylab = "log gamma")

> contour(x=c.log, y=g.log, mse,
+     lty = "solid", add = TRUE,
+     vfont = c(
+       "sans serif", "plain"))
```



The following is the R source code for searching optimal parameters in the epsilon regression RBF SVM model of `heart_scale`:

```
library(rpud)

hs <- read.svm.data(
+     "heart_scale", fac=FALSE)
x <- hs$x
y <- hs$y

c.log <- seq(-5, 11, by=2)
g.log <- seq(-13, 3, by=2)

nc <- length(c.log)
ng <- length(g.log)
mse <- matrix(nrow=nc, ncol=ng)

for (i in 1:nc) {
    for (j in 1:ng) {
        c <- 2^c.log[i]
        g <- 2^g.log[j]
        svm.model <- rpusvm(x, y,
            type="eps-regression",
              gamma=g, cost=c,
              scale=FALSE, cross=5)
        mse[i, j] = svm.model$tot.MSE
        cat("mse(", c, ",", g, ") =",
             mse[i, j], "\n")
    }
}

min.mse <- min(mse); mse
min.index <- which(mse == min.mse) - 1

log.cost  <-
    c.log[min.index %% nc + 1]
optimal.cost  <- 2^log.cost

log.gamma <-
    g.log[min.index %/% nc + 1]
optimal.gamma <- 2^log.gamma

cbind(log.cost, log.gamma,
    optimal.cost, optimal.gamma)

# contour map
plot(0, 0, type = "n",
    xlim = range(c.log),
    ylim = range(g.log),
    xlab = "log cost",
    ylab = "log gamma")
```

```
contour(x=c.log, y=g.log, mse,
    lty = "solid", add = TRUE,
    vfont = c("sans serif", "plain"))
```

30.8 Bayesian Classification with Gaussian Process

Despite prowess of the support vector machine, it is not specifically designed to extract features relevant to the prediction. For example, in network intrusion detection, we need to learn relevant network statistics for the network defense. In consumer credit rating, we would like to determine relevant financial records for the credit score. As for medical genetics research, we aim to identify genes relevant to the illness.

Bayesian learning has the **Automatic Relevance Determination** (ARD) capability built-in for this purpose. A particularly effective implementation is the variational Bayes approximation algorithm adopted in the R package vbmp. Using a Gaussian process prior on the function space, it is able to predict the posterior probability much more economically than plain MCMC.

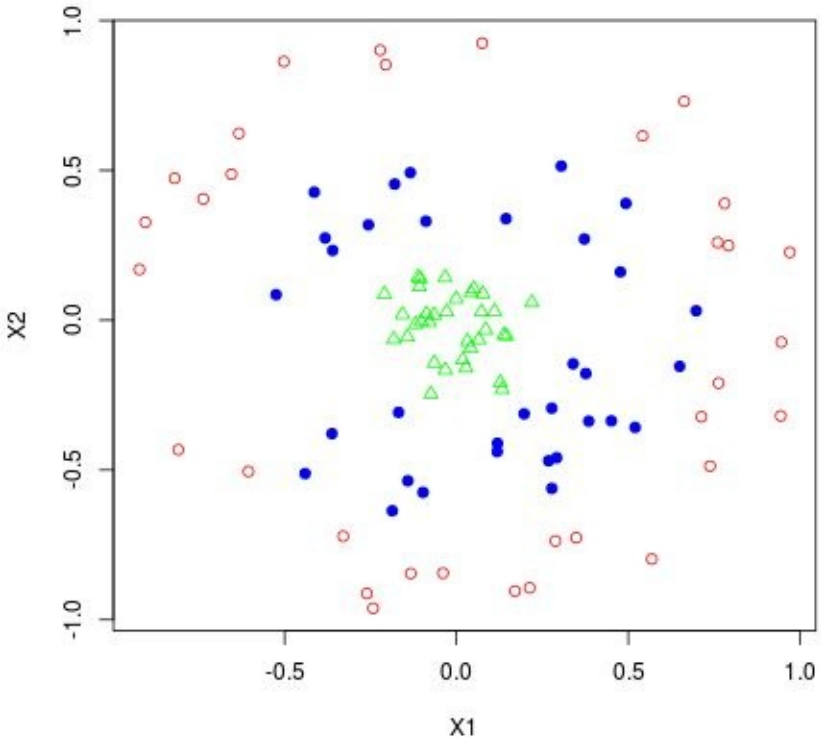
Example 1

For illustration, we begin with a toy example based on the `rvbm.sample.train` data set in `rpud`. The data set has two components, namely `X` and `t.class`. The first component `X` contains data points in a six dimensional Euclidean space, and the second component `t.class` classifies the data points of `X` into 3 different categories according to the squared sum of the first two coordinates of the data points. The other four coordinates in `X` serve only as noise dimensions. It is created with R code in the vbmp vignette.

We can visualize the data set is in a scatter plot as follows. The innermost layer is plotted in green triangles, the middle one is in blue solid dots, and the outermost layer is in red hollow dots.

```
> library(rpud)
> x1 <- rvbm.sample.train$X[, 1]
> x2 <- rvbm.sample.train$X[, 2]
> tc <- rvbm.sample.train$t.class

> plot(x1, x2, type="n",
+     xlab="X1", ylab="X2")
> points(x1[tc==1], x2[tc==1],
```

```
+   type="p", col="blue", pch=19)
> points(x1[tc==2], x2[tc==2],
+   type="p", col="red")
> points(x1[tc==3], x2[tc==3],
+   type="p", col="green", pch=24)
```



We will perform Gaussian process classification with the data set using the `vbmp` package. As it is in the Bioconductor repository, we have to install it with the `biocLite` remote script.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("vbmp")
```

Then we apply the `vbmp` method with the `bThetaEstimate` option for Automatic Relevance Determination. The training process takes about 200 seconds on an AMD X4 CPU.

```
> library(vbmp)
> system.time(res.vbmp <- vbmp(
+   rvbm.sample.train$X,
+   rvbm.sample.train$t.class,
```

```
+   rvbm.sample.test$X,
+   rvbm.sample.test$t.class,
+   theta = rep(1.,
+     ncol(rvbm.sample.train$X)),
+   control = list(
+     sKernelType="gauss",
+     bThetaEstimate=TRUE,
+     bMonitor=TRUE,
+     InfoLevel=1)
+   ))
      ......
  user  system elapsed
196.70    0.04  196.80
```

We then print out the covariance parameters with the `covParams` method, and note that the first two parameters are less than one, while the rest are all much larger than one. It thus indicates that only the first two parameters are relevant.

```
> covParams(res.vbmp)
[1] 1.979e-01 8.338e-02 3.009e+03
[4] 1.814e+03 2.245e+03 1.931e+03
```

Applying the `predError` method in `vbmp`, we found the error ratio to be 3.8%. Similarly, the `predLik` method shows that the posterior log likelihood is -0.3483.

```
> predError(res.vbmp)
[1] 0.038

> predLik(res.vbmp)
[1] -0.3483
```

We can run the same training process in much shorter time with the `rvbm` method in the `rpudplus` package. It takes about 20 seconds on an NVIDIA GTX 460 GPU.

```
> library(rpud)
> system.time(res.rvbm <- rvbm(
+   rvbm.sample.train$X,
+   rvbm.sample.train$t.class,
+   rvbm.sample.test$X,
+   rvbm.sample.test$t.class,
+   theta = rep(1.,
+     ncol(rvbm.sample.train$X)),
+   control = list(
+     sKernelType="gauss",
+     bThetaEstimate=TRUE,
+     bMonitor=TRUE,
+     InfoLevel=1)
```

```
+ ))
     ......
   user  system elapsed
 19.693   0.208  19.844

> summary(res.rvbm)
     ......

Covariance kernel hyperparameters:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      0     459    2020    1650    2440    3320

Posterior log likelihood:  -0.351
Prediction error rate:     3.8 %

> summary(model.rvbm)$covParams
[1] 2.073e-01 8.103e-02 3.324e+03
[4] 2.197e+03 2.517e+03 1.835e+03
```

Example 2

A more practical example is the `BRCA12` data set in `vbmp`. It contains the genetic test results of 30 cancer patients. Because of its large feature space with 8,080 genes and small sample size of 30, determining relevant genes is difficult with the support vector machine. With Bayesian methods, we do not have such constraint.

Let us load the data set into the workspace.

```
> library(vbmp)
> data(BRCA12)
```

As the data set is in the Bioconductor format, we need to install the `Biobase` package in order to extract the gene expression values. We then save the values in a new matrix `brca.x`. The matrix has 30 rows, each containing 8,080 gene expression values of a patient.

```
> source("http://bioconductor.org/biocLite.R")
> biocLite("Biobase")

> library(Biobase)
> brca.x <- t(exprs(BRCA12))
```

Next, we save the individual case categories of the patients in a vector `brca.y`.

```
> brca.y <- BRCA12$Target.class
```

Then we apply the `rvbm` method in `rpudplus` for the Gaussian process classification. Since the data lies in a high-dimensional Euclidean space, a linear kernel, instead of the usual Gaussian one, is more appropriate. On an GTX 460 GPU, the task takes about 2 minutes and a half to finish. Performing similar task with `vbmp` using the equivalent `iprod` kernel would take hours.

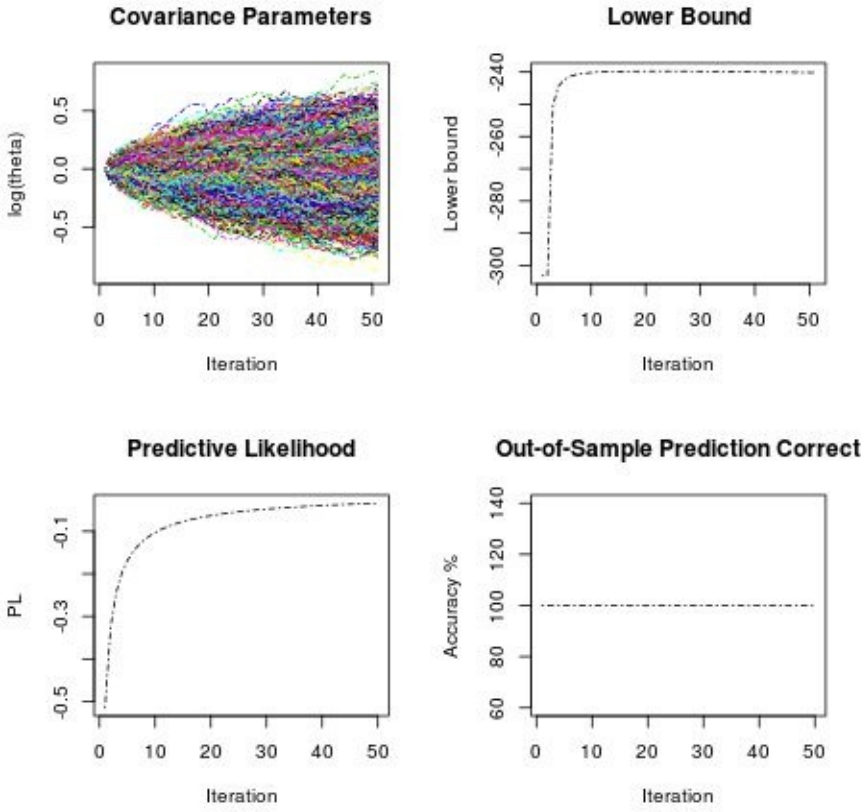
```
> library(rpud)
> system.time(brca.rvbm <- rvbm(
+   brca.x, brca.y,
+   brca.x, brca.y,
+   theta = rep(1.0, ncol(brca.x)),
+   control=list(
+     sKernelType="linear",
+     bThetaEstimate=TRUE,
+     bMonitor=TRUE,
+     InfoLevel=1)
+ ))
     ......
   user  system elapsed
148.562   3.656 152.205
```

The following indicates no extreme value in the kernel parameters, and confirms that all genes in the `BRCA12` data set are relevant in the study. The summary also shows that the posterior log likelihood is -0.0338, and the prediction error rate is zero.

```
> summary(brca.rvbm)
     ......

Covariance kernel hyperparameters:
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  0.391   0.839   0.978   1.000   1.140   2.390

Posterior log likelihood:  -0.03392
Prediction error rate:     0 %
```

Lastly we can plot the training history, and visually check the convergence of the lower bound estimate of marginal likelihood.

```
> plot(brca.rvbm)
```


Covariance Parameters
log(theta)
0.5
0.0
-0.5
0
10
20
30
40
50
Iteration
Lower Bound
Lower bound
-240
-260
-280
-300
0
10
20
30
40
50
Iteration
Predictive Likelihood
PL
-0.1
-0.3
-0.5
0
10
20
30
40
50
Iteration
Out-of-Sample Prediction Correct
Accuracy %
140
120
100
80
60
0
10
20
30
40
50
Iteration

30.9 Hierarchical Linear Model

Linear regression probably is the most familiar technique in data analysis, but its application is often hamstrung by model assumptions. For instance, if the data has a hierarchical structure, quite often the assumptions of linear regression are feasible only at local levels. We will investigate an extension of the linear model to bi-level hierarchies.

Example

We borrow an example from Rossi, Allenby and McCulloch (2005) for demonstration. It is based upon a data set called '`cheese`' from the `bayesm` package. The data set contains marketing data of certain brand name processed cheese, such as the weekly sales volume (`VOLUME`), unit retail price (`PRICE`), and display activity level (`DISP`) in various regional retailer accounts. For each account, we can define the following linear regression model of the log sales volume, where β_1 is the intercept term, β_2 is the display measure coefficient, and β_3 is the log price coefficient.

$$log(Volume) = \beta_1 + \beta_2 * Display + \beta_3 * log(Price) + \epsilon$$

The stochastic term ε relies on regional market conditions, and we would not expect it to have the same dispersion among retailers. For the same reason, we cannot expect identical regression coefficients for all accounts, or attempt to define a single linear regression model for the entire data set.

Nevertheless, we do expect regression coefficients of the retailer accounts to be related. A common approach to simulate the relationship is the hierarchical linear model, which treats the regression coefficients as random variables of yet another linear regression at the system level.

Problem

Fit the data set `cheese` with the hierarchical linear model, and estimate the average impact on sales volumes of the retailers if the unit retail price is to be

raised by 5%.

Solution

Our first task is to divide the data according to regional retailers. The following shows that the `RETAILER` column has the *factor* data type, which allow us to extract the entire list of retailer accounts with the `levels` method.

```
> library(bayesm)
> data(cheese)
> str(cheese)
'data.frame':  5555 obs. of 4 var ...
 $ RETAILER: Factor w/ 88 levels  ...
 $ VOLUME  : int 21374 6427 17302 ...
 $ DISP    : num 0.162 0.1241 ...
 $ PRICE   : num 2.58 3.73 2.71 ...

> retailer <-
+     levels(cheese$RETAILER)
> nreg <- length(retailer)
> nreg
[1] 88
```

Let *i* to be an integer between 1 and the number of retailer accounts. We define a filter for the *i*-th account as follows.

```
    filter <-
      cheese$RETAILER==retailer[i]
```

We now loop through the accounts, and create a list of data items consisting of the `X` and `y` components of the linear regression model in each account. The columns of `X` below contains the the intercept placeholder, the display measure, and log price data.

```
regdata <- NULL
for (i in 1:nreg) {
    filter <-
      cheese$RETAILER==retailer[i]

    y <- log(cheese$VOLUME[filter])
    X <- cbind(
      1,  # intercept placeholder
      cheese$DISP[filter],
      log(cheese$PRICE[filter]))
```

```
    regdata[[i]] <- list(y=y, X=X)
}
```

Finally, we wrap the `regdata` and the iteration parameters in lists, and invoke the `rhierLinearModel` method of the `bayesm` package. It takes less than 2 seconds for 2,000 MCMC iterations on a quad-core CPU.

```
> Data <- list(regdata=regdata)
> Mcmc <- list(R=2000)

> system.time(
+ out <- bayesm::rhierLinearModel(
+         Data=Data,
+         Mcmc=Mcmc))
.....................
   user  system elapsed
  2.374   3.804   1.639
```

We can perform the same MCMC simulation with an identically named method in `rpudplus`. The process finishes at about the same rate. The speedup will be more pronounced for larger data sets such as the one in **Exercise 3** below. Note the extra `output` option that we explicitly set to be '`bayesm`' for compatibility.

```
> library(rpud)
> system.time(
+ out <- rpud::rhierLinearModel(
+         Data=Data,
+         Mcmc=Mcmc,
+         output="bayesm"))
.....................
   user  system elapsed
  0.131   1.090   1.249
```

Observe that the log price data is in the third column of the `X` component. Hence we can estimate the log price coefficient from the third component of the second dimension in the `betadraw` attribute of the MCMC output. We also drop the first 10% for burn-in, *i.e.* 200 MCMC draws, before evaluating the mean of the simulation values.

```
> beta.3 <- mean(as.vector(
+   out$betadraw[, 3, 201:2000]))
> beta.3
[1] -2.146
```

A 5% increase of the unit price amounts to an increment of $log(1.05)$ in the log

price. Therefore the log sales volume will be adjusted by (-2.146) * *log*(1.05). The computation below shows that the sales volume is expected to decrease by 10% on average.

```
> exp(beta.3 * log(1.05))
[1] 0.9006
```

Exercise 1

Confirm MCMC convergence in the simulation of the hierarchical linear model of the `cheese` data set. As a hint, there is a '`coda`' output option in the `rpud::rhierLinearModel` method for this purpose. Make sure the `coda` package is installed beforehand.

Solution

With the `coda` package *installed*, we run the `rpud::rhierLinearModel` method again with the '`coda`' output option.

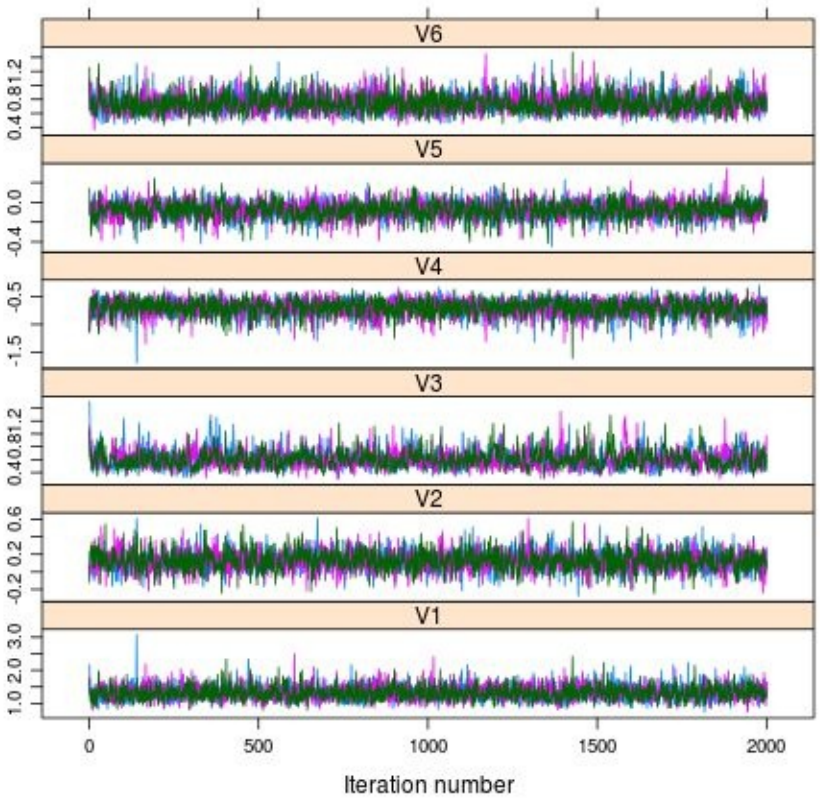
```
> out <- rpud::rhierLinearModel(
+   Data=Data,
+   Mcmc=Mcmc,
+   output="coda")
```

The result consists of four components, `Vbeta.mcmc`, `Delta.mcmc`, `tau.mcmc`, and `beta.mcmc`.

```
> attributes(out)
$names
[1] "Vbeta.mcmc" "Delta.mcmc"
[3] "tau.mcmc" "beta.mcmc"
```

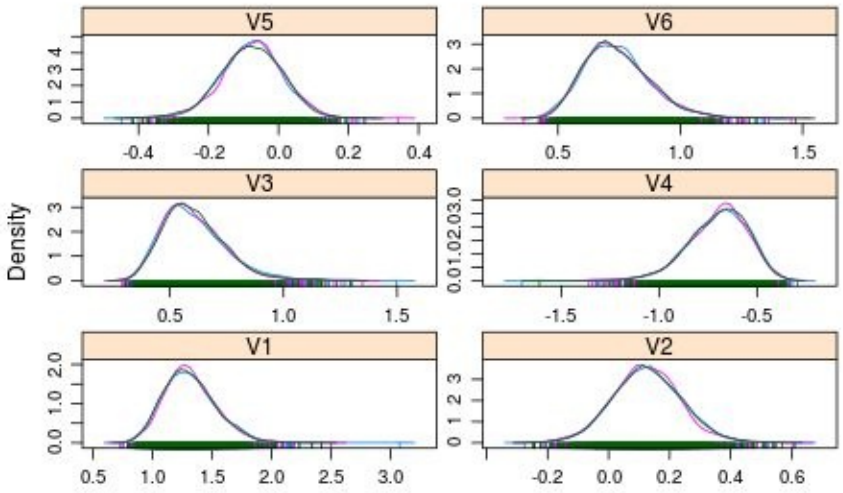
We need to confirm MCMC convergence for each of the component. For example, to perform convergence diagnostic for `Vbeta.mcmc`, we examine its `xyplot`, and observe well-mixed graphs with no obvious upward or downward trends.

```
> xyplot(out$Vbeta.mcmc)
```



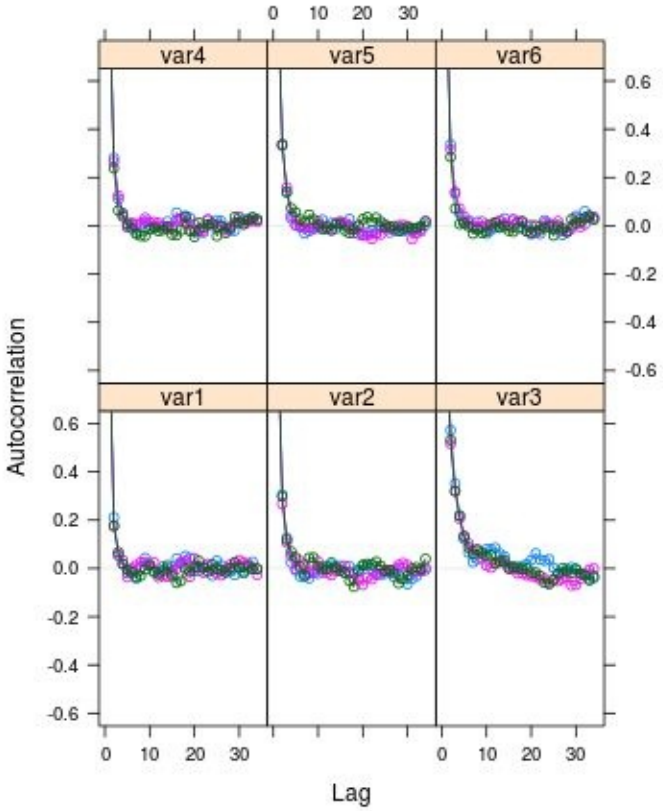
We also see in the density plot that the simulation chains of `Vbeta.mcmc` have almost completely overlapped common posterior distribution.

```
> densityplot(out$Vbeta.mcmc)
```



Furthermore, the auto-correlation graphs of the `Vbeta.mcmc` chains converge rapidly to zero in the acf plot.

```
> acfplot(out$Vbeta.mcmc)
```

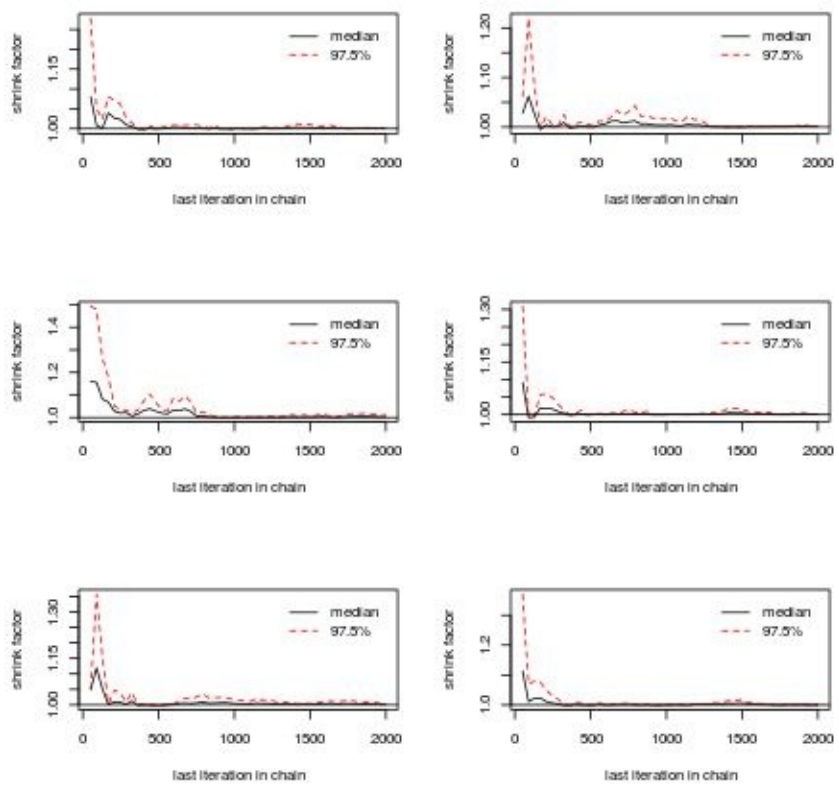


Lastly, the Gelman-Rubin diagnostic shows that the shrink factors are all below 1.05.

```
> gelman.diag(out$Vbeta.mcmc)
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]          1       1.00
[2,]          1       1.00
[3,]          1       1.01
[4,]          1       1.00
[5,]          1       1.00
[6,]          1       1.00

Multivariate psrf

1
```

We can also use the Gelman-Rubin-Brooks plot to visualize the convergence.

```
> gelman.plot(out$Vbeta.mcmc)
```



Similarly, we can confirm the convergence of `Delta.mcmc`. Since the `tau.mcmc` and `beta.mcmc` components have high dimensions, we might just want to sample some of their dimensions for convergence analysis.

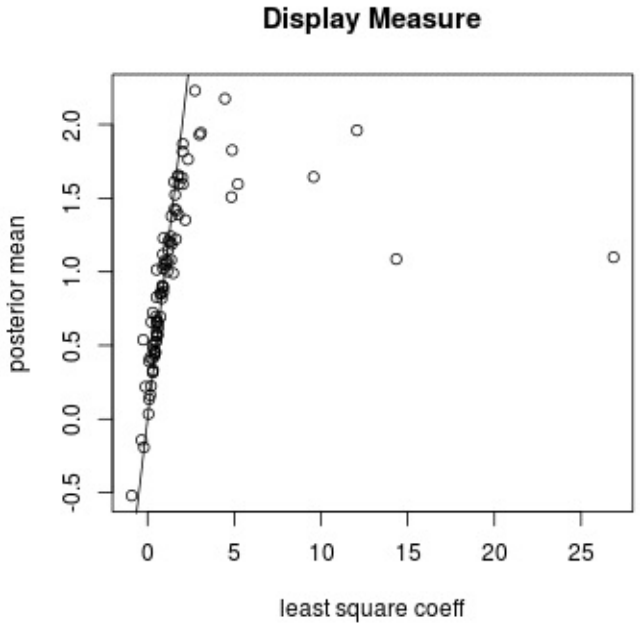
If we are in a hurry, the least we can do is to verify the *multivariate potential scale reduction factors* (`mpsrf`) to be less than 1.05. The `mpsrf` value of the last component in `beta.mcmc` suggests that we might need to increase the number of MCMC iterations to ensure convergence. Increasing the number of iterations to 4,000 would be sufficient here.

```
> Vbeta.res <-
+   gelman.diag(out$Vbeta.mcmc)
> Vbeta.res$mpsrf
[1] 1.003
>
> Delta.res <-
+   gelman.diag(out$Delta.mcmc)
> Delta.res$mpsrf
[1] 1.003
>
> tau.res <-
```

```
+   gelman.diag(out$tau.mcmc)
> tau.res$mpsrf
[1] 1.029
>
> beta.res <-
+   gelman.diag(out$beta.mcmc)
> beta.res$mpsrf
[1] 1.078
```

Exercise 2

Create a plot for the posterior mean of display measures and the matching least square coefficients of the `cheese` data set as illustrated in Fig 3.6 of Rossi, Allenby and McCulloch (2005). Define the Gibbs priors to be $\nu = 6$ and $V = 0.6$ I_3. Consult the text for an explanation of the prior parameters. Observe outliers of the least square coefficients.



Solution

Let `X` and `y` to be components of an item in `regdata` created previously in the example. We will create a linear regression model with the formula `y~X-1`. The extra `-1` term in the formula excludes implicit interception in the linear regression, as `X` already has it explicitly represented in the first column. We save the regression coefficients of the result in a matrix called `ls.regrs`. It has 88

columns, each of which consists of the least square coefficients of corresponding linear regression of the retailer account. Furthermore, the three matrix rows are the intercept coefficients, display measure effect, and log price effect estimates of the retailer accounts.

```
> ls.regrs <- sapply(regdata,
+   function(item) {
+     y <- item$y
+     X <- item$X
+     coefficients(lm(y ~ X-1))
+   })

> dim(ls.regrs)
[1]  3 88
```

We let `index` to be the row index of display measures, which is just equal to 2, and we save the corresponding least square coefficients in a vector called `ls.display`.

```
> index <- 2  # display measure
> ls.display <- ls.regrs[index,]
```

Denote the number of covariates in the linear regression by `nvar`. The default *v* prior in a hierarchical linear model is `nvar+3`. In our example, `nvar` is 3, hence the default *v* prior is 6. We also define the *V* prior to be 0.1*v* times the identity matrix of rank 3. The `rpud` runtime will implicitly promote a number to a diagonal matrix, and we can simply set *V* to be the numeric value of 0.6.

```
> # gibbs prior
> nvar <- ncol(regdata[[1]]$X)
> nu <- nvar+3; nu
[1] 6
> V <- 0.1*nu; V
[1] 0.6
> Prior <- list(nu=nu, V=V)
```

We now apply the `rpud::rhierLinearModel` method with the given prior parameters.

```
> out <- rpud::rhierLinearModel(
+   Data=Data,
+   Prior=Prior,
+   Mcmc=Mcmc,
+   output="bayesm")
```

In order to find the posterior mean, we remove the first 10% of the MCMC draws for burn-in, and compute the posterior mean of display measure from the `index` component in the second dimension of `betadraw`.

```
> R <- Mcmc$R
> burnin <- trunc(0.1*R)
> filter <- (burnin+1):R
> post.display <- apply(
+   out$betadraw[,index,filter],
+   1, mean)
```

Now we can create a scatter plot of the least square coefficients (`ls.display`) and posterior mean of the display measure (`post.display`).

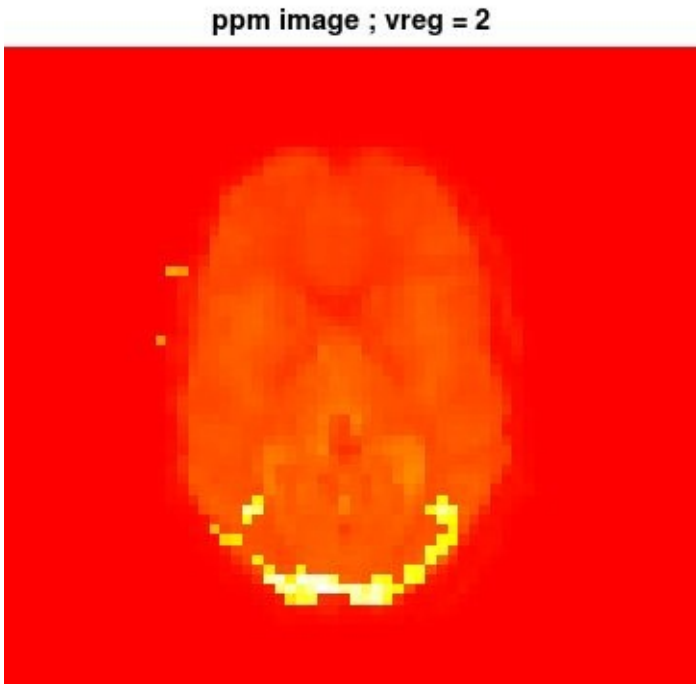
```
> plot(ls.display, post.display,
+   xlab="least square coeff",
+   ylab="posterior mean",
+   main="Display Measure",
+   pch=21)
```

Finally, we draw a line through the origin with unit slope for reference.

```
> lines(c(-15, 15), c(-15, 15))
```

Exercise 3

The `cudaBayesreg` package employs a hierarchical linear model for analysis of fMRI data. There is sample code in Silva (2010) that creates a brain activity image as below. Replace the `cudaMultireg.slice` method in the sample code by `rpud::rhierLinearModel` for the same effect.



Solution

Assume both the `cudaBayesreg` and `cudaBayesregData` packages are successfully installed. We can load the `fmri` data with the `read.fmrislice` method as follows, and thus find the design matrix from the `X` component of the output `slicedata`. We can also find the number of covariates (`nvar`) and the number of observations in each regression (`nobs`). Incidentally, these are the matrix dimensions of the design matrix `X`.

```
> library("cudaBayesreg")

> # load from cudaBayesregData
> slicedata <- read.fmrislice(
+   fbase = "fmri", slice = 3,
+   swap = TRUE)
Using slice n. 3

> X <- slicedata$X
> nvar <- slicedata$nvar; nvar
[1] 5
> nobs <- slicedata$nobs; nobs
[1] 45
```

We now read the pre-defined mask from `slicedata` using the method `ymaskdata`, and find the voxel matrix from its `yn` component. It must have `nobs` rows.

```
> ymaskdata <- premask(slicedata)
> yn <- ymaskdata$yn
> stopifnot(nobs == nrow(yn))
```

As shown below, for this particular data set, there are 1525 linear regressions involved. We collect the `X` and `y` components of the regressions in a list called `regdata`. Note that we use the same design matrix for all regressions in an fMRI analysis.

```
> nreg <- ymaskdata$nreg; nreg
[1] 1525
> regdata <- NULL
> for (i in 1:nreg) {
+   regdata[[i]] <- list(
+     y=yn[,i],
+     X=X)    # same design matrix
+ }
```

Then we assign the same MCMC parameters as in `cudaBayesreg` and apply the `rpud::rhierLinearModel` method using `bayesm` output option. It takes about 6 seconds on a medium speed CPU.

```
> # MCMC parameters
> R <- 3000
> keep <- 5
>
> Data <- list(regdata=regdata)
> Mcmc <- list(R=R,keep=keep)
>
> system.time(
+   out <- rpud::rhierLinearModel(
+   Data=Data, Mcmc=Mcmc,
+   output="bayesm"))
.............................
   user  system elapsed
  6.169   0.296   6.478
```

Finally, we can print out the brain activity image using the `post.ppm` method.

```
> post.ppm(out = out, vreg = 2,
+   slicedata = slicedata,
+   ymaskdata = ymaskdata,
+   col = heat.colors(256))
```

Although `rpud::rhierLinearModel` might be a tad slower than `cudaMultireg.slice`, the former is designed for general purpose usage, and is

much more versatile.

Note

Due to sequential nature of the MCMC algorithm and modest data size, the current implementation of the `rpud::rhierLinearModel` method is mostly CPU bound. Its only CUDA dependency is the random number generator for MCMC simulation. There is plenty of room for optimization with large data sets.

30.10 Hierarchical Multinomial Logit

A **multinomial logit model** describes the probability of selecting a choice among *p* alternatives, and is an important tool for marketing research. It can be expressed in the following formula, where V_i (i = 0, …, p-1) are linear functions of some covariates. Note that if we increment every V_i by a common constant, it will not affect the probability estimate P_i at all.

$$P_i = exp(V_i) / \sum_j exp(V_j)$$

Example

The data set '`margarine`' of the `baysem` package contains household purchase statistics of several margarine brands. We will use the multinomial logit model to analyze the customer purchase probability based on product retail price and customer demographic data.

We start by loading the data set and examine its two components, '`choicePrice`' and '`demos`'.

```
> library(bayesm)
> data(margarine)
> attributes(margarine)
$names
[1] "choicePrice" "demos"
```

The first component '`choicePrice`' is a data frame that records the margarine purchase. Its first column '`hhid`' identifies the customer households with artificial IDs. The second column '`choice`' contains the purchase choice made by the customer. The remaining columns are prices of available margarine brands at the time of purchase.

```
> str(margarine$choicePrice)
'data.frame': 4470 obs. of 12 var ...
 $ hhid    : int  2100016 2100016 ...
 $ choice  : num  1 1 1 1 1 4 1 1 ...
 $ PPk_Stk : num  0.66 0.63 0.29  ...
```

```
 $ PBB_Stk : num  0.67 0.67 0.5   ...
 $ PFl_Stk : num  1.09 0.99 0.99  ...
 $ PHse_Stk: num  0.57 0.57 0.57  ...
 $ PGen_Stk: num  0.36 0.36 0.36  ...
 $ PImp_Stk: num  0.93 1.03 0.69  ...
 $ PSS_Tub : num  0.85 0.85 0.79  ...
 $ PPk_Tub : num  1.09 1.09 1.09  ...
 $ PFl_Tub : num  1.19 1.19 1.19  ...
 $ PHse_Tub: num  0.33 0.37 0.59  ...
```

The second data frame '`demos`' contains the household demographic data. Its first column ('`hhid`') is the household ID, the second column ('`Income`') is the income level, and fifth column ('`Fam_Size`') is the family size.

```
> str(margarine$demos)
'data.frame': 516 obs. of  8 var...
 $ hhid     : num  2100016 2100024 ..
 $ Income   : num  32.5 17.5 37.5 ...
 $ Fs3_4    : int  0 1 0 0 0 0 0 ...
 $ Fs5.     : int  0 0 0 0 0 0 0 ...
 $ Fam_Size : int  2 3 2 1 1 2 2 ...
 $ college  : int  1 1 0 0 1 0 1 ...
 $ whtcollar: int  0 1 0 1 1 0 0 ...
 $ retired  : int  1 1 1 0 0 1 0 ...
```

Following an example in Rossi, Allenby and McCulloch (2005), we will track only the six margarine brands with major market shares, namely, Parkay (`PPk_Stk`), BlueBonnett (`PBB_Stk`), Fleischmanns (`PFl_Stk`), house brand (`PHse_Stk`), generic brand (`PGen_Stk`), and Shed Spread (`PSS_Tub`).

To do so, we apply the `subset` method to select relevant columns in `choicePrice`. Hence we include the data columns from `hhid` to `PSS_Tub`, but skip `PImp_Stk` and the rest. We save the result in a new data frame called `chpr`, and print out its first few rows as example.

```
> # select columns
> chpr <- subset(
+   margarine$choicePrice,
+   select=(hhid:PSS_Tub)[-PImp_Stk])

> head(chpr)
     hhid choice PPk_Stk PBB_Stk
1 2100016      1    0.66    0.67
2 2100016      1    0.63    0.67
3 2100016      1    0.29    0.50
4 2100016      1    0.62    0.61
```

```
5 2100016      1    0.50    0.58
6 2100016      4    0.58    0.45

  PFl_Stk PHse_Stk PGen_Stk PSS_Tub
1    1.09     0.57     0.36    0.85
2    0.99     0.57     0.36    0.85
3    0.99     0.57     0.36    0.79
4    0.99     0.57     0.36    0.85
5    0.99     0.45     0.33    0.85
6    0.99     0.45     0.33    0.85
```

Because we have fewer margarine brands in `chpr`, we need to eliminate unrelated purchase records. Hence we apply the `subset` method again with the `choice` values restricted to 1, …, 5, and 7.

```
> # select rows
> chpr <- subset(chpr,
+     choice %in% c(1:5,7))
```

Moreover, to reflect the new column position of `PSS_Tub`, we need to patch values in the `choice` column.

```
> # patch choice values
> L <- chpr[,2]==7
> chpr[L,2] <- 6
```

Now we need to define the multinomial logit model of individual household purchase. We first create score functions of the purchase probability in terms of the log price as below. The coefficients β_i (i = 0, …, 5) are intercept terms, and β is the log price effect. Obviously, the coefficients are dependent on individual household demographic.

$$\begin{aligned}
V_0 &= \beta_0 + \beta \times log(PPk_Stk) \\
V_1 &= \beta_1 + \beta \times log(PBB_Stk) \\
V_2 &= \beta_2 + \beta \times log(PFl_Stk) \\
V_3 &= \beta_3 + \beta \times log(PHse_Stk) \\
V_4 &= \beta_4 + \beta \times log(PGen_Stk) \\
V_5 &= \beta_5 + \beta \times log(PSS_Tub)
\end{aligned}$$

If we assume that the margarines have the *same* price, then we have

$$\begin{aligned}
log(P_1/P_0) &= V_1 - V_0 \\
&= \beta_1 - \beta_0
\end{aligned}$$

Since we can adjust the score functions V_i up to a common constant, we can further assume the intercept term of V_0 to be zero, *i.e.*, $\beta_0 = 0$. Hence $\beta_1 = log(P_1/P_0)$ is the log ratio of the purchase probabilities between BlueBonnett (`PBB_Stk`) and Parkay (`PPk_Stk`) margarines should they be sold at the *same* price.

We can similarly interpret the other coefficients β_2, …, β_5. Evidently, the remaining β coefficient reflects price sensitivity. Treating β_1, …, β_5, and β as unknown variables, the coefficient matrix of the linear system above becomes:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & log(PPk\_Stk) \\ 1 & 0 & 0 & 0 & 0 & log(PBB\_Stk) \\ 0 & 1 & 0 & 0 & 0 & log(PFl\_Stk) \\ 0 & 0 & 1 & 0 & 0 & log(PHse\_Stk) \\ 0 & 0 & 0 & 1 & 0 & log(PGen\_Stk) \\ 0 & 0 & 0 & 0 & 1 & log(PSS\_Tub) \end{bmatrix}$$

The values of β_1, …, β_5, and β are dependent on household profile. One approach to estimate them collectively is the **hierarchical multinomial logit model**, which treats the coefficients in question as dependent variables of a global linear regression governed by demographic data.

Problem

Consider only households with five or more purchase records in the data set `margarine`. Apply the hierarchical multinomial logit model and estimate the 95% credible interval of the mean log price effect β.

Solution

To consider only households with at least 5 purchase records, we first summarize the household purchase counts. The result shows that household '2100016' has 7 purchase records, household '2100024' has 10, household '2100495' has 11, and so forth ...

```
> hhid.nobs <-
+     table(chpr[,"hhid"])

> head(hhid.nobs)
2100016 2100024 2100495 ...
      7      10      11 ...
```

Then we save the household IDs with at least 5 purchase counts in a new vector called `hhid.target`. We denote its length by `nlgt`.

```
hhid.upper <-
    hhid.nobs[hhid.nobs >= 5]
hhid.target <- as.integer(
    dimnames(hhid.upper)[[1]])
nlgt <- length(hhid.target)
```

Let *i* to be an integer between 1 and `nlgt`. We can define a filter for the *i*-th household in `hhid.target`, and extract its log price data from `chpr`. We denote the result by `Xa`.

```
id <- hhid.target[i]
filter <- (chpr[,1]==id)
Xa <- log(chpr[filter, c(-1,-2)])
```

Denote the number of choices by `p`, we then create a design matrix `X` using the `createX` method in `bayesm`.

```
p <- ncol(chpr)-2
X <- createX(
    p  = p,
    na = 1,
    Xa = Xa,
    nd = NULL,
    Xd = NULL,
    base = 1)
```

To illustrate the design matrix with a concrete example, we set $i = 1$, and print out the first few rows of the design matrix `X` as below. We found that it matches the coefficient matrix of the linear system above with $\beta_0 = 0$. The last column of `X` is the log price data from `Xa`.

```
> i <- 1
> id <- hhid.target[i]
> filter <- (chpr[,1]==id)

> Xa <- log(
+     chpr[filter, c(-1,-2)])
> Xa[1,]
  PPk_Stk PBB_Stk PFl_Stk PHse_Stk
1 -0.4155 -0.4005 0.08618  -0.5621
  PGen_Stk PSS_Tub
1   -1.022 -0.1625
```

```
> X <- createX(
+     p  = p,
+     na = 1,
+     Xa = Xa,
+     nd = NULL,
+     Xd = NULL,
+     base = 1)

> head(X, n=p)
     [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    0    0    0
[2,]    1    0    0    0    0
[3,]    0    1    0    0    0
[4,]    0    0    1    0    0
[5,]    0    0    0    1    0
[6,]    0    0    0    0    1
         [,6]
[1,] -0.41552
[2,] -0.40048
[3,]  0.08618
[4,] -0.56212
[5,] -1.02165
[6,] -0.16252
```

In general, we can loop through the households in `hhid.target`, and create the input data for use by MCMC simulation.

```
lgtdata <- NULL
for (i in 1:nlgt) {
    id <- hhid.target[i]
    filter <- (chpr[,1]==id)

    # log price
    Xa <- log(
        chpr[filter, c(-1,-2)])

    # design matrix
    X <- createX(
        p  = p,
        na = 1,
        Xa = Xa,
        nd = NULL,
        Xd = NULL,
        base = 1)

    # choice
    y <- chpr[filter, 2]
    names(y) <- NULL
```

```
    lgtdata[[i]] <- list(
        y=y, X=X, hhid=id)
}
```

Lastly, we extract the demographic data of the target households in another matrix called `Z`. We also take logarithm of Z and center it with the column means.

```
demos <- as.matrix(
    margarine$demos[,c(1,2,5)])
Z <- matrix(0, nrow=nlgt, ncol=2)
for (i in 1:nlgt){
    id <- lgtdata[[i]]$hhid
    filter <- (demos[,1]==id)
    Z[i,] <- demos[filter, 2:3]
}

Z  <- log(Z)
zm <- apply(Z, 2, mean)
Z  <- sweep(Z, 2, zm)
```

Now we can apply the `rpud::rhierMnlRwMixture` method for MCMC simulation. It takes about 40 seconds on an AMD Phenom II X4 CPU for 20,000 iterations. Similar to our implementation of the hierarchical linear model, the current `rpud::rhierMnlRwMixture` method is mostly CPU bound.

```
keep <- 5
R <- 20000
mcmc1 <- list(keep=keep, R=R)

library(rpud)
system.time(
    out <- rpud::rhierMnlRwMixture(
        Data  = list(p=p,
            lgtdata=lgtdata, Z=Z),
        Prior = list(ncomp=1),
        Mcmc  = mcmc1
))
```

Answer

We can find MCMC samples of the mean parameters of β_1, …, β_5, β in the `mudraw` output component. We need to drop the first 10% for burn-in, *i.e.* 400 MCMC draws, before evaluating the 95% credible interval of the mean log price effect β. The result turns out to be [-4.385, -3.684], which is definitely below zero. It confirms our expectation that raising the price would lower purchase probability.

```
> beta.post <-
+    out$mudraw[401:4000, 6]

> quantile(beta.post, c(.025, .975))
  2.5%  97.5%
-4.385 -3.684

> summary(beta.post)
   Min. 1st Qu.  Median    Mean
  -4.69   -4.14   -4.02   -4.03
3rd Qu.    Max.
  -3.91   -3.48
```

Source Listing

```
library(bayesm)

data(margarine)
attributes(margarine)

str(margarine$choicePrice)
str(margarine$demos)

# select columns
chpr <- subset(
  margarine$choicePrice,
  select=(hhid:PSS_Tub)[-PImp_Stk])
head(chpr)

# select rows
chpr <- subset(chpr,
    choice %in% c(1:5,7))

# patch choice column
L <- chpr[,2]==7
chpr[L,2] <- 6

# row filter
hhid.nobs <-
    table(chpr[,"hhid"])
head(hhid.nobs, n=4)

hhid.upper <-
    hhid.nobs[hhid.nobs >= 5]
hhid.target <- as.integer(
    dimnames(hhid.upper)[[1]])
nlgt <- length(hhid.target); nlgt

# number of choices
```

```
p <- ncol(chpr)-2; p

# logit data
lgtdata <- NULL
for (i in 1:nlgt) {
    id <- hhid.target[i]
    filter <- (chpr[,1]==id)

    # log price
    Xa <- log(
        chpr[filter, c(-1,-2)])

    # design matrix
    X <- createX(
        p  = p,
        na = 1,
        Xa = Xa,
        nd = NULL,
        Xd = NULL,
        base = 1)

    # choice
    y <- chpr[filter, 2]
    names(y) <- NULL

    lgtdata[[i]] <- list(
        y=y, X=X, hhid=id)
}

# demographic data
demos <- as.matrix(
    margarine$demos[,c(1,2,5)])
Z <- matrix(0, nrow=nlgt, ncol=2)
for (i in 1:nlgt){
    id <- lgtdata[[i]]$hhid
    filter <- (demos[,1]==id)
    Z[i,] <- demos[filter, 2:3]
}

Z  <- log(Z)
zm <- apply(Z, 2, mean)
Z  <- sweep(Z, 2, zm)

# MCMC
keep <- 5
R <- 20000
mcmc1 <- list(keep=keep, R=R)

library(rpud)
system.time(
```

```
    out <- rpud::rhierMnlRwMixture(
        Data  = list(p=p,
            lgtdata=lgtdata, Z=Z),
        Prior = list(ncomp=1),
        Mcmc  = mcmc1
))

beta.post <- out$mudraw[401:4000, 6]
quantile(beta.post, c(.025, .975))
summary(beta.post)
```

Exercise

Investigate MCMC convergence of the mean parameters of the coefficients β_1, …, β_5, β in the hierarchical multinomial logit model of `margarine`.

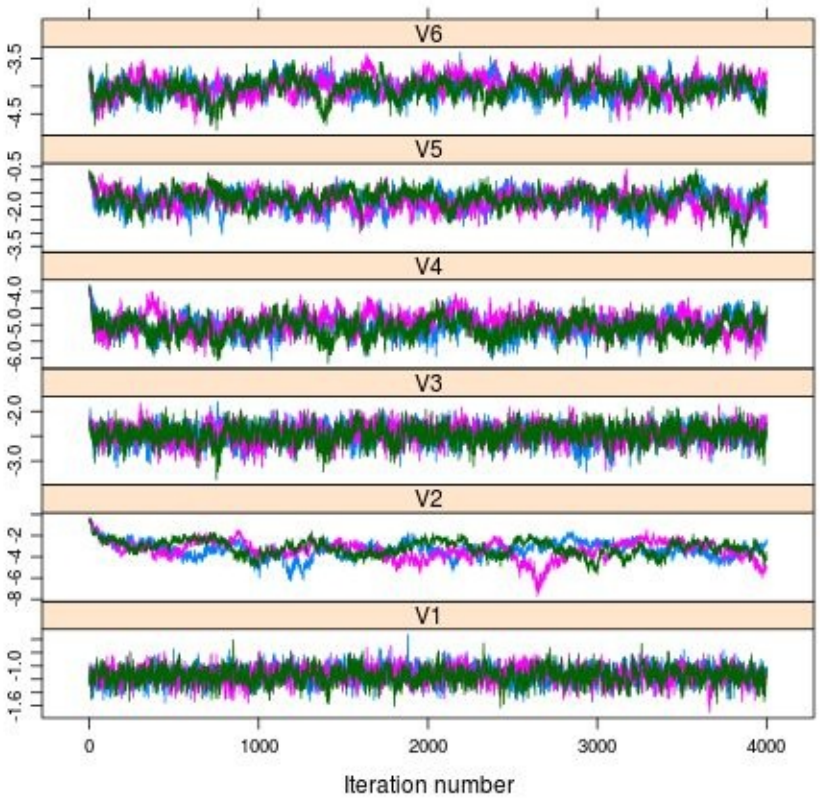
Solution

With the `coda` package *installed*, we run the `rpud::rhierMnlRwMixture` method again using the '`coda`' output option.

```
> out <- rpud::rhierMnlRwMixture(
+   Data = list(p=p,
+       lgtdata=lgtdata, Z=Z),
+   Prior  = list(ncomp=1),
+   Mcmc   = mcmc1,
+   output = "coda")

> attributes(out)
$names
[1] "loglike.mcmc" "beta.mcmc"
[3] "Delta.mcmc"   "prob.mcmc"
[5] "mu.mcmc"      "root.mcmc"
```

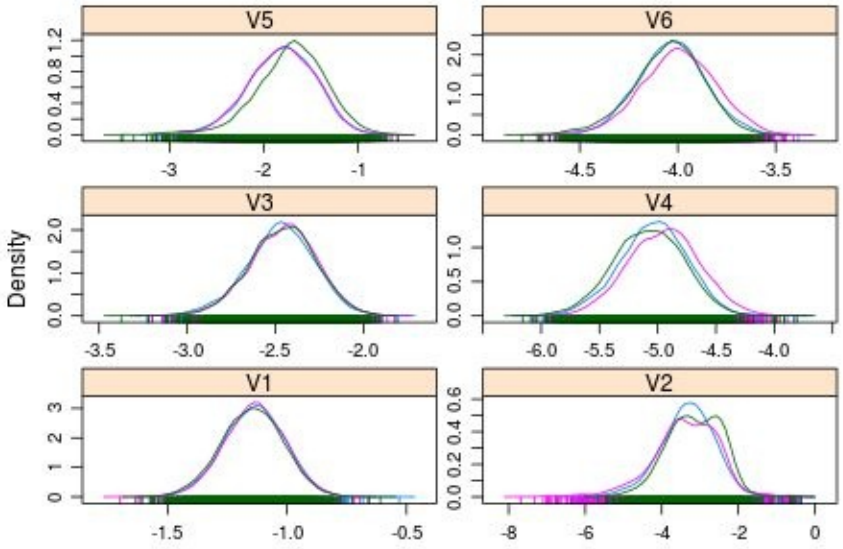
We can retrieve the posterior MCMC samples of the mean parameters of the coefficients from the `mu.mcmc` component of the output. By running the `xyplot`, we found the MCMC chains are pretty well-mixed except for β_2.

```
> xyplot(out$mu.mcmc)
```



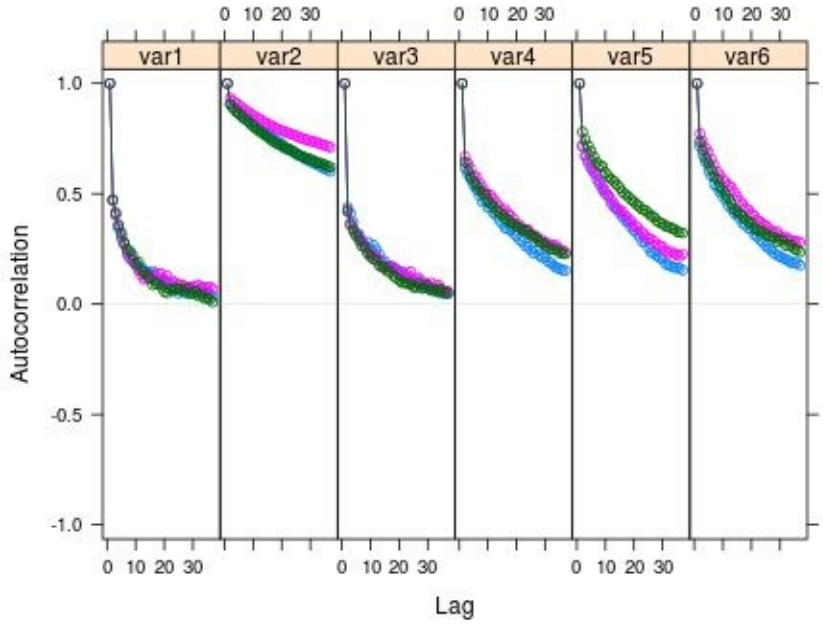
In addition, the posteriors of the MCMC chains in the density plots almost completely overlap except for β_2.

```
> densityplot(out$mu.mcmc)
```



The acf plot shows that the auto-correlations all fall below 0.5 with increasing lags except for β_2.

```
> acfplot(out$mu.mcmc)
```

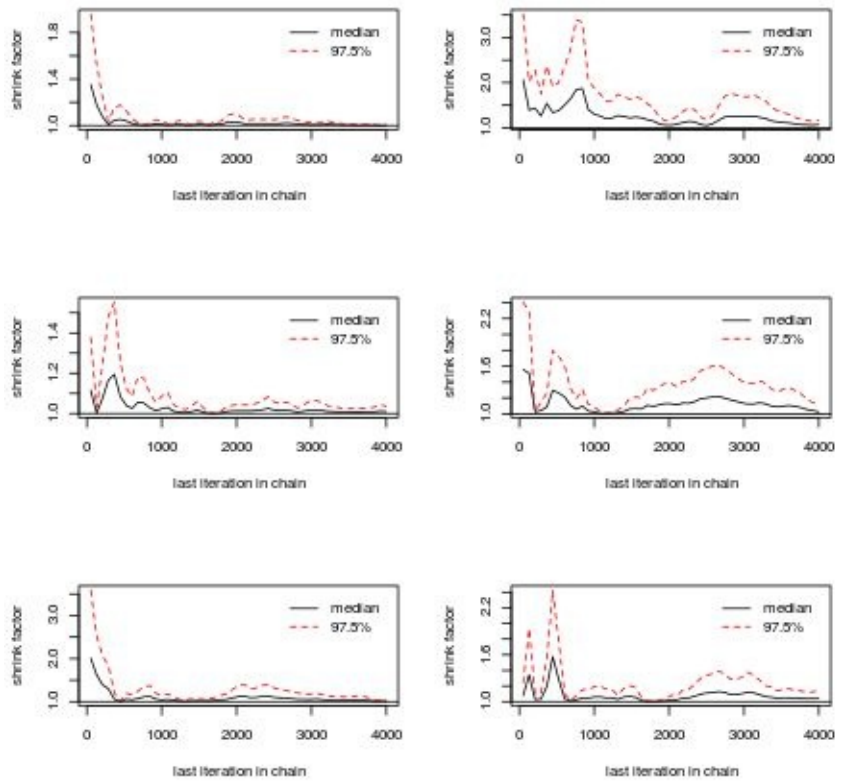


Lastly, the Gelman-Rubin diagnostic shows that the highest shrink factor is about 1.07 for β_2, and the multivariate shrink factor is 1.09.

```
> gelman.diag(out$mu.mcmc)
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]       1.00       1.00
[2,]       1.07       1.18
[3,]       1.01       1.04
[4,]       1.03       1.10
[5,]       1.02       1.04
[6,]       1.04       1.14

Multivariate psrf

1.09
```

We can also use the Gelman-Rubin-Brooks plot to visualize the result.

```
> gelman.plot(out$mu.mcmc)
```



Hence the result suggests likely MCMC convergence with 20,000 iterations for the mean parameters of β_1, …, β_5, and β. The only exception is β_2, which is marginally acceptable.

Appendix A
Installing GPU Packages

A1. Installing CUDA 7.5 on Fedora 21

The following explains how to install CUDA Toolkit 7.5 on 64-bit Fedora 21 Linux. I have tested it on a self-assembled desktop with NVIDIA GeForce GTX 550 Ti graphics card. The instruction assumes you have the necessary CUDA compatible hardware support and know how to use `sudo`. Depending on your system configuration, your mileage may vary.

CUDA Repository

Retrieve the CUDA repository package for Fedora 21 from the CUDA download site and install it in a terminal.

```
$ sudo rpm -Uhv \
 cuda-repo-fedora21-7.5-18.x86_64.rpm

$ sudo yum clean expire-cache
```

RPMFusion Repository

The CUDA driver relies on an external software framework, and you have to download and configure the RPMFusion free repository for the package dependency.

```
$ sudo rpm -Uhv \
 rpmfusion-free-release-21.noarch.rpm
```

Linux Development Tools

You also need to install the necessary background Linux tools.

```
$ sudo yum install \
    gcc-c++ kernel-devel
```

CUDA Toolkit

Then you can install the CUDA Toolkit using `yum`.

```
$ sudo yum install cuda
```

You should reboot the system afterward and verify the driver installation with the `nvidia-settings` utility.

Environment Variables

As part of the CUDA environment, you should add the following in the `.bashrc` file of your home folder.

```
export CUDA_HOME=/usr/local/cuda-7.5
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64

PATH=${CUDA_HOME}/bin:${PATH}
export PATH
```

CUDA SDK Samples

Now you can copy the SDK samples into your home directory, and build a test sample.

```
$ cuda-install-samples-7.5.sh  ~

$ cd ~/NVIDIA_CUDA-7.5_Samples

$ cd 1_Utilities/deviceQuery

$ make
```

If everything goes well, you should be able to verify your CUDA installation by running the `deviceQuery` sample.

A2. Installing CUDA 7.5 on Ubuntu 14.04

The following explains how to install CUDA Toolkit 7.5 on 64-bit Ubuntu 14.04 Linux. I have tested it on a self-assembled desktop with NVIDIA GeForce GTX 550 Ti graphics card. The instruction assumes you have the necessary CUDA compatible hardware support. Depending on your system configuration, your mileage may vary.

CUDA Repository

Retrieve the CUDA repository package for Ubuntu 14.04 from the CUDA download site and install it in a terminal.

```
$ sudo dpkg -i \
 cuda-repo-ubuntu1404_7.5-18_amd64.deb

$ sudo apt-get update
```

CUDA Toolkit

Then you can install the CUDA Toolkit using `apt-get`.

```
$ sudo apt-get install cuda
```

You should reboot the system afterward and verify the driver installation with the `nvidia-settings` utility.

Environment Variables

As part of the CUDA environment, you should add the following in the `.bashrc` file of your home folder.

```
export CUDA_HOME=/usr/local/cuda-7.5
export LD_LIBRARY_PATH=${CUDA_HOME}/lib64

PATH=${CUDA_HOME}/bin:${PATH}
export PATH
```

CUDA SDK Samples

Now you can copy the SDK samples into your home directory, and build a test sample.

```
$ cuda-install-samples-7.5.sh  ~

$ cd ~/NVIDIA_CUDA-7.5_Samples

$ cd 1_Utilities/deviceQuery

$ make
```

If everything goes well, you should be able to verify your CUDA installation by running the `deviceQuery` sample.

A3. Installing RPUD and RPUDPLUS

After installing the CUDA Toolkit and R, you can download and extract the latest RPUD package in a local folder, and proceed to install `rpudplus` on your operating system.

Windows

For Windows users, in the `R` main window, you can select the menu item *"Packages > Install package(s) from local zip files"*. Then navigate to the extraction folder you have just created, and install the two binary packages `rpud_0.5.2.zip` and `rpudplus_0.5.2.zip` in turn. The `rpud_0.5.2_src.zip` package is for your reference only. It is not meant to be installed.

Mac OS X

For Mac OS X users, you should open an OS X terminal and change current directory to the extraction folder.

```
$ tar xf rpux_0.5.2_mac.tgz

$ cd rpux_0.5.2_mac
```

And you can install the binary packages.

```
$ R CMD INSTALL rpud_0.5.2.tgz

$ R CMD INSTALL rpudplus_0.5.2.tgz
```

You can *optionally* install the `rpud` source package in a terminal if you already have the Xcode command line tools installed.

```
$ R CMD INSTALL \
      rpud_0.5.2_src.tgz
```

Linux

The R installation process is a little bit different on Linux. For Fedora users, enter the following in a terminal.

```
$ sudo yum install R-devel
```

For Ubuntu users, use the following instead.

```
$ sudo apt-get install r-base-dev
```

Ubuntu users should also follow the CRAN instruction for latest R release.

Then you can open an R console and change working directory to the extraction folder.

```
> setwd("<extraction folder>/rpux_0.5.2_linux")
```

And you can install the binary packages in turn.

```
> install.packages("rpud_0.5.2.tar.gz")
```

```
> install.packages("rpudplus_0.5.2.tar.gz")
```

You can *optionally* install the rpud source package in R.

```
> install.packages("rpud_0.5.2_src.tar.gz")
```

Post Installation

If you need SVM and Bayesian inferences, you should meet their dependencies on coda and SparseM in R.

```
> install.packages(
+     c("coda", "SparseM"))
```

Now you may verify your rpudplus installation.

```
> library(rpud)
Rpudplus 0.5.2
http://www.r-tutor.com
Copyright (C) 2010-2015 Chi Yau. All Rights Reserved.
Rpudplus is free for academic use only. There is absolutely NO wa
```

RPUDPLUS License

You will need an official license to use `rpudplus`. To apply an individual user license, you should identify your home folder in an `R` console by typing the following command, and copy the license file to your home folder location.

```
> Sys.getenv("HOME")
```

For Windows users, the home folder of R is usually the Document folder: "*C:\Users\<username>\Documents*". For Mac OS X users, it is usually "*/Users/<username>*". As for Linux users, it is usually "*/home/<username>*".

To apply the license in a cluster environment, you should copy the license file to the installation folder of `rpudplus` instead: "*<libpath>/rpudplus/rpudplus.LICENSE*". The individual user license will override the cluster system license if both are present.

Note

The `rpudplus` GPU package requires double precision arithmetic hardware support. In order to fully exploit its capabilities, you should ensure that the compute capability of your CUDA GPU exceeds *2.0 or above* according to the CUDA hardware page.

Furthermore, the Windows version of `rpudplus` prefers a Tesla GPU running in TCC mode. Since the GeForce hardware does not support TCC mode, the performance is suboptimal.

Appendix B
Installing OpenBUGS

The following explains how to install OpenBUGS on Windows and Linux. The Mac support for OpenBUGS is very limited.

Windows Installation

Since there is a Windows installer available, installing OpenBUGS on Windows is fairly simple. You can just download and run the setup program OpenBUGS<version>setup.exe.

Linux Installation

The Linux installation of OpenBUGS is a little bit more involved. If you are using 64-bit Linux, you need to prepare a 32-bit environment for OpenBUGS first.

For 64-bit Fedora users, do the following in a terminal.

```
$ sudo yum install glibc-devel.i686 \
    libgcc.i686
```

For 64-bit Ubuntu users, do the following instead.

```
$ sudo apt-get install gcc-multilib
```

With the 32-bit environment ready, you can then download and extract the OpenBUGS Linux package in a temporary folder for installation.

```
$ tar xf OpenBUGS-<version>.tar.gz
$ cd OpenBUGS-<version>
$ ./configure
$ make
$ sudo make install
```

R2OpenBUGS

Now you can install the `R2OpenBUGS` extension in R.

```
> install.packages("R2OpenBUGS")
```

And you may verify OpenBUGS with the command `validateInstallOpenBUGS`.

```
> library(R2OpenBUGS)
> validateInstallOpenBUGS()
```

References

- W. M. Bolstad
  Introduction to Bayesian Statistics, Second Edition (2007)
  Wiley-Interscience, New York, NY

- BUGS Team
  OpenBUGS
  http://www.openbugs.net

- C. J. C. Burges
  A Tutorial on Support Vector Machines for Pattern Recognition
  Data Mining and Knowledge Discovery, 2 (1998) 121-167

- A. Carpenter
  CuSVM: A CUDA Implementation of Support Vector Classification and Regression
  http://patternsonascreen.net/cuSVM.html

- C.-C. Chang and C.-J. Lin
  LIBSVM - A Library for Support Vector Machines
  http://www.csie.ntu.edu.tw/~cjlin/libsvm/

- A. J. Dobson
  An Introduction to Generalized Linear Models, Third Edition (2008)
  Chapman and Hall/CRC, Boca Raton, FL

- R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin
  LIBLINEAR - A Library for Large Linear Classification
  http://www.csie.ntu.edu.tw/~cjlin/liblinear/

- A. Gelman
  Prior Distributions for Variance Parameters in Hierarchical Models
  Bayesian Analysis, 1, Number 3 (2006) 515-533

- A. Gelman, J. B. Carlin, H. S. Stern, and D. B. Rubin
  Bayesian Data Analysis, Second Edition (2004)
  Chapman and Hall/CRC, Boca Raton, FL

- Z. Ghahramani
  A Tutorial on Gaussian Processes (or Why I Don't Use SVMs)
  http://mlss2011.comp.nus.edu.sg/uploads/Site/lect1gp.pdf

- M. Girolami and S. Rogers
  Variational Bayesian Multinomial Probit Regression with Gaussian Process Priors
  Neural Computation, 18 (2006) 1790-1817

- J. Gill
  Generalized Linear Models: A Unified Approach (2000)
  SAGE Publications, Thousand Oaks, CA

- C.-W. Hsu, C.-C. Chang, and C.-J. Lin
  A Practical Guide to Support Vector Classification
  http://www.csie.ntu.edu.tw/~cjlin/papers/guide/guide.pdf

- W. P. Krijnen
  Applied Statistics for Bioinformatics Using R (2009)
  http://cran.r-project.org/doc/contrib/Krijnen-IntroBioInfStatistics.pdf

- M. Kery
  Introduction to WinBUGS for Ecologists:
  Bayesian Approach to Regression, ANOVA, Mixed Models and Related Analyses (2010)
  Academic Press, San Diego, CA

- N. Lama and M. Girolami
  Variational Bayesian Multinomial Probit Regression
  http://www.bioconductor.org/packages/release/bioc/html/vbmp.html

- J. Magus and H. Neudecker
  Matrix Differential Calculus with Applications in Statistics and Econometrics, Second Edition (1999)
  Wiley-Interscience, New York, NY

- R Development Core Team
  An Introduction to R

http://cran.r-project.org/doc/manuals/R-intro.pdf

- C. E. Rasmussen and C. K. I. Williams
  Gaussian Processes for Machine Learning (2006)
  MIT Press, Cambridge, MA

- P. E. Rossi, G. M. Allenby, and R. McCulloch
  Bayesian Statistics and Marketing (2005)
  Wiley-Interscience, New York, NY

- A. F. da Silva
  cudaBayesreg: Bayesian Computation in CUDA (2010)
  The R Journal, Vol. 2/2, 48-55
  http://journal.r-project.org/archive/2010-2/RJournal_2010-2_Ferreira~da~Silva.pdf

- D. Spiegelhalter, A. Thomas, N. Best, and D. Lunn
  WinBUGS User Manual
  http://www.mrc-bsu.cam.ac.uk/bugs/winbugs/manual14.pdf

- K. E. Train
  Discrete Choice Methods with Simulation, Second Edition (2009)
  Cambridge University Press, New York, NY

- V. N. Vapnik
  Statistical Learning Theory (1998)
  Wiley-Interscience, New York, NY

- C. Yau
  R Tutorial - An R Introduction to Statistics
  http://www.r-tutor.com

- J. Verzani
  SimpleR - Using R for Introductory Statistics
  http://cran.r-project.org/doc/contrib/Verzani-SimpleR.pdf