R Textbook Companion for Operations Research: An Introduction by Hamdy A Taha[1]

Created by
Georgey John
M.Tech.
Others
NIT, Calicut
Cross-Checked by
R TBC Team

May 29, 2020

[1]Funded by a grant from the National Mission on Education through ICT - http://spoken-tutorial.org/NMEICT-Intro. This Textbook Companion and R codes written in it can be downloaded from the "Textbook Companion Project" section at the website - https://r.fossee.in.

Book Description

Title: Operations Research: An Introduction

Author: Hamdy A Taha

Publisher: Pearson

Edition: 9

Year: 2014

ISBN: 9780132555937

R numbering policy used in this document and the relation to the above book.

Exa Example (Solved example)

Eqn Equation (Particular equation of the above book)

For example, Exa 3.51 means solved example 3.51 of this book. Sec 2.3 means an R code whose theory is explained in Section 2.3 of the book.

Contents

List of R Codes

Chapter 2

Modelling with Linear Programming

R code Exa 2.2.1 Graphical Solution to LP Problems

```
##Chapter 2 : Modelling with Linear Programming
##Example 2-1 : Page 16

#To plot the line, we have to consider them as equation instead of inequality and express/
#them in terms of x2 :

#Constraint 1 : 6 * x1 + 4 * x2 <= 24
#Con1         : x2 = (24 - 6 * x1)/4
con1 <- function(x1) (24 - 6 * x1)/4
plot (con1, xlab = "x1", ylab = "x2", xlim = c(0,7), ylim = c(0,7), col = "red",
      main = "Example 2-1", yaxs= "i", xaxs = "i")
#xlab & ylab  : x and y label respectively
#xlim         : limits of x value on the plot
#ylim         : limits of y value on the plot
#col          : color of the line
#main         : Title of the plot
#yaxs & xaxs  : the style of axis interval
```

```
calculation to be used by R. The default
#value is a 4% gap at each end of axis

#Constraint 2 : x1 + 2 * x2 <=6
#Con2 : x2 = (6 - x1)/2
con2 <- function(x1) (6 - x1)/2
plot (con2, add=T, xlim = c(0,7), ylim = c(0,7), col = "blue")
#add            : adds to an existing plot


#Constraint 3 : -x1 + x2 <= 1
#Con3           : x2 = (1 + x1)
con3 <- function(x1) (1 + x1)
plot (con3, add=T, xlim = c(0,7), ylim = c(0,7), col = "green")

#Constraint 4 : x2 <= 2
#Con4           : x2 = 2
con4 <- function(x1) (2 + 0*x1)
plot (con4, add=T, xlim = c(0,7), ylim = c(0,7), col = "green")
#h              : horizontal line at y=2

#Points of intersections of constraints : (0,1),(1,2),(2,2),(2,1.5),(4,0)
points(c(0,1,2,3,4),c(1,2,2,1.5,0))

#Add a shaded area
polygon(c(0,1,2,3,4,0),c(1,2,2,1.5,0,0), col = rgb(0.48, 0.46, 0.46, 0.5),
        border=NA)
#border         : option to add border to the shaded area

#Adding "solution space" text to the shaded area at (2,1)
text(2,1,"Solution \nSpace")
```

```

##Objective function :Max 5 *x1 + 4 * x2
# maximum objective is 21, Therefore 5 *x1 + 4 * x2 =21
# Obj : x2 = (21 - 5*x1)/4
Obj <- function(x1) (21 - 5*x1)/4
plot (Obj, add=T, xlim = c(0,7), ylim = c(0,7), lty =2 )
#lty : option to set the type of line ,2 for dashed line

text(3,3,"x1 = 3 \nx2 = 1.5 \n z=21")
```

R code Exa 2.2.2 Graphical solution of LP for Diet Problem

```
##Chapter 2 : Modelling with Linear Programming
##Example 2-2 : Page 24

#To plot the line , we have to consider them as equation instead of inequality and express/
#them in terms of x2 :

#Constraint 1 : x1 + x2 <= 800
#Con1 : x2 = (800 - x1)
con1 <- function(x1) (800 - x1)
plot (con1, xlab = "x1", ylab = "x2", xlim = c(0,1500), ylim = c(0,1500), col = "red",
      main = "Example 2-2", yaxs= "i", xaxs = "i")
#xlab & ylab : x and y label respectively
#xlim : limits of x value on the plot
#ylim : limits of y value on the plot
#col : color of the line
#main : Title of the plot
#yaxs & xaxs : the style of axis interval calculation to be used by R. The default
```

```
#value is a 4% gap at each end of axis

#Constraint 2 : 0.21 * x1 - 0.3 * x2 <= 0
#Con2 : x2 = (0.21 * x1)/0.3
con2 <- function(x1) (0.21 * x1)/0.3
plot (con2, add=T, xlim = c(0,1500), ylim = c(0,1500), col = "blue")
#add          : adds to an existing plot


#Constraint 3 : 0.03 * x1 - 0.01 * x2 >= 0
#Con3 : x2 = (0.03 * x1)/0.01
con3 <- function(x1) (0.03 * x1)/0.01
plot (con3, add=T, xlim = c(0,1500), ylim = c(0,1500), col = "green")

#Points of intersections of constraints : (0,1),(1,2),(2,2),(2,1.5),(4,0)
points(c(470.6,200),c(329.4,600))

#Add a shaded area
polygon(c(470.6,200,500,15000/7),c(329.4,600,1500,1500), col = rgb(0.48, 0.46, 0.46, 0.5),
        border=NA)
#border : option to add border to the shaded area

#Adding "solution space" text to the shaded area at (750,1000)
text(750,1000,"Solution \nSpace")

##Objective function  :Min 0.3 *x1 + 0.9 * x2
# Minimum objective is 437.64, Therefore 0.3 *x1 + 0.9 * x2 = 437.64
# Obj  : x2 = (437.64 - 0.3 * x1)/0.9
Obj <- function(x1) (437.64 - 0.3 * x1)/0.9
plot (Obj, add=T, xlim = c(0,1500), ylim = c(0,1500), lty =2 )
```

```
#lty    : option to set the type of line ,2 for dashed line

text(1000,250,"x1 = 470.6 \nx2 = 329.4 \n z=437.64")
```

R code Exa 2.4.1 LP solution for Bank Loan Problem

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-1 : Page 35

# Objective function :Max (0.126 * x1 +0.1209 * x2 + 0.1164 * x3 + 0.11875 * x4 + 0.098 * x5)
#                - (0.1 * x1 + 0.07 * x2 + 0.03 * x3 + 0.05 * x4 + 0.02 * x5)
a <- c(0.126-0.1,0.1209-0.07,0.1164-0.03,0.11875-0.05,0.098-0.02)

# Constraint 1 : x1 + x2 + x3 + x4 + x5 <=12
C1 <- c(1,1,1,1,1)
bc1<- 12

# Constraint 2 : 0.4*x1 + 0.4*x2 + 0.4*x3 - 0.6*x4 - 0.6*x5 <=0
C2 <- c(0.4,0.4,0.4,-0.6,-0.6)
bc2<-0

# Constraint 3 : 0.5 * x1 + 0.5 * x2 - 0.5 * x3 <=0
C3 <- c(0.5,0.5,-0.5,0,0)
bc3<-0

# Constraint 4 : 0.06 * x1 + 0.03 * x2 - 0.01 * x3 +0.01 * x4 -0.02 * x5 <=0
C4 <- c(0.06,0.03,-0.01,0.01,-0.02)
bc4<-0
```

```
library("boot")
##Simplex is a function which uses dual simplex method
simplex(a , A1 = rbind(C1, C2, C3, C4), b1 = c(bc1, bc2, bc3, bc4), maxi = TRUE)
```

R code Exa 2.4.2 LP solution for Single period Production Model

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-2 : Page 41

# Objective function :Max (30 * x1 + 40 * x2 + 20 * x3 + 10 * x4)
#                       - (15 * s1 + 20 * s2 + 10 * s3 + 8 * s4)
a <- c(30,40,20,10,-15,-20,-10,-8)

# Constraint 1 : 0.3 * x1 + 0.3 * x2 + 0.25 * x3 + 0.15 * x4  <= 1000
C1 <- c(0.3,0.3,0.25,0.15,0,0,0,0)
bc1<- 1000

# Constraint 2 : 0.25 * x1 + 0.35 * x2 + 0.3 * x3 + 0.1 * x4 <= 1000
C2 <- c(0.25,0.35,0.3,0.1,0,0,0,0)
bc2<-1000

# Constraint 3 : 0.45 * x1 + 0.5 * x2 + 0.4 * x3 + 0.22 * x4 <=1000
C3 <- c(0.45,0.5,0.4,0.22,0,0,0,0)
bc3<-1000

# Constraint 4 : 0.15 * x1 + 0.15 * x2 + 0.1 * x3 + 0.05 * x4 <=0
```

```
C4 <- c(0.15,0.15,0.1,0.05,0,0,0,0)
bc4<-1000

# Constraint 5 : x1 + s1 = 800
C5 <- c(1,0,0,0,1,0,0,0)
bc5<-800

# Constraint 6 : x2 + s2 = 750
C6 <- c(0,1,0,0,0,1,0,0)
bc6<-750

# Constraint 7: x3 + s3 = 600
C7 <- c(0,0,1,0,0,0,1,0)
bc7<-600

# Constraint 8 : x4 + s4 = 500
C8 <- c(0,0,0,1,0,0,0,1)
bc8<-500

library("boot")

simplex(a , A1 = rbind(C1, C2, C3, C4), b1 = c(bc1,
   bc2, bc3, bc4),
        A2=NULL, b2=NULL, A3 = rbind(C5,C6,C7,C8),
           b3 = c(bc5,bc6,bc7,bc8), maxi = TRUE)
```

R code Exa 2.4.3 LP solution for Multi period Production Inventory Model

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-3 : Page 41

# Objective function :Min (50 * x1 + 45 * x2 + 55 *
   x3 + 48 * x4 + 52 * x5 + 50 * x6)
#                      + 8(I1 +I2 +I3 +I4 + I5 + I6)
a <- c(50,45,55,48,52,50,8,8,8,8,8,8)
```

```

# Constraint 1 : x1 -I1 = 100
C1 <- c(1,0,0,0,0,0,-1,0,0,0,0,0)
bc1<- 100

# Constraint 2 : I1 + x2 - I2 = 250
C2 <- c(0,1,0,0,0,0,1,-1,0,0,0,0)
bc2<-250

# Constraint 3 : I2 + x3 - I3 = 190
C3 <- c(0,0,1,0,0,0,0,1,-1,0,0,0)
bc3<-190

# Constraint 4 : I3 + x4 - I4 = 140
C4 <- c(0,0,0,1,0,0,0,0,1,-1,0,0)
bc4<-140

# Constraint 5 : I4 + x5 - I5 =220
C5 <- c(0,0,0,0,1,0,0,0,0,1,-1,0)
bc5<-220

# Constraint 6 : I5 + x6 = 110
C6 <- c(0,0,0,0,0,1,0,0,0,0,1,0)
bc6<-110

library("boot")

simplex(a , A1=NULL,b1=NULL, A2=NULL, b2=NULL, A3 = rbind(C1, C2, C3, C4,C5,C6),
        b3 = c(bc1, bc2, bc3, bc4,bc5,bc6), maxi = F)
```

R code Exa 2.4.4 LP solution for Multi period Production smoothing Model

```
##Chapter 2 : Modelling with Linear Programming
```

```
##Example 4-4 : Page 43

# Objective function :Min 50(I1 + I2 + I3 + I4) + 200(Sm1 +Sm2 +Sm3 +Sm4) + 400(Sp1 +Sp2 +Sp3 +Sp4)
a <- c
    (0,0,0,0,50,50,50,50,200,200,200,200,400,400,400,400)

# Constraint 1 : 10*x1 -I1 = 400
C1 <- c(10,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0)
bc1<- 400

# Constraint 2 : I1 + 10 * x2 - I2 = 600
C2 <- c(0,10,0,0,1,-1,0,0,0,0,0,0,0,0,0,0)
bc2<-600

# Constraint 3 : I2 + 10 * x3 - I3 = 400
C3 <- c(0,0,10,0,0,0,1,-1,0,0,0,0,0,0,0,0)
bc3<-400

# Constraint 4 : I3 + 10 * x4 = 500
C4 <- c(0,0,0,10,0,0,0,1,0,0,0,0,0,0,0,0)
bc4<-500

# Constraint 5 : x1 -Sm1 + Sp1 = 0
C5 <- c(1,0,0,0,0,0,0,0,-1,0,0,0,1,0,0,0)
bc5<-0

# Constraint 6 : x2 -x1 -Sm2 +Sp2 = 0
C6 <- c(-1,1,0,0,0,0,0,0,0,-1,0,0,0,1,0,0)
bc6<-0

# Constraint 7 : x3 -x2 -Sm3 +Sp3 = 0
C7 <- c(0,-1,1,0,0,0,0,0,0,0,-1,0,0,0,1,0)
bc7<-0

# Constraint 8 : x4 -x3 -Sm4 +Sp4 = 0
C8 <- c(0,0,-1,1,0,0,0,0,0,0,0,-1,0,0,0,1)
```

```
bc8<-0

library("boot")

simplex(a , A1=NULL,b1=NULL, A2=NULL, b2=NULL, A3 =
   rbind(C1, C2, C3, C4, C5, C6, C7, C8),
        b3 = c(bc1, bc2, bc3, bc4, bc5, bc6, bc7,
           bc8), maxi = F)
```

R code Exa 2.4.5 LP solution for Bus Scheduling Problem

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-5 : Page 50

# Objective function :Min x1 + x2 + x3 + x4 + x5 +
   x6
a <- c(1,1,1,1,1,1)

# Constraint 1 : x1 + x6 >= 4
C1 <- c(1,0,0,0,0,1)
bc1<- 4

# Constraint 2 : x1 + x2 >= 8
C2 <- c(1,1,0,0,0,0)
bc2<-8

# Constraint 3 : x2 + x3 >= 10
C3 <- c(0,1,1,0,0,0)
bc3<-10

# Constraint 4 : x3 + x4 >= 7
C4 <- c(0,0,1,1,0,0)
bc4<-7

# Constraint 5 : x4 + x5 >= 12
```

```
C5 <- c(0,0,0,1,1,0)
bc5<-12

# Constraint 6 : x5 + x6 >= 4
C6 <- c(0,0,0,0,1,1)
bc6<-4

library("lpSolve")

solution=lp("min",a, rbind(C1, C2, C3, C4, C5, C6),
    rep(">=",6),c(bc1, bc2, bc3, bc4, bc5, bc6))

solution
solution$solution
```

R code Exa 2.4.6 LP solution for Urban Renewal Model Problem

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-6 : Page 52

# Objective function :Max 1000 * x1 + 1900 * x2 +
    2700 * x3 + 3400 * x4
a <- c(1000, 1900, 2700, 3400, 0)

# Constraint 1 : 0.18 * x1 + 0.28 * x2 + 0.4 * x3 +
    0.5 * x4 -0.2125 * x5 <= 0
C1 <- c(0.18, 0.28, 0.4, 0.5, -0.2125)
bc1<- 0

# Constraint 2 : x5 <= 300
C2 <- c(0,0,0,0,1)
bc2<-300

# Constraint 3 : -0.8 * x1 + 0.2 * x2 + 0.2 * x3 +
    0.2 * x4 <= 0
```

```
C3 <- c(-0.8, 0.2, 0.2, 0.2, 0)
bc3<-0

# Constraint 4 : 0.1 * x1 - 0.9 * x2 + 0.1 * x3 +
    0.1 * x4 <= 0
C4 <- c(0.1, -0.9, 0.1, 0.1, 0)
bc4<-7

# Constraint 5 : 0.25 * x1 + 0.25 * x2 - 0.75 * x3 -
     0.75 * x4  <= 0
C5 <- c(0.25,0.25,-0.75,-0.75,0)
bc5<-0

# Constraint 6 : 50 * x1 + 70 * x2 + 130 * x3 + 160
    * x4 + 2* x5 <= 15000
C6 <- c(50, 70, 130, 160, 2)
bc6<-15000

#to install the lpSolve package,run the following
    command
#install.packages("lpSolve")
library("lpSolve")

solution = lp("max", a, rbind(C1, C2, C3, C4, C5, C6
    ), rep("<=",6),
                c(bc1, bc2, bc3, bc4, bc5, bc6))
solution
solution$solution
```

R code Exa 2.4.7 LP solution for Crude Oil Refining and Gasoline Blending Problem

```
##Chapter 2 : Modelling with Linear Programming
##Example 4-7 : Page 57

```

```
# Objective function :Max 6.7 * (x11 +x21) + 7.2 * (x12 + x22) + 8.1 * (x13 + x23)
a <- c(6.7, 7.2, 8.1, 6.7, 7.2, 8.1)

# Constraint 1 : 5 *(x11 + x12 + x13) + 10 * (x21 + x22 + x23) <= 1500000
C1 <- c(5, 5, 5, 10, 10, 10)
bc1<- 1500000

# Constraint 2 : 2 * (x21 + x22 + x23) <=200000
C2 <- c(0, 0, 0, 2, 2, 2)
bc2<-200000

# Constraint 3 : (x11 + x21) <= 50000
C3 <- c(1,0,0,1,0,0)
bc3<-50000

# Constraint 4 : (x12 + x22) <= 30000
C4 <- c(0, 1, 0, 0, 1, 0)
bc4<-30000

# Constraint 5 : (x13 + x23) <= 40000
C5 <- c(0, 0, 1, 0, 0, 1)
bc5<-40000

# Constraint 6 : 5 * x11 - 11 * x21 <= 0
C6 <- c(5,0,0,-11,0,0)
bc6<-0

# Constraint 7 : 7* x12 - 9 * x22 <= 0
C7 <- c(0, 7, 0, 0, -9, 0)
bc7<-0
# Constraint 8 : 10 * x12 - 6 * x22 <= 0
C8 <- c(0, 0, 10, 0, 0, -6)
bc8<-0

#to install the lpSolve package,run the following command
```

```r
#install.packages("lpSolve")
library("lpSolve")

solution = lp("max", a, rbind(C1, C2, C3, C4, C5, C6
    , C7, C8), rep("<=",8),
                c(bc1, bc2, bc3, bc4, bc5, bc6, bc7,
                  bc8))
solution
solution$solution
```

Chapter 3

The Simplex Method and Sensitivity Analysis

R code Exa 3.2.1 Graphical solution of LP with 2 variables

```
##Chapter 3 : The Simplex Method and Sensitivity
   Analysis
##Example 2-1 : Page 83

#To plot the line, we have to consider them as
   equation instead of inequality and express/
#them in terms of x2 :

#Constraint 1 : 2 * x1 + x2 <= 4
#Con1         : x2 = (4 - 2 * x1)
con1 <- function(x1) (4 - 2 * x1)
plot (con1, xlab = "x1", ylab = "x2", xlim = c(0,6),
    ylim = c(0,5), col = "red",
      main = "Example 2-1", yaxs= "i", xaxs = "i")
#xlab & ylab  : x and y label respectively
#xlim         : limits of x value on the plot
#ylim         : limits of y value on the plot
#col          : color of the line
#main         : Title of the plot
```

```
#yaxs & xaxs  : the style of axis interval calculation to be used by R. The default
#value is a 4% gap at each end of axis

#Constraint 2 : x1 + 2 * x2 <= 5
#Con2 : x2 = (5 − x1)/2
con2 <- function(x1) (5 - x1)/2
plot (con2, add=T, xlim = c(0,6), ylim = c(0,5), col = "blue")
#add            : adds to an existing plot

#Points of intersections of constraints and the axis : (0,0),(0,2.5),(1,2),(2,0),(5,0),(0,4)
points(c(0,0,1,2,5,0),c(0,2.5,2,0,0,4))

#Inserting texts to name the points
text(c(0,0,1,2,5,0)+0.1,c(0,2.5,2,0,0,4)+0.2,LETTERS[1:6],cex = 0.7)
#First two arguments are x(+0.1) and y(+0.2) coordinates
#LETTERS        : a stored array variable of all alphabets
#cex            : ratio of modification to font size

#Add a shaded area
polygon(c(0,0,1,2),c(0,2.5,2,0), col = rgb(0.48, 0.46, 0.46, 0.5),
        border=NA)
#border         : option to add border to the shaded area

#Adding "solution space" text to the shaded area at (0.7,1)
text(0.7,1,"Solution \nSpace",cex = 0.9)

##Objective function  :Max 2 *x1 + 3 * x2
# maximum objective is 8, Therefore 2 *x1 + 3 * x2 = 8
```

```
# Obj          : x2 = (8 - 2*x1)/3
Obj <- function(x1) (8 - 2*x1)/3
plot (Obj, add=T, xlim = c(0,6), ylim = c(0,5), lty
   =2 )
#lty          : option to set the type of line ,2 for
     dashed line

#Adding text at (3,3)
text(3,3,"x1 = 1 \nx2 = 2 \n z = 8")
```

R code Exa 3.3.1 LP solution for Reddy Mikks Problem using lpSolve

```
##Chapter 3 : The Simplex Method and Sensitivity
   Analysis
##Example 3-1 : Page 89

# Objective function :Max 5*x1 + 4*x2 + 0*s1 + 0*s2
   + 0*s3 + 0*s4
a <- c(5,4,0,0,0,0)

# Constraint 1 : 6*x1 + 4*x2 +s1 = 24
C1 <- c(6,4,1,0,0,0)
bc1<- 24

# Constraint 2 : x1 + 2*x2 + s2 = 6
C2 <- c(1,2,0,1,0,0)
bc2<-6

# Constraint 3 : - x1 + x2 + s3 = 1
C3 <- c(-1,1,0,0,1,0)
bc3<-1

# Constraint 4 : 1*x2 + s4  = 2
C4 <- c(0,1,0,0,0,1)
bc4<-2
```

```
library("lpSolve")

solution=lp("max",a, rbind(C1, C2, C3, C4), rep("=",4),c(bc1, bc2, bc3, bc4))

solution
solution$solution
```

R code Exa 3.4.1 LP solution using M Method

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 4-1 : Page 100

# Objective function :Min 4 * x1 + x2 + M * R1 + M * R2
BigM <- 1000
a <- c(4,1,0,0,BigM,BigM)

# Constraint 1 : 3*x1 + x2 + R1 = 3
C1 <- c(3,1,0,0,1,0)
bc1<- 3

# Constraint 2 : 4*x1 + 3*x2 -x3 + R2 >= 6
C2 <- c(4,3,-1,0,0,1)
bc2<-6

# Constraint 3 : x1 + 2*x2 +x4 <=4
C3 <- c(1,2,0,1,0,0)
bc3<-4

library("boot")

simplex(a,NULL,NULL,NULL,NULL, rbind(C1,C2,C3),c(bc1
```

```
,bc2,bc3) ,maxi = F)
```

R code Exa 3.4.2 LP solution using Two Phase Method

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 4-2 : Page 105

# Objective function :Min 4 * x1 + x2
a <- c(4,1)

# Constraint 1 : 3*x1 + x2 = 3
C1 <- c(3,1)
bc1<- 3

# Constraint 2 : 4*x1 + 3*x2 >= 6
C2 <- c(4,3)
bc2<-6

# Constraint 3 : x1 + 2*x2 <=4
C3 <- c(1,2)
bc3<-4

library("boot")

simplex(a,C3,bc3,C2,bc2, C1,bc1 ,maxi = F)

#The method employed by this function is the two phase tableau simplex method.
#If there are >= or equality constraints an initial feasible solution is not
#easy to find. To find a feasible solution an artificial variable is introduced
#into each >= or equality constraint and an auxiliary objective function is defined
```

```
#as the sum of these artificial variables. If a feasible solution to the set of
#constraints exists then the auxiliary objective will be minimized when all of the
#artificial variables are 0. These are then discarded and the original problem solved
#starting at the solution to the auxiliary problem. If the only constraints are of the
#<= form, the origin is a feasible solution and so the first stage can be omitted.
##Refrence : https://stat.ethz.ch/R-manual/R-devel/library/boot/html/simplex.html
```

R code Exa 3.5.1 Degenerate Optimal Solution

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 5-1 : Page 108

# Objective function :Max 3 * x1 + 9 * x2
a <- c(3,9)

# Constraint 1 : x1 + 4 * x2 <= 8
C1 <- c(1,4)
bc1<- 8

# Constraint 2 : x1 + 2*x2 <= 4
C2 <- c(1,2)
bc2<-4


library("boot")

solution=simplex(a,rbind(C1,C2),c(bc1,bc2) ,maxi = T)
```

```
##The simplex function arbitrarily breaks the tie in minimum ratio for the leaving variable.
```

R code Exa 3.5.2 Infinite Solutions to an LP

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 5-2 : Page 112

# Objective function :Max 2 * x1 + 4 * x2
a <- c(2,4)

# Constraint 1 : 1 * x1 + 2 * x2 <= 5
C1 <- c(1,2)
bc1<- 5

# Constraint 2 : x1 + x2 <= 4
C2 <- c(1,1)
bc2<-4

library("boot")

simplex(a,rbind(C1,C2),c(bc1,bc2) ,maxi = T)
#The simplex function as well as lpsolve stops as soon as a feasible optima is obtained and doesn't evaluate
#alternate optima
```

R code Exa 3.5.3 Unbounded objective value

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 5-3 : Page 115
```

```

# Objective function :Max 2 * x1 +  x2
a <- c(2,1)

# Constraint 1 :  x1 -  x2 <= 10
C1 <- c(1,-1)
bc1<- 10

# Constraint 2 : 2 * x1 <= 40
C2 <- c(2,0)
bc2<-40

library("lpSolve")

solution = lp("max", a, rbind(C1, C2), rep("<=",2),
    c(bc1, bc2))
#Error: status 3 implies that the model is unbounded
```

R code Exa 3.5.4 Infeasible Solution Space

```
##Chapter 3 : The Simplex Method and Sensitivity
    Analysis
##Example 5-4 : Page 116

# Objective function :Max 3 * x1 +  2 * x2
a <- c(3,2)

# Constraint 1 :  2 * x1 +  x2 <= 2
C1 <- c(2,1)
bc1<- 2

# Constraint 2 : 3 * x1 + 4 * x2 >= 12
C2 <- c(3,4)
bc2<-12

```

```
library(”lpSolve”)

lp(”max”, a, rbind(C1, C2), c(”<=”,”>=”), c(bc1, bc2))
#Error: no feasible solution found
```

R code Exa 3.6.1 Changing RHS for Sensitivity analysis

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 6-1 : Page 118

# Objective function :Max 30 * x1 +  20 * x2
a <- c(30,20)

# Constraint 1 :  2 * x1 +  x2 <= 8
C1 <- c(2,1)
bc1<- 8

# Constraint 2 : x1 + 3 * x2 <= 8
C2 <- c(1,3)
bc2<-8

library(”lpSolve”)

solution=lp(”max”, a, rbind(C1, C2), rep(”<=”,2), c(bc1, bc2),compute.sens = 1)

##Unit worth of resources in $/hr
solution$duals[1:2]

##Lower limit of hours resource for respective machines for which dual prices are valid
solution$duals.from[1:2]

```

```
##Upper limit of hours resource for respective machines for which dual prices are valid
solution$duals.to[1:2]
```

R code Exa 3.6.2 Changing objective coefficient for Sensitivity analysis

```
##Chapter 3 : The Simplex Method and Sensitivity Analysis
##Example 6-2 : Page 121

# Objective function :Max 30 * x1 +  20 * x2
a <- c(30,20)

# Constraint 1 :  2 * x1 +  x2 <= 8
C1 <- c(2,1)
bc1<- 8

# Constraint 2 : x1 + 3 * x2 <= 8
C2 <- c(1,3)
bc2<-8

library("lpSolve")

solution=lp("max", a, rbind(C1, C2), rep("<=",2), c(bc1, bc2),compute.sens = 1)

##Lower limit of respective Objective coefficient for which the objective value will not change
solution$sens.coef.from
##Upper limit of respective Objective coefficient for which the objective value will not change
solution$sens.coef.to
```

R code Exa 3.6.3 Sensitivity Analysis using TOYCO Model

```
##Chapter 3 : The Simplex Method and Sensitivity
    Analysis
##Example 6-3 : Page 124

# Objective function :Max 3 * x1 +  2 * x2 + 5 * x3
a <- c(3,2,5)

# Constraint 1 :   x1 +  2 * x2 + x3 <= 430
C1 <- c(1,2,1)
bc1<- 430

# Constraint 2 : 3 * x1 + 2 * x3 <= 460
C2 <- c(3,0,2)
bc2<-460

# Constraint 3 : 1 * x1 + 4 * x2 <= 420
C3 <- c(1,4,0)
bc3<-420

library("lpSolve")
solution=lp("max", a, rbind(C1, C2, C3), rep("<=",3)
    , c(bc1, bc2, bc3),compute.sens = 1)

##Lower limit of hours resource for respective
    machines for which dual prices are valid
solution$duals.from

##Upper limit of hours resource for respective
    machines for which dual prices are valid
solution$duals.to

##Refer Footnote 10, Page 127
```

Chapter 4

Duality and Post optimality Analysis

R code Exa 4.2.1 Solving primal and dual problem using linprog pacakge

```
##Chapter 4 : Duality and Post−optimality Analysis
##Example 2−1 : Page 160

# Objective function :Max 5 ∗ x1 +  12 ∗ x2 + 4 ∗ x3
a <- c(5,12,4)

# Constraint 1 :   x1 +  2 ∗ x2 + x3 <= 10
C1 <- c(1,2,1)
bc1<- 10

# Constraint 2 : 2 ∗ x1 − x2 + 3 ∗ x3 = 8
C2 <- c(2,-1,3)
bc2<-8

library("linprog")

solveLP(a,c(bc1, bc2), rbind(C1, C2), c("<=","="),
   maximum = T,lpSolve = T)

```

```
## At the moment the dual problem can not be solved with equality constraints in the function solveLP
##Nevertheless we change the equality constrait to two inequality constraints. i.e.
## f1(x)=b ==> f1(x)<=b , f1(x)>=b
solveLP(a,c(bc1, bc2, bc2), rbind(C1, C2, C2), c("<=","<=",">="), maximum = T,lpSolve = T,solve.dual = T)
```

R code Exa 4.2.2 Associated objective values for arbitrary feasible primal and dual solutions

```
##Chapter 4 : Duality and Post-optimality Analysis
##Example 2-2 : Page 162

##Returns primal objective value
PrimalObj<-function(x1,x2,x3){
    return(5*x1+12*x2+4*x3)
}

#Returns dual objective value
DualObj<-function(y1,y2){
    return(10*y1+8*y2)
}

##Calling primal and dual functions with the arbitrary feasible solutions
PrimalObj(0,0,8/3)
DualObj(6,0)
```

R code Exa 4.2.3 Getting the simplex tableau at any iteration from the original data and inverse of the iteration

```
##Chapter 4 : Duality and Post-optimality Analysis
##Example 2-3 : Page 166

#Optimal Inverse
OptimalInv=matrix(c(2/5,-1/5,1/5,2/5),nrow=2,byrow=T)

#Original X1 column
OrigX1=c(1,2)

##Optimal X1 column using formaula 1
X1Optimal=OptimalInv%*%OrigX1
X1Optimal

##Functions to calculate Z coefficients of X1 and R
ZCoefX1<-function(y1,y2) {return(y1+2*y2-5)}
ZCoefR<-function(y1,y2) {return(paste(y2,"+ M"))}

## Calling the function with optimal dual values
ZCoefX1(29/5,-2/5)
ZCoefR(29/5,-2/5)
```

R code Exa 4.3.1 Economic interpretation of Dual variables of Reddy Mikks Problem

```
##Chapter 4 : Duality and Post-optimality Analysis
##Example 2-1 : Page 171

# Objective function :Max 5 * x1 + 4 * x2
a <- c(5,4)

# Constraint 1 :   6 * x1 +  4 * x2 <= 24
C1 <- c(6,4)
bc1<- 24

```

```
# Constraint 2 : x1 + 2 * x2 <= 6
C2 <- c(1,2)
bc2<-6

# Constraint 3 : -x1 + x2 <= 1
C3 <- c(-1,1)
bc3<-1

# Constraint 4 : x2 <= 2
C4 <- c(0,1)
bc4<-2

library("linprog")

solveLP(a,c(bc1, bc2, bc3, bc4), rbind(C1, C2, C3, C4), rep("<=",4), maximum = T,lpSolve = T)

##solve.dual arguement can be passed to solveLP function to solve the dual of the problem
solveLP(a,c(bc1, bc2, bc3, bc4), rbind(C1, C2, C3, C4), rep("<=",4), maximum = T,lpSolve = T, solve.dual = T)
```

R code Exa 4.3.2 Economic interpretation of Dual constraints of Reddy Mikks Problem

```
##Chapter 4 : Duality and Post-optimality Analysis
##Example 3-2 : Page 173

# Objective function :Max 3 * x1 + 2 * x2 + 5 * x3
a <- c(3,2,5)

# Constraint 1 : x1 + 2 * x2 + x3 <= 430
C1 <- c(1,2,1)
bc1<- 430
```

```

# Constraint 2 : 3 * x1 + 2 * x3 <= 460
C2 <- c(3,0,2)
bc2<-460

# Constraint 3 : 1 * x1 + 4 * x2 <= 420
C3 <- c(1,4,0)
bc3<-420

library("linprog")
solveLP(a,c(bc1, bc2, bc3), rbind(C1, C2, C3), rep("<=",3), maximum = T,lpSolve = T)

##solve.dual arguement can be passed to solveLP function to solve the dual of the problem
solveLP(a,c(bc1, bc2, bc3), rbind(C1, C2, C3), rep("<=",3), maximum = T,lpSolve = T,solve.dual = T)
```

R code Exa 4.4.1 Dual Simplex using glpk library

```
##Chapter 4 : Duality and Post-optimal Analysis
##Example 2-1 : Page 175

##Please note that the constraints are not given in the example(printing error) and
##it has been deduced from simplex tableau

##We will be using glpkAPI library. If you are using Debian system, run the following command
##before running the R script :: sudo apt-get install libglpk-dev
##For other linux and windows systems, install GLPK which is available at https://www.gnu.org/software/glpk/
##or from their respective repositories
```

```
library(glpkAPI)

# Objective function :Min 3 * x1 + 2 * x2 + 1 * x3
a <- c(3,2,1)

# Constraint 1 :   3 * x1 +  1 * x2 + x3 >= 3
# Standardizing constraint by multiplying by -1
C1 <- c(-3,-1,-1)
bc1<- -3

# Constraint 2 : -3 * x1 + 3 * x2 + x3 >= 6
# Standardizing constraint by multiplying by -1
C2 <- c(3,-3,-1)
bc2<--6

# Constraint 3 : x1 + x2 + x3 <= 3
C3 <- c(1,1,1)
bc3<-3

#upper bound vector
bc <- c(bc1,bc2,bc3)

#Constraint matrix
ConstraintMatrix <- rbind(a,C1,C2,C3)

#Initiating row and coloumn index variable as well
    as constraint coefficient value variable
rowindex <- numeric()
colindex <- numeric()
value <- numeric()

#initiate GLPK object and name
dualSimplex <- initProbGLPK()
setProbNameGLPK(dualSimplex, "Example 4-1")
setObjNameGLPK(dualSimplex, "Minimize using Dual
    Simplex")

#Setting objective direction and number of coloumns
```

```
setObjDirGLPK(dualSimplex, GLP_MIN)
addColsGLPK(dualSimplex, 3)

#setting decision variable names,bounds and coefficients
for (i in 1:3) {
  setColsNamesGLPK(dualSimplex,i,toString(c("x",i)))
  setColBndGLPK(dualSimplex, i, GLP_LO, 0.0, 0.0)
  setObjCoefsGLPK(dualSimplex, i, a[i])
}

#add 4 rows (including the objective)
addRowsGLPK(dualSimplex, 4)

#set row name as objective name itself
setRowsNamesGLPK(dualSimplex, 1, getObjNameGLPK(dualSimplex))

#set row names and bounds for constraint
for (i in 1:3){
  setRowsNamesGLPK(dualSimplex, i+1, toString(c("Constraint", i)))
  setRowBndGLPK(dualSimplex, i+1, GLP_UP, 0, bc[i])
}

#initiating row and coloumn index and the values
counter=1
for (i in 1:4) {
  for (j in 1:3) {
    rowindex[counter] <- i
    colindex[counter] <- j
    value[counter] <- ConstraintMatrix[i,j]
    counter=counter+1
  }
}

#change the soving algorithm to dual simplex
setSimplexParmGLPK(METH,GLP_DUAL)
```

```
#shows the current solver parameters
getSimplexParmGLPK()

#load and initiate all the data
loadMatrixGLPK(dualSimplex, 12, rowindex, colindex, value)

#Solve
solveSimplexGLPK(dualSimplex)

#Prints the status, optimal objective value and decisision variable value
getSolStatGLPK(dualSimplex)
getObjValGLPK(dualSimplex)
getColsPrimGLPK(dualSimplex)

#prints the summary of the optimization to your working directory
printSolGLPK(dualSimplex,'textfile.txt')

#deletes the glpk object
delProbGLPK(dualSimplex)
```

R code Exa 4.5.1 Changes in RHS

```
##Chapter 4 : Duality and Post−optimal Analysis
##Example 5−1 : Page 182

##Situation 1
Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1))
ConstraintNNewRightHandSide <-rbind(600,640,590)
TableauNewRightHandSide <- Inverse %*% ConstraintNNewRightHandSide
```

```
TableauNewRightHandSide

##Situation 2
ConstraintNNewRightHandSide <-rbind(450,460,400)
TableauNewRightHandSide <- Inverse %*% ConstraintNNewRightHandSide
TableauNewRightHandSide
```

R code Exa 4.5.2 Addition of a new constraint

```
##Chapter 4 : Duality and Post−optimal Analysis
##Example 5−2 : Page 185

##Situation 1 is a theoretical explanation

##Situation 2
tableau=matrix(c(-0.25,1,0,0.5,-0.25,0,0,100,1.5,0,1,0,1.5,0,0,230,2,0,0,-2,1,1,0,nrow=4,ncol =8,byrow=T )

OldX7Row <- tableau[4,]
CoeffX2 <-OldX7Row[2]
CoeffX3 <-OldX7Row[3]
X2Row <- c(-0.25,1,0,0.5,-0.25,0,0,100)
X3Row <- c(1.5,0,1,0,1.5,0,0,230)
tableau[4,] <- OldX7Row - (CoeffX2 %*% tableau[1,] + CoeffX3 %*% tableau[2,])
tableau
```

R code Exa 4.5.3 Changes in objective coefficient

```
##Chapter 4 : Duality and Post−optimal Analysis
##Example 5−3 : Page 187
```

```r

##Situation 1
NewObjCoeffBasic <- c(3,4,0)
Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
  )

NewDualVariables <-NewObjCoeffBasic %*%  Inverse
NewDualVariables

##Element-wise multiplication to get Reduced Costs
ReducedCostX1<-sum(c(1,3,1)*NewDualVariables) -2
ReducedCostX4<-sum(c(1,0,0)*NewDualVariables) -0
ReducedCostX5<-sum(c(0,1,0)*NewDualVariables) -0
ReducedCostX1
ReducedCostX4
ReducedCostX5


CurrentOptimal <- c(0,100,230)
NewObjCoeff<-c(2,3,4)

#Optimal Objective value
NewRevenue <- sum(NewObjCoeff*CurrentOptimal)
NewRevenue

##Situation 2
NewObjCoeffBasic <- c(3,4,0)
Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
  )

NewDualVariables <-NewObjCoeffBasic %*%  Inverse
NewDualVariables

##Element-wise multiplication to get Reduced Costs
ReducedCostX1<-sum(c(1,3,1)*NewDualVariables) -6
ReducedCostX4<-sum(c(1,0,0)*NewDualVariables) -0
ReducedCostX5<-sum(c(0,1,0)*NewDualVariables) -0
ReducedCostX1
```

```
ReducedCostX4
ReducedCostX5

NewObjCoeff<-c(-0.75,0,0,1.5,1.25,0)
C1<-c(-0.25,1,0,0.5,-0.25,0)
bc1<-100
C2<-c(1.5,0,1,0,0.5,0)
bc2<-230
C3<-c(2,0,0,-2,1,1)
bc3<-20
library("lpSolve")
solution=lp("max", NewObjCoeff, rbind(C1, C2, C3),
    rep("=",3), c(bc1, bc2, bc3))
solution
```

R code Exa 4.5.4 Addition of a new activity

```
##Chapter 4 : Duality and Post-optimal Analysis
##Example 5-4 : Page 189
OptimalDualValues<-c(1,2,0)

ReducedCostX7 <-sum(c(1,1,2)*OptimalDualValues)-4
ReducedCostX7

Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
    )
OldX7Col<-rbind(1,1,2)
NewX7Col<-Inverse%*%OldX7Col
NewX7Col
```

Chapter 5

Transportation Model and its Variant

R code Exa 5.1.1 A general transportation model

```r
##Chapter 5 : Transportation Model and its Variant
##Example 1-1 : Page 195
##Refer to the transportation tableau in Page 197
costs <- matrix (c(80,215,100,108,102,68), 3, 2, byrow = T)

#Constraints 1,2 & 3 are row constraints as they corresponds to rows of transportation tableau
row.signs <-rep("=",3)
row.rhs <-c(1000,1500,1200)

#Constraints 4 & 5 are coloumn constraints as they corresponds to coloumns of transportation tableau
col.signs <-rep("=",2)
col.rhs <-c(2300,1400)

library(lpSolve)
##lpSolve library has lp.transport function specially for problems which can be formulated as
```

```
    a transportation tableau
solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
solution$objval
solution$solution
```

R code Exa 5.1.2 Balancing transportation models

```
##Chapter 5 : Transportation Model and its Variant
##Example 1-2 : Page 197

##Part1- Adding a dummy origin
costs <- matrix (c(80,215,100,108,102,68,0,0), 4, 2,
    byrow = T)

#Constraints 1 to 4 are row constraints as they
    corresponds to rows of transportation tableau
row.signs <-rep("=",4)
row.rhs <-c(1000,1300,1200,200)

#Constraints 5 & 6 are coloumn constraints as they
    corresponds to coloumns of transportation tableau
col.signs <-rep("=",2)
col.rhs <-c(2300,1400)

library(lpSolve)
solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
solution$solution


##Part2 - Adding a dummy destination
costs <- matrix (c(80,215,0,100,108,0,102,68,0), 3,
    3,byrow = T)

```

```
#Constraints 1,2 & 3 are row constraints as they corresponds to rows of transportation tableau
row.signs <-rep(”=”,3)
row.rhs <-c(1000,1500,1200)

#Constraints 4 & 5 are coloumn constraints as they corresponds to coloumns of transportation tableau
col.signs <-rep(”=”,3)
col.rhs <-c(1900,1400,400)

library(lpSolve)
solution <- lp.transport (costs, ”min”, row.signs, row.rhs, col.signs, col.rhs)
solution$solution
```

R code Exa 5.2.1 Production inventory control

```
##Chapter 5 : Transportation Model and its Variant
##Example 2−1 : Page 202

costs <- 40 + matrix (c(0,0.5,1,1.5,2,0,0.5,1,4,2,0,0.5,6,4,2,0), 4, 4, byrow = T)

#Constraints 1,2,3 & 4 are row constraints as they corresponds to rows of transportation tableau
row.signs <-rep(”=”,4)
row.rhs <-c(50,180,280,270)

#Constraints 4 to 8 are coloumn constraints as they corresponds to coloumns of transportation tableau
col.signs <-rep(”=”,4)
col.rhs <-c(100,200,180,300)

library(lpSolve)
```

```
solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
solution$solution
solution$objval
```

R code Exa 5.2.2 Tool sharpening

```
##Chapter 5 : Transportation Model and its Variant
##Example 2-2 : Page 203

#BigM is Big
BigM=1000

#Initializing the matrix with all values as M
costs<-matrix(BigM,8,8,byrow = T)
#All values of first row are 12
costs[1,]=c(rep(12,8))
#All values of 8th column is 0
costs[,8]=rbind(rep(0,8))

##Adding the rest of the values
for (i in 2:7) {
  costs[i,]=c(rep(BigM,i-1),c(6,5,rep(3,4))[1:(8-i)
      ],0)
}

#Constraints 1 to 8 are row constraints as they
    corresponds to rows of transportation tableau
row.signs <-rep("=",8)
row.rhs <-c(124,24,12,14,20,18,14,22)

#Constraints 8 to 16 are coloumn constraints as they
     corresponds to coloumns of transportation
    tableau
col.signs <-rep("=",8)
```

```
col.rhs <-c(24,12,14,20,18,14,22,124)

library(lpSolve)
solution <- lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)
solution$solution
```

R code Exa 5.3.1 Sunray Transport

```
##Chapter 5 : Transportation Model and its Variant
##Example 3-1 : Page 207
costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18), 3, 4,byrow = T)

#Constraints 1,2 & 3 are row constraints as they corresponds to rows of transportation tableau
row.signs <-rep("=",3)
row.rhs <-c(15,25,10)

#Constraints 4 & 5 are coloumn constraints as they corresponds to coloumns of transportation tableau
col.signs <-rep("=",4)
col.rhs <-c(5,15,15,15)

library(lpSolve)
solution <- lp.transport (costs, "min", row.signs, row.rhs, col.signs, col.rhs)
solution$objval
solution$solution
```

R code Exa 5.3.2 Northwest corner method

```
##Chapter 5 : Transportation Model and its Variant
```

```
##Example 3-2 : Page 208
costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
    3, 4,byrow = T)
row.rhs <-c(15,25,10)
col.rhs <-c(5,15,15,15)
i=1;j=1
allotment <- matrix (rep(0,12), 3, 4,byrow = T)

while (sum(row.rhs) & sum(col.rhs)) {
  ##Till we reach the last row and coloumn
  while (i<=3 & j<=4) {
    ##if demand is >= supply
    if (row.rhs[i] >= col.rhs[j]) {
      ##assign the demand to that cell
      allotment[i,j]=col.rhs[j]
      ##deduct supply from demand
      row.rhs[i]=row.rhs[i]-col.rhs[j]
      ##assign zero to supply
      col.rhs[j]=0
    }else {
      ##assign the supply to that cell
      allotment[i,j]=row.rhs[i]
      ##deduct demand from supply
      col.rhs[j]=col.rhs[j]-row.rhs[i]
      ##assign zero to demand
      row.rhs[i]=0

    }
    #if demand=0, go to the next demand,else go to
       next supply
    ifelse(row.rhs[i] == 0,(i=i+1),(j=j+1))
  }
}
allotment
```

R code Exa 5.3.3 Least cost method

```
##Chapter 5 : Transportation Model and its Variant
##Example 3-3 : Page 209
costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
   3, 4,byrow = T)
row.rhs <-c(15,25,10)
col.rhs <-c(5,15,15,15)
costsdup <- costs
allotment <- matrix (rep(0,12), 3, 4,byrow = T)

#until there are supply and demand
while (sum(row.rhs) & sum(col.rhs)) {
  #index of min cost
  index=which.min(costsdup)
  #get the row index
  rowindex=index %% length(row.rhs)
  #get the coloumn index
  colindex=ceiling(index/length(row.rhs))
  #if row index=0 ,assign 3(since we are takinf
     modulus)
  if (rowindex==0) {rowindex=3}

  #if demand > supply
  if(row.rhs[rowindex]>=col.rhs[colindex]){
    #allocate supply to that cell
    allotment[index] <- col.rhs[colindex]
    ##deduct supply from demand
    row.rhs[rowindex] <- row.rhs[rowindex]-col.rhs[
       colindex]
    ##assign zero to supply
    col.rhs[colindex]=0
  }else{
    #allocate demand to that cell
    allotment[index] <- row.rhs[rowindex]
    ##deduct demand from supply
    col.rhs[colindex] <- col.rhs[colindex]-row.rhs[
       rowindex]
```

```
    ##assign zero to demand
    row.rhs[rowindex]=0
  }
  costsdup[index]=1000

}


allotment
```

R code Exa 5.3.5 Method of multipliers

```
##Chapter 5 : Transportation Model and its Variant
##Example 3-5 : Page 212
#u-v incidence matrix with columns - u1,v1,v2,u2,v3,
   v4,u3 and rows as uv equations
a=matrix(c
   (1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,1,
   ,6,7,byrow = T)

#cost matrix
costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
    3, 4,byrow = T)
#rhs of uv equations
n=c(10,2,7,9,20,18)

library(limSolve)
##Least Squares with Equality and Inequality
   Constraints
Sol=lsei(E=a, F=n, A = diag(7), B = rep(0, 7),G=diag
   (7),H=rep(0, 7), verbose = FALSE)$X

c(Sol[1]+Sol[5]-20,Sol[1]+Sol[6]-11,Sol[4]+Sol
   [2]-12,Sol[7]+Sol[2]-4,Sol[7]+Sol[3]-14,Sol[7]+
   Sol[5]-16)
```

R code Exa 5.4.1 Hungarian method 1

```
##Chapter 5 : Transportation Model and its Variant
##Example 4-1 : Page 221
costs <- matrix (c(15,10,9,9,15,10,10,12,8), 3, 3,
   byrow = T)
rowmin<-numeric()
colmin<-numeric()

##Subtracting minimum cost element of row from all
   the elements of rows
for(i in 1:nrow(costs)){
  costs[i,]<-costs[i,]-min(costs[i,])
}

##Subtracting minimum cost element of column from
   all the elements of column
for(i in 1:ncol(costs)){
  costs[,i]<-costs[,i]-min(costs[,i])
}

##logic1 is a boolean matrix which contains true if
   cost matrix after the above operations is 0 and 0
    otherwise
logic1<-costs==0
eqnrow<-numeric()

##We formulate it as a transportation tableau such
   that only one zero is selected from every row
   and every column
row.signs <-rep("=",nrow(costs))
row.rhs <-rep(1,nrow(costs))

col.signs <-rep("=",ncol(costs))
```

```
col.rhs <-rep(1,ncol(costs))

library(lpSolve)
Solution<-lp.transport (logic1, "max", row.signs,
   row.rhs, col.signs, col.rhs)
if (Solution$objval==nrow(costs)){
  Solution$solution
}
```

R code Exa 5.4.2 Hungarian method 2

```
##Chapter 5 : Transportation Model and its Variant
##Example 4-2 : Page 222
costs <- matrix (c
   (1,4,6,3,9,7,10,9,4,5,11,7,8,7,8,5), 4, 4,byrow =
    T)
rowmin<-numeric()
colmin<-numeric()
costsdup<-costs

##Subtracting minimum cost element of row from all
   the elements of rows
for(i in 1:nrow(costsdup)){
  costsdup[i,]<-costsdup[i,]-min(costsdup[i,])
}

##Subtracting minimum cost element of column from
   all the elements of column
for(i in 1:ncol(costsdup)){
  costsdup[,i]<-costsdup[,i]-min(costsdup[,i])
}


##We formulate it as a transportation tableau such
   that only one zero is selected from every row
```

```
   and every column
##maketable function returns the constraint matrix
eqnrow<-numeric()
maketable<-function(costsdup){
  logic1<-costsdup==0
  return(logic1)
}

matr<-maketable(costsdup)

row.signs <-rep("=",nrow(costsdup))
row.rhs <-rep(1,nrow(costsdup))

col.signs <-rep("=",ncol(costsdup))
col.rhs <-rep(1,ncol(costsdup))
library(lpSolve)

Solution<-lp.transport (matr, "max", row.signs, row.
   rhs, col.signs, col.rhs)
while(Solution$objval!=nrow(costsdup)){
  costsdup<-costsdup-min(costsdup[costsdup>0])
  for (i in 1:length(costsdup)){
    ifelse(costsdup[i]<0, (costsdup[i]=0),0)
  }
  matr<-maketable(costsdup)
  Solution<-lp.transport (matr, "max", row.signs,
     row.rhs, col.signs, col.rhs)
}
Solution$solution
```

Chapter 6

Network Model

R code Exa 6.2.1 Minimal spanning tree algorithm

```
##Chapter 6 : Network Model
##Example 2-1 : Page 239

#If you have trouble installing the package/library, please reinstall R form the following link:https://cran.r-project.org/bin/
library(igraph)
#creating the undirected graph with 6 nodes
A=graph(edges=c(1,2,1,5,1,3,1,4,2,5,2,3,2,4,3,4,3,6,4,5,4,6), n=6, directed=F )
#mst function generates the minimum spanning tree
MST<-mst(A,weights = c(1,9,5,7,3,6,4,5,10,8,3))

#The Fruchterman-Reingold Algorithm is a force-directed layout algorithm. The idea of a force directed layout algorithm is to consider
#a force between any two nodes. In this algorithm, the nodes are represented by steel rings and the edges are springs between them.
#The attractive force is analogous to the spring
```

```
force and the repulsive force is analogous to the
    electrical force. The basic idea is
#to minimize the energy of the system by moving the
   nodes and changing the forces between them.
#Here we create the cordinates for our graph using
   layout.fruchterman.reingold function
lay <- layout.fruchterman.reingold(A)

#Assigning the coordinates to the nodes of A
V(A)$x <- lay[, 1]
V(A)$y <- lay[, 2]

#assigning range of x and y
xlim <- range(lay[,1])
ylim <- range(lay[,2])

#plot graph A
plot.igraph(A, layout = lay,vertex.size=20,
            xlim = xlim, ylim = ylim, rescale =
              FALSE)
#plot MST with red edges and nodes over the previous
    graph
plot.igraph(MST, layout = lay,vertex.color="red",
   edge.color="red",vertex.size=20,
            add = TRUE, rescale = FALSE)
```

R code Exa 6.3.1 Equipment replacement as shortest route problem

```
##Chapter 6 : Network Model
##Example 3-1 : Page 243

#If you have trouble installing the package/library,
    please reinstall R form the following link:https
   ://cran.r-project.org/bin/
library(igraph)
```

```r

#creating the directed graph with 5 nodes
A=graph(edges=c(1,2,1,3,1,4,2,3,2,4,2,5,3,4,3,5,4,5), n=5, directed=T )
#creating weights vector for each edges of the graph
weightsg<-c(4000,5400,9800,4300,6200,8700,4800,7100,4900)
#get shortest path from 1 to 5
sP<-get.shortest.paths(A, 1, to=5,weights =weightsg)$vpath
#get shortest path cost from 1 to 5
sPCost<-shortest.paths(A, 1, to=5,weights =weightsg)
sP
sPCost

#creating coordinates for layout
l<-cbind(seq(0,9,2),0)

#plot the graph with straight edges(edge 1,4,7 & 9 are straight edges.The order of edges is taken from  graph initialization in line 6)
plot(delete.edges(A,c(2,3,5,6,8)),layout=l*2,vertex.size=15,edge.arrow.width = 0.2, asp = 0.5,edge.label=weightsg[c(1,4,7,9)])
#plot the graph with straight edges over the previous graph
plot(delete.edges(A,c(1,4,7,9)),edge.curved=.8,layout=l*2,vertex.size=15,edge.arrow.width = 0.2,asp = 0.5,edge.label=weightsg[c(2,3,5,6,8)],add=T)
A[] <- 0
#assign color red to each node of the graph in the shortest path
for (ed in 1:(length(sP[[1]])-1)){
  A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red")
  V(A)[sP[[1]][ed]]$color<-"red"
}
```

```
V(A)[5]$color<-"red"

#plot the shortest path over the previous graph
plot(A,layout=l*2,vertex.size=15,edge.arrow.width =
    0.2,edge.curved=.8,edge.color="orange", asp =
    0.5,add=T)
```

R code Exa 6.3.2 Most reliable route

```
##Chapter 6 : Network Model
##Example 3-2 : Page 244

#If you have trouble installing the package/library,
    please reinstall R form the following link:https
    ://cran.r-project.org/bin/
library(igraph)
#creating the directed graph with 7 nodes
A=make_directed_graph(edges=c
    (1,2,1,3,2,4,2,3,3,4,3,5,4,5,4,6,5,7,6,7), n=7)
#creating probability vector
prob<-c(.2,.9,.8,.6,.1,.3,.4,.35,.25,.5)
#creating weight vector from probability vector
weightsg<--round(log10(prob),digits = 5)
##calculating shortest path and its cost
sP<-get.shortest.paths(A, 1, to=7,weights =weightsg)
    $vpath
sPCost<-shortest.paths(A, 1, to=7,weights =weightsg,
    mode = "out")
sP
10^(-sPCost)
#plotting the graph A
l<-layout.auto(A)
plot(A,vertex.size=15,layout=l,edge.arrow.width =
    0.2, asp = 0.5,edge.label=weightsg)

```

```
#plotting the shortest path over graph A
A[] <- 0
for (ed in 1:(length(sP[[1]])-1)){
  A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red")
  V(A)[sP[[1]][ed]]$color<-"red"
}
V(A)[7]$color<-"red"
plot(A,vertex.size=15,layout=l, edge.arrow.width = 0.2,edge.color="orange", asp = 0.5,add=T)
```

R code Exa 6.3.3 Three jub puzzle

```
##Chapter 6 : Network Model
##Example 3-3 : Page 245
library(igraph)

#creating the directed graph with 15 nodes
A=make_directed_graph(edges=c(1,2,2,3,3,4,4,5,5,6,6,7,7,8,8,9,1,10,10,11,11,12,12,13,13,14,14,
, n=15)
#weights vector
weightsg<-rep(1,21)
#shortest path
sP<-get.shortest.paths(A, 1, to=9,weights =weightsg)$vpath
#shortest path cost
sPCost<-shortest.paths(A, 1, to=9,weights =weightsg, mode = "out")
sP
sPCost
#circle layout of graph
l<-layout.circle (A)
#plot graph A
plot(A,vertex.size=15,layout=l,edge.arrow.size =
```

```
0.2, asp = 0.5)

#make an empty graph and add edges of the shortest path to the graph with red color nodes
A[] <- 0
for (ed in 1:(length(sP[[1]])-1)){
  A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red")
  V(A)[sP[[1]][ed]]$color<-"red"
}
V(A)[9]$color<-"red"
#plot the shortest path over the previous graph
plot(A,vertex.size=15,layout=l, edge.arrow.size = 0.2,edge.color="orange", asp = 0.5,add=T)
```

R code Exa 6.3.4 Djiktras algorithm

```
##Chapter 6 : Network Model
##Example 3-4 : Page 248

# Initializing no of nodes and the distance matrix
n = 5
d<-matrix(Inf,nrow=n,ncol = n)
d[1,2]=100;d[1,3]=30;d[2,3]=20;d[3,4]=10;d[3,5]=50;d[4,2]=15;d[4,5]=50

#initalizing the djiktra's algorithm table as shown in the book
djiktable<-matrix(c(1:n,rep(0,3*n)),nrow=n)

now=1
djiktable[1,4]=1

#if any of the node doesn't have any outgoing edge, make it permanent. Ex: node 5
```

```
for (i in 1:5){
  if (length(which((d[i,]!=Inf) %in% TRUE))==0){
    djiktable[i,4]=1
  }
}
#while there are nodes with temporary status
while(sum(djiktable[,4])!=n){
  #find all the possibile nodes form current node
  possibles<-which((d[now,]!=Inf) %in% TRUE)
  #for each node in possible nodes
  for (i in possibles){
    #assign current node to temp
    temp<-djiktable[i,2]
    #if current node is not assigned
    if(djiktable[i,2]!=0){
      #the minimum of distance is added
      djiktable[i,2]=min(djiktable[i,2],djiktable[
         now,2]+ d[now,i])
    }else{
      djiktable[i,2]=djiktable[now,2]+ d[now,i]
    }
    #if there is no change in the next node
    if(djiktable[i,2]!=temp){
      #backtrack
      djiktable[i,3]=now
    }
  }

  #assign permanent status to the minimum index
  min.indx<-which.min(djiktable[possibles,2])
  djiktable[possibles[min.indx],4]=1
  now<- possibles[min.indx]
}

#prints out the shortest route
djiktable[now,2]
path<-character()
while (now!=0){
```

```
  path<-paste("->",now,path)
  now=djiktable[now,3]
}
path
```

R code Exa 6.3.5 Floyds algorithm

```
##Chapter 6 : Network Model
##Example 3-5 : Page 252
#Initializing the nodes and floyd's D matrix and S matrix
n = 5
floydD<-array(Inf,dim= c(5,5))
diag(floydD) <- 0
floydD[1,2]=3;floydD[1,3]=10;floydD[2,4]=5;floydD[3,4]=6;floydD[4,5]=4;

#symmetric matrix
for(i in 1:5){
  j=1
  while(j<i){
    floydD[i,j]=floydD[j,i]
    j=j+1
  }
}
floydseq<-matrix(1:n,nrow=n,ncol=n,byrow = T)
diag(floydseq) <- 0

##Floyd's Algorithm
for(k in 1:n){
  for(i in 1:n){
    for(j in 1:n){
      if(i!=k & j!=k & i!=j){
        if(floydD[i,k]+ floydD[k,j]< floydD[i,j]){
          floydD[i,j] <- floydD[i,k]+ floydD[k,j]
```

```
            floydseq[i,j] <- k
          }
        }
      }
    }
}
floydD
floydseq

##Printing the shortest route
path<-character()
i=1;j=5
while(floydseq[i,j]!=j){
  path<-paste("->",j,path)
  j=floydseq[i,j]
}
path<-paste(i,"->",j,path)
path
```

R code Exa 6.3.6 LP formulation and solution of shortest route problem

```
##Chapter 6 : Network Model
##Example 3-6 : Page 256

# Objective function :Min 100 * x12 +  30 * x13 +20* x23 + 10*x34 + 60 * x35 + 15 * x42 + 50 * x45
a <- c(100,30,20,10,60,15,50)

# Constraint 1 :  x12 +  x13 = 1
C1 <- c(1,1,0,0,0,0,0)
bc1<- 1

# Constraint 2 : x12 + x42 - x23 = 1
C2 <- c(1,0,-1,0,0,1,0)
bc2<-1
```

```

# Constraint 3 : x13 + x23 − x34 −x35 = 0
C3 <- c(0,1,1,-1,-1,0,0)
bc3<-0

# Constraint 4 : x34 − x42 − x45 = 0
C4 <- c(0,0,0,1,0,-1,-1)
bc4<-0

# Constraint 5 : x35 + x45 = 0
C5 <- c(0,0,0,0,1,0,1)
bc5<-0

library("lpSolve")

solution<-lp("min", a, rbind(C1, C2,C3,C4,C5), rep("
   =",5), c(bc1, bc2,bc3,bc4,bc5))
solution$objval
solution$solution
```

R code Exa 6.4.2 Maximal flow algorithm

```
##Chapter 6 : Network Model
##Example 4−2 : Page 263

#If you have trouble installing the package/library,
    please reinstall R form the following link:https
   ://cran.r−project.org/bin/
library(optrees)
#edge matrix  with weights
arcs<-matrix(c(1,2,20,
     1,3,30,
     1,4,10,
     1,3,30,
     1,4,10,
```

```
    2,3,40,
    2,5,30,
    2,5,30,
    3,5,20,
    4,5,20),
byrow=T,ncol=3)
#Finds the min cut for maximal flow
findMinCut(1:5, arcs, algorithm = "Ford-Fulkerson",
   source.node = 1,sink.node = 5, directed = T)$max.
   flow
```

R code Exa 6.4.3 LP formulation and solution of Maximal flow Mode

```
##Chapter 6 : Network Model
##Example 4-3 : Page 271

# Objective function 1 :Max x12 + x13 + x14
a <- c(1,1,1,0,0,0,0,0,0)

# Objective function 2 :Max x25 + x35 + x45
b <- c(0,0,0,0,1,0,1,0,1)

# Constraint 1 :  x12 -  x23 - x25 = 0
C1 <- c(1,0,0,-1,-1,0,0,0,0)
bc1<- 0

# Constraint 2 : x13 + x23 - x34 -x35 + x43 = 0
C2 <- c(0,1,0,1,0,-1,-1,1,0)
bc2<-0

# Constraint 3 : x14 + x34 - x43 -x45 = 0
C3 <- c(0,0,1,0,0,1,0,-1,-1)
bc3<-0

# Constraint 4 : x12 <= 20
```

```
C4 <- c(1,0,0,0,0,0,0,0,0)
bc4<-20

# Constraint 5 : x13 <= 30
C5 <- c(0,1,0,0,0,0,0,0,0)
bc5<-30

# Constraint 6 : x14 <= 10
C6 <- c(0,0,1,0,0,0,0,0,0)
bc6<-10

# Constraint 7 : x23 <= 40
C7 <- c(0,0,0,1,0,0,0,0,0)
bc7<-40

# Constraint 8 : x25 <= 30
C8 <- c(0,0,0,0,1,0,0,0,0)
bc8<-30

# Constraint 9 : x34 <= 10
C9 <- c(0,0,0,0,0,1,0,0,0)
bc9<-10

# Constraint 10 : x35 <= 20
C10 <- c(0,0,0,0,0,0,1,0,0)
bc10<-20

# Constraint 11 : x43 <= 5
C11 <- c(0,0,0,0,0,0,0,1,0)
bc11<-5

# Constraint 12 : x45 <= 20
C12 <- c(0,0,0,0,0,0,0,0,1)
bc12<-20

library(”lpSolve”)

solution<-lp(”max”, a, rbind(C1, C2,C3,C4,C5,C6,C7,
```

```
    C8,C9,C10,C11,C12), c(rep("=",3),rep("<=",9)), c(
    bc1, bc2,bc3,bc4,bc5,bc6,bc7,bc8,bc9,bc10,bc11,
    bc12))
solution$objval
solution$solution

solution<-lp("max", b, rbind(C1, C2,C3,C4,C5,C6,C7,
    C8,C9,C10,C11,C12), c(rep("=",3),rep("<=",9)), c(
    bc1, bc2,bc3,bc4,bc5,bc6,bc7,bc8,bc9,bc10,bc11,
    bc12))
solution$objval
solution$solution
```

R code Exa 6.5.1 Network representation for PERT and CPM

```
##Chapter 6 : Network Model
##Example 4-3 : Page 275
library(igraph)
#create a directed graph
A=make_directed_graph(edges=c
    (1,2,3,4,4,6,6,7,7,8,8,9,1,3,1,5,5,7,1,8), n=9)

#creating layout for plot
l<-cbind(c(1,1.5,3,5,4.5,7,9,11,13),c
    (1,3,3,3,2,3,2,1,1))

#creating plot
plot(A,vertex.size=15,layout=l,edge.arrow.size =
    0.5, asp = 0.2,edge.label=c("A -3","E - 2","F - 2
    ","G - 2","I - 2","J - 4","B - 2","D - 3","H - 1"
    ,"C - 4")      )
plot(graph(c(2,3),n=9),layout=l,edge.arrow.size =
    0.5,asp = 0.2,edge.lty=2,add=T)
```

R code Exa 6.5.2 Critical path method

```
##Chapter 6 : Network Model
##Example 5-2 : Page 280

#create distance matrix with -1 meaning no edge and
    0 meaning no cost edge
D<-matrix(c
    (-1,5,6,-1,-1,-1,-1,-1,3,8,-1,-1,-1,-1,-1,-1,2,11,-1,-1,-1,-1,0,1
    ,nrow=6,ncol = 6,byrow = T)

##Forward pass
ECT<-numeric()
ECT[1]<-0
for(i in 2:6){
  index<-which((D[,i]>=0) %in% TRUE)
  ECT[i]<-max(ECT[index]+D[index,i])
}

##Backward pass
BP<-numeric()
BP[6]<-ECT[6]
for(i in 5:1){
  index<-which((D[i,]>=0) %in% TRUE)
  BP[i]<-min(BP[index]-D[i,index])
}

##Finding critical nodes
critical<-character()
criti<-numeric()
critj<-numeric()
for(i in 1:6){
  for(j in 1:6){
    if(BP[i]==ECT[i] & BP[j]==ECT[j] & BP[j]-ECT[i
```

```
    ]==D[i,j]){
      critical<-cbind(critical,paste(i,"-",j))
      criti=cbind(criti,i)
      critj=cbind(critj,j)
    }
  }
}
critical

##Duration of the project
TotalDays<-0
for (i in 1:length(criti)){
  TotalDays<-TotalDays+D[criti[i],critj[i]]
}
TotalDays
```

R code Exa 6.5.3 Preliminary schedule

```
##Chapter 6 : Network Model
##Example 5-3 : Page 282
#Add an empty plot
plot(1, type="n", axes=T, xlab="Days", ylab="", yaxt
    ='n',xlim=c(0,25),ylim=c(0,10), yaxs= "i", xaxs =
     "i")

#Add line segment for each segment
segments(0, 9, x1 = 5, y1 = 9)
segments(5, 8, x1 = 13, y1 = 8)
segments(13, 7, x1 = 25, y1 = 7)
segments(0, 6, x1 = 11, y1 = 6,lty = 2)
segments(5, 5, x1 = 11, y1 = 5,lty = 2)
segments(8, 4, x1 = 13, y1 = 4,lty = 2)
segments(8, 3, x1 = 25, y1 = 3,lty = 2)
segments(13, 2, x1 = 25, y1 = 2,lty = 2)
#Add text at specific coordinates
```

```
text(c(2.5,5,7.5,8,10.5,16,19,19),c(9,6,5,8,4,3,2,7)+0.3,labels = c("A - 5","B -6","C - 3","D -8","E - 2","F - 11","G - 1","H - 12"),cex = 0.6)
#Add legend
legend('topright', c("Critical","Non-Critical") ,lty=c(1,2), bty='n', cex=.75)
```

R code Exa 6.5.4 Determination of floats and red flag rule

```
##Chapter 6 : Network Model
##Example 5-4 : Page 284

#create distance matrix with -1 meaning no edge and 0 meaning no cost edge
D<-matrix(c(-1,5,6,-1,-1,-1,-1,-1,3,8,-1,-1,-1,-1,-1,-1,2,11,-1,-1,-1,-1,0,1
,nrow=6,ncol = 6,byrow = T)

##Forward pass
ECT<-numeric()
ECT[1]<-0
for(i in 2:6){
  index<-which((D[,i]>=0) %in% TRUE)
  ECT[i]<-max(ECT[index]+D[index,i])
}

##Backward pass
BP<-numeric()
BP[6]<-ECT[6]
for(i in 5:1){
  index<-which((D[i,]>=0) %in% TRUE)
  BP[i]<-min(BP[index]-D[i,index])
}

##Finding critical nodes
```

```
critical<-character()
criti<-numeric()
critj<-numeric()
for(i in 1:6){
  for(j in 1:6){
      if(BP[i]==ECT[i] & BP[j]==ECT[j] & BP[j]-ECT[i]==D[i,j]){
        critical<-cbind(critical,paste(i,"-",j))
        criti=cbind(criti,i)
        critj=cbind(critj,j)
    }
  }
}
critical

##Duration of the project
TotalDays<-0
for (i in 1:length(criti)){
  TotalDays<-TotalDays+D[criti[i],critj[i]]
}
TotalDays

#Calculating total float and free float for non-critical activities
NonCritical<-matrix(c(1,3,2,3,3,5,3,6,4,6),ncol=2, byrow = T)
NCA<-character()
duration<-numeric()
TotalF<-numeric()
FreeF<-numeric()
for(i in 1:length(NonCritical[,1])){
  j<-NonCritical[i,1]
  k<-NonCritical[i,2]
  NCA[i]<-paste(j,"-->",k)
  duration[i]<- D[j,k]
  TotalF[i]<- BP[k]-ECT[j]-D[j,k]
  FreeF[i]<-ECT[k]-BP[j]-D[j,k]
}
```

```r

#Displaying df
df<-data.frame(NCA,duration,TotalF,FreeF)
names(df)<-c("Non-Critical Activity","Duration","Total Float","Free float")
df
```

R code Exa 6.5.5 LP formulation and solution of project scheduling problem

```r
##Chapter 6 : Network Model
##Example 5-5 : Page 288

# Objective function :Max 6*x12 + 6*x13 + 3*x23 + 8*x24 + 2*x35 + 11*x36 +1 *x46 + 12*x56
a <- c(6,6,3,8,2,11,0,1,12)

# Constraint 1 : -x12 - x13 = -1
C1 <- c(-1,-1,0,0,0,0,0,0,0)
bc1<- -1

# Constraint 2 : x12 - x23 - x24 = 0
C2 <- c(1,0,-1,-1,0,0,0,0,0)
bc2<-0

# Constraint 3 : x13 + x23 - x35 - x36 = 0
C3 <- c(0,1,1,0,-1,-1,0,0,0)
bc3<-0

# Constraint 4 : x24 -x45 -x46 = 0
C4 <- c(0,0,0,1,0,0,-1,-1,0)
bc4<-0

# Constraint 5 : x35 + x45 - x56 = 0
C5 <- c(0,0,0,0,1,0,1,0,-1)
```

```
bc5<-0

# Constraint 6 : x35 + x46 + x56 = 1
C6 <- c(0,0,0,0,0,1,0,1,1)
bc6<-1
library("lpSolve")
solution<-lp("max", a, rbind(C1, C2,C3,C4,C5,C6),
   rep("=",6), c(bc1, bc2,bc3,bc4,bc5,bc6))
solution$objval
solution$solution
```

Chapter 7

Advanced Linear Programming

R code Exa 7.1.2 All basic feasible and infeasible solutions of an equation

```
##Chapter 7 : Advanced Linear Programming
##Example 1-2 : Page 300

#Creating the A and b matrix
A=matrix(c(1,3,-1,2,-2,-2),nrow=2,byrow=T)
b=matrix(c(4,2),nrow=2)


s<-character()
basic<-character()
Type<-character()

for(i in 3:1){
  for(j in i:3){
    ##if i!=j and ith and jth column are not linearly dependent
    if(i!=j & !all(abs(A[,i]/A[,j])==c(1,1))){
      ##Solve for X with the ith and jth column
      a<-solve(A[,c(i,j)])%*%b
      s<-rbind(s,toString(a))
      basic<-rbind(basic,c(paste(i,"&",j)))
```

```
        #if all values in a>=0,then feasible ,
            infeasible otherwise
        Type<-rbind(Type,ifelse(all(a >= 0),"Feasible"
            ,"Infeasible"))
      }
    }
}

cbind(basic,s,Type)
```

R code Exa 7.1.3 Simmplex tableau in matrix form

```
##Chapter 7 : Advanced Linear Programming
##Example 1-3 : Page 303

# Objective function :Max x1 + 4*x2 + 2*x3 + 5*x4
a <- c(1,4,7,5)

# Constraint 1 :  2*x1 +  x2 + 2*x3 + 4*x4 = 10
C1 <- c(2,1,2,4)
bc1<- 10

# Constraint 2 : 3*x1 - x2 - 2*x3 + 6*x4 = 5
C2 <- c(3,-1,-2,6)
bc2<-5

A<-rbind(C1,C2)
b<-rbind(bc1,bc2)

Binv<-solve(A[,c(1,2)])
X<-Binv%*%b

Binv
X
```

R code Exa 7.2.1 Revised simplex algoithmr

```
##Chapter 7 : Advanced Linear Programming
##Example 2-1 : Page 309

# Objective function
a <- c(5,4,0,0,0,0)

#Constraints
C<-rbind(c(6,4,1,0,0,0),c(1,2,0,1,0,0),c
    (-1,1,0,0,1,0),c(0,1,0,0,0,1))
b<-rbind(24,6,1,2)
library("lpSolve")

solution<-lp("max", a, C, rep("=",4), b)
solution$objval
solution$solution

##This solver is based on the revised simplex method
```

R code Exa 7.3.1 Bounded variables algorithm

```
##Chapter 7 : Advanced Linear Programming
##Example 3-1 : Page 315
# Objective function
a <- c(3,5,2)

#Constraints
C<-rbind(c(1,1,2),c(2,4,3),c(1,0,0),c(0,1,0),c
    (0,-1,0),c(0,0,1))
b<-rbind(14,43,4,10,-7,3)
library("lpSolve")
```

```
solution<-lp("max", a, C, rep("<=",6), b)
solution$objval
solution$solution
```

R code Exa 7.4.1 Dual simplex algorithm

```
##Chapter 7 : Advanced Linear Programming
##Example 4-1 : Page 322
# Objective function
w <- c(3,5,0,0)

#Constraints
C<-rbind(c(1,2,1,0),c(-1,3,0,1))
b<-rbind(5,2)

B<-C[,c(1,4)]
Binv<-solve(B)

#Associated primal and dual variables are
    assocprimal and assocdual
assocprimal<-Binv%*%b
assocdual<-w[c(1,4)]%*%Binv

##the objective values are
primalobj<-c(3,0)%*%assocprimal
dualobj<-assocdual%*%c(5,0)

primalobj
dualobj
```

Chapter 8

Integer Linear Programming

R code Exa 8.1.1 Project selection

```
##Chapter 8 : Integer Linear Programming
##Example 1−1 : Page 336

# Objective function
a <- c(20,40,20,15,30)

# Constraint 1 :
C1 <- c(5,4,3,7,8)
bc1<- 25

# Constraint 2
C2 <- c(1,7,9,4,6)
bc2<-25

# Constraint 3
C3 <- c(8,10,2,1,10)
bc3<-25

library("lpSolve")
solution<-lp("max", a, rbind(C1, C2,C3), rep("<=",3)
    , c(bc1, bc2,bc3),all.bin=T)
```

```
solution$objval
solution$solution
```

R code Exa 8.1.2 Installing security phones

```
##Chapter 8 : Integer Linear Programming
##Example 1-2 : Page 340

# Objective function
a <- rep(1,8)

# Constraint 1 :
C1 <- c(1,1,rep(0,6))
bc1<- 1

# Constraint 2
C2 <- c(0,1,1,rep(0,5))
bc2<-1

# Constraint 3
C3 <- c(0,0,0,1,1,0,0,0)
bc3<-1

# Constraint 4
C4 <- c(rep(0,6),1,1)
bc4<-1

# Constraint 5
C5 <- c(rep(0,5),1,1,0)
bc5<-1

# Constraint 6
C6 <- c(0,1,0,0,0,1,0,0)
bc6<-1

```

```
# Constraint 7
C7 <- c(1,0,0,0,0,1,0,0)
bc7<-1

# Constraint 8
C8 <- c(0,0,0,1,0,0,1,0)
bc8<-1

# Constraint 9
C9 <- c(0,1,0,1,0,0,0,0)
bc9<-1

# Constraint 10
C10 <- c(0,0,0,0,1,0,0,1)
bc10<-1

# Constraint 11
C11 <- c(0,0,1,0,1,0,0,0)
bc11<-1

library("lpSolve")
solution<-lp("min", a, rbind(C1, C2,C3,C4,C5,C6,C7,
   C8,C9,C10,C11), rep(">=",11), c(bc1, bc2,bc3,bc4,
   bc5,bc6,bc7,bc8,bc9,bc10,bc11),all.bin=T)
solution$objval
solution$solution
```

R code Exa 8.1.3 Choosing a telephone company

```
##Chapter 8 : Integer Linear Programming
##Example 1-3 : Page 346

# Objective function
a <- c(0.25,0.21,0.22,16,25,18)

```

```
# Constraint 1 :
C1 <- c(1,1,1,0,0,0)
bc1<- 200

# Constraint 2
C2 <- c(1,0,0,-200,0,0)
bc2<-0

# Constraint 3
C3 <- c(0,1,0,0,-200,0)
bc3<-0

# Constraint 4
C4 <- c(0,0,1,0,0,-200)
bc4<-0

library("lpSolve")
solution<-lp("min", a, rbind(C1, C2,C3,C4), c("=","<=","<=","<="), c(bc1, bc2,bc3,bc4),int.vec=c(4,5,6))
solution$objval
solution$solution
```

R code Exa 8.1.4 Job scheeduling model

```
##Chapter 8 : Integer Linear Programming
##Example 1-4 : Page 350

M<-1000
# Objective function
a <- c(0,0,0,0,0,0,19,0,12,0,34,0)

# Constraint 1 :
C1 <- c(1,-1,0,M,0,0,0,0,0,0,0,0)
bc1<- 20
```

```
# Constraint 2
C2 <-c(-1,1,0,-M,0,0,0,0,0,0,0,0)
bc2<-5-M

# Constraint 3
C3 <- c(1,0,-1,0,M,0,0,0,0,0,0,0)
bc3<-15

# Constraint 4
C4 <- c(-1,0,1,0,-M,0,0,0,0,0,0,0)
bc4<-5-M

# Constraint 5
C5 <- c(0,1,-1,0,M,0,0,0,0,0,0,0)
bc5<-15

# Constraint 6
C6 <- c(0,-1,1,0,-M,0,0,0,0,0,0,0)
bc6<-20-M


# Constraint 7
C7 <- c(1,0,0,0,0,0,-1,1,0,0,0,0)
bc7<-20

# Constraint 8
C8 <- c(0,1,0,0,0,0,0,0,-1,1,0,0)
bc8<-2

# Constraint 9
C9 <- c(0,0,1,0,0,0,0,0,0,0,-1,1)
bc9<-20

library(”lpSolve”)
solution<-lp(”min”, a, rbind(C1, C2,C3,C4,C5,C6,C7,
   C8,C9), c(rep(”>=”,6),rep(”=”,3)), c(bc1, bc2,bc3
   ,bc4,bc5,bc6,bc7,bc8,bc9),int.vec=c(4,5,6))
```

```
solution$objval
solution$solution
```

R code Exa 8.2.1 Branch and bound algorithm

```
##Chapter 8 : Integer Linear Programming
##Example 2-1 : Page 356

# Objective function
a <- c(5,4)

# Constraint 1 :
C1 <- c(1,1)
bc1<- 5

# Constraint 2
C2 <-c(10,6)
bc2<-45

library("lpSolve")
solution<-lp("max", a, rbind(C1, C2), rep("<=",2), c
    (bc1, bc2),int.vec=c(1,2))
solution$objval
solution$solution

## lpSolve solver is based on the revised simplex
    method and a branch-and-bound (B&B) approach.
```

R code Exa 8.2.2 Cutting plane algorithm

```
##Chapter 8 : Integer Linear Programming
##Example 2-2 : Page 364

```

```
# Objective function
a <- c(7,10)

# Constraint 1 :
C1 <- c(-1,3)
bc1<- 6

# Constraint 2
C2 <-c(7,1)
bc2<-35

library("lpSolve")
solution<-lp("max", a, rbind(C1, C2), rep("<=",2), c(bc1, bc2),int.vec=c(1,2))
solution$objval
solution$solution

## lpSolve solver is based on the revised simplex method and a branch-and-bound (B&B) approach.
```

Chapter 9

Heuristic Programming

R code Exa 9.2.1 Discrete variable heuristics

```
##Chapter 8 : Heuristic Programming
##Example 2-1 : Page 382

##Initializing function F, Random number R and index
F=c(90,60,50,80,100,40,20,70)
R=0.1002
index=ceiling(R*8)
#while any of neighbouring values are lower,keep
    looping
while (F[index]>=F[index+1] | F[index]>=F[ifelse(
    index==1,1,index-1)]){
  #if the next index is less than the current index
  if(F[index+1]<F[index]){
    index=index+1
  }
  #if the previous index is less than the current
     index
  if(F[index-1]<F[index]){
    index=index+1
  }
}
```

```
index
F[index]
```

R code Exa 9.2.2 Random walk heuristic

```
##Chapter 8 : Heuristic Programming
##Example 2-2 : Page 383

##Initializing function F, Random number R, current
   index and index
F=c(90,60,50,80,100,40,20,70)
R=0.1002
count=0
current=ceiling(R*8)
index<-ceiling(runif(1,min = 0,max = 8))
#while count < 3,keep looping
while (count<3){

  count=count+1
  if(F[index]<F[current]){
    #set current to index and a new random value is
       alloted to index
    current=index
    index<-ceiling(runif(1,min = 0,max = 8))
    #count is set to 0 when another index lesser
       than current is found
    count=0
  }
}
current
F[current]
```

R code Exa 9.2.3 Random walk heuristic for continuous variables

```r
##Chapter 8 : Heuristic Programming
##Example 2-3 : Page 385

#Function F returns  the value of F for a given x
F <- function(x){return(x**5-10*x**4+35*x**3-50*x**2+24*x)}

#Intial values
x=0.5
newx=5
count=0
#while count<3
while(count<3){
  #generate newx random uniform sampling
  while(newx>4 | newx<0){newx<-x+(runif(1)-0.5)*(4-0)}
  #if the new value is better than the old
  if (F(newx)<F(x)){
    #assign newx as x,newx as 5(so that it enters newx while loop in next iteration) and count to 0
    x=newx
    newx=5
    count=0
  }else{
    #else increment count
    count=count+1
  }
}

print(paste("Optimal using uniform distribution = ",x))

x=0.5
newx=5
count=0
#while count<3
while(count<3){
```

```
  #generate newx random uniform sampling
  while(newx>4 | newx<0){newx<-x+(4*rnorm(1)/6)}
  if (F(newx)<F(x)){
    #assign newx as x,newx as 5(so that it enters
       newx while loop in next iteration) and count
       to 0
    x=newx
    newx=5
    count=0
  }else{
    #else increment count
    count=count+1
    newx=5
  }
}

print(paste("Optimal using normal distribution = ",x
   ))
```

R code Exa 9.3.1 Minimization of single varibale function using tabu search algorithm

```
##Chapter 8 : Heuristic Programming
##Example 3-1 : Page 388

G<-rep(0,8)
F=c(90,60,50,80,100,40,20,70)
##function to return the neighbourhood of an index
N<-function(index){
  A<-numeric()
  for(j in 1:2){
    for(i in 1:4){
      if(index+((-1)**j)*i<=8 & index+((-1)**j)*i>0)
         {
        A<-c(A,index+((-1)**j)*i)
```

```
      }
    }
  }
  return(A)
}

#for 100 repetitions
for(m in 1:100){
  #take a random uniform index
  index=ceiling(runif(1)*8)
  Tenure=3
  L=c(0)
  #a random index from the neighbourhood not in tabu list
  i=ceiling(runif(1)*sum(sum(!(N(index) %in% L))))

  #while new index is lesser than old
  while(F[index]>F[N(index)[!(N(index) %in% L)]][i]){
    #add current node to tabu list
    L<-c(L,index)
    #pick new node
    index=N(index)[!(N(index) %in% L)][i]
    #remove the node after their tenure
    if(i>2){
      L<-L[-1]
    }
    ##pick the next index
    i=ceiling(runif(1)*sum(sum(!(N(index) %in% L))))
  }
  #counter for bar plot
  G[index]=G[index]+1
}

barplot(G/sum(G),names.arg = 1:8)
```

R code Exa 9.3.3 Minimization of single varibale function using simulated annealing algorithm

```
##Chapter 8 : Heuristic Programming
##Example 3-3 : Page 396
library(GenSA)
#function that returns the function value
Fx<-function(x){
  F=c(90,60,50,80,100,40,20,70)
  return(F[ceiling(x)])
}
#GenSA is a continues function ,But as you see in the
    return value in function Fx, we return the value
    of the ceiling of the input
Solution=GenSA(fn=Fx,lower=c(1),upper=c(8),control =
    list(smooth =F))
Solution$value
ceiling(Solution$par)
```

R code Exa 9.3.4 Job scheduling using simulated annealing algorithm

```
##Chapter 8 : Heuristic Programming
##Example 3-1 : Page 388

#function to return the cost of a sequence
Fx<-function(index1){
  cost<-0
  date<-0
  ctable=matrix(c
     (10,15,3,10,8,20,2,22,6,10,5,10,7,30,4,8),nrow
     = 4,byrow=T)

```

```
  for(i in 1:4){
    date<-date+ctable[index1[i],1]

    if(date > ctable[index1[i],2]){
      cost<-cost+(date-ctable[index1[i],2])*ctable[index1[i],4]
    }else{
      cost<-cost+(ctable[index1[i],2]-date)*ctable[index1[i],3]
    }
  }
  return(cost)
}

#function to get the neighbourhoods of a sequence
N<-function(index1){
  swap <- function(x,i) {x[c(i,i+1)] <- x[c(i+1,i)]; x}
  A<-array(numeric(),c(0,4))
  for(i in 1:3){
    A<-rbind(A,swap(index1,i))
  }
  return(A)
}

#Initializations
index1=sample(c(1,2,3,4))
i=1;Ti<-Fx(index1)

#for 60 repetitions
while(i<=50){
  Ti=0.5*Ti
  p=0
  ##Step 2 of the algorithm
  if(p<3){
    Neighbours<-N(index1)
    RandomN<-Neighbours[ceiling(runif(1,0,3)),]
    if(Fx(index1)<Fx(RandomN)){
```

```
      if(runif(1)<exp(-abs(Fx(index1)-Fx(RandomN))/
         Ti)){
        index1<-RandomN
        p=p+1
        i=i+1
      }else{
        i=i+1
      }
    }else{
      index1<-RandomN
      p=p+1
      i=i+1
    }
  }else{
    i=i+1
  }
}
Fx(index1)
```

Chapter 10

Travelling Salesman Problem

R code Exa 10.3.1 Travelling Salesman Problem using branch and bound algorithm

```
##Chapter 10 : Travelling Salesman Problem
##Example 3-1 : Page 438

#TSP library
library("TSP")
#Distance matrix
dij<-matrix(c(Inf,10,3,6,9,5,Inf,5,4,2,4,9,Inf
    ,7,8,7,1,3,Inf,4,3,2,6,5,Inf),nrow=5,byrow = T)
#making a asymetric TSP instance
atsp<-ATSP(dij)
#Solve TSP
d=solve_TSP(atsp,method = "nearest_insertion",
    control = list(start,"1"))
d
d[1:5]
```

R code Exa 10.3.2 Travelling Salesman Problem using cutting plane algorithm

```
##Chapter 10 : Travelling Salesman Problem
##Example 3-2 : Page 441

#TSP library
library("TSP")
#Distance matrix
dij<-matrix(c(Inf,13,21,26,10,Inf,29,20,30,20,Inf
    ,5,12,30,7,Inf),nrow=4,byrow = T)
#making a asymetric TSP instance
atsp<-ATSP(dij)
#Solve TSP
d=solve_TSP(atsp,method = "nearest_insertion",
    control = list(start,"1"))
d
d[1:4]
```

R code Exa 10.4.1 Travelling Salesman Problem using nearest neighbour heuristic

```
##Chapter 10 : Travelling Salesman Problem
##Example 3-1 : Page 443

#Distance matrix
dij<-matrix(c(Inf,120,220,150,210,120,Inf
    ,100,110,130,220,80,Inf,160,185,150,Inf,160,Inf
    ,190,210,130,185,Inf,Inf),nrow=5,byrow = T)
#Source node
source=3
#All edges leading to 3 has infinite length
dij[,3]=Inf
tour<-source
##Nearest neighbour heuristic
for(i in 1:4){
    #choose nearest neighbour to the node
  mini<-which.min(dij[source,])
```

```
  #add it to the tour
  tour<-c(tour,mini)
  #make it as the source for next iteration
  source=mini
  #set all edges leading to current node as Inf
  dij[,mini]=Inf
}
```

Chapter 11

Deterministic Dynamic Programming

R code Exa 11.1.1 Shortest route problem using dynamic programming

```
##Chapter 11 : Deterministic Dynamic Programming
##Example 1-1 : Page 461

#Create a matrix with all Inf
d<-matrix(Inf,7,7)
#Add edge weights/lengths
d[8]=7;d[15]=8;d[22]=5;d[30:32]=c(12,8,7);d[38:39]=c
   (9,13);d[47:48]=c(9,6)
#Dynamic algorithm for Shortest path algorithm
ShortestDistance<-function(node,stage){

  index<-which(d[1:7,node]!=Inf)
  #if it is node 1 at stage 2, return 1
  if(c(1) %in% index & stage==2){
    return(d[1,node])
  #else return the minimum distance of all possible
     nodes from current node
  }else{
    dist<-numeric()
```

```
    for(i in 1:length(index)){
      dist[i]<-(min(d[index[i],node]+
         ShortestDistance(index[i],stage-1)))
    }
    return(min(dist))
  }
}
ShortestDistance(7,4)
```

Chapter 12

Deterministic Inventory Modelling

R code Exa 12.2.1 Analyzing the nature of demand

```
##Chapter 12 : Deterministic Inventory Modelling
##Example 2-1 : Page 493

##Natural gas consumption details
d<-matrix(c
   (100,110,90,70,65,50,40,42,56,68,88,95,110,125,98,80,60,53,44,45,
            88,79,56,57,38,39,60,70,82,90,121,130,95,90,70,58,41,44,
            68,55,43,41,65,79,88,94,130,122,100,85,73,58,42,43,64,75
            55,45,40,67,78,98,97,130,115,100,95,80,60,49,48,64,85,96
            39,69,90,100,110,87,80,78,75,69,48,39,41,50,70,88,93)
               ,byrow=T,ncol=12)
#row and coloumn names
rownames(d)<-1990:1999
colnames(d)<-month.abb

```

```
#assigns mean,standards deviation and median to a
    dataframe
tmp <- do.call(data.frame,
               list(mean = apply(d, 2, mean),
                    sd = apply(d, 2, sd),
                    median = apply(d, 2, median)))
rbind(d,t(tmp))
```

R code Exa 12.3.1 Classic EOQ model problem

```
##Chapter 12 : Deterministic Inventory Modelling
##Example 3-1 : Page 495

#input parametres
D=100
K=100
h=0.02
L=12
#optimal quantity
y=sqrt(2*K*D/h)

print(paste("EOQ =",y," units"))
#cycle length
t=y/D
print(paste("Cycle length =",t," days"))
#number of cycles
L0=L- floor(L/t)*t

print(paste("Reorder point =",L0*D," units"))
print(paste("Daily inventory cost = $",(K*D/y)+h*y/
    2))
```

R code Exa 12.3.2 Optimal order policy

```
##Chapter 12 : Deterministic Inventory Modelling
##Example 3-2 : Page 501

#input parameters
D=187.5
h=0.02
K=20
L=2
c1=3
c2=2.5
q=1000
#optimal quantity
ym=sqrt(2*K*D/h)
#function to calculate the least cost
if(q<ym){
  Y=ym
}else{
  x=polyroot(c(2*K*D/h,(2*(c2*D-(c1*D+(K*D/ym)+(h*ym
     /2) ))/h),1))
  if(Re(x[2])>q){
    Y=q
  }else{
    Y=ym
  }
}
Y
```

R code Exa 12.4.1 Dynamic EOQ models with no setup

```
##Chapter 12 : Deterministic Inventory Modelling
##Example 4-1 : Page 508

#Setting up initial cost matrix and other values
cost<-matrix(Inf,8,5)
for(i in 1:4){
```

```
  for (j in i:4){
    cost[2*i-1,j]=6 + (j-i)*0.1
    cost[2*i,j]=9 + (j-i)*0.1
  }
}
cost[,5]=0
totalcost=0
Regular=c(90,100,120,110)
Overtime=c(50,60,80,70)
Demand=c(100,190,210,160,20)
Supply= c(rbind(Regular,Overtime))

#Setting up the resultant table
rown=numeric()
for(i in 1:4){rown=c(rown,paste("R",i,sep = ""),paste("O",i,sep = ""))}
allocation=matrix(0,8,5,dimnames = list(rown,c(1:4,"Surplus")))


for(i in 1:5){

  while(Demand[i]>0){
    minindex=which.min(cost[,i])
    #if it can still meet the demand
    if(Supply[minindex]>0){
      #allocate min of demand or supply
      allocation[minindex,i]=min(Demand[i],Supply[minindex])
      #calculate cost
      totalcost=totalcost+allocation[minindex,i]*cost[minindex,i]
      ##Subtract the allocated form supply and demand
      tmp=Demand[i]
      Demand[i]=Demand[i]-min(Demand[i],Supply[minindex])
      Supply[minindex]=Supply[minindex]-min(tmp,
```

```
           Supply[minindex])
        cost[minindex,i]=Inf
      #else set all cost of that week to Inf so that
         it doesnt get chosen again
      }else{
        cost[minindex,]=Inf
      }


    }
}
allocation
totalcost
```

Chapter 13

Decision Analysis and Games

R code Exa 13.1.1 Overall Idea of AHP

```
##Chapter 13 : Decision Analysis and Games
##Example 1-1 : Page 527

#Setting up input values
LocationC=c(12.9,27.2,59.4)
ReputationC=c(54.5,27.3,18.2)
LocationP=0.17
ReputationP=0.83
Criteria=c(LocationP,ReputationP)
#composite weights
Choice=Criteria%*%rbind(LocationC,ReputationC)

colnames(Choice)<-c("UofA","UofB","UofC")



#Setting up the data tree for the particular problem
#If you have trouble installing the package/library, please reinstall R form the following link:https://cran.r-project.org/bin/
library(data.tree)
SelectAUniversity <- Node$new("Select a University")
```

```
  Location<-SelectAUniversity$AddChild(paste("Location",LocationP))
    for(child in 1:3){
      Location$AddChild(paste("Uof",LETTERS[child],LocationC[child]))
    }
  Reputation<-SelectAUniversity$AddChild(paste("Reputation",ReputationP))
    for(child in 1:3){
      Reputation$AddChild(paste("Uof",LETTERS[child],LocationC[child]))
    }
print(SelectAUniversity)
print(Choice)
```

R code Exa 13.1.2 Weights and consistency for AHP

```
##Chapter 13 : Decision Analysis and Games
##Example 1-2 : Page 530

A=matrix(c(1,0.2,5,1),nrow = 2,byrow = T)
rowMeans(sweep(A,2,colSums(A),'/'))
AL=matrix(c(1,0.5,0.2,2,1,0.5,5,2,1),nrow = 3,byrow = T)
rowMeans(sweep(AL,2,colSums(AL),'/'))
AR=matrix(c(1,2,3,0.5,1,1.5,1/3,2/3,1),nrow = 3,byrow = T)
rowMeans(sweep(AR,2,colSums(AR),'/'))
```

R code Exa 13.1.3 Consistency ratio

```
##Chapter 13 : Decision Analysis and Games
##Example 1-3 : Page 533
```

```
AL=matrix(c(1,0.5,0.2,2,1,0.5,5,2,1),nrow = 3,byrow
   = T)
wi=round(rowMeans(sweep(AL,2,colSums(AL),'/')),3)
nmax=sum(round((AL%*%wi),4))
#Consistency index
CI=(nmax-3)/2
#random consistency of A
RI=1.98/3
#Consistency ratio
CR=CI/RI
CR
```

R code Exa 13.2.1 Decision tree

```
##Chapter 13 : Decision Analysis and Games
##Example 2-1 : Page 538

CompanyA=c(5000,-2000)
CompanyB=c(1500,500)
#Probability of occurance
POfOccurance=c(0.6,0.4)
#Espected return of stock A and B
ExpectedStockA=POfOccurance%*%CompanyA
ExpectedStockB=POfOccurance%*%CompanyB
ExpectedStockA
ExpectedStockB
```

R code Exa 13.2.2 Expected value criterion

```
##Chapter 13 : Decision Analysis and Games
##Example 2-2 : Page 544

```

```
#Probability matrix
CondP=matrix(c(0.9,0.1,0.5,0.5),nrow=2,byrow = T)

PriorP=c(0.6,0.4)
JointP=CondP*PriorP
AbsP=colSums(JointP)
BayesP=sweep(JointP,2,AbsP,‘/‘)

CompanyA=c(5000,-2000)
CompanyB=c(1500,500)

ExpectedStockAat4=BayesP[,1]%*%CompanyA
ExpectedStockBat5=BayesP[,1]%*%CompanyB
ExpectedStockAat6=BayesP[,2]%*%CompanyA
ExpectedStockBat7=BayesP[,2]%*%CompanyB

ExpectedStockAat4
ExpectedStockBat5
ExpectedStockAat6
ExpectedStockBat7
```

R code Exa 13.3.1 Decision under uncertainity

```
##Chapter 13 : Decision Analysis and Games
##Example 3-1 : Page 553

#Cost matrix
costmatrix<-matrix(c
    (5,10,18,25,8,7,12,23,21,18,12,21,30,22,19,15),
    nrow = 4,byrow = T)

#Laplace criterion
print(”Laplace”)
P=1/4
E=rowSums(costmatrix)*P
```

```
min(E)*1000

#Minimax criterion
print("Minimax")
E=apply(costmatrix,1,max)
min(E)*1000

#Savage criterion
print("Savage")
r=sweep(costmatrix,2,apply(costmatrix,2,min))
E=apply(r,1,max)
min(E)
```

R code Exa 13.4.1 Two person zero sum game

```
##Chapter 13 : Decision Analysis and Games
##Example 4-1 : Page 556

#payoff matrix
payoffmatrix=matrix(c(8,-2,9,-3,6,5,6,8,-2,4,-9,5),
   nrow = 3,byrow = T)

RowMin=apply(payoffmatrix,1,min)
print("Maximin")
max(RowMin)

ColMax=apply(payoffmatrix,2,max)
print("Maximin")
min(ColMax)
```

R code Exa 13.4.3 Mixed strategy games

```
##Chapter 13 : Decision Analysis and Games
```

```
##Example 4-3 : Page 559

#plot an empty graph
plot(1, type="n", axes=F,xlab = "x1", ylab = "x2", xlim = c(-0.5,1.5), ylim = c(-2,7), col = "red",
    yaxs= "i", xaxs = "i")

#Add custom axis on sides 1 and 2
axis(side = 1,pos = 0,at = seq(0,1,0.25))
axis(side = 2,pos = 1,at = seq(-2,9,2),padj=3.5)
axis(side = 2,pos = 0,at = seq(-2,9,2))

#Add line segments for the constriants
segments (0,6,1,-1,col = "red")
segments (0,4,1,2,col = "yellow")
segments (0,3,1,2,col = "green")
segments (0,2,1,3,col = "blue")
#adding a line segment to indicate the maximum
segments (0.5,0,0.5,2.5,col = "black",lwd = 1.5)
#Adding the name of the constriants on the graph
text(0,4,"B1",pos=4,cex=0.5)
text(0,3,"B2",pos=4,cex=0.5)
text(0,2,"B3",pos=4,cex=0.5)
text(0,6,"B4",pos=4,cex=0.5)
```

Chapter 14

Probabilistic Inventory Models

R code Exa 14.1.1 Probabilistic Inventory Models with normal distribution of demand

```
##Chapter 14 : Probabilistic Inventory Models
##Example 1-1 : Page 575

#Daily demand
D=100
#standard deviation
dev=10
#probability of running out of stock
alpha=0.05
#lead time
L=2
#average demand during lead time
muL=D*L
#standard deviation of demand during lead time
devL=sqrt((dev**2)*L)
#optimal inventory level for reordering
x=qnorm(0.05,100,10,lower.tail = F)
z=(x-D)/dev
B=devL*z
B
```

R code Exa 14.2.1 Newsvendor problem

```
##Chapter 14 : Probabilistic Inventory Models
##Example 2-1 : Page 582

#holding cost
h=25
#penalty cost
p=45
#critical ratio
CR=p/(p+h)

D=matrix(c(200,220,300,320,340,0.1,0.2,0.4,0.2,0.1),
   nrow = 2,byrow = T)

#The demand is a normal distribuion
print("Case A")
ystar=qnorm(0.643,300,20)

#The demand is a discrete PDF
print("Case B")
CDF=numeric()
for(i in 1:5){CDF[i]=sum(D[2,1:i])}
rbind(D,CDF)
```

Chapter 15

Markov Chain

R code Exa 15.2.1 Absolute probabilities after transitions

```
##Chapter 15 : Markov Chain
##Example 2-1 : Page 596

#Library expm is needed to get element-wise power of a matrix
library(expm)
P=matrix(c(0.3,0.6,0.1,0.1,0.6,0.3,0.05,0.4,0.55),
  nrow = 3,byrow = T)
P8=P%^%8
P16=P%^%16

#Steady-state probabilities
a1=c(1,0,0)%*%P
a8=c(1,0,0)%*%P8
a16=c(1,0,0)%*%P16

a1
a8
a16
```

R code Exa 15.3.1 Absorbing and transient states

```r
##Chapter 15 : Markov Chain
##Example 3-1 : Page 598

##Editing mistake in textbook
##The last element of P should be one as (sum over j
    )(p)=1 for all i
P=matrix(c(0.2,0.5,0.3,0,0.5,0.5,0,0,1),nrow = 3,
    byrow = T)
library(expm)
P%^%100
```

R code Exa 15.3.2 Periodic states

```r
##Chapter 15 : Markov Chain
##Example 3-2 : Page 599

#transition matrix
P=matrix(c(0,0.6,0.4,0,1,0,0.6,0.4,0),nrow = 3,byrow
     = T)
library(expm)
#n=2
P%^%2
#n=3
P%^%3
#n=4
P%^%4
#n=5
P%^%5
```

R code Exa 15.4.1 Steady state probabilities

```
##Chapter 15 : Markov Chain
##Example 4-1 : Page 601

#transition matrix
A=matrix(c(0.6,-0.4,0.4,0.1,0.3,-0.45,1,1,1),nrow =
    3,byrow = T)
b=c(rep(0,2),1)
#solve the linear equations
pi=solve(A,b)
pi
#mean recurrance time
mu=1/pi
mu
```

R code Exa 15.4.2 Cost model

```
##Chapter 15 : Markov Chain
##Example 4-2 : Page 602

#transition matrix
A=matrix(c(0.6,-0.4,0.4,0.1,0.3,-0.45,1,1,1),nrow =
    3,byrow = T)
b=c(rep(0,2),1)

#solve the linear equations
pi=solve(A,b)
#mean recurrance time
mu=1/pi

#expected abbual cost of fertilizer
```

```
bagsOfFertilizer=c(2,2*1.25,2*1.6)
sum(bagsOfFertilizer*50*pi)
```

R code Exa 15.5.1 Mean first passage time

```
##Chapter 15 : Markov Chain
##Example 5-1 : Page 606

#transition matrix
P=matrix(c(0.3,0.6,0.1,0.1,0.6,0.3,0.05,0.4,0.55),
    nrow = 3,byrow = T)
N1=P[-1,-1]
#mean first-passage time
mu=solve(diag(2)- N1)%*%rep(1,2)
```

R code Exa 15.6.1 Analysis of absorbing states

```
##Chapter 15 : Markov Chain
##Example 6-1 : Page 610

#transition matrix
P=matrix(c
    (0,0.95,0,0,0.05,0,0.07,0,0.9,0,0.03,0,0,0,0,0.95,0.05,0,0,0,0.07
    ,nrow=6,byrow = T)
N=P[1:4,1:4]
A=P[1:4,5:6]
#Espexted time of absorption
ExpT=solve(diag(4)-N)
#Pobability of absorption
POfAbsorption=ExpT %*% A
ExpT
POfAbsorption
```

Chapter 16

Queuing Systems

R code Exa 16.4.1 Pure birth model

```
##Chapter 16 : Queuing Systems
##Example 4-1 : Page 629

#birth-rate per day
lambdaday=24*60/12
#birth-rate per year
lambdayear=lambdaday*365
#probability of no births during a day
P=((lambdaday*1)^0 * exp(-120*1))/factorial(0)
P
#probability of issuing 50 birth certificate in 3
   hrs given that 40 certificates were issued during
    the first 2 hrs off 3 hr period
P1=((60/12 *1)^10 *exp(-5*1))/factorial(10)
P1
```

R code Exa 16.4.2 Pure death model

```r
##Chapter 16 : Queuing Systems
##Example 4-1 : Page 632

#probability of placing an order in any one day of the week
Pfunction <- function(mu,t,n,Slimit){
  #empty matrix for the table
  A<-matrix(0,nrow = 3,ncol = 7)
  p0=0
  for(t in 1:7){
    P=p0
    for(n in 1:5){
      P=P+((mu*t)^(Slimit-n) *exp(-mu*t))/factorial(Slimit-n)
    }
    #appending to the table
    A[1,t]=t
    A[2,t]=mu*t
    A[3,t]=round(P,digits = 4)
  }
  return(A)
}
Pfunction(3,7,5,18)

#average number of dozen roses discarded at the end of the week
discardedRoses<-function(N,mu,n,t){
  P=0
  for(i in 1:n){
    P=P+i*((mu*t)^(N-i) * exp(-mu*t))/factorial(N-i)
  }
  return(P)
}
discardedRoses(18,3,18,7)
```

R code Exa 16.6.1 Measures of performance

```
##Chapter 16 : Queuing Systems
##Example 6-1 : Page 642

#Function to calculate p0
P0<-function(n,servers,temp){
  x=1
  for(i in 1:n){
    if (i<6){
     x=x+(temp^i)/factorial(i)
    }else{
      x=x+(temp^i)/(factorial(servers)*servers^(i-
         servers))
    }
  }
  return(1/x)
}
p0=P0(8,5,3)
p0

#calculate Pn
temp=3
servers=5
A=matrix(0,2,8)
for(i in 1:8){
  if (i<6){
    A[1,i]=i
    A[2,i]=p0*(3^i)/factorial(i)
  }else{
    A[1,i]=i
    A[2,i]=p0*(temp^i)/(factorial(servers)*servers^(
       i-servers))
  }
}
A
#arrival rate
lambda=6
```

```
#arrivals lost
lambdalost=lambda*A[2,8]
lambdalost
#effective arrivals
lamdaeff=lambda-lambdalost
lamdaeff
#average lengths in the systems
Ls=0*p0+sum(1:8*A[2,])
Ls
#waiting time in the systems
Ws=Ls/lamdaeff
Ws
#average lengths in the queue
Wq=Ws-1/2
Wq
#average number of occupied spaces
cbar=lamdaeff/2
cbar
#parking lot utilization
utilization=cbar/servers
utilization
```

R code Exa 16.6.2 MM1 GDInfInf model

```
##Chapter 16 : Queuing Systems
##Example 6−2 : Page 645

#Queueing library to process different queueing
   models
library(queueing)
#creating a MM1 instance with the following
   parameters
x=NewInput.MM1(lambda=4, mu=6,n=25)
#Solving the model which returns a list
y=QueueingModel(x)
```

```
#probability of each of the n customers in the
    system
P=y$Pn

#number of parking spaces such that an arriving car
    finds a place atleast 90% of times
cummP=numeric()
for(i in 1:length(P)){cummP[i]=sum(P[1:i])}
min(which(cummP > 0.9))-1
```

R code Exa 16.6.4 MM1 GDNInf model

```
##Chapter 16 : Queuing Systems
##Example 6-4 : Page 649

#Queueing library to process different queueing
    models
library(queueing)
#creating a MMK instance with the following
    parameters
x=NewInput.MM1K(lambda=4, mu=6,k=5)
#Solving the model which returns a list
y=QueueingModel(x)
summary(y)
```

R code Exa 16.6.5 MMc GDInfInf model

```
##Chapter 16 : Queuing Systems
##Example 6-5 : Page 653

#Queueing library to process different queueing
    models
library(queueing)
```

```
#creating a MMc instance with the following
    parameters
x=NewInput.MMC(lambda=8, mu=5,c=2)
#Solving the model which returns a list
y=QueueingModel(x)
summary(y)

#creating a MMc instance with the following
    parameters
x=NewInput.MMC(lambda=16, mu=5,c=4)
#Solving the model which returns a list
y=QueueingModel(x)
summary(y)
```

R code Exa 16.6.6 MMc GDNInf model

```
##Chapter 16 : Queuing Systems
##Example 6−6 : Page 657

#Queueing library to process different queueing
    models
library(queueing)

#creating a MMc instance with the following
    parameters
x=NewInput.MMCK(lambda=16, mu=5,c=4,k=10)
#Solving the model which returns a list
y=QueueingModel(x)
summary(y)
```

R code Exa 16.6.7 MMInf GDInfNEInf or Slef service models

```
##Chapter 16 : Queuing Systems
##Example 6−7 : Page 660

#Queueing library to process different queueing
   models
library(queueing)
#creating a MMc instance with the following
   parameters
x=NewInput.MMInf(lambda=12, mu=0.333)
#Solving the model which returns a list
y=QueueingModel(x)
summary(y)
estimate=(0.25*round(y$L)*1000)*(1-0.2)+(0.75*round(
   y$L)*1000)*(1+0.12)
estimate
##The answer for the estimate is given wrong in the
   book
```

R code Exa 16.6.8 MMR GDKK or Machine servicing model

```
##Chapter 16 : Queuing Systems
##Example 6−8 : Page 662

#Queueing library to process different queueing
   models
library(queueing)

#Empty matrix table for Machine productivity
McProductivity=matrix(0,2,4)
#Making a matrix for different number of
   repairpersons
for(i in 1:4){
  #creating a MMc instance with the following
     parameters
  x=NewInput.MMCKK(lambda=0.5, mu=5, c=i, k=22)
```

```
  #Solving the model which returns a list
  y=QueueingModel(x)
  McProductivity[1,i]=(22-y$L)*100/22
  if(i>1){
    McProductivity[2,i]=McProductivity[1,i]-
       McProductivity[1,i-1]
  }
}
McProductivity
```

R code Exa 16.7.1 MG1 GDInfInf model or Pollaczek Khintchine formula

```
##Chapter 16 : Queuing Systems
##Example 7-1 : Page 664

#arrival rate
lambda=4
#Expectation of arrival
EOfT=1/6
#variance of arrival
VarOfT=0
#Length of the system
Ls=lambda*EOfT+((lambda^2 *(EOfT^2 + VarOfT))/(2*(1-
   lambda*EOfT)))
#Length of the queue
Lq=Ls-lambda*EOfT
#Waiting time in the system
Ws=Ls/lambda
#Waiting time in the queue
Wq=Lq/lambda
Ls
Lq
Ws
Wq
```

R code Exa 16.9.1 Cost models

```
##Chapter 16 : Queuing Systems
##Example 9-1 : Page 667

#Queueing library to process different queueing models
library(queueing)
#inputs
Speed=c(30,36,50,66)
OperatingCost=c(15,20,24,27)

#service rate for each model
ServiceRate=numeric()
for(i in 1:4){
  ServiceRate[i]=24/(10000/(Speed[i]*60))
}
ServiceRate
Ls=numeric()

#length of system for each model
for(i in 1:4){
  #creating a MMc instance with the following parameters
  y=NewInput.MM1(lambda=4, mu=ServiceRate[i])
  #Solving the model which returns a list
  z=QueueingModel(y)
  Ls[i]=z$L
}
Ls
#cost for the 4 models
cost=matrix(0,4,3)
colnames(cost)=c("EOC","EWC","ETC")
for(i in 1:4){
```

```
  cost[i,1]=24*OperatingCost[i]
  cost[i,2]=80*Ls[i]
  cost[i,3]=cost[i,1]+cost[i,2]
}
cost
```

R code Exa 16.9.2 Practical problem for MMc GDInfInf model

```
##Chapter 16 : Queuing Systems
##Example 9-2 : Page 670

#Queueing library to process different queueing
   models
library(queueing)

Ls=numeric()
ETC=numeric()
for (c in 2:6){
  #creating a MMc instance with the following
     parameters
  y=NewInput.MMC(lambda=17.5, mu=10,c)
  #Solving the model which returns a list
  z=QueueingModel(y)
  Ls[c]=z$L
  ETC[c]=12*c + 50*Ls[c]
}
ETC[2:6]
```

R code Exa 16.9.3 Aspiration level model

```
##Chapter 16 : Queuing Systems
##Example 9-3 : Page 672

```

```
#Queueing library to process different queueing
   models
library(queueing)
Ws=numeric()
RO=numeric()


for (c in 2:8){
  #creating a MMc instance with the following
     parameters
  y=NewInput.MMC(lambda=17.5, mu=10,c)
  #Solving the model which returns a list
  z=QueueingModel(y)
  #waiting time in the system
  Ws[c]=z$W*60
  #idleness percentage of the system
  RO[c]=(1-z$RO)*100
}
rbind(Ws,RO)[,2:8]
```

Chapter 17

Simulation Modelling

R code Exa 17.1.1 Monte carlo sampling

```
##Chapter 17 : Simulation Modelling
##Example 1-1 : Page 681

#number of trials
n=1000
#Setting up parameters for the plot
par(pty="s")
#plot empty graph
plot(1, axes=T,asp = 1, xlim = c(-4, 6),ylim = c(-3,
    7))
#getting a sample of 1000 random uniform numbers
   within the given range
x=-4+(10)*runif(n)
y=-3+10*runif(n)
#counting the number of true values for the give
   expression
m=sum((x-1)^2+(y-2)^2<25)
#plotting the points on the graph
points(x, y)
#plotting the circle
symbols(x=1,y=2,circles = 5,add = T,inches = F)
```

```
#plotting  the square
symbols(x=1,y=2,squares =10,add = T,inches = F)
#Area of the circle
ApproxArea=m*100/n
ApproxArea
```

R code Exa 17.3.3 Erlang distribution

```
##Chapter 17 : Simulation Modelling
##Example 3-3 : Page 691

#Function which returns an erlang distribution
erlang=function(m,lambda){
  R=c(0.0589,0.6733,0.4799)
  y=-log(prod(R))/lambda
  return(y)
}
erlang(3,4)
```

R code Exa 17.4.1 Multiplicative congruential method

```
##Chapter 17 : Simulation Modelling
##Example 3-3 : Page 691

#inputs for the random number generator
b=9
c=5
m=12


u=numeric()
#initial random  number
u[1]=11
```

```
R=numeric()
for(i in 2:4){
  u[i]=(b*u[i-1]+c)%%m
  R[i-1]=u[i]/m
}
R
```

Chapter 18

Classical Optimization theory

R code Exa 18.1.1 Necessary and sufficient conditions

```r
##Chapter 18 : Classical Optimization theory
##Example 1-1 : Page 714

#Function to be given as input to optim function
minimize<-function(x){
  x1=x[1]
  x2=x[2]
  x3=x[3]
  return(-(x1+2*x2+x2*x3-x1^2-x2^2-x3^2))
}

#Calculates the optimal value
optim(c(0,0,0),minimize,hessian = T )
#the Jacobian for each
f=expression(x1+2*x2+x2*x3-x1^2-x2^2-x3^2)
for(i in 1:3){
  x=D(f,paste("x",i,sep = ""))
  print(x)
}
```

R code Exa 18.1.3 Newton Raphson method

```
##Chapter 18 : Classical Optimization theory
##Example 1-3 : Page 717

#The function expression has to be given as function
    to nlm function
g<-function(x){
  return((3*x-2)^2 * (2*x-3)^2)
}
#calling nlm function with function g and initial
   guess 0
nlm(g,0)
```