

R Companion to Introductory Nonparametrics

Paul Rippon



PAUL RIPPON

R COMPANION TO INTRODUCTORY NONPARAMETRICS

R Companion to Introductory Nonparametrics

1st edition

© 2018 Paul Rippon & bookboon.com

ISBN 978-87-403-2260-6

Peer review by John Rayner

Honorary Professorial Fellow, National Institute for Applied Statistics

Research Australia, University of Wollongong, Wollongong, Australia

and Conjoint Professor, School of Mathematical and Physical

Sciences, University of Newcastle, Newcastle, Australia

CONTENTS

	Preface	6
	Using R	7
	Examples and R Scripts	16
1	A First Perspective on Nonparametric Testing	17
1.2	The sign tests	17
1.3	Runs tests	24
1.4	The median test	31
2	Nonparametric Testing in the Completely Randomised, Randomised Blocks and Balanced Incomplete Block Designs	46
2.2	The Kruskal-Wallis test	46
2.3	The Friedman test	50
2.4	The Durbin test	53

2.5	Relationships of Kruskal-Wallis, Friedman and Durbin tests with ANOVA F tests	55
2.6	Orthogonal contrasts: Page and umbrella tests	57
3	Permutation Testing	61
3.1	What is permutation testing and why it is important?	61
3.2	Nonparametric multifactor ANOVA when the levels of the factors are unordered	64
3.3	Revisiting some previous examples	72
	Bibliography	84

Preface

This book is written specifically as a companion to [Rayner \(2016\)](#). It is not a standalone document. It is assumed that readers will be accessing this book as they work through the material in [Rayner \(2016\)](#). The notation and terminology are consistent with [Rayner \(2016\)](#) and there is no unnecessary repetition of material.

As a reader studies the theory and calculations for a particular non-parametric example in [Rayner \(2016\)](#), the identically numbered section in this volume will provide the necessary computer code in the R language to complete the analysis for that example. Although this book does not repeat the detailed exposition of methods contained in [Rayner \(2016\)](#), the code and supporting text have been written with the aim of facilitating understanding of the calculations and algorithms. Similarly, graphs have also often been provided to aid understanding.

Those readers who are more interested in just being able to access a library of functions that will do the necessary calculations should also find the book useful. However, this has not been the main aim. The code aims for clarity in some cases rather than computational efficiency. Nor has extensive error checking been included in functions to warn users if the function parameters they provide don't make sense.

It is expected that readers will have a range of mathematical, statistical and computer programming skills. Readers familiar with R should be very comfortable with the material. Those familiar with other programming languages should also find it reasonably easy to follow and to learn the minimal R skills they need to run the code or to adapt it to other languages they may be more comfortable with. Those new to computer programming and R will obviously have additional skills to learn. As noted above the code has been written and commented to facilitate understanding but novices are likely to need additional help. This book does not purport to teach R but the section on [Using R \(p 7\)](#) gives some additional explanation about a number of aspects of the R language that are likely to be particularly useful and suggests sources for those wishing to learn more.

It has always been a pleasure to work with John Rayner and I thank him for inviting me to write this companion volume to his book. (I'm sure neither of us thought it would take this long!)

Using R

This book does not purport to teach R but this section gives some additional explanation about a number of aspects of the R language that are likely to be particularly useful in working through the example code in this book and suggests learning resources for those who are new to R.

Introduction

What is R?

The R Project for Statistical Computing (<https://www.r-project.org>) describes R as “*a language and environment for statistical computing and graphics.*” It is available for download at no cost and runs on a wide variety of UNIX platforms, Windows and MacOS. Follow the links from this website to download the software from one of many servers around the world hosting mirrored websites of the Comprehensive R Archive Network (CRAN).



 **MTHøjgaard**

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



What is RStudio?

RStudio (<https://www.rstudio.com/>) is an integrated development environment (IDE) that helps you develop programs in R. It offers additional tools compared to the more limited development environment that is installed with the base R system. Some users may find this IDE helpful but it is not essential. The example R code and any associated instructions contained within this document make no assumptions about which environment is being used.

R Version

R is regularly updated. The output shown in this document was generated using R version 3.3.3 (2017-03-06).

Getting Started

Information is readily available in many books and online resources to help the new user learn R. Many of these are free. For example, you might like to access [A Step-by-Step R Tutorial: An introduction into R applications and programming](#) by [le Roux and Lubbet \(2015\)](#). [An Introduction to R](#) by [Venables et al. \(2017\)](#) is another good starting point. The latter is also available from the Help menu within the R environment.

Using R and this book

As a reader studies the theory and calculations for a particular non-parametric example in [Rayner \(2016\)](#), the identically numbered section in this document will provide the necessary R commands to complete the analysis for that example. The relevant blocks or chunks of code are printed within a lightly shaded box as shown in the simple example below. Any output is also shown within the shaded box, immediately after the command that generates it, on lines starting with `##`.

```
# This is a comment within the R code.  
y <- rnorm(10, mean=0, sd=1)  
x <- y + 3  
cat("The mean of x is", mean(x))  
  
## The mean of x is 3.033985
```

Note the colour coding of the code structure. R objects are displayed in black as is any output. Function names are red. Constant values (including reserved words like TRUE and NULL) are magenta. Character strings are blue. Names of function parameters are green. Comments are also coloured differently.

Note the R assignment operator, `<-`, used to assign values to objects. In this example, the objects `y` and `x` did not previously exist and so are created as part of the assignment operation.

For convenience, all of the code listed in this document is also contained within a set of files as listed in [Table 1](#) on page 16. These R scripts also contain some minor additional code that is not listed in this document - for example, code that creates some of the illustrative graphs.

It is assumed that readers will be accessing this book as they work through the material in [Rayner \(2016\)](#) and so the notation and terminology are consistent with [Rayner \(2016\)](#) and there is no unnecessary repetition of the detailed exposition of methods contained there. However the code, including code comments, and the supporting text have been written with the aim of facilitating understanding of the calculations and algorithms. The code aims for clarity in some cases rather than computational efficiency.

Writing R Code

Scripts

R commands can be typed one at a time at the R console. All of the R provided in this document can be run using this interactive approach which is useful when starting to learn about R and also when debugging lines of code. However I strongly recommend that all analyses be carried out using completely self-contained R scripts which are appropriately commented and use meaningful variable names. Saving such scripts then provides a complete record of the analysis carried out.

R Input and Output

For simplicity in this document, the data for any example from [Rayner \(2016\)](#) is contained within the relevant R code for that example. For larger data sets this is not a very practical approach and it is likely to be necessary to first read the data from a data file of some kind. The [R Data Import/Export](#) document by [Team \(2017\)](#) explains how this is carried out. This document is also available from the Help menu within the R environment.

R Help

I have already mentioned the R Help menu which provides ready access to a range of useful documents which are part of every R installation. Detailed documentation is also available for any R function by typing a question mark followed by the function name at the R console. For example,

```
?rnorm
```

provides a description of the `rnorm` function used in the previous code chunk.

R Functions

Like most other programming languages, R uses the concept of a function to provide a way of re-using the same code again and again, often with slight variations in the parameters. There are functions that are already available built-in to the language as well as functions that you define yourself. For example, in the following code chunk the `ls` function is called without any arguments. As shown in the output below, it returns a vector of character strings, `x`, `y`, which forms a list of names of the objects that exist in the current R environment. These objects were created in a previous code chunk.

```
ls()  
  
## [1] "x" "y"
```

In this example, the function call has no arguments and so function parameters take their default values as specified in the function definition. You can use `?ls` to get more detailed information about this function including a list of the parameters and their default values.

The `rm` function can be used to remove objects as shown below.

```
rm(x)  
ls()  
  
## [1] "y"
```

Here there is only one argument provided to the `rm` function specifying that the object `x` is to be deleted from the current environment. Arguments specified in a function call are matched to function parameters by the order in which they are specified or are given in `name=object` form.

I usually use the `name=object` form to improve the readability of the code except for the simplest of cases like `mean(y)` which is clearer than `mean(x=y)` in my opinion.

Consider the example below.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



```
rm(list=ls())
```

Here the `ls` function is evaluated to find the vector of character strings corresponding to the names of the objects that exist in the current R environment. This vector forms the argument to be passed to the `list` parameter of the `rm` function. The result is that all of the objects in the current R environment are removed.

Many of the R scripts provided in this document start with this command as a simple way of initialising the environment ensuring no contamination of an analysis by retained objects from a previous analysis.

Data Structures

The simplest data structure in R is a vector. A vector is a single entity consisting of an ordered collection of elements which are all of the same type. For example,

```
(y <- 1:10)

## [1] 1 2 3 4 5 6 7 8 9 10
```

creates a vector `y` consisting of the integers from 1 to 10. (Note the `:` operator for generating regular sequences. Adding parentheses around the entire assignment statement is a simple way of echoing the value being assigned.)

R functions and operators typically operate in a vectorized fashion as shown in the following example.

```
(x <- y + 3)

## [1] 4 5 6 7 8 9 10 11 12 13

(z <- x * y)

## [1] 4 10 18 28 40 54 70 88 108 130

log(z)

## [1] 1.386294 2.302585 2.890372 3.332205 3.688879 3.988984 4.248495
## [8] 4.477337 4.682131 4.867534
```

This is very efficient but can take some getting used to if you are familiar with other languages that always operate on scalar values and would typically require looping through each individual element of the vector for an example like this.

More complex data structures include data frames and lists. A list is a more general form of vector where each element need not be of the same type and can themselves be vectors or lists. The results of many R functions are returned as lists. A data frame is a very useful structure for data tables that typically occur in statistical analysis where columns correspond to variables and rows to observational units. Examples are given below.

```
(examplelist <- list(x, z, pets=c("cat", "dog", "bird", "fish")))
```

```
## [[1]]
##  [1]  4  5  6  7  8  9 10 11 12 13
##
## [[2]]
##  [1]  4 10 18 28 40 54 70 88 108 130
##
## $pets
## [1] "cat" "dog" "bird" "fish"
```

```
(exampledataframe <- data.frame(x, y, z, fred=LETTERS[y],
                                stringsAsFactors=FALSE))
```

```
##      x  y  z fred
## 1   4  1  4    A
## 2   5  2 10    B
## 3   6  3 18    C
## 4   7  4 28    D
## 5   8  5 40    E
## 6   9  6 54    F
## 7  10  7 70    G
## 8  11  8 88    H
## 9  12  9 108   I
## 10 13 10 130   J
```

Note that each column of a data frame must have the same number of elements but lists are more general. Setting `stringsAsFactors=FALSE` ensures that the column labelled `fred` is a vector of character strings. If `stringsAsFactors=TRUE` were specified instead (which is the default setting) then character vectors are converted to factors. Factors are used within R for storing categorical data (about which group a particular observational unit may belong to).

Indexing

Subsets of the elements of a vector may be selected by using an index vector as shown in the following example.

```
x
```

```
##  [1]  4  5  6  7  8  9 10 11 12 13
```

```
x[1]
```

```
##  [1]  4
```

```
x[1:3]
```

```
##  [1]  4  5  6
```

```
x[7:10]

## [1] 10 11 12 13

x[c(1:3,7:10)]

## [1]  4  5  6 10 11 12 13

x[-c(1:3,7:10)]

## [1] 7 8 9
```

Note the use of the `c` function to create an index vector by concatenating values together.

Instead of using a vector of positive integers to specify the elements to include, or a vector of negative integers to specify the elements to exclude, a vector of logical values can be used as shown below.

```
y

## [1] 1 2 3 4 5 6 7 8 9 10

y > 5 # logical vector indicating which elements of y are > 5

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

y[y>5] # select those elements

## [1] 6 7 8 9 10
```

Data frames are two-dimensional and so both dimensions must be specified in some way as shown in the following example.

```
exampledataframe

##      x  y  z fred
## 1   4  1  4    A
## 2   5  2 10    B
## 3   6  3 18    C
## 4   7  4 28    D
## 5   8  5 40    E
## 6   9  6 54    F
## 7  10  7 70    G
## 8  11  8 88    H
## 9  12  9 108   I
## 10 13 10 130   J

exampledataframe[1, 4]
```

```
## [1] "A"

exampledataframe$fred[1] # £ notation accesses the named column

## [1] "A"

exampledataframe[1, "fred"]

## [1] "A"

exampledataframe[exampledataframe$y>7, c("x", "z")]

##      x      z
## 8   11   88
## 9   12  108
## 10  13  130
```

More details about indexing specific elements of data structures are contained in [Venables et al. \(2017\)](#) or many other R references. Understanding the concept of indexing and the methods used to index specific elements of different data structures is critical to understanding the code provided in this document.



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

R packages

All R functions are stored in packages. Only when a package is loaded are its contents available.

Most of the functions used in this document are part of the standard or base packages that are installed with every R installation and are loaded when you start R.

However there are thousands of other packages available that have been created by members of the R community around the world. These packages greatly extend the functionality of R and are available for download from any of the available CRAN servers around the world using the `install.packages` command. (Alternatively there are menu options to facilitate installation of packages in both R and R Studio.)














A package only needs to be installed once, which downloads the package from the appropriate CRAN server and stores the package components as part of your R installation. To access package functions you then need to load the package using the `library` function. This step needs to be done once in every R session that uses functions from that package.

Examples and R Scripts

R code and corresponding output is displayed within the context of the worked examples throughout the document. Many of the examples are revisited and extended several times as additional methods are covered. All of the examples are listed in Table 1 below with page references. All of the code relating to a particular example has been combined into a single R script which is also linked in the table below.

Note that these files are attached as pdf annotations. Depending on your pdf reader you may need to click, double-click or right-click to download the file. NB Some limited pdf readers like those built into web browsers may not offer this functionality. In this case you should open this file with the free Acrobat Reader software available at <https://get.adobe.com/reader/>.

Table 1: List of examples.

achievement test	35	 achieve.r
corn	31	 corn.r
examination	28	 exam.r
flints	24, 41	 flints.r
heart rates	22	 hrates.r
herbicide	61	 herbicide.r
ice cream	53, 60, 69, 80	 icecream.r
lemonade	50, 56, 59, 67, 77	 lemonade.r
monkey	17	 monkey.r
speed	29	 speed.r
teaching methods	44	 teaching.r
tomato	46, 56, 57, 64, 72	 tomato.r
tyres	38	 tyre.r

1 A First Perspective on Nonparametric Testing

1.2 The sign tests

1.2.1 A one-sample sign test

Monkey example

```
# create data objects; calculate test statistic
rm(list=ls()) # remove any existing objects from environment

# create vector of female monkey weights
y <- c(8.30, 9.50, 9.60, 8.75, 8.40, 9.10, 9.25, 9.80,
      10.05, 8.15, 10.00, 9.60, 9.80, 9.20, 9.30)
n <- length(y) # number of measurements
hmed <- 8.41 # hypothesized median
S <- length(y[y>hmed]) # number of measurements greater than hypothesized
                        # median
```

This code chunk determines that there are $n = 15$ measurements in the given vector of monkey weights — as shown in the (jittered) dot plot in Figure 1.1. The $S = 12$ measurements greater than the hypothesized median of 8.41kg are highlighted in red.

If the null hypothesis is true, that the population median female monkey weight is 8.41kg, then $S \sim \text{bin}(n = 15, p = 0.5)$. This null distribution is shown in Figure 1.2 and will always be symmetric. This is because the binomial success probability will always be 0.5 since there is an equal probability of a value being above or below the median.

There are three possible p-values based on the particular alternative hypothesis specified.

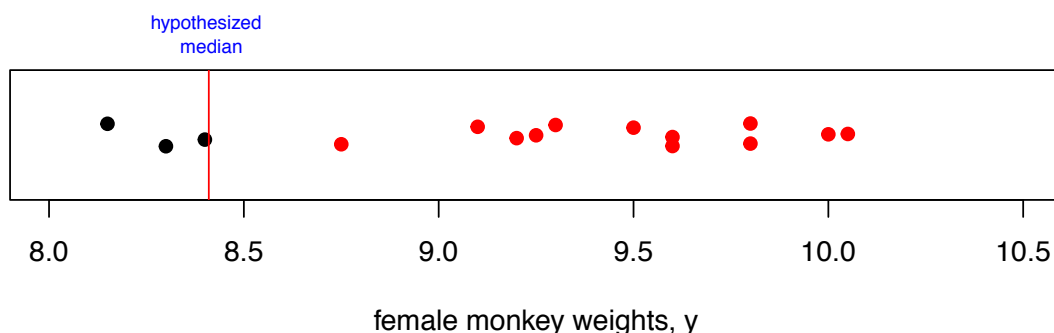


Figure 1.1: Dot plot of weights and hypothesized median.

Alternative	p-value	Comment
median > 8.41	$P(S \geq 12) = 0.0175781$	the highlighted right tail probability in above graph
median < 8.41	$P(S < 12) = 0.9824219$	the left tail probability in above graph
median \neq 8.41	$P(S \leq 3 \text{ or } S \geq 12) = 0.0351563$	double the highlighted right tail probability in above graph

These are easily calculated in R using the function for cumulative binomial probabilities.

```
# calculate p-values
left.tail <- pbinom(q=S-1, size=n, prob=0.5)
right.tail <- pbinom(q=S-1, size=n, prob=0.5, lower.tail=FALSE)
cat("pval(less)=", left.tail, "\n", "pval(greater)=", right.tail,
    "\n", "pval(two.sided)=", 2*right.tail, sep="")

## pval(less)=0.9824219
## pval(greater)=0.01757813
## pval(two.sided)=0.03515625
```

Perhaps even easier is to use the existing `binom.test()` function for conducting an exact binomial test, as shown here for a one-sided alternative hypothesis.

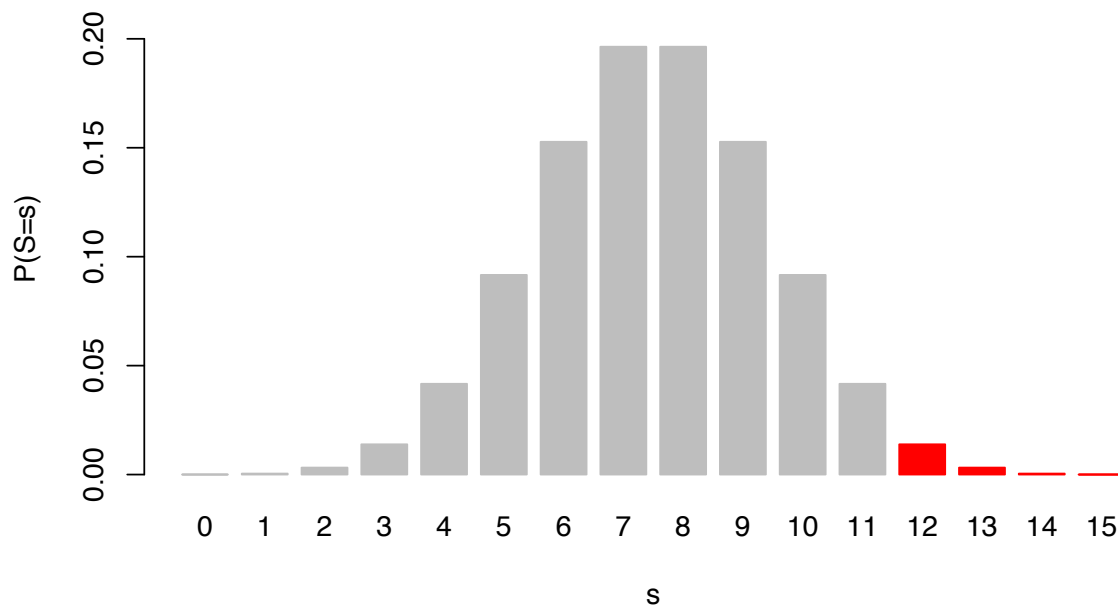


Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

Figure 1.2: Null distribution of test statistic, S .

```
# alternative using existing function
binom.test(x=S, n=n, p=0.5, alternative="greater")

##
## Exact binomial test
##
## data: S and n
## number of successes = 12, number of trials = 15, p-value = 0.01758
## alternative hypothesis: true probability of success is greater than 0.5
## 95 percent confidence interval:
##  0.5602156 1.0000000
## sample estimates:
## probability of success
##                0.8
```

Using the normal approximation to the binomial

Instead of using exact binomial probabilities, it is possible to approximate the binomial distribution with a normal distribution having the same mean ($np = 15 \times 0.5 = 7.5$) and variance ($np(1-p) = 15 \times 0.5 \times 0.5 = 3.75$) as shown in Figure 1.3.

We can then use this normal distribution to calculate an approximate p-value,

$$P(S \geq 12) \approx P(Y \geq 11.5)$$

where $Y \sim N(\mu = 7.5, \sigma^2 = 3.75)$. As we are using the continuous normal distribution to approximate the discrete binomial distribution it is common practice to adjust the normal

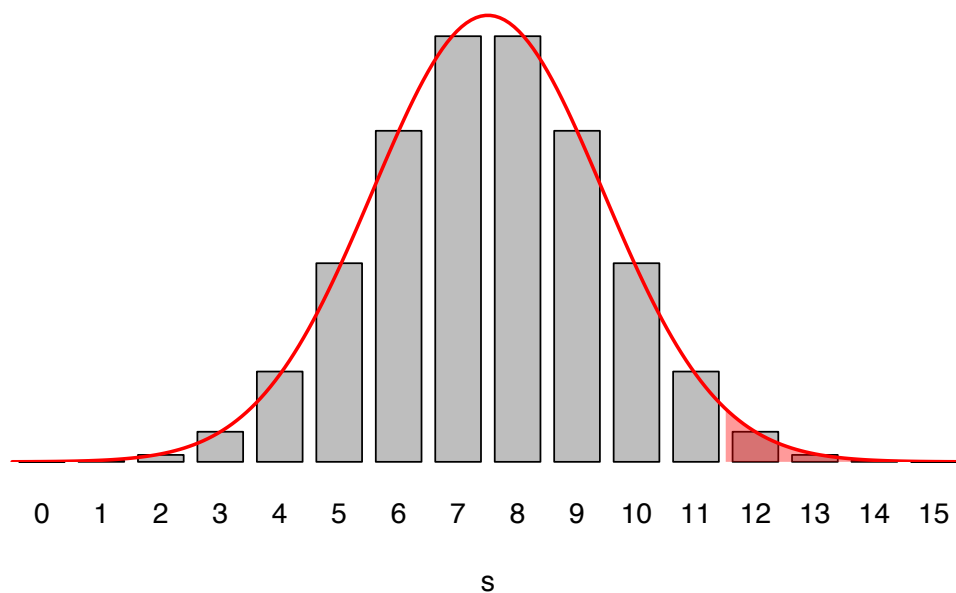


Figure 1.3: Normal approximation to the binomial null distribution for test statistic, S .

quantile corresponding to the test statistic by 0.5 so that it is midway between the adjacent discrete values. This concept of continuity correction is illustrated in Figure 1.3 where the shaded area is bounded at $S - 0.5 = 11.5$. Again, the required probability value is easily calculated in R, this time using the function for cumulative normal probabilities.

```
# p-value based on normal approximation
right.tail <- pnorm(q=S-0.5, mean=n*0.5, sd=0.5*sqrt(n), lower.tail=FALSE)
cat("P(Y>", S-0.5, ")=", right.tail, sep="")

## P(Y>11.5)=0.01943355
```

In the days of calculators and probability tables, the normal approximation was commonly used to simplify the determination of the p-value. When using a computer however, it is probably just as easy to calculate the exact binomial probabilities.

Analogous parametric analysis

```
# parametric alternative is the t-test
t.test(x=y, alternative = "two.sided", mu = hmed, conf.level = 0.95)

##
## One Sample t-test
##
## data: y
## t = 5.3453, df = 14, p-value = 0.0001033
## alternative hypothesis: true mean is not equal to 8.41
```



```
## 95 percent confidence interval:  
## 8.914946 9.591721  
## sample estimates:  
## mean of x  
## 9.253333
```

The parametric analysis analogous to the sign test is the one sample t test — applied here to the monkey weight data using the `t.test()` function. Strictly speaking this analysis assumes that the data has been sampled from a normal distribution. Although the t test is generally considered quite robust unless there are major departures from normality, it can be useful to assess this assumption. One way to do this is the Shapiro-Wilk test as shown below.

```
# assess normality of monkey weights  
shapiro.test(x=y)  
  
##  
## Shapiro-Wilk normality test  
##  
## data: y  
## W = 0.92648, p-value = 0.2416
```



 MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



1.2.2 A two-sample sign test

Heart rates example

The two-sample sign test is just an application of the one-sample sign test to a sample comprising the differences between pairs of measurements.

```
# create data objects and conduct sign test
rm(list=ls()) # remove any existing objects from environment

# Vectors of heart rate measurements
alone <- c(463,462,462,456,450,426,418,415,409,402)
together <- c(523,494,461,535,476,454,448,408,470,437)
y <- alone - together # calculate the difference in rates
n <- length(y) # number of differences
hmed <- 0 # hypothesized median difference
S <- length(y[y>hmed]) # number of measurements greater than hypothesized
                        # median
(ebt <- binom.test(x=S, n=n, p=0.5, alternative="less")) # exact binomial test

##
## Exact binomial test
##
## data: S and n
## number of successes = 2, number of trials = 10, p-value = 0.05469
## alternative hypothesis: true probability of success is less than 0.5
## 95 percent confidence interval:
##  0.0000000 0.5069013
## sample estimates:
## probability of success
##                0.2
```

Once the differences are calculated the process is exactly as for a one sample sign test with a hypothesized median difference of zero. Figure 1.4 illustrates the $S = 2$ differences that are greater than the hypothesized median of zero.

The sign test is telling us that if the population median difference in heart rates really is zero, then the probability of seeing two or fewer positive differences, $P(S \leq 2) = 0.0546875$. At the

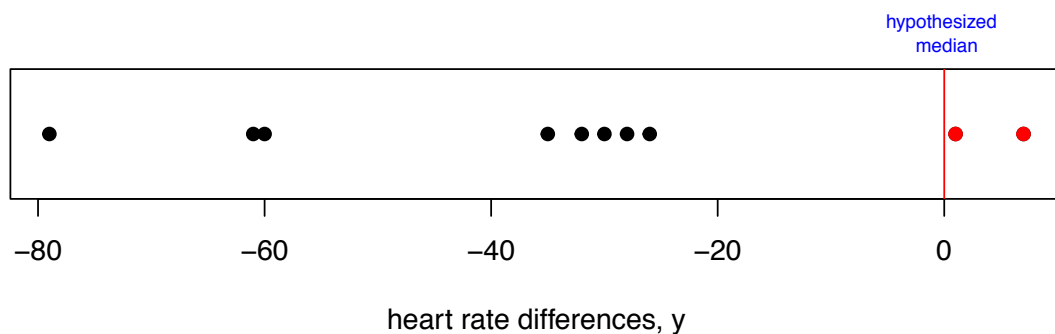


Figure 1.4: Dot plot of heart rate differences and hypothesized median.

0.05 significance level this is not enough evidence to reject the null hypothesis of no difference. This method greatly reduces the effect of the variation between individual rats on the analysis and concentrates on the variation related to the different conditions, being alone or in the presence of another rat.

Analogous parametric analysis

```
# parametric alternative
# using calculated differences
t.test(x=y, alternative = "less", mu=0, conf.level = 0.95)

##
## One Sample t-test
##
## data: y
## t = -4.0498, df = 9, p-value = 0.001443
## alternative hypothesis: true mean is less than 0
## 95 percent confidence interval:
##      -Inf -18.77423
## sample estimates:
## mean of x
##      -34.3

# using original data vectors
t.test(x=alone, y=together, alternative = "less", mu = 0, paired = TRUE,
      conf.level = 0.95)

##
## Paired t-test
##
## data: alone and together
## t = -4.0498, df = 9, p-value = 0.001443
## alternative hypothesis: true difference in means is less than 0
## 95 percent confidence interval:
##      -Inf -18.77423
## sample estimates:
## mean of the differences
##      -34.3

shapiro.test(x=y) # assess normality of differences

##
## Shapiro-Wilk normality test
##
## data: y
## W = 0.94148, p-value = 0.5695
```

Note that the paired t test can be carried out by supplying the `t.test()` function directly with the differences between each pair, or by supplying each vector separately in which case the differences are calculated within the function. Again, the strict assumption is that the set of differences has been sampled from a normal distribution which could be assessed using the Shapiro-Wilk test as shown here.

1.3 Runs tests

1.3.1 A two-sample runs test

Flints example

```
# create data objects and determine no of runs
rm(list=ls()) # remove any existing objects from environment

area <- c("A", "A", "A", "B", "A", "B", "B", "B", "B") # data vector
i <- 2 # starting at second element of vector
r <- 1 # initial run length
cv <- area[1] # current value
while (i <= length(area)) { # count the length of each run
  if (area[i] == cv)
    r[1] <- r[1] + 1
  else {
    r <- c(1,r)
    cv <- area[i]
  }
  i <- i + 1
}
r <- rev(r) # vector of run lengths is determined in reverse order
T <- length(r) # number of runs
```

This code counts the run lengths as 3, 1, 1, 4 for this example and therefore determines that there are 4 runs in total. The following code is more efficient if you aren't interested in determining the vector of run lengths.

```
# alternative determination of no of runs
area.t <- (area == area[1]) # convert to TRUE and FALSE, ie 1 and 0
area.d <- diff(area.t) # non-zero elements of the difference vector
# indicate the end of a run
T <- length(area.d[area.d != 0])+1
```

Is $T = 4$ a smaller number of runs than expected if the flint hardness distributions at both sites were identical? The required p-value is calculated below using the exact probability distribution for the number of runs.

```
# calculate runs test p-value
tab.area <- table(area) # table of counts
M <- tab.area[1]
N <- tab.area[2]

# function for exact probability distribution
PT <- function (t, m, n) {
  k <- t %% 2 # note use of the integer division operator
  if (t %% 2 == 0) # t is even
    prob <- 2 * choose(m-1, k-1) * choose(n-1, k-1) / choose(m+n, n)
  else # t is odd
    prob <- ( choose(m-1, k) * choose(n-1, k-1) +
              choose(m-1, k-1) * choose(n-1, k) ) / choose(m+n, n)
  return(prob)
}

cat("p-value = P(T <= ", T, ") = ",
    PT(t=2,m=M,n=N) + PT(t=3,m=M,n=N) + PT(t=4,m=M,n=N), sep="")

## p-value = P(T <= 4) = 0.2619048
```

The runs test is telling us that 4 or less runs is not unlikely for a random arrangement of this data and so there is no evidence against the null hypothesis of identical flint hardness distributions at each site.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



Alternatively, the p-value could be calculated using the normal approximation with continuity correction.

```
# using the normal approximation
E.T <- 1 + 2*M*N/(M+N)
var.T <- 2*M*N*(2*M*N-M-N)/(M+N)^2/(M+N-1)
z <- (T + 0.5 - E.T) / sqrt(var.T)
cat("p-value = P(Z < ", z, ") = ", pnorm(z), sep="")

## p-value = P(Z < -0.6827364) = 0.2473867
```

Figure 1.5 illustrates these distributions with shading indicating the p-values.

For easier application of this test, this code could be wrapped into a function with some basic error checking added and made more efficient. (The above calculation of the exact p-value is certainly not efficient.) Alternatively you could choose to use the `runs.test()` function included in the `randtests` package. Loading packages extends the capability of R by adding additional functions. See [page 15](#) for more information about R packages.

```
# using a pre-existing function
library(randtests) # loads the required package.
# calculating the exact p-value
runs.test(x=as.numeric(area.t), alternative="left.sided", pvalue="exact",
          threshold = 0.5)

##
## Runs Test
##
## data: as.numeric(area.t)
## statistic = -1.0442, runs = 4, n1 = 4, n2 = 5, n = 9, p-value =
## 0.2619
## alternative hypothesis: trend

# using the normal approximation (no continuity correction)
runs.test(x=as.numeric(area.t), alternative="left.sided", pvalue="normal",
          threshold = 0.5)

##
## Runs Test
##
## data: as.numeric(area.t)
## statistic = -1.0442, runs = 4, n1 = 4, n2 = 5, n = 9, p-value =
## 0.1482
## alternative hypothesis: trend
```

Note:

1. The implementation of the runs test in the `randtests` package is designed to test for the randomness of patterns in numerical data. The context for this example is a little different.

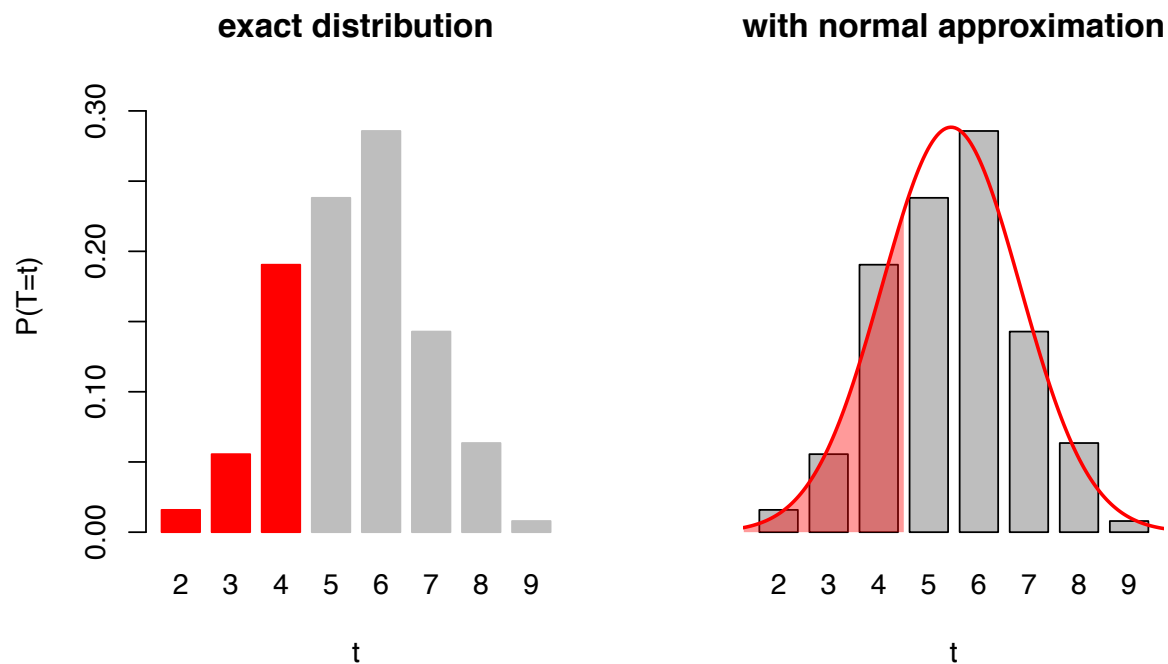


Figure 1.5: Determination of the p-value from the exact null distribution or normal approximation.

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

CISO Conference
Produced by **Inspired**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

Therefore some manipulation of the original data is required to give this function the numeric data it expects. In the previous code chunk, the data is provided to the function as a sequence of 0's and 1's and a threshold value of 0.5 is provided to change this numeric data back into dichotomous categorical data. This is redundant processing as the original data in this example is already dichotomous categorical. This is unfortunately necessary in order to use this pre-existing function.

2. Note that this function's calculation of the p-value using the normal approximation does not match the previous calculation. Unfortunately this function does not seem to implement the continuity correction when using the normal approximation. For large sample sizes, the absence of the continuity correction should not make a big difference to the p-value. For small sample sizes, the option specifying exact p-value calculation is recommended.
3. There seem to be a number of other implementations of the runs test in other packages. These may be more appropriate depending on the specific context. For example, the implementation in the `tseries` package has the advantage of accepting the data as a dichotomous factor but it also does not use a continuity correction. Nor does it calculate exact p-values.

1.3.2 A runs test for randomness

Examination example

In this example, the dichotomous data is coded directly as 1 or 0 representing True or False responses to an examination.

```
# create and process examination data
rm(list=ls()) # remove any existing objects from environment

T <- 1
F <- 0
y <- c(T, F, F, T, F, T, F, T, T, F, T, F, F, T, F, T, F, T, T, F)
yd <- diff(y) # non-zero elements of the difference vector
           # indicate the end of a run
T <- length(yd[yd != 0])+1
ty <- table(y) # table of counts
M <- ty[1]
N <- ty[2]
```

Here there are $M = 10$ False responses, $N = 10$ True responses and $T = 16$ runs. Does this suggest non-randomness in the answers?

A two-sided p-value is calculated below using the normal approximation with continuity correction.

```
# two-sided p-value, based on normal approx with continuity correction
E.T <- 1 + 2*M*N/(M+N)
var.T <- 2*M*N*(2*M*N-M-N)/(M+N)^2/(M+N-1)
z <- (T - 0.5 - E.T) / sqrt(var.T)
```

```
p1 <- pnorm(z, lower.tail=FALSE)
cat("p-value = 2 P(Z > ", z, ") = 2 * ", p1, " = ", 2*p1, sep="")

## p-value = 2 P(Z > 2.067607) = 2 * 0.01933848 = 0.03867696
```

The runs test is indicating that if the 10 False and 10 True responses were sequenced randomly, 16 runs (or something more extreme) is fairly unlikely. The low p-value (0.038677) provides evidence against the null hypothesis of a random sequence. The code below uses the `runs.test()` function from the `randtests` package to calculate the exact p-value for this example.

```
library(randtests)
runs.test(x=y, alternative="two.sided", pvalue="exact", threshold = 0.5)

##
##  Runs Test
##
## data:  y
## statistic = 2.2973, runs = 16, n1 = 10, n2 = 10, n = 20, p-value =
## 0.03704
## alternative hypothesis: nonrandomness
```

Speed example

```
# create data object and conduct runs test
rm(list=ls()) # remove any existing objects from environment
library(randtests)

speeds <- c(46, 58, 60, 56, 70, 66, 48, 54, 62, 41, 39, 52, 45, 62, 53, 69, 65,
           65, 67, 76, 52, 52, 59, 59, 67, 51, 46, 61, 40, 43, 42, 77, 67, 63,
           59, 63, 63, 72, 57, 59, 42, 56, 47, 62, 67, 70, 63, 66, 69, 73)
runs.test(x=speeds, alternative="two.sided", pvalue="exact",
          threshold=median(speeds))

##
##  Runs Test
##
## data:  speeds
## statistic = -1.7146, runs = 20, n1 = 25, n2 = 25, n = 50, p-value
## = 0.1158
## alternative hypothesis: nonrandomness
```

This is the kind of example this implementation of the runs test function is designed for. In fact, you don't even need to specify the median as the threshold to transform the data into a dichotomous vector as this is the default behaviour. [Rayner \(2016\)](#) gives a p-value of 0.116 for this example based on the normal approximation with continuity correction. This is a very good approximation to the exact p-value calculated here. The normal approximation without the continuity correction is not particularly good as shown below.

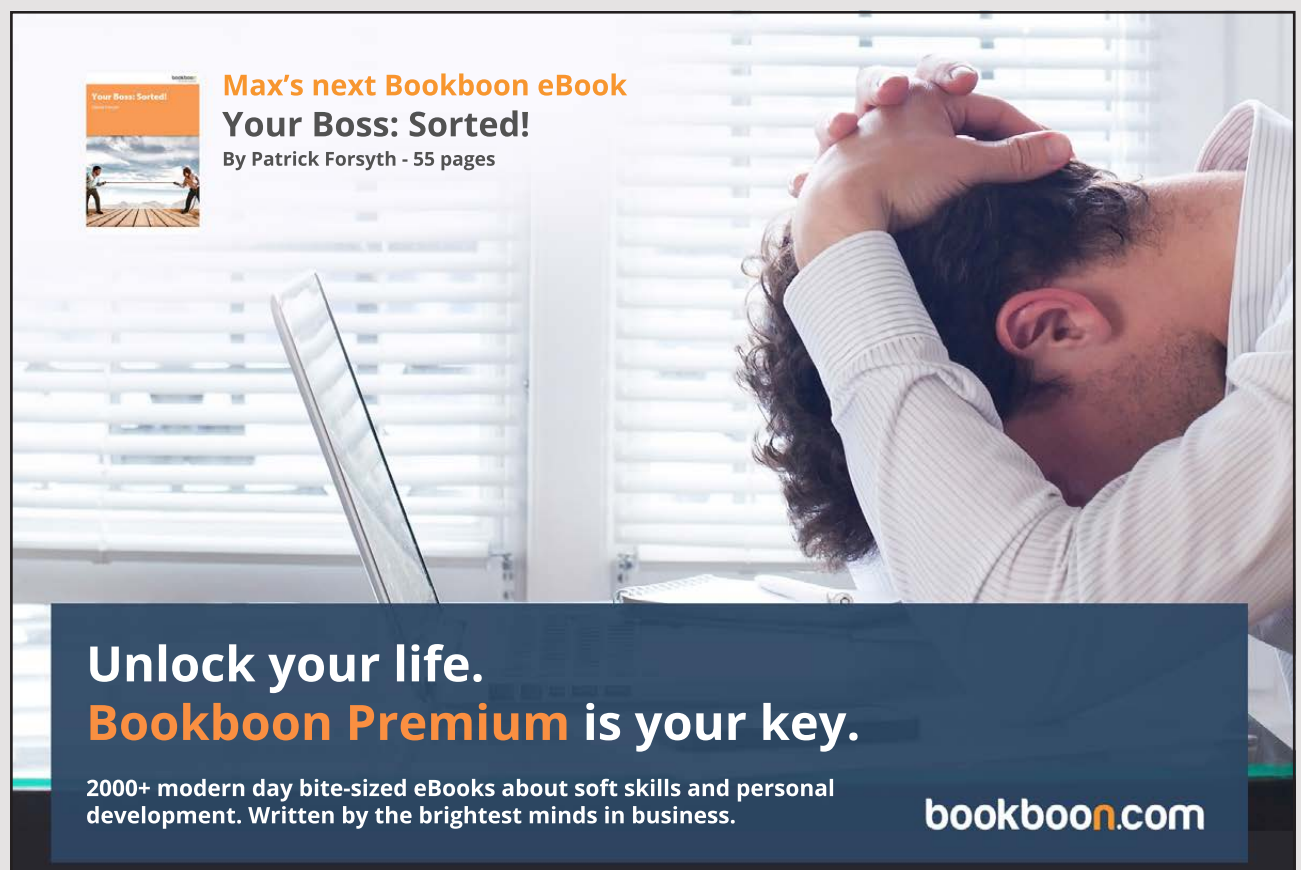
```
# using the normal approximation
runs.test(x=speeds, alternative="two.sided", pvalue="normal",
          threshold=median(speeds))

##
## Runs Test
##
## data: speeds
## statistic = -1.7146, runs = 20, n1 = 25, n2 = 25, n = 50, p-value
## = 0.08641
## alternative hypothesis: nonrandomness
```

For the modified example using a threshold of 55.1 mph:

```
# using a different threshold
runs.test(x=speeds, alternative="two.sided", pvalue="exact", threshold=55.1)

##
## Runs Test
##
## data: speeds
## statistic = -1.7361, runs = 18, n1 = 33, n2 = 17, n = 50, p-value
## = 0.116
## alternative hypothesis: nonrandomness
```



Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com



Again the normal approximation with continuity correction provided by [Rayner \(2016\)](#) is very good ($p=0.115$) but the normal approximation without continuity correction (as provided by this function) is poor as shown below.

```
# using the normal approximation
runs.test(x=speeds, alternative="two.sided", pvalue="normal", threshold=55.1)

##
##  Runs Test
##
## data:  speeds
## statistic = -1.7361, runs = 18, n1 = 33, n2 = 17, n = 50, p-value
## = 0.08255
## alternative hypothesis: nonrandomness
```

1.4 The median test

Corn example

```
# create data object
rm(list=ls()) # remove any existing objects from environment

# list of data vectors
corn <- list(m1=c(83, 91, 94, 89, 89, 96, 91, 92, 90),
            m2=c(91, 90, 81, 83, 84, 83, 88, 91, 89, 84),
            m3=c(101, 100, 91, 93, 96, 95, 94),
            m4=c(78, 82, 81, 77, 79, 81, 80, 81))

corn <- stack(corn) # create data frame in long form
names(corn) <- c("yield", "method") # change column names

# display a subset of the data
corn[c(1,2,10,11,20,21,27,28),]

##      yield method
## 1       83      m1
## 2       91      m1
## 10      91      m2
## 11      90      m2
## 20     101      m3
## 21     100      m3
## 27      78      m4
## 28      82      m4
```

The code above uses the `stack()` function to re-arrange the original list of four vectors into a data frame with a column of corn yields and a second column indicating the growing method. A small subset of the data is also shown. This is commonly known as *long* form as compared

to *wide* form which would have four columns, each corresponding to one of the original four vectors. (NB A list is used instead of a data frame for the original four vectors because they are of different lengths.)

For each growing method, the following code tabulates the number of observed corn yields that are above or below the combined median value. A chi-squared test is then carried out based on the resultant contingency table. Essentially we are now testing the null hypothesis of independent rows and columns (i.e. that whether a yield is above or below the median is independent of growing method).

```
# conduct median test
om <- median(corn$yield) # observed median of combined corn yields

# use the cut function to create a factor based on whether yield
# is above or below the median
corn$cut <- cut(corn$yield, breaks=quantile(corn$yield, probs=c(0,0.5,1)),
               labels=paste(c("<=", ">"), om), include.lowest=TRUE)

# create and print contingency table
(corn.xt <- xtabs(~cut+method,data=corn))

##           method
## cut      m1 m2 m3 m4
## <= 89    3  7  0  8
## > 89    6  3  7  0

# perform chi-squared test
(corn.chisq <- chisq.test(corn.xt))

## Warning in chisq.test(corn.xt): Chi-squared approximation may be incorrect

##
## Pearson's Chi-squared test
##
## data:  corn.xt
## X-squared = 17.543, df = 3, p-value = 0.0005464

# show expected values for comparison to Rayner(2016)
corn.chisq$expected

##           method
## cut      m1      m2      m3      m4
## <= 89 4.764706 5.294118 3.705882 4.235294
## > 89  4.235294 4.705882 3.294118 3.764706
```

The test provides strong evidence against the null hypothesis that all growing methods provide the same median corn yield ($p = 5.4637 \times 10^{-4}$). This low p-value is not surprising looking at the (jittered) dot plot in Figure 1.6.

Note that the `chisq.test()` function gives a warning that the assumed χ^2_3 distribution for the test statistic may not be a good approximation in this case. This is a conservative warning

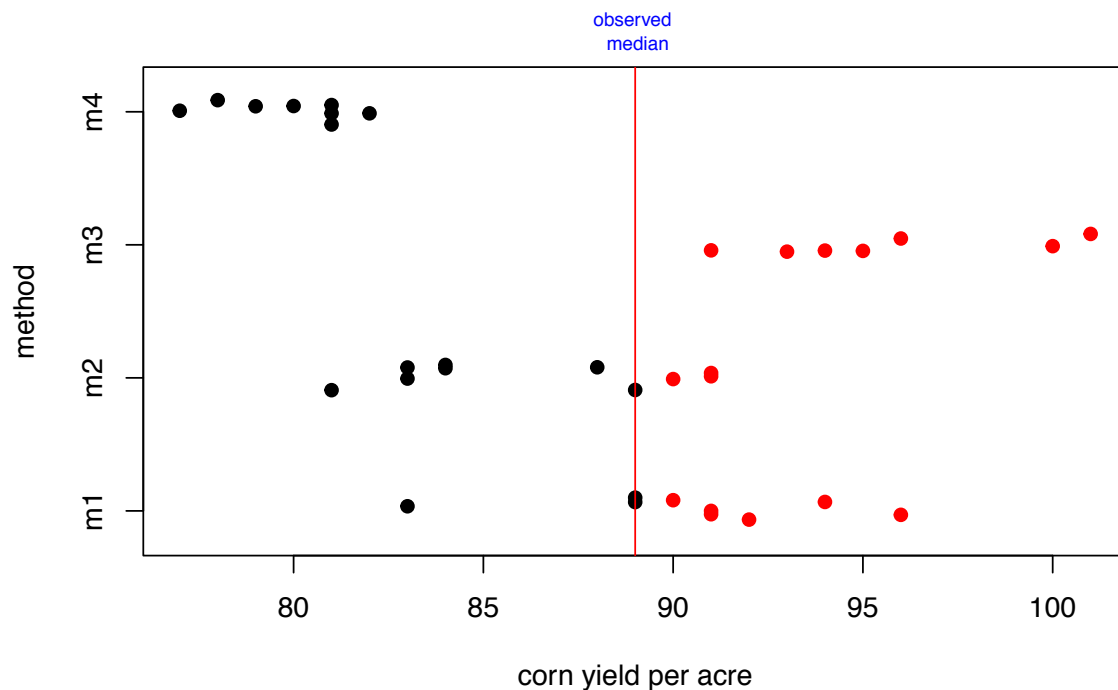


Figure 1.6: Dot plot of corn yields for different growing methods compared to observed overall median.




MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



based on the existence of small expected values (< 5) in the contingency table. Instead of using this approximation, the `chisq.test()` function allows estimation of the p-value by Monte-Carlo simulation — as demonstrated below. By default the simulated p-value is based on 2000 replications but this can be modified. As with any method involving random simulation, results may change slightly each time the calculations are performed so don't expect your results to exactly match the p-value shown here.

```
# alternatively with a simulated p-value
chisq.test(corn.xt, simulate.p.value=TRUE)

##
## Pearson's Chi-squared test with simulated p-value (based on 2000
## replicates)
##
## data:  corn.xt
## X-squared = 17.543, df = NA, p-value = 0.0004998
```

Note also that this data set has three observations exactly the same as the overall median. These have been counted with those observations less than the median. This seems to be common practice for the median test but it does introduce some bias. If there are a relatively large number of ties with the median in any particular group it may be appropriate to repeat the test with the ties counted with those observations greater than the median. This provides a simple robustness check. As shown below, using this approach for the corn data set provides even more evidence against the null hypothesis of identical medians.

```
# alternatively count ties with median in the greater than group
corn$cut2 <- cut(corn$yield, breaks=quantile(corn$yield, probs=c(0,0.5,1)),
               labels=paste(c("<", ">="), om), include.lowest=TRUE,
               right=FALSE)

(corn.xt2 <- xtabs(~cut2+method,data=corn))

##           method
## cut2      m1 m2 m3 m4
## < 89       1  6  0  8
## >= 89      8  4  7  0

chisq.test(corn.xt2)

## Warning in chisq.test(corn.xt2): Chi-squared approximation may be incorrect

##
## Pearson's Chi-squared test
##
## data:  corn.xt2
## X-squared = 20.66, df = 3, p-value = 0.0001239
```

Achievement test example

The code for this example is very similar.

```
# create data and conduct median test
rm(list=ls()) # remove any existing objects from environment

# list of data vectors
achieve <- list(s1=c(43, 80, 99, 86, 68, 70, 85, 93, 98, 96,
                    75, 81, 32, 92, 96, 64, 79, 97, 76, 80),
               s2=c(76, 65, 73, 95, 77, 99, 55, 35, 72, 83,
                    70, 65, 86, 60, 62, 90, 71, 65, 89, 71, 80, 76, 93, 94))

achieve <- stack(achieve)
names(achieve) <- c("result", "school")

# the data frame achieve now has a column of achievement test results and a
# column indicating the school.

om <- median(achieve$result) # observed median of combined test results

# use the cut function to create a factor based on whether each student's
# result is above or below the observed median
achieve$cut <- cut(achieve$result,
                  breaks=quantile(achieve$result, probs=c(0,0.5,1)),
                  labels=paste(c("<=", ">"), om), include.lowest=TRUE)

# create and print contingency table
(achieve.xt <- xtabs(~cut+school,data=achieve))

##           school
## cut      s1 s2
## <= 78    7 15
## > 78   13  9

# perform chi-squared test
(achieve.chisq <- chisq.test(achieve.xt))

##
## Pearson's Chi-squared test with Yates' continuity correction
##
## data:  achieve.xt
## X-squared = 2.2917, df = 1, p-value = 0.1301

# show expected values for comparison to Rayner(2016)
achieve.chisq$expected

##           school
## cut      s1 s2
## <= 78   10 12
## > 78   10 12
```

As noted by [Rayner \(2016\)](#), a continuity correction improves the chi-squared approximation to the exact distribution. The Yates' continuity correction is applied by default by the `chisq.test()` function for a two by two contingency table as shown in the previous results. The uncorrected result is shown below.

```
# without Yates continuity correction for comparison with Rayner(2016)
chisq.test(achieve.xt, correct=FALSE)

##
## Pearson's Chi-squared test
##
## data:  achieve.xt
## X-squared = 3.3, df = 1, p-value = 0.06928
```

A simulated p-value could also be used as demonstrated previously or the exact p-value based on the hypergeometric distribution could be calculated. Rather than calculating the individual probabilities using the `dhyper()` function and then adding them together as demonstrated in [Rayner \(2016\)](#), we can use the `phyper()` function to directly access the cumulative distribution function. The two-sided p-value is highlighted in Figure 1.7 which shows the relevant hypergeometric distribution for this example.

```
# calculating exact p-value based on hypergeometric distribution
cat("P(A1 =< 7) = ", phyper(q=7, m=20, n=24, k=22), sep="")

## P(A1 =< 7) = 0.06460399

cat("p-value = 2 P(A1 =< 7) = 2 * ", phyper(q=7, m=20, n=24, k=22),
    " = ", 2*phyper(q=7, m=20, n=24, k=22), sep="")

## p-value = 2 P(A1 =< 7) = 2 * 0.06460399 = 0.129208
```

Using the exact distribution like this to test the null hypothesis of independent rows and columns in a contingency table is commonly known as Fisher's Exact test. Thus an even easier practice is to use the pre-existing `fisher.test()` function directly on the contingency table as shown below.

```
# equivalently use Fisher's exact test.
fisher.test(achieve.xt)

##
## Fisher's Exact Test for Count Data
##
## data:  achieve.xt
## p-value = 0.1292
## alternative hypothesis: true odds ratio is not equal to 1
## 95 percent confidence interval:
##  0.07807363 1.29887821
## sample estimates:
## odds ratio
##  0.331928
```

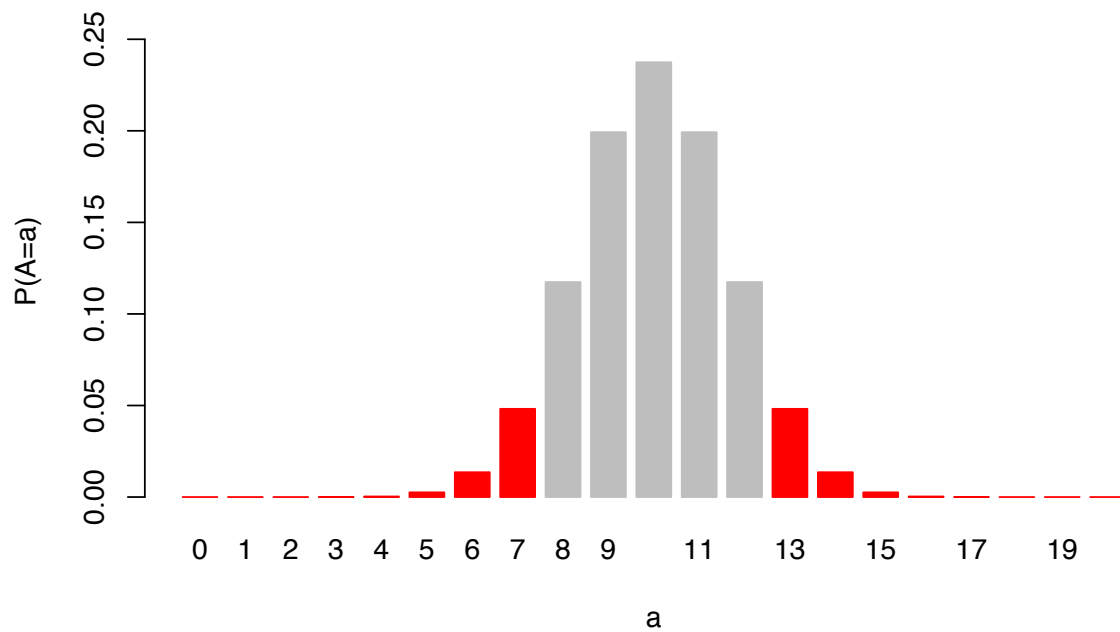


Figure 1.7: Hypergeometric null distribution for achievement test example.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



1.5 The Wilcoxon tests

1.5.1 The signed ranks test

Tyre example

```
# create data and determine signed ranks
rm(list=ls()) # remove any existing objects from environment

# data vector
tyres <- data.frame(miles=c(27900,35100,29800,27700,26700,30700,26900,32400,
                           24800,27400,24900,33300,31600,24300,28300,27600))
hmed <- 30000 # hypothesized median
tyres$y <- (tyres$miles - hmed)/1000
# The scaling of this difference by a factor of 1000 makes no difference.
# This just matches Rayner(2016) and simplifies the presentation of some
# of the numbers.
tyres$r <- rank(x=abs(tyres$y), ties.method="average")
tyres$s <- sign(tyres$y)
```

The previous code calculates the (scaled) difference between the observed tyre miles of service and the hypothesized median value of 30000 miles. The data is plotted in Figure 1.8 on both the original scale and in terms of the scaled differences.

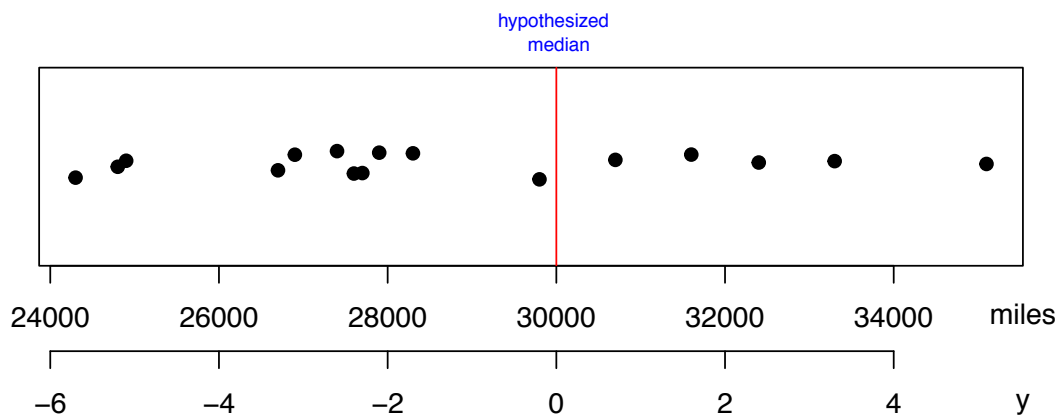


Figure 1.8: Dot plot of miles of service.

The signs and ranks of the absolute differences are also calculated with any tied ranks being averaged. The results are shown below.

```
##   miles    y    r    s
## 1  27900 -2.1  5.0 -1
## 2  35100  5.1 13.5  1
## 3  29800 -0.2  1.0 -1
## 4  27700 -2.3  6.0 -1
```

```
## 5  26700 -3.3 11.5 -1
## 6  30700  0.7  2.0  1
## 7  26900 -3.1 10.0 -1
## 8  32400  2.4  7.5  1
## 9  24800 -5.2 15.0 -1
## 10 27400 -2.6  9.0 -1
## 11 24900 -5.1 13.5 -1
## 12 33300  3.3 11.5  1
## 13 31600  1.6  3.0  1
## 14 24300 -5.7 16.0 -1
## 15 28300 -1.7  4.0 -1
## 16 27600 -2.4  7.5 -1
```

The `rank()` function has done most of the work here. The following code completes the test.

```
# perform Wilcoxon signed ranks test
W.plus <- sum(tyres$r[tyres$s==1])
W.minus <- sum(tyres$r[tyres$s==-1])
n <- nrow(tyres) # number of observations

# mean and variance of approximately normal null distribution for W.plus
mu <- n*(n+1)/4
sigma2 <- mu*(2*n+1)/6

# z score corresponding to W.plus
z <- (W.plus - mu)/sqrt(sigma2)

p.val <- pnorm(z)
cat("p-value = P(Z < ", z, ") = ", p.val, sep="")

## p-value = P(Z < -1.577117) = 0.05738425
```

The test statistic, W_+ , is the sum of the ranks for the positive differences. In this example, $W_+ = 37.5$. As suggested by [Rayner \(2016\)](#), a check of

$$W_+ + W_- = 37.5 + 98.5 = 136$$

should match the sum

$$1 + 2 + \dots + n = 1 + 2 + \dots + 16 = \frac{n(n+1)}{2} = \frac{16 \times 17}{2} = 136.$$

If the hypothesized median is correct then repeated samples from the population should yield test statistics that are approximately normally distributed with mean 68 and variance 374. Thus the calculated test statistic corresponds to a Z score of

$$Z = \frac{37.5 - 68}{\sqrt{374}} = -1.5771175$$

giving a p-value,

$$P(Z < -1.5771175) = 0.0573843$$

for this one sided alternative hypothesis. This code has been written to match the detail of the calculations shown in Rayner (2016). In practice however, it would be more usual to use the pre-defined R function that implements this test as shown below.

```
# using pre-defined function
wilcox.test(x=tyres$miles, alternative="less", mu=30000)

## Warning in wilcox.test.default(x = tyres$miles, alternative = "less", mu = 30000):
## cannot compute exact p-value with ties

##
## Wilcoxon signed rank test with continuity correction
##
## data:  tyres$miles
## V = 37.5, p-value = 0.06033
## alternative hypothesis: true location is less than 30000
```

By default, an exact p-value is computed if the samples contain less than 50 finite values and there are no tied ranks. In this case, there are several tied ranks (as indicated by the warning message) and so the normal approximation is used. Note that the W_+ test statistic is actually a discrete random variable and so when approximating using a normal distribution, a continuity correction is applied by default. Results are shown below without the continuity correction.



The advertisement features a night-time photograph of the Apollo Hotel building. Overlaid on the image is a red lightbulb icon with a white filament. To the right of the icon, the text 'CISO Conference' is displayed in white, with 'Produced by Inspired' in red below it. Further right, a white box contains the text 'Apollo Hotel 1, Groenlandsekade Vinkeveen, Amsterdam, NL' and 'Dec 5th 2019' in blue. At the bottom, a white banner contains the text 'Listen, learn & build relationships with our Network of CISOs & Cyber Security Leaders' and the 'Inspired' logo, which consists of a blue lightbulb icon and the word 'Inspired' in blue.

CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

```
# without continuity correction
wilcox.test(x=tyres$miles, alternative="less", mu=30000,
            exact=FALSE, correct=FALSE)

##
##  Wilcoxon signed rank test
##
## data:  tyres$miles
## V = 37.5, p-value = 0.05729
## alternative hypothesis: true location is less than 30000
```

Note that this still doesn't match the previous p-value (0.0573843) exactly. The `wilcox.test()` function in R also makes a slight correction to the variance of the normal approximation for W_+ when there are tied ranks.

For comparison, the result of the analogous parametric test, the one-sample t-test is also shown.

```
# using analogous parametric alternative
t.test(x=tyres$miles, alternative="less", mu=30000)

##
##  One Sample t-test
##
## data:  tyres$miles
## t = -1.6265, df = 15, p-value = 0.06233
## alternative hypothesis: true mean is less than 30000
## 95 percent confidence interval:
##      -Inf 30100.15
## sample estimates:
## mean of x
##      28712.5
```

The sign test could also be used for this example by adapting the code provided earlier in this chapter.

1.5.2 The two-sample Wilcoxon test

Flints example continued

The following code uses the `area` vector previously created for the [flints example](#) (p 24) to create the `flints` data frame. As this vector is already in order of hardness, assigning the ranks is trivial as shown below.

```
# create flints data frame
flints <- data.frame(area)
flints$r <- 1:nrow(flints) # ranks
flints
```

```
##   area r
## 1    A 1
## 2    A 2
## 3    A 3
## 4    B 4
## 5    A 5
## 6    B 6
## 7    B 7
## 8    B 8
## 9    B 9
```

The test statistic is based on summing the combined rankings for each separate area.

```
# conduct Wilcoxon rank sum test
W.A <- sum(flints$r[flints$area=="A"]) # sum of ranks for area A flints
W.B <- sum(flints$r[flints$area=="B"]) # sum of ranks for area B flints

counts <- table(flints$area)
m <- counts[1] # number of area A flints
n <- counts[2] # number of area B flints

# determine parameters for normal approximation to null distribution of W.A
mu.A <- m*(m+n+1)/2
sig2.A <- m*n*(m+n+1)/12

# calculate z score and p value corresponding to W.A
# note continuity correction
z <- (W.A + 0.5 - mu.A)/sqrt(sig2.A)
p.val <- pnorm(z)
cat("p-value = P(Z < ", z, ") = ", p.val, sep="")

## p-value = P(Z < -2.082066) = 0.01866821
```

In this case there are $m = 4$ flints from area A with summed ranks $W_A = 11$ and $n = 5$ flints from area B with summed ranks $W_B = 34$. As [Rayner \(2016\)](#) suggests, when doing calculations manually the following check step is useful.

Check: $W_A + W_B = 45$ and $(m + n)(m + n + 1)/2 = 9 \times 10/2 = 45$.

If the null hypothesis of identical hardness distributions is correct, then repeated samples from these two populations of flints should yield values of W_A that are approximately normally distributed with mean 20 and variance 16.6666667. Thus the observed value $W_A = 11$ corresponds to a z score of

$$Z = \frac{11 + 0.5 - 20}{\sqrt{16.6666667}} = -2.0820663$$

(note continuity correction) giving a p-value, $P(Z < -2.0820663) = 0.0186682$, for a one sided alternative hypothesis that the hardness for area A flints is higher than for Area B flints in some way. This could correspond to a shift in means (or medians) but could be a more complex difference in distributions.

Details of the exact distribution for this test statistic are not covered in [Rayner \(2016\)](#) but the probability mass function and cumulative distribution functions are available as the R functions

`dwilcox()` and `pwilcox()`. The `dwilcox()` function has been used to create Figure 1.9 showing the null distribution for W_A for this example, with and without the normal approximation and with the p-value highlighted.

The same R function that was previously used for the signed ranks test will also perform the rank sum test so in practice the following code is the easiest way to perform the test.

```
# using pre-defined function
(wt <- wilcox.test(x=c(1,2,3,5),y=c(4,6,7,8,9), alternative="less"))

##
## Wilcoxon rank sum test
##
## data:  c(1, 2, 3, 5) and c(4, 6, 7, 8, 9)
## W = 1, p-value = 0.01587
## alternative hypothesis: true location shift is less than 0
```

Note that this function uses a slightly different definition of the test statistic which is shifted to the left by $\frac{m(m+1)}{2} = \frac{4 \times 5}{2} = 10$. Thus the $W = 1$ value here corresponds to the $W_A = 11$ value calculated previously from first principles. By default, for small sample sizes like this, this p-value is based on the exact distribution of W_A under the null hypothesis of equal hardness distributions for flints from the two areas. Results based on the normal approximation, with and without continuity correction are shown below.



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

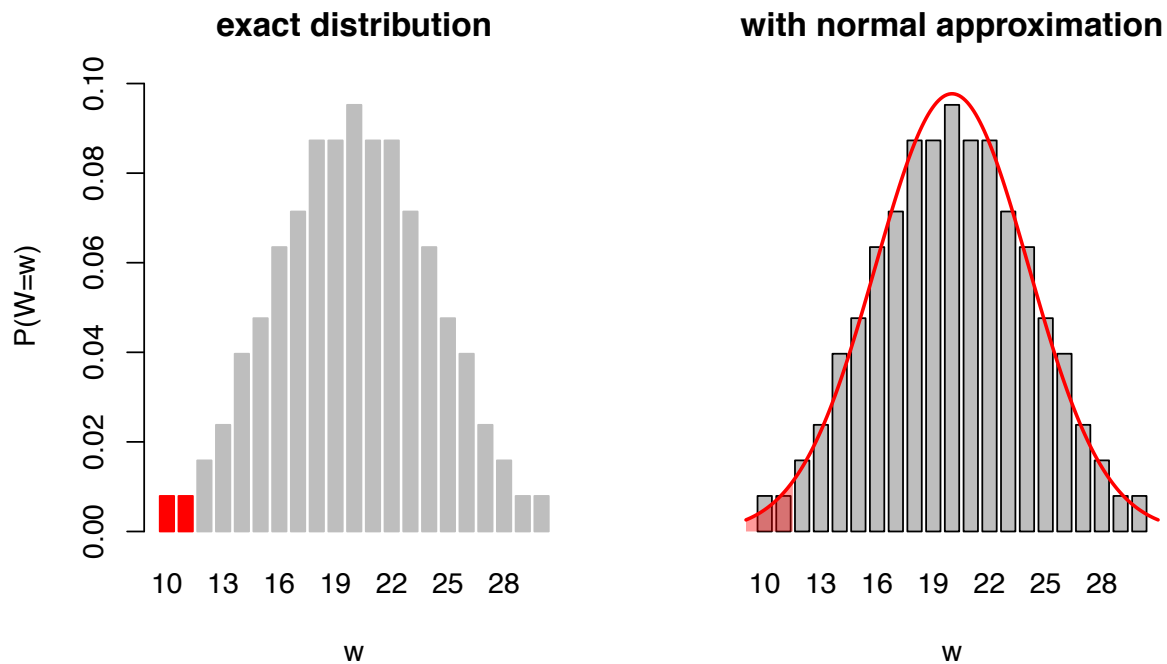


Figure 1.9: Exact null distribution and normal approximation for the Wilcoxon rank sum test statistic for flints example.

```
# using normal approx, with and without continuity correction
wilcox.test(x=c(1,2,3,5),y=c(4,6,7,8,9), alternative="less", exact=FALSE)

##
## Wilcoxon rank sum test with continuity correction
##
## data: c(1, 2, 3, 5) and c(4, 6, 7, 8, 9)
## W = 1, p-value = 0.01867
## alternative hypothesis: true location shift is less than 0

wilcox.test(x=c(1,2,3,5),y=c(4,6,7,8,9), alternative="less", exact=FALSE,
            correct=FALSE)

##
## Wilcoxon rank sum test
##
## data: c(1, 2, 3, 5) and c(4, 6, 7, 8, 9)
## W = 1, p-value = 0.01374
## alternative hypothesis: true location shift is less than 0
```

Teaching Methods example

The following code provides a straightforward application of the Wilcoxon rank sum test to the teaching methods example using the pre-existing function. Results are shown using the exact distribution and the normal approximation with continuity correction.

```
# applying the Wilcoxon rank sum test
rm(list=ls()) # remove any existing objects from environment
X <- c(70,72,74,79,80,82,86,91,93,95)
Y <- c(65,66,68,69,71,73,75,76,78,84,90)

(wt <- wilcox.test(x=X, y=Y))

##
## Wilcoxon rank sum test
##
## data: X and Y
## W = 85, p-value = 0.0357
## alternative hypothesis: true location shift is not equal to 0
# using the normal approximation with continuity correction
wilcox.test(x=X, y=Y, exact=FALSE)

##
## Wilcoxon rank sum test with continuity correction
##
## data: X and Y
## W = 85, p-value = 0.03777
## alternative hypothesis: true location shift is not equal to 0
```

Again note the different definition of the test statistic which is shifted to the left by $\frac{m(m+1)}{2} = \frac{10 \times 11}{2} = 55$. Thus the $W = 85$ value here corresponds to $W_X = 140$, the sum of the X ranks.

2 Nonparametric Testing in the Completely Randomised, Randomised Blocks and Balanced Incomplete Block Designs

2.2 The Kruskal-Wallis test

Tomato example

```
# create and re-arrange data frame
rm(list=ls()) # removes any existing objects from environment

# flavour scores
tom1 <- data.frame(
  Floradade = c(43, 5, 74, 64, 10, 16, 75, 20, 36, 76, 60, 57,
                55, 29, 82, 91, 66, 27, 72, 108, 84, 50, 82, 39),
  Momotara = c(74, 112, 64, 101, 105, 12, 33, 90, 129, 37, 50, 44,
                18, 24, 48, 62, 88, 50, 73, 119, 109, 50, 12, 37),
  Summit = c(109, 25, 48, 91, 52, 35, 42, 100, 22, 122, 105, 119,
              29, 26, 102, 48, 108, 53, 57, 82, 105, 108, 13, 74),
  Rutgers = c(39, 82, 100, 62, 126, 26, 24, 35, 74, 19, 113, 56,
              61, 21, 6, 13, 118, 91, 60, 88, 15, 32, 134, 29)
)

tom2 <- stack(tom1)
names(tom2) <- c("flavour", "variety") # change default column names to
                                         # something more meaningful
```

This initial code chunk creates the necessary data structures. The `tom1` data frame has separate columns for the flavour scores for each tomato variety. This “wide” format is useful for display in tables as shown below.

##	Floradade	Momotara	Summit	Rutgers
## 1	43	74	109	39
## 2	5	112	25	82
## 3	74	64	48	100
## 4	64	101	91	62
## 5	10	105	52	126
## 6	16	12	35	26
## 7	75	33	42	24
## 8	20	90	100	35
## 9	36	129	22	74

## 10	76	37	122	19
## 11	60	50	105	113
## 12	57	44	119	56
## 13	55	18	29	61
## 14	29	24	26	21
## 15	82	48	102	6
## 16	91	62	48	13
## 17	66	88	108	118
## 18	27	50	53	91
## 19	72	73	57	60
## 20	108	119	82	88
## 21	84	109	105	15
## 22	50	50	108	32
## 23	82	12	13	134
## 24	39	37	74	29

The `tom2` data frame is the result of *stacking* these multiple columns into a single column and creating a factor indicating where each observation originated. A selection of rows from this data frame is shown below. This “long” form is not as useful for display purposes (too many rows) but is much more useful for analysis. This is the usual format that data will need to be in for analysis in R or most other statistical software. A `for` loop was used in the Chapter 1 examples to convert data into this format but using the `stack()` function as shown here is more efficient.



 **MTHøjgaard**

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang

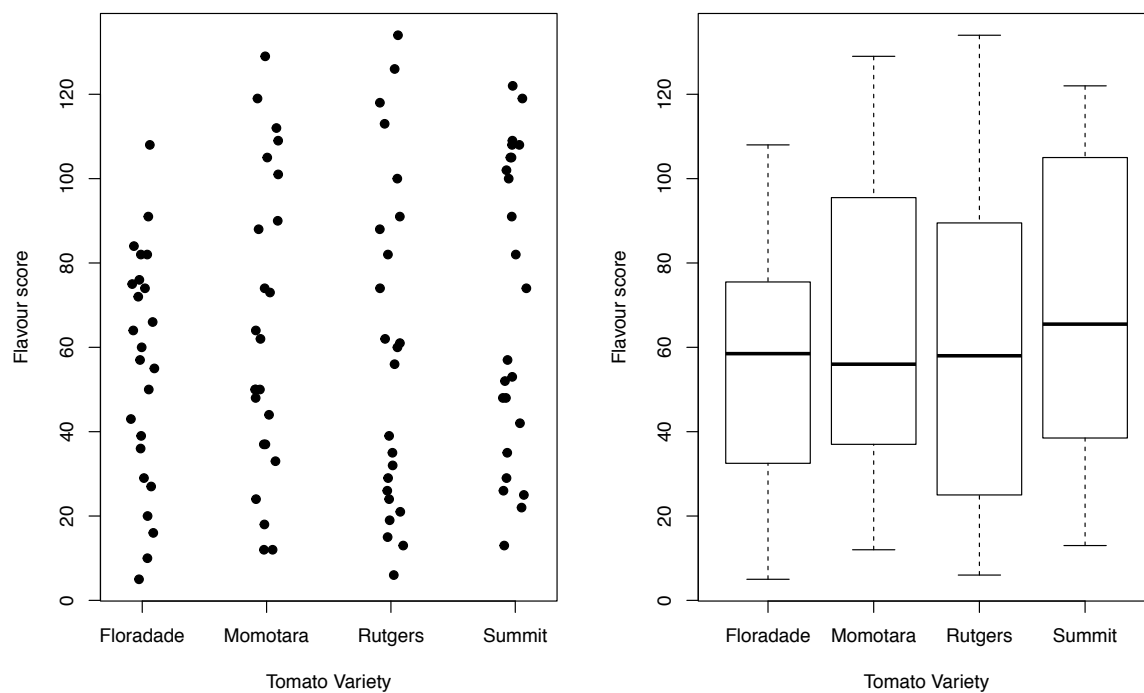


Figure 2.1: Dot and box plots of flavour scores for each of four varieties.

```
## flavour variety
## 1      43 Floradade
## 2       5 Floradade
## 25     74 Momotara
## 26    112 Momotara
## 49    109 Summit
## 50     25 Summit
## 73     39 Rutgers
## 74     82 Rutgers
```

Note that the original data in this example is in the form of a data frame, `tom1`, which requires component vectors to be the same length. In the [herbicide example at the start of Chapter 3](#), a list data structure is used instead because the source vectors are of different lengths.

Graphing the data, as shown in Figure 2.1, is a useful first step in any analysis. Dot plots were used in the Chapter 1 examples and are shown again here but box plots as shown in the right hand graph are a better choice for larger numbers of data points. These are sometimes known as box and whisker plots. The bottom of the box indicates the first quartile, the top of the box the third quartile and the center line the median. The whiskers extend below the box to the minimum value and above the box to the maximum value.

The following code ranks the combined flavour scores, sums the ranks for each treatment group and calculates the Kruskal Wallis test statistic and p-value.

```
# perform Kruskal-Wallis test
tom2$r <- rank(tom2$flavour, ties.method = "average")

# sum of ranks for different tomato varieties
F <- sum(tom2$r[tom2$variety=="Floradade"])
M <- sum(tom2$r[tom2$variety=="Momotara"])
S <- sum(tom2$r[tom2$variety=="Summit"])
R <- sum(tom2$r[tom2$variety=="Rutgers"])

# Kruskal-Wallis test statistic and p-value
n.F <- n.M <- n.S <- n.R <- 24
n <- n.F + n.M + n.S + n.R
KW <- 12/n/(n+1) * (F^2/n.F + M^2/n.M + S^2/n.S + R^2/n.R) - 3*(n+1)
pval <- pchisq(q=KW, df=3, lower.tail=FALSE)
```

The resultant rank sums are $F = 1058$, $M = 1200$, $S = 1303.5$, $R = 1094.5$ and the Kruskal-Wallis test statistic is $KW = 1.9771531$. Comparing this to the chi-squared distribution with 3 degrees of freedom gives a p-value of $P(KW > 1.9771531) = 0.5771622$. Based on this large p-value there is no evidence of any difference in distribution of flavour scores for the different varieties.

This code has been written just to demonstrate the calculations. In practice, it would be more usual (and certainly simpler) to use the pre-existing `kruskal.test` function in R.

```
# using the pre-existing function
kruskal.test(flavour~variety, data=tom2)

##
##  Kruskal-Wallis rank sum test
##
## data:  flavour by variety
## Kruskal-Wallis chi-squared = 1.978, df = 3, p-value = 0.577
```

There is a very minor difference in the value of the test statistic calculated by the pre-existing function as it is also applying a correction for the number of tied ranks in the data set. This improves the accuracy of the chi-square approximation to the null distribution of the test statistic.

Note that the Wilcoxon rank sum test covered in Chapter 1 is essentially a special case of the Kruskal-Wallis test for two treatment groups. You may wish to review [the flints example \(p 24\)](#) and [the teaching methods example \(p 44\)](#) using the Kruskal-Wallis method. Your results should be consistent with those calculated previously using the Wilcoxon rank sum test.

For comparison, the one-way ANOVA F test results are presented below for the tomato example.

```
# alternative parametric analysis
m.tom <- lm(flavour~variety, data=tom2) # fit the underlying linear model
anova(m.tom) # present model results as an ANOVA table

## Analysis of Variance Table
##
## Response: flavour
##           Df Sum Sq Mean Sq F value Pr(>F)
## variety    3   2906   968.65   0.7986 0.4978
## Residuals  92 111592 1212.96
```

2.3 The Friedman test

Lemonade example

The data for the lemonade example is shown below. This is the transpose of the tabular presentation used in [Rayner \(2016\)](#) but is still a reasonable way to present the data set.

```
# creating data frame
rm(list=ls()) # removes any existing objects from environment
sdrink <- data.frame(L1 = c(5,3,4,5,3,4,5,3,1,3),
                     L2 = c(2,5,3,2,5,3,2,5,4,1),
                     L3 = c(1,2,2,1,2,2,1,2,2,2),
                     L4 = c(3,1,5,3,1,5,3,1,5,4),
                     L5 = c(4,4,1,4,4,1,4,4,3,5),
                     judge = 1:10)

sdrink
```

```
##      L1 L2 L3 L4 L5 judge
## 1      5  2  1  3  4      1
## 2      3  5  2  1  4      2
## 3      4  3  2  5  1      3
## 4      5  2  1  3  4      4
## 5      3  5  2  1  4      5
## 6      4  3  2  5  1      6
## 7      5  2  1  3  4      7
## 8      3  5  2  1  4      8
## 9      1  4  2  5  3      9
## 10     3  1  2  4  5     10
```

Again however this data frame needs to be converted into a stacked or long form for subsequent analysis. As we also have to take into account the `judge` variable, the simple `stack()` function used in the previous example is insufficient. The code below uses the `melt()` function from the `reshape2` package and displays a selection of rows from the new data frame. (NB There are other functions that could be used to achieve a similar result but the `reshape2` functions seem to work well.)

```
# rearrange data frame to more useful long form
library(reshape2) # load required package
sdrink2 <- melt(data=sdrink, id.vars="judge", value.name="rank")
names(sdrink2)[2] <- "type" # rename column
sdrink2$judge <- as.factor(sdrink2$judge)
sdrink2[c(1,2,11,12,21,22,31,32,41,42),]
```

```
##      judge type rank
## 1         1   L1    5
## 2         2   L1    3
## 11        1   L2    2
## 12        2   L2    5
## 21        1   L3    1
## 22        2   L3    2
## 31        1   L4    3
## 32        2   L4    1
## 41        1   L5    4
## 42        2   L5    4
```

The following code demonstrates the calculation of the Friedman test statistic.

```
# perform Friedman test
b <- nlevels(sdrink2$judge) # no of blocks (ie no of judges)
t <- nlevels(sdrink2$type)  # no of treatments, ie lemonade types

# sum the ranks for each of the five lemonades over all 10 judges
Ri <- aggregate(rank~type, FUN=sum, data=sdrink2)

# calculate Friedman test statistic and p-value
S <- 12/b/t/(t+1)*sum(Ri$rank^2) - 3*b*(t+1)
pval <- pchisq(q=S, df=t-1, lower.tail=FALSE)
```

Note the use of the `aggregate()` function to efficiently calculate the sum of the ranks for each lemonade over all 10 judges without using a loop. It applies the nominated function `sum()` to the `rank` column for the subsets defined by `type`.

The Friedman test statistic is $S = 9.04$. Comparing this to the chi-squared distribution with 4 degrees of freedom gives a p-value of $P(S > 9.04) = 0.0601074$. This p-value suggests weak evidence of a difference in ratings of lemonades.

The previous code demonstrates the calculations. In practice, it would be more usual to use the pre-existing `friedman.test()` function in R.

```
# using pre-existing function
friedman.test(rank~type|judge, data=sdrink2)

##
##  Friedman rank sum test
##
## data:  rank and type and judge
## Friedman chi-squared = 9.04, df = 4, p-value = 0.06011
```



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



2.4 The Durbin test

Ice Cream example

```
# creating and arranging data
rm(list=ls()) # removes any existing objects from environment

x <- scan(what=list(judge=character(), variety=character(), rank=integer()),
          nlines=22, text="1 S 2
                          1 U 3
                          1 W 1
                          2 U 3
                          2 V 1
                          2 X 2
                          3 V 2
                          3 W 1
                          3 Y 3
                          4 W 1
                          4 X 2
                          4 Z 3
                          5 S 3
                          5 X 1
                          5 Y 2
                          6 U 3
                          6 Y 1
                          6 Z 2
                          7 S 3
                          7 V 1
                          7 Z 2")

ice.cream <- as.data.frame(x)
```

The previous code creates the data frame `ice.cream` containing the taste test data involving seven ice cream varieties coded S, U, V, W, X, Y and Z, and presented three at a time to each of seven judges. Usually data like this would be read from an external file. The use of the `scan()` function here is simply to enable the data to be defined within the code statements eliminating the need for additional files.

```
# calculation of Durbin test statistic
b <- nlevels(ice.cream$judge) # no of blocks (the 7 judges)
t <- nlevels(ice.cream$variety) # no of treatments
                                # (the 7 different ice creams)
k <- length(unique(ice.cream$rank)) # no treatments in each block
                                # (each block contains 3 varieties)

# determine no of blocks containing each treatment
# (each icecream is judged by 3 judges)
check.r <- aggregate(variety~judge, data=ice.cream, FUN=length)
r <- check.r$variety[1]
```



```
# sum the ranks for each of the 7 ice creams over all 7 judges
Ri <- aggregate(rank~variety, FUN=sum, data=ice.cream)

# calculate Durbin test statistic and p-value
D <- 12*(t-1)/b/k/(k^2-1)*sum(Ri$rank^2) - 3*r*(t-1)*(k+1)/(k-1)
pval <- pchisq(q=D, df=t-1, lower.tail=FALSE)
```

This code assumes the data is correct. There is no checking that the data meets the constraints implied by the balanced incomplete block design. The sum of the ranks for each of the 7 ice creams over all 7 judges is 8, 9, 4, 3, 5, 6, 7. The Durbin test statistic is $D = 12$ and comparing to a chi-squared distribution with 6 degrees of freedom yields a p-value of $P(D > 12) = 0.0619688$ providing weak evidence of a difference in the distribution of rankings for the ice cream varieties.

The previous code demonstrates the calculations. There is also a pre-existing `durbin.test()` function in the `agricolae` package. Use of this function is demonstrated below.

```
# using pre-defined function
library(agricolae)
comparison <- durbin.test(judge=ice.cream$judge, trt=ice.cream$variety,
                          evaluation=ice.cream$rank, console=TRUE)

##
## Study: ice.cream$rank ~ ice.cream$judge + ice.cream$variety
## ice.cream$variety, Sum of ranks
##
##      sum
## S      8
## U      9
## V      4
## W      3
## X      5
## Y      6
## Z      7
##
## Durbin Test
## =====
## Value      : 12
## Df 1       : 6
## P-value    : 0.0619688
## Alpha     : 0.05
## Df 2       : 8
## t-Student  : 2.306004
##
## Least Significant Difference
## between the sum of ranks: 2.824267
##
## Parameters BIB
## Lambda     : 1
## treatmeans : 7
## Block size : 3
```

```
## Blocks      : 7
## Replication: 3
##
## Groups, Treatments and sum of the ranks
##
## a    U    9
## ab   S    8
## abc  Z    7
## bcd  Y    6
## cde  X    5
## de   V    4
## e    W    3
```

Note that the Durbin test collapses to the Friedman test when applied to a randomized complete block design.

2.5 Relationships of Kruskal-Wallis, Friedman and Durbin tests with ANOVA F tests

Transformations of the test statistics KW, S and D based on their relationships with the ANOVA F test of the ranks may be used to improve approximate sampling distributions.



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

2.5.1 The completely randomised design

Tomato example

Using objects previously calculated for [the tomato example \(p 46\)](#), the following code calculates a transformation of the Kruskal Wallis test statistic ($KW = 1.9771531$) based on an ANOVA F test of the ranks.

```
# transformation of KW test statistic
t <- nlevels(tom2$variety) # no of tomato varieties
n <- nrow(tom2) # total number of observations
d <- 1 - 6*(n+1)/(n-1)/(5*n+6) # adjustment to F degrees of freedom

F <- KW / (n-1-KW) * (n-t) / (t-1) # based on KW test statistic

pval <- pf(F, df1=d*(t-1), df2=d*(n-t), lower.tail=FALSE) # note adjusted df
```

Algebraically,

$$F = \frac{KW(n-t)}{(n-1-KW)(t-1)} = \frac{1.9771531(96-4)}{(96-1-1.9771531)(4-1)} = 0.6518043.$$

Using the $F_{3,92}$ distribution, the p-value based on this transformed test statistic is $P(F > 0.6518043) = 0.5838219$. Adjusting the degrees of freedom by the multiplicative factor $d = 0.9873944$ yields an improved $F_{2.9621832,90.8402859}$ approximation to the null distribution giving a more accurate p-value of $P(F > 0.6518043) = 0.5819512$. In this example there is not much difference between these two values or the value $P(KW > 1.9771531) = 0.5771622$ used previously based on the χ^2_3 distribution.

2.5.2 The randomised block design

Lemonade example

Using objects previously calculated for [the lemonade example \(p 50\)](#), the following code calculates a transformation of the Friedman test statistic ($S = 9.04$) based on an ANOVA F test of the ranks.

```
# transformation of Friedman test statistic
F <- S * (b-1) / (b * (t-1) - S) # based on S test statistic
pval <- pf(F, df1=(t-1), df2=(b-1)*(t-1), lower.tail=FALSE)
```

Algebraically,

$$F = \frac{S(b-1)}{b(t-1) - S} = \frac{9.04(10-1)}{10(5-1) - 9.04} = 2.627907.$$

Using the $F_{4,36}$ distribution, the p-value based on this transformed test statistic is $P(F > 2.627907) = 0.0503702$ which is slightly less than the previously calculated p-value $P(S > 9.04) = 0.0601074$ based on the χ^2_4 distribution.

2.5.3 The balanced incomplete block design

Ice Cream example

Using objects previously calculated for [the ice cream example \(p 53\)](#), the following code calculates a transformation of the Durbin test statistic ($D = 12$) based on an ANOVA F test of the ranks.

```
# transformation of Durbin test statistic
F <- D*(b*k-b-t+1)/(b*(k-1)-D)/(t-1) # based on D test statistic

pval <- pf(F, df1=(t-1), df2=(b*k-b-t+1), lower.tail=FALSE)
```

Algebraically,

$$F = \frac{D}{(b(k-1) - D)} \frac{(bk - b - t + 1)}{(t - 1)} = \frac{12}{(7(3 - 1) - 12)} \frac{(7 \times 3 - 7 - 7 + 1)}{(7 - 1)} = 8.$$

Using the $F_{6,8}$ distribution, the p-value based on this transformed test statistic is $P(F > 8) = 0.0049044$ which is considerably less than the previously calculated p-value $P(D > 12) = 0.0619688$ based on the χ^2_6 distribution.

2.6 Orthogonal contrasts: Page and umbrella tests

[Rayner \(2016, Table 2.4\)](#) gives linear and quadratic coefficients for constructing Page and umbrella tests for balanced designs. The R function `contr.poly()` returns a contrast matrix based on orthogonal polynomials as shown below for designs with three treatments.

```
contr.poly(n=3)

##           .L           .Q
## [1,] -7.071068e-01  0.4082483
## [2,] -7.850462e-17 -0.8164966
## [3,]  7.071068e-01  0.4082483
```

Note that the first column of the returned matrix corresponds to the contrast coefficients $\frac{-1}{\sqrt{2}}, 0, \frac{1}{\sqrt{2}}$ for a linear trend test whereas the second column corresponds to the contrast coefficients $\frac{1}{\sqrt{6}}, \frac{-2}{\sqrt{6}}, \frac{1}{\sqrt{6}}$ for a quadratic trend test.

Tomato example

The following code chunk uses objects created previously in [the tomato example \(p 46\)](#). This example corresponds to a completely randomized design. The `tom2` data frame contains columns for the flavour score, the tomato variety and the rank of that flavour score within the set of combined scores.

```

# calculation of orthogonal contrasts
# sum of ranks for different tomato varieties
Ri <- aggregate(r~variety, FUN=sum, data=tom2)

ni <- xtabs(~variety, data=tom2) # no of observations for each variety
n <- sum(ni) # total no of observations

t <- nlevels(tom2$variety) # no of treatments, ie tomato varieties

nRi <- (Ri$r - (n+1)*ni/2)/sqrt(n*(n+1)*ni/12) # normalized Ri values

# calculate test statistics and p-values
Z <- nRi[c(1,2,4,3)] %%% contr.poly(n=t) # note the use of the index to change
                                         # ordering of the tomato varieties to
                                         # match Rayner(2016).
pvals <- 2*pnorm(q=abs(Z), lower.tail=FALSE)

```

The code shown here has calculated test statistics and p-values corresponding to tests of a linear trend ($Z_1 = 0.3490022$, with two=sided p-value $2P(Z > 0.3490022) = 0.7270877$), a quadratic trend ($Z_2 = -1.285999$, with two=sided p-value $2P(Z > 1.285999) = 0.1984434$) and cubic trend ($Z_3 = -0.4489511$, with two=sided p-value $2P(Z > 0.4489511) = 0.6534669$).

Note that matrix calculations used here provide all $t - 1 = 3$ orthogonal contrasts at once but as suggested by Rayner (2016), it is very rare that anything other than the first two contrasts would be specifically considered with the rest perhaps being aggregated into a residual.



Max's next Bookboon eBook
Your Boss: Sorted!
 By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

These results provide no evidence against the null hypotheses of no linear, quadratic or cubic effects.

This example assumes that the ordering of the treatments (ie tomato varieties) as presented in [Rayner \(2016\)](#) was meaningful in some way. The concepts of linear or quadratic trend are nonsensical if there is no meaningful ordering of categories. Note that the ordering of the rank sums in these calculations needed to be changed in order to match the results presented in [Rayner \(2016\)](#). (This is just an illustrative example. There is nothing in the information provided that suggests that there is any meaningful ordering of these categories, unlike the next example.)

Lemonade example

The following code chunk uses objects created previously in [the lemonade example \(p 50\)](#). This example corresponds to a randomized block design. The `sdrink2` data frame contains columns for the type of lemonade, the identification of the person judging, and the rank given to that lemonade by that judge. The `Ri` data frame contains the sum of the ranks for each of the five lemonades over all 10 judges.

```
# calculation of orthogonal contrasts - test statistics and p-values  
obs.Z <- Ri$rank %*% contr.poly(n=t) / sqrt(b*t*(t+1)/12)  
pvals <- 2*pnorm(q=abs(obs.Z), lower.tail=FALSE)
```

The code shown here has calculated test statistics and p-values corresponding to tests of a linear trend ($Z_1 = -0.3162278$, with two=sided p-value $2P(Z > -0.3162278) = 0.7518296$), and quadratic trend ($Z_2 = 2.2984467$, with two=sided p-value $2P(Z > 2.2984467) = 0.0215364$). (Cubic and quartic polynomial contrasts are also calculated but are not of any particular interest.)

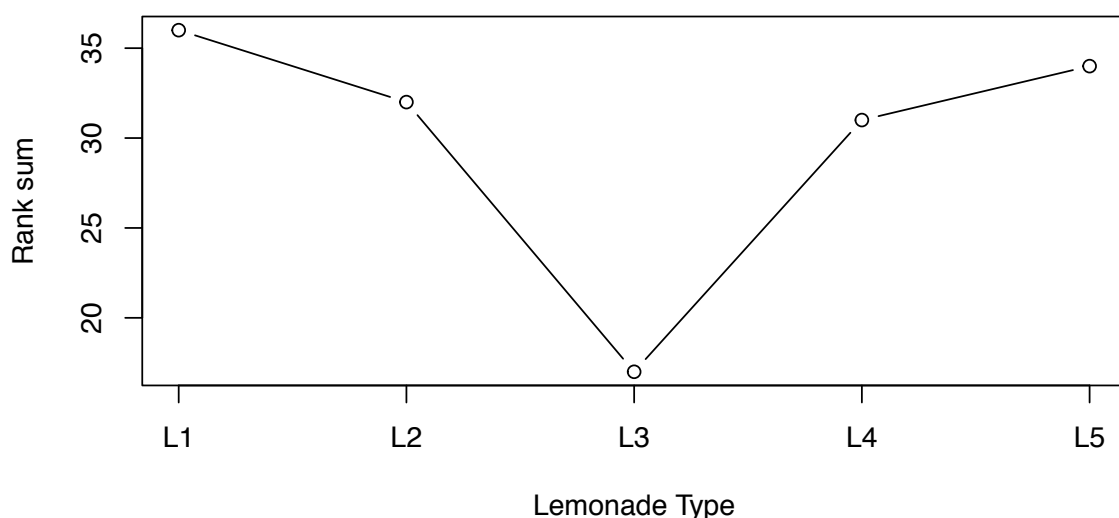


Figure 2.2: Plot of rank sums.

Clearly there is evidence of a quadratic or umbrella effect but not of a linear effect. This is consistent with the plot of the rank sums shown in [Figure 2.2](#). Remember that the lemonades were in order of increasing sugar content. So the judges rank the sweeter lemonades more highly (lower ranks) up to a certain point, after which they are considered too sweet and are ranked lower.

Ice Cream example

The following code chunk uses objects created previously in [the ice cream example \(p 53\)](#). This example corresponds to a a balanced incomplete block design. The `ice.cream` data frame contains columns for the type of ice cream, the identification of the person judging and the rank given to that lemonade by that judge (within the sub-group considered by that judge). The `Ri` data frame contains the sum of the ranks for each of the for each of the seven ice creams over all seven judges.

```
# calculation of orthogonal contrasts - test statistics and p-values  
obs.Z <- Ri$rank %*% contr.poly(n=t) / sqrt(b*k*(k^2-1)/12/(t-1))  
pvals <- 2*pnorm(q=abs(obs.Z), lower.tail=FALSE)
```

The code shown here has calculated test statistics and p-values corresponding to tests of a linear trend ($Z_1 = -0.9897433$, with two=sided p-value $2P(Z > -0.9897433) = 0.3222996$), and quadratic trend ($Z_2 = 2.5714286$, with two=sided p-value $2P(Z > 2.5714286) = 0.010128$). (Higher order contrasts are also calculated but are not of any particular interest.)

Clearly there is evidence of a quadratic or umbrella effect but not of a linear effect. This is consistent with the plot of the rank sums shown in [Figure 2.3](#). NB This is just an illustrative example. There is nothing in the information provided in [Rayner \(2016\)](#) that suggests that there is any meaningful ordering of these categories. This apparent umbrella effect only has meaning if there is some information in the ordering of the ice cream categories.

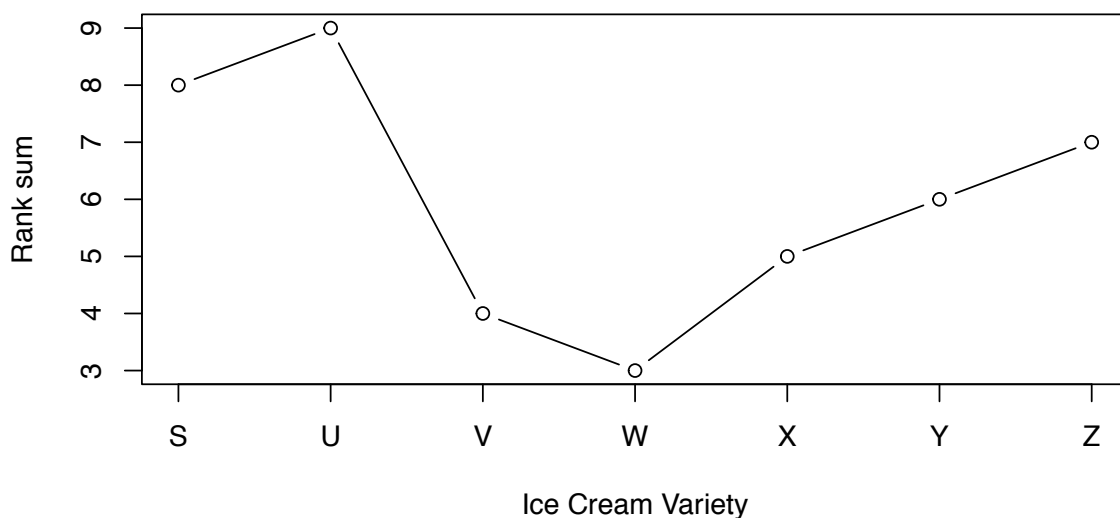


Figure 2.3: Plot of rank sums.

3 Permutation Testing

3.1 What is permutation testing and why it is important?

Herbicide example

```
# creating data frames
rm(list=ls()) # removes any existing objects from environment

strawberry1 <- list(herbicide=c(0.44, 0.47, 0.51, 0.52, 0.58, 0.59,
                                0.60, 0.65, 0.66),
                    none=c(0.55, 0.63, 0.67, 0.68, 0.79, 0.81, 0.85)
)

strawberry2 <- stack(strawberry1)
# change default column names to something more meaningful
names(strawberry2) <- c("dry.weight", "treatment")
```



MTHøjgaard

**BEDRE
LØSNINGER**

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang



Although already covered in some detail in [Rayner \(2016\)](#), the herbicide example is repeated here with slightly different R code with explanatory comments.

The `strawberry2` data frame created here is in stacked form with a single column for the dry weights of each strawberry plant and another column indicating whether the plant was treated with herbicide or not.

Note that this method of creating the stacked data frame is almost identical to that used previously in the [the tomato example \(p 46\)](#). The only difference is that the original data that is passed to the `stack()` function here is in the form of a list which allows the component vectors to be of different lengths. In the tomato example the original data was in the form of a data frame which requires component vectors to be the same length.

```
# rank the combined dry weight values and sum the ranks for untreated plants
strawberry2$r <- rank(strawberry2$dry.weight)
teststat.obs <- sum(strawberry2$r[strawberry2$treatment=="none"])
```

After ranking the combined dry weight values, the sum of the ranks for the untreated strawberry plants is 84. Does this provide evidence that the untreated plants have higher dry weights?

```
# use random sampling to approximate null distribution of test statistic
nperm <- 10000 # no of permutations
n.t <- nrow(strawberry2) # total number of plants
n.u <- nrow(strawberry2[strawberry2$treatment=="none",]) # no untreated plants
# function to generate random sample of n.u ranks and sum
sr <- function(X) sum(sample(x=1:n.t, size=n.u))
# generate approximate null distribution of test statistic based on nperm
# random samples
sr.dist <- sapply(1:nperm, FUN=sr)
# compare observed test statistic to approx distribution to determine p-value
pval <- length(sr.dist[sr.dist>=teststat.obs])/nperm
```

Figure 3.1 shows an approximation of the distribution of the test statistic under the null hypothesis of no difference in weights. Under this constraint, the ranks would be a random arrangement and so the distribution is approximated by taking `nperm`= 10^4 random samples of 7 ranks (the number of untreated plants) from the set 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and summing the ranks in each sample to effectively get 10^4 randomly sampled values from the null distribution of the test statistic.

In the code above, the defined `sr()` function will create a single such random sample using the `sample()` function and sum the ranks. The next line uses the `sapply()` function to repeat this `nperm` times. This is more efficient in R than using a for loop.

Clearly the observed test statistic of 84 is relatively unlikely. The shading in Figure 3.1 illustrates the estimated one tailed p-value, $P(S_{\text{none}} \geq 84) = 0.0046$. This is the proportion of the `nperm`= 10^4 random samples that result in a test statistic more extreme than that calculated from the observed sample.

This is very similar to the exact p-value that can be found using the `wilcox.test()` function.

```
wilcox.test(dry.weight~treatment, data=strawberry2, alternative="less",
            exact=TRUE)
```

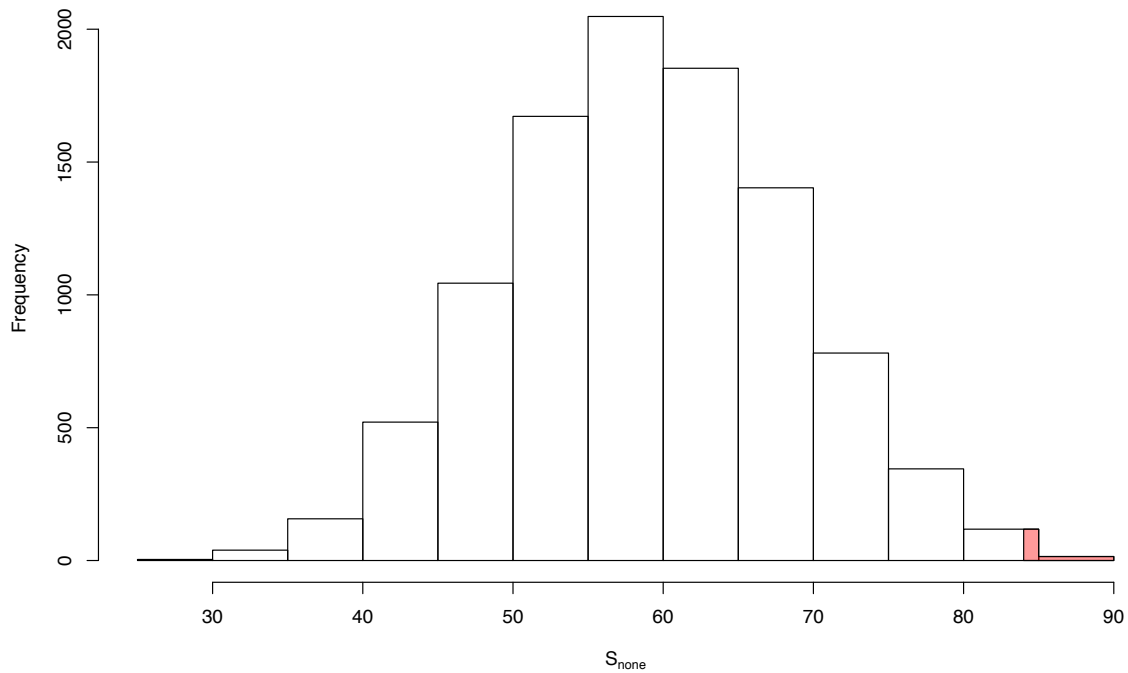


Figure 3.1: Approximate null distribution of test statistic based on random permutations.



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



```
##
## Wilcoxon rank sum test
##
## data: dry.weight by treatment
## W = 7, p-value = 0.003934
## alternative hypothesis: true location shift is less than 0
```

Remember that any p-value that is based on simulation of random sampling is likely to be a little different each time the code is run. Increasing the size of the simulation will reduce the variability in the p-value output but will require extra computing time.

3.2 Nonparametric multifactor ANOVA when the levels of the factors are unordered

Tomato example.

Using objects previously calculated for [the tomato example \(p 46\)](#), the following code calculates the central sample moments up to order 6.

```
# calculating central moments for nonparametric ANOVA
x <- tom2$flavour

m1 <- mean(x)
dv <- x - m1 # deviations from the mean
t <- 1
cm <- rep(NA, 6)
for (i in 1:6) {
  t <- t * dv
  cm[i] <- mean(t)
}
cm

## [1] 0.000000e+00 1.192689e+03 9.539087e+03 2.747648e+06 5.156285e+07
## [6] 8.071673e+09
```

Alternatively, the `moments` package provides a convenient function as shown below.

```
# alternative method for calculating central moments
library(moments)
cm <- all.moments(tom2$flavour, order.max=6, central=TRUE)[-1]
# note the index of -1 to discard the 0th central moment.
cm

## [1] 0.000000e+00 1.192689e+03 9.539087e+03 2.747648e+06 5.156285e+07
## [6] 8.071673e+09
```

Using either method the computed central moments are the same. The next step is to transform the data using the orthonormal polynomials.

```
# calculating orthonormal polynomial transformation
a1 <- dv / sqrt(cm[2])
d <- cm[4] - cm[3]^2/cm[2] - cm[2]^2
a2 <- (dv^2 - cm[3]*dv/cm[2] - cm[2]^2) / sqrt(d)
a <- (cm[5] - cm[3]*cm[4]/cm[2] - cm[2]*cm[3]) / d
b <- (cm[4]^2/cm[2] - cm[2]*cm[4] - cm[3]*cm[5]/cm[2] + cm[3]^2) / d
c <- (2*cm[3]*cm[4] - cm[3]^3/cm[2] - cm[2]*cm[5]) / d
e <- cm[6] - 2*a*cm[5] + (a^2-2*b)*cm[4] + 2*(a*b-c)*cm[3] +
  (b^2+2*a*c)*cm[2] + c^2
a3 <- (dv^3 - a*dv^2 - b*dv - c) / sqrt(e)

# reproduce Rayner(2016) Table 3.2
sset <- order(tom2$flavour)[1:8]
data.frame(response=tom2$flavour, variety=tom2$variety, first.order=a1,
  second.order=a2, third.order=a3)[sset,]
```

##	response	variety	first.order	second.order	third.order
## 2	5 Floradade	-1.653198	2.258255	-2.7114063	
## 87	6 Rutgers	-1.624242	2.149813	-2.4676581	
## 5	10 Floradade	-1.508419	1.733943	-1.5893158	
## 30	12 Momotara	-1.450507	1.536747	-1.2058779	
## 47	12 Momotara	-1.450507	1.536747	-1.2058779	
## 71	13 Summit	-1.421551	1.440833	-1.0275281	
## 88	13 Rutgers	-1.421551	1.440833	-1.0275281	
## 93	15 Rutgers	-1.363640	1.254374	-0.6967605	

We can now use the linear model function, `lm()`, to calculate the relevant F statistics from the vectors `a1`, `a2` and `a3` which contain the transformed responses using the first, second and third orthonormal polynomials respectively.

```
# calculate F statistics
(aov.a1 <- anova(lm(a1~tom2$variety)))

## Analysis of Variance Table
##
## Response: a1
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3  2.436  0.81216  0.7986 0.4978
## Residuals   92 93.564  1.01699

F.a1 <- aov.a1["tom2$variety", "F value"]
(aov.a2 <- anova(lm(a2~tom2$variety)))

## Analysis of Variance Table
##
## Response: a2
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3  4.394  1.46453  1.4708 0.2277
## Residuals   92 91.606  0.99572
```

```
F.a2 <- aov.a2["tom2$variety", "F value"]
(aov.a3 <- anova(lm(a3~tom2$variety)))

## Analysis of Variance Table
##
## Response: a3
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3  1.548 0.51587  0.5025 0.6815
## Residuals   92 94.452 1.02666

F.a3 <- aov.a3["tom2$variety", "F value"]
```

Alternatively, a much simpler approach is to use the standard function `poly()` to calculate the polynomials. NB These polynomials are orthogonal but not orthonormal. However this is just a scaling factor and so does not affect the calculation of the p-values as shown below.

```
# alternative calculations using poly function
tom.A123 <- poly(tom2$flavour, degree=3)
(tom.aov.A1 <- anova(lm(tom.A123[,1]~tom2$variety)))

## Analysis of Variance Table
##
## Response: tom.A123[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3 0.02538 0.008460  0.7986 0.4978
## Residuals   92 0.97462 0.010594

tom.F.A1 <- tom.aov.A1["tom2$variety", "F value"]
(tom.aov.A2 <- anova(lm(tom.A123[,2]~tom2$variety)))

## Analysis of Variance Table
##
## Response: tom.A123[, 2]
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3 0.04577 0.015256  1.4708 0.2277
## Residuals   92 0.95423 0.010372

tom.F.A2 <- tom.aov.A2["tom2$variety", "F value"]
(tom.aov.A3 <- anova(lm(tom.A123[,3]~tom2$variety)))

## Analysis of Variance Table
##
## Response: tom.A123[, 3]
##           Df Sum Sq Mean Sq F value Pr(>F)
## tom2$variety 3 0.01612 0.0053737  0.5025 0.6815
## Residuals   92 0.98388 0.0106943

tom.F.A3 <- tom.aov.A3["tom2$variety", "F value"]
```


As noted by [Rayner \(2016\)](#), these p-values are based on the usual F distribution. Permutation based p-values are calculated for this example on [page 72](#).

Lemonade example

The following code applies this non-parametric ANOVA methodology to objects previously created for [the lemonade example \(p 50\)](#).

```
# non-parametric ANOVA
sdrink.A123 <- poly(sdrink2$rank, degree=3)
(sdrink.aov.A1 <- anova(lm(sdrink.A123[,1]~sdrink2$type+sdrink2$judge)))

## Analysis of Variance Table
##
## Response: sdrink.A123[, 1]
##              Df Sum Sq Mean Sq F value    Pr(>F)
## sdrink2$type   4  0.226  0.0565   2.6279 0.05037 .
## sdrink2$judge   9  0.000  0.0000   0.0000 1.00000
## Residuals     36  0.774  0.0215
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

sdrink.F.A1 <- sdrink.aov.A1["sdrink2$type", "F value"]
(sdrink.aov.A2 <- anova(lm(sdrink.A123[,2]~sdrink2$type+sdrink2$judge)))
```



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired


```
## Analysis of Variance Table
##
## Response: sdrink.A123[, 2]
##           Df Sum Sq Mean Sq F value Pr(>F)
## sdrink2$type  4 0.02429  0.0060714    0.224 0.9232
## sdrink2$judge  9 0.00000  0.0000000    0.000 1.0000
## Residuals    36 0.97571  0.0271032

sdrink.F.A2 <- sdrink.aov.A2["sdrink2$type", "F value"]
(sdrink.aov.A3 <- anova(lm(sdrink.A123[,3]~sdrink2$type+sdrink2$judge)))

## Analysis of Variance Table
##
## Response: sdrink.A123[, 3]
##           Df Sum Sq Mean Sq F value Pr(>F)
## sdrink2$type  4  0.334  0.0835  4.5135 0.00466 **
## sdrink2$judge  9  0.000  0.0000  0.0000 1.00000
## Residuals    36  0.666  0.0185
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

sdrink.F.A3 <- sdrink.aov.A3["sdrink2$type", "F value"]
```

Results match [Rayner \(2016, Table 3.4\)](#). The p-values suggest that there are first and third order effects.

Code which reproduces other results from [Rayner \(2016, Tables 3.4 and 3.5\)](#) is shown below.

```
# Shapiro-Wilk normality tests of residuals from ANOVA models
# reproduces second row in Rayner(2016) Table 3.4
shapiro.test(residuals(lm(sdrink.A123[,1]~sdrink2$type+sdrink2$judge)))

##
## Shapiro-Wilk normality test
##
## data: residuals(lm(sdrink.A123[, 1] ~ sdrink2$type + sdrink2$judge))
## W = 0.94488, p-value = 0.0211

shapiro.test(residuals(lm(sdrink.A123[,2]~sdrink2$type+sdrink2$judge)))

##
## Shapiro-Wilk normality test
##
## data: residuals(lm(sdrink.A123[, 2] ~ sdrink2$type + sdrink2$judge))
## W = 0.83334, p-value = 5.64e-06

shapiro.test(residuals(lm(sdrink.A123[,3]~sdrink2$type+sdrink2$judge)))
```

```
##
## Shapiro-Wilk normality test
##
## data: residuals(lm(sdrink.A123[, 3] ~ sdrink2$type + sdrink2$judge))
## W = 0.93917, p-value = 0.0125

# averages of the transformed ranks for each lemonade type.
# this provides similar information as Rayner(2016) Table 3.5 but the scale
# is different because the polynomials are only orthogonal not orthonormal.
aggregate(as.data.frame(sdrink.A123), by=list(sdrink2$type), FUN=mean)

##   Group.1      1      2      3
## 1      L1  0.06 -1.690309e-02 -0.02
## 2      L2  0.02  5.548405e-18  0.06
## 3      L3 -0.13 -8.451543e-03  0.11
## 4      L4  0.01  4.225771e-02 -0.02
## 5      L5  0.04 -1.690309e-02 -0.13
```

Ice Cream example

The following code applies this non-parametric ANOVA methodology to objects previously created for [the ice cream example \(p 53\)](#).

```
ice.cream$judge <- factor(ice.cream$judge)

ice.cream$A12 <- poly(ice.cream$rank, degree=2)
(ice.cream.aov.A1 <- anova(lm(A12[,1] ~ judge + variety, data=ice.cream)))

## Analysis of Variance Table
##
## Response: A12[, 1]
##           Df Sum Sq Mean Sq F value    Pr(>F)
## judge      6 0.00000  0.000000      0 1.000000
## variety    6 0.85714  0.142857      8 0.004904 **
## Residuals  8 0.14286  0.017857
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

shapiro.test(residuals(lm(A12[,1] ~ judge + variety, data=ice.cream)))

##
## Shapiro-Wilk normality test
##
## data: residuals(lm(A12[, 1] ~ judge + variety, data = ice.cream))
## W = 0.95628, p-value = 0.4445

ice.cream.F.A1 <- ice.cream.aov.A1["variety", "F value"]
(ice.cream.aov.A2 <- anova(lm(A12[,2] ~ judge + variety, data=ice.cream)))
```

```
## Analysis of Variance Table
##
## Response: A12[, 2]
##           Df Sum Sq Mean Sq F value Pr(>F)
## judge      6 0.00000 0.000000  0.0000 1.0000
## variety    6 0.36735 0.061224  0.7742 0.6118
## Residuals  8 0.63265 0.079082

shapiro.test(residuals(lm(A12[,2] ~ judge + variety, data=ice.cream)))

##
## Shapiro-Wilk normality test
##
## data:  residuals(lm(A12[, 2] ~ judge + variety, data = ice.cream))
## W = 0.96396, p-value = 0.5992

ice.cream.F.A2 <- ice.cream.aov.A2["variety", "F value"]
```

Results match those in [Rayner \(2016\)](#). The p-values suggest a first order but no second order effects.

Note that this balanced incomplete block design can be considered as a block design with missing data. In such cases, the effects of treatments and blocks are not orthogonal. As the anova table provided by R is meant to be interpreted sequentially, to get the correct F statistic and p-value



Max's next Bookboon eBook
Your Boss: Sorted!
By Patrick Forsyth - 55 pages

Unlock your life.
Bookboon Premium is your key.

2000+ modern day bite-sized eBooks about soft skills and personal development. Written by the brightest minds in business.

bookboon.com

for the effect of the ice cream variety it needs to be the last factor added to the model. This issue is demonstrated below for the first order polynomial transformation. Compare the anova table output when the ice cream variety is the first term in the model (below) to that where it is the last term in the model (above). Alternatively, if we use the `drop1()` function we get a summary of the model where each row shows the effect of dropping that particular term from the complete model. This gives the same results irrespective of the sequence of terms in the model.

```
# sequence of terms in model important. This line
# gives invalid F statistic and p-value
anova(lm(A12[,1] ~ variety + judge, data=ice.cream))

## Analysis of Variance Table
##
## Response: A12[, 1]
##           Df Sum Sq Mean Sq F value Pr(>F)
## variety    6 0.66667 0.111111  6.2222 0.01074 *
## judge      6 0.19048 0.031746  1.7778 0.22100
## Residuals   8 0.14286 0.017857
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

drop1(lm(A12[,1] ~ judge + variety, data=ice.cream), test="F")

## Single term deletions
##
## Model:
## A12[, 1] ~ judge + variety
##           Df Sum of Sq    RSS    AIC F value    Pr(>F)
## <none>                0.14286 -78.799
## judge    6    0.19048 0.33333 -73.006  1.7778 0.221005
## variety  6    0.85714 1.00000 -49.935  8.0000 0.004904 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

drop1(lm(A12[,1] ~ variety + judge, data=ice.cream), test="F")

## Single term deletions
##
## Model:
## A12[, 1] ~ variety + judge
##           Df Sum of Sq    RSS    AIC F value    Pr(>F)
## <none>                0.14286 -78.799
## variety  6    0.85714 1.00000 -49.935  8.0000 0.004904 **
## judge    6    0.19048 0.33333 -73.006  1.7778 0.221005
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Averages of the polynomial transformed ranks are given below for each ice-cream variety. The

umbrella effect previously identified in [section 2.6](#) for this example is apparent in the first order averages. There is no apparent pattern in the second order averages.

```
# averages of the transformed ranks for each ice-cream variety.
(d <- aggregate(as.data.frame(ice.cream$A12), by=list(ice.cream$variety),
               FUN=mean))
```

##	Group.1	1	2
## 1	S	1.781742e-01	2.035409e-16
## 2	U	2.672612e-01	1.543033e-01
## 3	V	-1.781742e-01	0.000000e+00
## 4	W	-2.672612e-01	1.543033e-01
## 5	X	-8.908708e-02	-1.543033e-01
## 6	Y	1.328148e-18	0.000000e+00
## 7	Z	8.908708e-02	-1.543033e-01

NB There appears to be an error in [Rayner \(2016, Table 3.6\)](#) which should show similar information but on a different scale due to use of orthonormal polynomials.

3.3 Revisiting some previous examples

Tomato example

The following sections provide example code for all of the permutation tests summarised in [Rayner \(2016, Table 3.7\)](#).

Using the ANOVA F Test Statistic

Under the null hypothesis for the tomato example (a completely randomized design), there is no difference in the distribution of flavour scores for the four tomato varieties and so any of the flavour scores could just as easily have been associated with any of the varieties. In the code below, the function `F.perm()` creates a random permutation of the flavour scores and then calculates an F statistic based on this random ordering of scores and the original ordering of the tomato varieties. Repeating this process a large number of times (set by the value of `nperm`), we develop an approximation for the distribution of the F test statistic under the null hypothesis. We calculate the required p-value by comparing the F statistic for the observed ordering of the flavour scores with this simulated null distribution.

```
# permutation test based on ANOVA F test statistic
nperm <- 10000 # no of permutations
F.perm <- function(X) {
  # random permutation of the flavour scores
  flperm <- sample(x=tom2$flavour)
  # F statistic based on random permutation
  return(anova(lm(flperm~tom2$variety))["tom2$variety", "F value"])
}

Fp.dist <- sapply(1:nperm, FUN=F.perm) # apply the F.perm function nperm times
```

```
(tom.aov <- anova(lm(flavour~variety, data=tom2)))

## Analysis of Variance Table
##
## Response: flavour
##           Df Sum Sq Mean Sq F value Pr(>F)
## variety    3   2906   968.65   0.7986 0.4978
## Residuals  92 111592 1212.96

# test statistic for observed sample
observed.tom.F <- tom.aov["variety", "F value"]
# right tail permutation p-value
pval <- length(Fp.dist[Fp.dist >= observed.tom.F])/nperm
```

In this example, the p-value based on distribution theory is $P(F_{3,92} > 0.7985839) = 0.4977698$ and the permutation p-value based on 10^4 random permutations is 0.5013 (illustrated by the shading in the histogram of the approximate null distribution shown in Figure 3.2). Again remember that any permutation p-value like this is based on computer simulation of random sampling and so is likely to be a little different each time the code is run. Increasing the size of the simulation will reduce the variability in the reported p-value but will require extra computing time.

Kruskal-Wallis Test

The following code uses the same basic process to calculate a permutation p-value for the tomato example but based on the Kruskal-Wallis test statistic.

```
# permutation test based on Kruskal Wallis test statistic
nperm <- 10000 # no of permutations
KW.perm <- function(X) {
  # random permutation of the flavour scores
  flperm <- sample(x=tom2$flavour)
  # KW statistic based on random permutation
  return(kruskal.test(flperm~tom2$variety)$statistic)
}

# apply the KW.perm function nperm times
KWp.dist <- sapply(1:nperm, FUN=KW.perm)

(tom.KW <- kruskal.test(flavour~variety, data=tom2))

##
## Kruskal-Wallis rank sum test
##
## data: flavour by variety
## Kruskal-Wallis chi-squared = 1.978, df = 3, p-value = 0.577

observed.tom.KW <- tom.KW$statistic # test statistic for observed sample
```

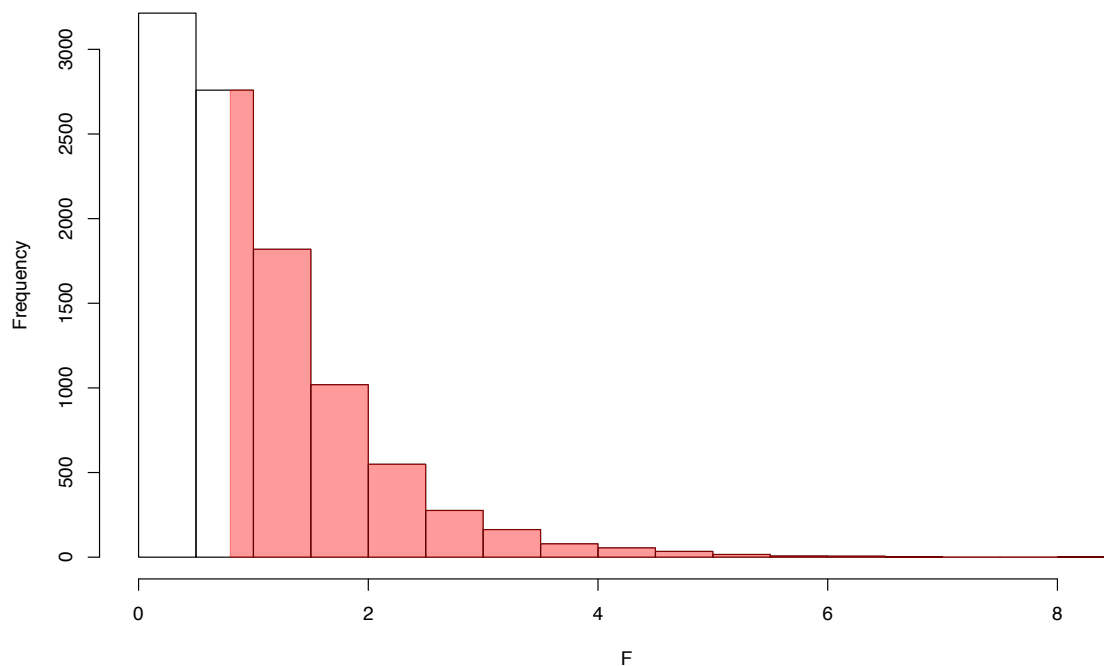


Figure 3.2: Approximate null distribution of ANOVA F test statistic based on random permutations.



MTHøjgaard

BEDRE LØSNINGER

I MT Højgaard insisterer vi på, at der findes en bedre løsning. Vi udvikler og anvender metoder og teknologier, der sætter nye standarder for bygge- og anlægsbranchen. Vi har fokus på hele tiden at videreudvikle vores medarbejdere, så vi gennem nye teknologier og nye samarbejdsformer kan transformere bygge- og anlægsbranchen. Vil du med på holdet?

mth.dk/vorestilgang




```
# right tail permutation p-value
pval <- length(KWp.dist[KWp.dist >= observed.tom.KW])/nperm
```

The p-value based on distribution theory is $P(\chi_3^2 > 1.9780252) = 0.5769802$ and the permutation p-value based on 10^4 random permutations is 0.5728.

Page and Umbrella Tests

The following code again uses the same basic process based on the Page and umbrella test statistics applied to the tomato example.

```
# permutation test for Page and Umbrella test statistics
ni <- xtabs(~variety, data=tom2) # no of observations for each variety
n <- sum(ni) # total no of observations
t <- nlevels(tom2$variety) # no of treatments, ie tomato varieties
tom2$r <- rank(tom2$flavour, ties.method = "average")

# function to sum ranks for different varieties
Ri <- function(r) aggregate(r~tom2$variety, FUN=sum)

Z <- function (R) {# calculate test statistics
  nR <- (R$r - (n+1)*ni/2)/sqrt(n*(n+1)*ni/12) # normalized rank sums
  return(nR[c(1,2,4,3)] %*% contr.poly(n=t)) # note use of the index to change
                                              # the ordering of varieties
}

Z.perm <- function(X) {
  rperm <- sample(tom2$r) # random permutation of the ranks
  R <- Ri(rperm) # sum ranks
  return(Z(R)) # Z statistics based on random permutation
}

nperm <- 10000
Zp.dist <- sapply(1:nperm, FUN=Z.perm) # apply the Z.perm function nperm times

observed.tom.Z <- Z(Ri(tom2$r))

# two tail permutation p-values
Zp.dist <- abs(Zp.dist)
observed.tom.Z <- abs(observed.tom.Z)
pval.L <- length(Zp.dist[1,Zp.dist[1,] >= observed.tom.Z[1]])/nperm
pval.Q <- length(Zp.dist[2,Zp.dist[2,] >= observed.tom.Z[2]])/nperm
pval.C <- length(Zp.dist[3,Zp.dist[3,] >= observed.tom.Z[3]])/nperm
```

As shown in Chapter 2, when using the theoretical normal distribution we double the one-sided p-values to get the required two-sided p-values $2P(Z > 0.3490022) = 0.7270877$ for the test of a linear trend and $2P(Z > 1.285999) = 0.1984434$ for the test of a quadratic trend.

A similar process is followed here but rather than just using one side of the simulated distribution we use the distribution of the absolute values of the test statistics for the permuted

samples. This gives us a more precise result as we are using all 10^4 permutations rather than just half of them. The permutation p-values are 0.7226 for testing a linear trend, 0.1984 for testing a quadratic trend and 0.6514 for testing a cubic trend.

First, Second and Third order NP ANOVA

Earlier in this chapter (p 64), the matrix `tom.A123` was created containing the values of the orthogonal polynomials up to order 3 for the tomato example. We now use these values to conduct permutation tests analogous to the F tests conducted previously. The rows of the matrix are randomly permuted and F statistics are calculated for each of a_1 , a_2 and a_3 . This process is repeated many times (controlled by the variable `nperm`) to simulate the distribution of the test statistics. The p-value is calculated by comparing the previously calculated test statistics based on the observed data to the simulated distributions.

```
# permutation tests for non-parametric ANOVA
n.obs <- nrow(tom.A123)
nperm <- 10000

F.perm <- function(X) {
  rperm <- sample(1:n.obs) # random permutation of the rows
  F.A1 <- anova(lm(tom.A123[rperm,1]~tom2$variety))["tom2$variety", "F value"]
  F.A2 <- anova(lm(tom.A123[rperm,2]~tom2$variety))["tom2$variety", "F value"]
  F.A3 <- anova(lm(tom.A123[rperm,3]~tom2$variety))["tom2$variety", "F value"]
  return(c(F.A1, F.A2, F.A3)) # F statistics based on random permutation
}

opF.dist <- sapply(1:nperm, FUN=F.perm) # apply the F.perm fun nperm times

pval.A1 <- length(opF.dist[1,opF.dist[1,] >= tom.F.A1])/nperm
pval.A2 <- length(opF.dist[2,opF.dist[2,] >= tom.F.A2])/nperm
pval.A3 <- length(opF.dist[3,opF.dist[3,] >= tom.F.A3])/nperm
```

Permutation test p-values are 0.5099, 0.2406 and 0.6798 for first, second and third order effects respectively suggesting no difference between tomato flavour varieties.

More Efficient Code

These code snippets have been written to illustrate the principles and not for maximum efficiency. Examples are still likely to only take a few seconds to run on a modern PC, but for applications with larger data sets or where many such tests need to be carried, the time spent writing more efficient code may be justified. For example, when fitting a large number of similar linear models it may be more efficient to use the lower level `lm.fit()` function rather than the wrapper function `lm`. For the Kruskal-Wallis statistic the above code creates permutations of the flavour scores which are operated on by the existing `kruskal.test` function. It would be more efficient to just permute the ranks to eliminate unnecessary ranking for each permutation. Other improvements are also possible.

Lemonade example

The following sections provide example code for all of the permutation tests summarised in [Rayner \(2016, Table 3.8\)](#).

ANOVA F Test

Under the null hypothesis for the lemonade example (a randomized block design), there is no difference in the lemonades and so any rank could just as easily have been associated with any of the five lemonades within each block (ie for each judge). In the code below, the function `F.perm` creates a random permutation of the ranks within each block and then calculates an F statistic based on these. Repeating this process a large number of times (set by the value of `nperm`), we develop an approximation for the distribution of the F test statistic under the null hypothesis. We calculate the required p-value by comparing the F statistic for the observed lemonade ranks with this distribution.

```
# permutation test based on ANOVA F test statistic
b <- nlevels(as.factor(sdrink2$judge)) # no of blocks, ie judges
t <- nlevels(sdrink2$type) # no of treatments, ie lemonades
nperm <- 10000 # no of permutations
F.perm <- function(X) {
  # random perm. of ranks within each block
  sdrink2$rp.rank <- c(t(replicate(b, sample(1:t))))
  # F statistic based on random permutation
  return(anova(lm(rp.rank~judge+type, data=sdrink2))["type", "F value"])
}

# apply the F.perm function nperm times
Fp.dist <- sapply(1:nperm, FUN=F.perm)

(sdrink.aov <- anova(lm(rank~judge+type, data=sdrink2)))

## Analysis of Variance Table
##
## Response: rank
##          Df Sum Sq Mean Sq F value    Pr(>F)
## judge      9    0.0    0.00  0.0000 1.00000
## type       4   22.6    5.65  2.6279 0.05037 .
## Residuals 36   77.4    2.15
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# test statistic for observed sample
observed.sdrink.F <- sdrink.aov["type", "F value"]
# right tail permutation p-value
pval <- length(Fp.dist[Fp.dist >= observed.sdrink.F])/nperm
```

In this example, the p-value based on distribution theory is $P(F_{4,36} > 2.627907) = 0.0503702$ and the permutation p-value based on 10^4 random permutations is 0.0575.

Friedman Test

The code below uses the same basic process to calculate a permutation p-value for the lemonade example based on the Friedman test statistic.

```
# permutation test based on Friedman test statistic
nperm <- 10000 # no of permutations

friedman.perm <- function(X) {
  # random perm. of ranks within each block
  sdrink2$rp.rank <- c(t(replicate(b, sample(1:t))))
  # test statistic based on random permutation
  return(friedman.test(rp.rank~type|judge, data=sdrink2)$statistic)
}

# apply the friedman.perm function nperm times to simulate distribution
frp.dist <- sapply(1:nperm, FUN=friedman.perm)

(sdrink.friedman <- friedman.test(rank~type|judge, data=sdrink2))

##
## Friedman rank sum test
##
## data: rank and type and judge
## Friedman chi-squared = 9.04, df = 4, p-value = 0.06011
```



Ses vi til DSE-Aalborg?

Kom forbi vores stand den
9. og 10. oktober 2019.

Vi giver en is og fortæller
om jobmulighederne hos
os.

banedanmark



```
# test statistic for observed sample
observed.sdrink.friedman <- sdrink.friedman$statistic

# right tail permutation p-value
pval <- length(frp.dist[frp.dist >= observed.sdrink.friedman])/nperm
```

The p-value based on distribution theory is $P(\chi_4^2 > 9.04) = 0.0601074$ and the permutation p-value based on 10^4 random permutations is 0.0575.

Page and Umbrella Tests

Previously in Chapter 2 (p 59), we calculated test statistics and p-values corresponding to tests of a linear trend ($Z_1 = -0.3162278$, with two-sided p-value $2P(Z > 0.3162278) = 0.7518296$), and quadratic trend ($Z_2 = 2.2984467$, with two-sided p-value $2P(Z > 2.2984467) = 0.0215364$).

Permutation p-values could be determined instead as shown below.

```
# permutation test for Page and Umbrella test statistics
ni <- xtabs(~type, data=sdrink2) # no of observations for each treatment
n <- sum(ni) # total no of observations
t <- nlevels(sdrink2$type) # no of treatments, ie lemonade varieties

Z.perm <- function(X) {
  # random perm. of ranks within each block
  sdrink2$rp.rank <- c(t(replicate(b, sample(1:t))))
  # function to sum ranks for diff treatments
  R <- aggregate(rp.rank~type, FUN=sum, data=sdrink2)
  nR <- R$rp.rank/sqrt(b*t*(t+1)/12) # normalized rank sums
  return(nR %*% contr.poly(n=t))
}

nperm <- 10000
Zp.dist <- sapply(1:nperm, FUN=Z.perm) # apply the Z.perm function nperm times

# two tail permutation p-values
Zp.dist <- abs(Zp.dist)
obs.Z <- abs(obs.Z)
pval.L <- length(Zp.dist[1,Zp.dist[1,] >= obs.Z[1]])/nperm
pval.Q <- length(Zp.dist[2,Zp.dist[2,] >= obs.Z[2]])/nperm
```

Again the basic process is the same. Here the permutation p-values are 0.7761 for the test of a linear trend and 0.0225 for the test of a quadratic trend.

First, Second and Third order NP ANOVA

Again we consider random permutation of ranks within each block and then calculate F statistics calculated based on the orthogonal polynomials up to order 3 for the permutation. This process is repeated many times (controlled by the variable `nperm`) to simulate the distribution of the test statistics. The p-value is calculated by comparing the previously calculated test statistics based on the observed data to the simulated distributions.

```
# permutation tests for non-parametric ANOVA
nperm <- 10000

F.perm <- function(X) {
  # random perm. of ranks within each block
  sdrink2$rp.rank <- c(t(replicate(b, sample(1:t))))
  A123 <- poly(sdrink2$rp.rank, degree=3)

  F.A1 <- anova(lm(A123[,1]~sdrink2$type+sdrink2$judge))["sdrink2$type",
                                                         "F value"]
  F.A2 <- anova(lm(A123[,2]~sdrink2$type+sdrink2$judge))["sdrink2$type",
                                                         "F value"]
  F.A3 <- anova(lm(A123[,3]~sdrink2$type+sdrink2$judge))["sdrink2$type",
                                                         "F value"]

  return(c(F.A1, F.A2, F.A3)) # F statistics based on random permutation
}

opF.dist <- sapply(1:nperm, FUN=F.perm) # apply F.perm function nperm times

pval.A1 <- length(opF.dist[1,opF.dist[1,] >= sdrink.F.A1])/nperm
pval.A2 <- length(opF.dist[2,opF.dist[2,] >= sdrink.F.A2])/nperm
pval.A3 <- length(opF.dist[3,opF.dist[3,] >= sdrink.F.A3])/nperm
```

Permutation test p-values are 0.0487, 0.9185 and 0.0067 for first, second and third order effects respectively.

Ice Cream example

The following sections provide example code for all of the permutation tests summarised in [Rayner \(2016, Table 3.9\)](#).

ANOVA F Test

Under the null hypothesis for this balanced incomplete block design there is no difference in the $t = 7$ ice-creams and so any rank could just as easily have been associated with any of the $k = 3$ ice-creams presented within each of the $b = 7$ blocks (ie to each judge). In the code below, the function `F.perm` creates a random permutation of the ranks within each block and then calculates an F statistic based on these. Repeating this process a large number of times (set by the value of `nperm`), we develop an approximation for the distribution of the F test statistic under the null hypothesis. We calculate the required p-value by comparing the F statistic for the observed ice-cream ranks with this distribution.

```
nperm <- 10000 # no of permutations
F.perm <- function(X) {
  # random perm. of ranks within each block
  ice.cream$rp.rank <- c(replicate(b, sample(1:k)))
  # F statistic based on random permutation
```



```

return(anova(lm(rp.rank~judge+variety, data=ice.cream))["variety",
                                                         "F value"])
}

Fp.dist <- sapply(1:nperm, FUN=F.perm) # apply the F.perm function nperm times

(ice.cream.aov <- anova(lm(rank~judge+variety, data=ice.cream)))

## Analysis of Variance Table
##
## Response: rank
##          Df Sum Sq Mean Sq F value    Pr(>F)
## judge      6      0    0.00      0 1.000000
## variety     6     12    2.00      8 0.004904 **
## Residuals   8      2    0.25
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

# test statistic for observed sample
observed.ice.cream.F <- ice.cream.aov["variety", "F value"]

# right tail permutation p-value
pval <- length(Fp.dist[Fp.dist >= observed.ice.cream.F])/nperm

```



CISO Conference
Produced by **Inspired**

**Apollo Hotel 1, Groenlandsekade
Vinkeveen, Amsterdam, NL
Dec 5th 2019**

**Listen, learn & build relationships with our
Network of CISOs & Cyber Security Leaders**

Inspired

The p-value based on distribution theory for this example is $P(F_{6,8} > 8) = 0.0049044$, and the permutation p-value, based on 10^4 random permutations, is 0.0032.

Durbin Test

```
nperm <- 10000 # no of permutations

durbin.perm <- function(X) {
  # random perm. of ranks within each block
  ice.cream$rp.rank <- c(replicate(b, sample(1:k)))
  # sum the permuted ranks for each of the 7 ice creams over all 7 judges
  Ri <- aggregate(rp.rank~variety, FUN=sum, data=ice.cream)
  # test statistic based on random permutation
  return(12*(t-1)/b/k/(k^2-1)*sum(Ri$rp.rank^2) - 3*r*(t-1)*(k+1)/(k-1))
}

# apply the durbin.perm function nperm times to simulate distribution
dp.dist <- sapply(1:nperm, FUN=durbin.perm)

obs.Ri <- aggregate(rank~variety, FUN=sum, data=ice.cream)
obs.D <- 12*(t-1)/b/k/(k^2-1)*sum(obs.Ri$rank^2) - 3*r*(t-1)*(k+1)/(k-1)

pval <- length(dp.dist[dp.dist >= obs.D])/nperm # right tail p-value
```

The code above uses the same basic process to calculate a permutation p-value for the ice-cream example based on the Durbin test statistic. The p-value based on distribution theory is $P(\chi_6^2 > 12) = 0.0619688$ and the permutation p-value based on 10^4 random permutations is 0.0161.

Page and Umbrella Tests

Previously in Chapter 2 (p60), we calculated test statistics and p-values corresponding to tests of a linear trend ($Z_1 = -0.9897433$, with two-sided p-value $2P(Z > 0.9897433) = 0.3222996$), and quadratic trend ($Z_2 = 2.5714286$, with two-sided p-value $2P(Z > 2.5714286) = 0.010128$).

Using the same basic process as before, permutation p-values could be determined instead.

```
Z.perm <- function(X) {
  # random perm. of ranks within each block
  ice.cream$rp.rank <- c(replicate(b, sample(1:k)))
  R <- aggregate(rp.rank~variety, FUN=sum, data=ice.cream)
  return(R$rp.rank %*% contr.poly(n=t) / sqrt(b*k*(k^2-1)/12/(t-1)))
}

nperm <- 10000
Zp.dist <- sapply(1:nperm, FUN=Z.perm) # apply the Z.perm function nperm times

# two tail permutation p-values
Zp.dist <- abs(Zp.dist)
```

```
pval.L <- length(Zp.dist[1,Zp.dist[1,] >= abs(obs.Z[1]))/nperm
pval.Q <- length(Zp.dist[2,Zp.dist[2,] >= abs(obs.Z[2]))/nperm
```

Here the permutation p-values are 0.3205 for the test of a linear trend and 0.0064 for the test of a quadratic trend.

First and Second Order NP ANOVA

Again we consider random permutation of ranks within each block and then calculate F statistics calculated based on the orthogonal polynomials up to order 2 for the permutation. This process is repeated many times (controlled by the variable `nperm`) to simulate the distribution of the test statistics. The p-value is calculated by comparing the previously calculated test statistics based on the observed data to the simulated distributions.

```
nperm <- 10000

F.perm <- function(X) {
  # random perm. of ranks within each block
  ice.cream$rp.rank <- c(replicate(b, sample(1:k)))
  A123 <- poly(ice.cream$rp.rank, degree=2)

  F.A1 <- anova(lm(A123[,1]~ice.cream$judge+ice.cream$variety))["ice.cream$variety", "F value"]
  F.A2 <- anova(lm(A123[,2]~ice.cream$judge+ice.cream$variety))["ice.cream$variety", "F value"]

  return(c(F.A1, F.A2)) # F statistics based on random permutation
}

opF.dist <- sapply(1:nperm, FUN=F.perm) # apply F.perm function nperm times

pval.A1 <- length(opF.dist[1,opF.dist[1,] >= ice.cream.F.A1])/nperm
pval.A2 <- length(opF.dist[2,opF.dist[2,] >= ice.cream.F.A2])/nperm
```

Permutation test p-values are 0.009 and 0.6272 for first and second order effects respectively.

Bibliography

Niel J le Roux and Sugnet Lubbet. *A Step-by-Step R Tutorial: An introduction into R applications and programming*. Bookboon, 2015.

J.C.W. Rayner. *Introductory Nonparameterics*. BookBoon, 2nd edition, 2016.

The R Core Team. *R Data Import/Export*. CRAN, 2017.

W. N. Venables, D. M. Smith, and The R Core Team. *An Introduction to R Notes on R: A Programming Environment for Data Analysis and Graphics*. CRAN, version 3.3.0 (2016-05-03) edition, 2017.