

R Textbook Companion for  
Operations Research: An Introduction  
by Hamdy A Taha<sup>1</sup>

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

May 26, 2020

<sup>1</sup>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                               | 4   |
| 2 Modelling with Linear Programming           | 5   |
| 3 The Simplex Method and Sensitivity Analysis | 19  |
| 4 Duality and Post optimality Analysis        | 30  |
| 5 Transportation Model and its Variant        | 41  |
| 6 Network Model                               | 52  |
| 7 Advanced Linear Programming                 | 71  |
| 8 Integer Linear Programming                  | 75  |
| 9 Heuristic Programming                       | 82  |
| 10 Travelling Salesman Problem                | 90  |
| 11 Deterministic Dynamic Programming          | 93  |
| 12 Deterministic Inventory Modelling          | 95  |
| 13 Decision Analysis and Games                | 100 |
| 14 Probabilistic Inventory Models             | 106 |
| 15 Markov Chain                               | 108 |

|    |                               |     |
|----|-------------------------------|-----|
| 16 | Queuing Systems               | 112 |
| 17 | Simulation Modelling          | 123 |
| 18 | Classical Optimization theory | 126 |

# List of R Codes

|           |  |    |
|-----------|--|----|
| Exa 2.2.1 | Graphical Solution to LP Problems . . . . .  | 5  |
| Exa 2.2.2 | Graphical solution of LP for Diet Problem . . . . .  | 7  |
| Exa 2.4.1 | LP solution for Bank Loan Problem . . . . .  | 9  |
| Exa 2.4.2 | LP solution for Single period Production Model . . . . .   | 10 |
| Exa 2.4.3 | LP solution for Multi period Production Inventory Model . . . . .  | 11 |
| Exa 2.4.4 | LP solution for Multi period Production smoothing Model . . . . .  | 12 |
| Exa 2.4.5 | LP solution for Bus Scheduling Problem . . . . .   | 14 |
| Exa 2.4.6 | LP solution for Urban Renewal Model Problem . . . . .  | 15 |
| Exa 2.4.7 | LP solution for Crude Oil Refining and Gasoline Blending Problem . . . . .                                 | 16 |
| Exa 3.2.1 | Graphical solution of LP with 2 variables . . . . .  | 19 |
| Exa 3.3.1 | LP solution for Reddy Mikks Problem using lpSolve . . . . .  | 21 |
| Exa 3.4.1 | LP solution using M Method . . . . .   | 22 |
| Exa 3.4.2 | LP solution using Two Phase Method . . . . .   | 23 |
| Exa 3.5.1 | Degenerate Optimal Solution . . . . .  | 24 |
| Exa 3.5.2 | Infinite Solutions to an LP . . . . .  | 25 |
| Exa 3.5.3 | Unbounded objective value . . . . .  | 25 |
| Exa 3.5.4 | Infeasible Solution Space . . . . .  | 26 |
| Exa 3.6.1 | Changing RHS for Sensitivity analysis . . . . .  | 27 |
| Exa 3.6.2 | Changing objective coefficient for Sensitivity analysis . . . . .  | 28 |
| Exa 3.6.3 | Sensitivity Analysis using TOYCO Model . . . . .   | 29 |
| Exa 4.2.1 | Solving primal and dual problem using linprog package . . . . .  | 30 |
| Exa 4.2.2 | Associated objective values for arbitrary feasible primal and dual solutions . . . . .                     | 31 |
| Exa 4.2.3 | Getting the simplex tableau at any iteration from the original data and inverse of the iteration . . . . . | 31 |
| Exa 4.3.1 | Economic interpretation of Dual variables of Reddy Mikks Problem . . . . .                                 | 32 |

|           |  |    |
|-----------|--|----|
| Exa 4.3.2 | Economic interpretation of Dual constraints of Reddy Mikks Problem . . . . . | 33 |
| Exa 4.4.1 | Dual Simplex using glpk library . . . . .                                    | 34 |
| Exa 4.5.1 | Changes in RHS . . . . .   | 37 |
| Exa 4.5.2 | Addition of a new constraint . . . . .                                       | 38 |
| Exa 4.5.3 | Changes in objective coefficient . . . . .                                   | 38 |
| Exa 4.5.4 | Addition of a new activity . . . . .   | 40 |
| Exa 5.1.1 | A general transportation model . . . . .                                     | 41 |
| Exa 5.1.2 | Balancing transportation models . . . . .                                    | 42 |
| Exa 5.2.1 | Production inventory control . . . . .                                       | 43 |
| Exa 5.2.2 | Tool sharpening . . . . .  | 44 |
| Exa 5.3.1 | Sunray Transport . . . . .   | 45 |
| Exa 5.3.2 | Northwest corner method . . . . .  | 45 |
| Exa 5.3.3 | Least cost method . . . . .  | 46 |
| Exa 5.3.5 | Method of multipliers . . . . .  | 48 |
| Exa 5.4.1 | Hungarian method 1 . . . . .   | 49 |
| Exa 5.4.2 | Hungarian method 2 . . . . .   | 50 |
| Exa 6.2.1 | Minimal spanning tree algorithm . . . . .                                    | 52 |
| Exa 6.3.1 | Equipment replacement as shortest route problem . .                          | 53 |
| Exa 6.3.2 | Most reliable route . . . . .  | 55 |
| Exa 6.3.3 | Three jub puzzle . . . . .   | 56 |
| Exa 6.3.4 | Djiktras algorithm . . . . .   | 57 |
| Exa 6.3.5 | Floyds algorithm . . . . .   | 59 |
| Exa 6.3.6 | LP formulation and solution of shortest route problem                        | 60 |
| Exa 6.4.2 | Maximal flow algorithm . . . . .   | 61 |
| Exa 6.4.3 | LP formulation and solution of Maximal flow Mode .                           | 62 |
| Exa 6.5.1 | Network representation for PERT and CPM . . . . .                            | 64 |
| Exa 6.5.2 | Critical path method . . . . .   | 65 |
| Exa 6.5.3 | Preliminary schedule . . . . .   | 66 |
| Exa 6.5.4 | Determination of floats and red flag rule . . . . .                          | 67 |
| Exa 6.5.5 | LP formulation and solution of project scheduling problem . . . . .          | 69 |
| Exa 7.1.2 | All basic feasible and infeasible solutions of an equation                   | 71 |
| Exa 7.1.3 | Simmplex tableau in matrix form . . . . .                                    | 72 |
| Exa 7.2.1 | Revised simplex algoithmr . . . . .  | 73 |
| Exa 7.3.1 | Bounded variables algorithm . . . . .  | 73 |
| Exa 7.4.1 | Dual simplex algorithm . . . . .   | 74 |
| Exa 8.1.1 | Project selection . . . . .  | 75 |

|            |   |     |
|------------|---|-----|
| Exa 8.1.2  | Installing security phones . . . . .  | 76  |
| Exa 8.1.3  | Choosing a telephone company . . . . .  | 77  |
| Exa 8.1.4  | Job scheeduling model . . . . .   | 78  |
| Exa 8.2.1  | Branch and bound algorithm . . . . .  | 80  |
| Exa 8.2.2  | Cutting plane algorithm . . . . .   | 80  |
| Exa 9.2.1  | Discrete variable heuristics . . . . .  | 82  |
| Exa 9.2.2  | Random walk heuristic . . . . .   | 83  |
| Exa 9.2.3  | Random walk heuristic for continuous variables . . . . .                                  | 83  |
| Exa 9.3.1  | Minimization of single varibale function using tabu search<br>algorithm . . . . .         | 85  |
| Exa 9.3.3  | Minimization of single varibale function using simulated<br>annealing algorithm . . . . . | 87  |
| Exa 9.3.4  | Job scheduling using simulated annealing algorithm . . . . .                              | 87  |
| Exa 10.3.1 | Travelling Salesman Problem using branch and bound<br>algorithm . . . . .                 | 90  |
| Exa 10.3.2 | Travelling Salesman Problem using cutting plane algo-<br>rithm . . . . .                  | 90  |
| Exa 10.4.1 | Travelling Salesman Problem using nearest neighbour<br>heuristic . . . . .                | 91  |
| Exa 11.1.1 | Shortest route problem using dynamic programming . . . . .                                | 93  |
| Exa 12.2.1 | Analyzing the nature of demand . . . . .  | 95  |
| Exa 12.3.1 | Classic EOQ model problem . . . . .   | 96  |
| Exa 12.3.2 | Optimal order policy . . . . .  | 96  |
| Exa 12.4.1 | Dynamic EOQ models with no setup . . . . .  | 97  |
| Exa 13.1.1 | Overall Idea of AHP . . . . .   | 100 |
| Exa 13.1.2 | Weights and consistency for AHP . . . . .   | 101 |
| Exa 13.1.3 | Consistency ratio . . . . .   | 101 |
| Exa 13.2.1 | Decision tree . . . . .   | 102 |
| Exa 13.2.2 | Expected value criterion . . . . .  | 102 |
| Exa 13.3.1 | Decision under uncertainty . . . . .  | 103 |
| Exa 13.4.1 | Two person zero sum game . . . . .  | 104 |
| Exa 13.4.3 | Mixed strategy games . . . . .  | 104 |
| Exa 14.1.1 | Probabilistic Inventory Models with normal distribution<br>of demand . . . . .            | 106 |
| Exa 14.2.1 | Newsvendor problem . . . . .  | 107 |
| Exa 15.2.1 | Absolute probabilities after transitions . . . . .  | 108 |
| Exa 15.3.1 | Absorbing and transient states . . . . .  | 109 |
| Exa 15.3.2 | Periodic states . . . . .   | 109 |



|   |     |
|---|-----|
| Exa 15.4.1 Steady state probabilities . . . . .                         | 110 |
| Exa 15.4.2 Cost model . . . . .   | 110 |
| Exa 15.5.1 Mean first passage time . . . . .                            | 111 |
| Exa 15.6.1 Analysis of absorbing states . . . . .                       | 111 |
| Exa 16.4.1 Pure birth model . . . . .                                   | 112 |
| Exa 16.4.2 Pure death model . . . . .                                   | 112 |
| Exa 16.6.1 Measures of performance . . . . .                            | 113 |
| Exa 16.6.2 MM1 GDInfInf model . . . . .                                 | 115 |
| Exa 16.6.4 MM1 GDNInf model . . . . .                                   | 116 |
| Exa 16.6.5 MMc GDInfInf model . . . . .                                 | 116 |
| Exa 16.6.6 MMc GDNInf model . . . . .                                   | 117 |
| Exa 16.6.7 MMInf GDInfNEInf or Slef service models . . . . .            | 117 |
| Exa 16.6.8 MMR GDKK or Machine servicing model . . . . .                | 118 |
| Exa 16.7.1 MG1 GDInfInf model or Pollaczek Khintchine formula . . . . . | 119 |
| Exa 16.9.1 Cost models . . . . .  | 120 |
| Exa 16.9.2 Practical problem for MMc GDInfInf model . . . . .           | 121 |
| Exa 16.9.3 Aspiration level model . . . . .                             | 121 |
| Exa 17.1.1 Monte carlo sampling . . . . .                               | 123 |
| Exa 17.3.3 Erlang distribution . . . . .                                | 124 |
| Exa 17.4.1 Multiplicative congruential method . . . . .                 | 124 |
| Exa 18.1.1 Necessary and sufficient conditions . . . . .                | 126 |
| Exa 18.1.3 Newton Raphson method . . . . .                              | 127 |

## Chapter 2

# Modelling with Linear Programming

R code Exa 2.2.1 Graphical Solution to LP Problems

```
1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 2-1 : Page 16
3
4 #To plot the line , we have to consider them as
   equation instead of inequality and express /
5 #them in terms of x2 :
6
7 #Constraint 1 :  $6 * x1 + 4 * x2 \leq 24$ 
8 #Con1          :  $x2 = (24 - 6 * x1) / 4$ 
9 con1 <- function(x1) (24 - 6 * x1) / 4
10 plot (con1, xlab = "x1", ylab = "x2", xlim = c(0,7),
       ylim = c(0,7), col = "red",
11       main = "Example 2-1", yaxs = "i", xaxs = "i")
12 #xlab & ylab   : x and y label respectively
13 #xlim          : limits of x value on the plot
14 #ylim          : limits of y value on the plot
15 #col           : color of the line
16 #main          : Title of the plot
17 #yaxs & xaxs   : the style of axis interval
```

```

    calculation to be used by R. The default
18 #value is a 4% gap at each end of axis
19
20 #Constraint 2 :  $x_1 + 2 * x_2 \leq 6$ 
21 #Con2 :  $x_2 = (6 - x_1)/2$ 
22 con2 <- function(x1) (6 - x1)/2
23 plot (con2, add=T, xlim = c(0,7), ylim = c(0,7), col
      = "blue")
24 #add          : adds to an existing plot
25
26
27 #Constraint 3 :  $-x_1 + x_2 \leq 1$ 
28 #Con3          :  $x_2 = (1 + x_1)$ 
29 con3 <- function(x1) (1 + x1)
30 plot (con3, add=T, xlim = c(0,7), ylim = c(0,7), col
      = "green")
31
32 #Constraint 4 :  $x_2 \leq 2$ 
33 #Con4          :  $x_2 = 2$ 
34 con4 <- function(x1) (2 + 0*x1)
35 plot (con4, add=T, xlim = c(0,7), ylim = c(0,7), col
      = "green")
36 #h            : horizontal line at y=2
37
38 #Points of intersections of constraints : (0,1)
      ,(1,2),(2,2),(2,1.5),(4,0)
39 points(c(0,1,2,3,4),c(1,2,2,1.5,0))
40
41 #Add a shaded area
42 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),
43         border=NA)
44 #border        : option to add border to the shaded
      area
45
46 #Adding "solution space" text to the shaded area at
      (2,1)
47 text(2,1,"Solution \nSpace")

```

```

48
49 ##Objective function :Max 5 *x1 + 4 * x2
50 # maximum objective is 21, Therefore 5 *x1 + 4 * x2
    =21
51 # Obj          : x2 = (21 - 5*x1)/4
52 Obj <- function(x1) (21 - 5*x1)/4
53 plot (Obj, add=T, xlim = c(0,7), ylim = c(0,7), lty
    =2 )
54 #lty          : option to set the type of line ,2 for
    dashed line
55
56 text(3,3,"x1 = 3 \nx2 = 1.5 \n z=21")

```

---

### R code Exa 2.2.2 Graphical solution of LP for Diet Problem

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 2-2 : Page 24
3
4 #To plot the line , we have to consider them as
    equation instead of inequality and express/
5 #them in terms of x2 :
6
7 #Constraint 1 : x1 + x2 <= 800
8 #Con1          : x2 = (800 - x1)
9 con1 <- function(x1) (800 - x1)
10 plot (con1, xlab = "x1", ylab = "x2", xlim = c
    (0,1500), ylim = c(0,1500), col = "red",
11     main = "Example 2-2", yaxs="i", xaxs = "i")
12 #xlab & ylab : x and y label respectively
13 #xlim       : limits of x value on the plot
14 #ylim       : limits of y value on the plot
15 #col        : color of the line
16 #main       : Title of the plot
17 #yaxs & xaxs : the style of axis interval
    calculation to be used by R. The default

```

```

18 #value is a 4% gap at each end of axis
19
20 #Constraint 2 :  $0.21 * x1 - 0.3 * x2 \leq 0$ 
21 #Con2 :  $x2 = (0.21 * x1)/0.3$ 
22 con2 <- function(x1) (0.21 * x1)/0.3
23 plot (con2, add=T, xlim = c(0,1500), ylim = c
      (0,1500), col = "blue")
24 #add          : adds to an existing plot
25
26
27 #Constraint 3 :  $0.03 * x1 - 0.01 * x2 \geq 0$ 
28 #Con3 :  $x2 = (0.03 * x1)/0.01$ 
29 con3 <- function(x1) (0.03 * x1)/0.01
30 plot (con3, add=T, xlim = c(0,1500), ylim = c
      (0,1500), col = "green")
31
32 #Points of intersections of constraints : (0,1)
      ,(1,2),(2,2),(2,1.5),(4,0)
33 points(c(470.6,200),c(329.4,600))
34
35 #Add a shaded area
36 polygon(c(470.6,200,500,15000/7),c
      (329.4,600,1500,1500), col = rgb(0.48, 0.46,
      0.46, 0.5),
37         border=NA)
38 #border : option to add border to the shaded area
39
40 #Adding "solution space" text to the shaded area at
      (750,1000)
41 text(750,1000,"Solution \nSpace")
42
43 ##Objective function :  $\text{Min } 0.3 * x1 + 0.9 * x2$ 
44 # Minimum objective is 437.64, Therefore  $0.3 * x1 +$ 
       $0.9 * x2 = 437.64$ 
45 # Obj :  $x2 = (437.64 - 0.3 * x1)/0.9$ 
46 Obj <- function(x1) (437.64 - 0.3 * x1)/0.9
47 plot (Obj, add=T, xlim = c(0,1500), ylim = c(0,1500)
      , lty =2 )

```

```

48 #lty      : option to set the type of line ,2 for
    dashed line
49
50 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

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-1 : Page 35
3
4 # Objective function :Max (0.126 * x1 +0.1209 * x2 +
    0.1164 * x3 + 0.11875 * x4 + 0.098 * x5)
5 #
    - (0.1 * x1 + 0.07 * x2 + 0.03
    * x3 + 0.05 * x4 + 0.02 * x5)
6 a <- c
    (0.126-0.1,0.1209-0.07,0.1164-0.03,0.11875-0.05,0.098-0.02)

7
8 # Constraint 1 : x1 + x2 + x3 + x4 + x5 <=12
9 C1 <- c(1,1,1,1,1)
10 bc1<- 12
11
12 # Constraint 2 : 0.4*x1 + 0.4*x2 + 0.4*x3 - 0.6*x4 -
    0.6*x5 <=0
13 C2 <- c(0.4,0.4,0.4,-0.6,-0.6)
14 bc2<-0
15
16 # Constraint 3 : 0.5 * x1 + 0.5 * x2 - 0.5 * x3 <=0
17 C3 <- c(0.5,0.5,-0.5,0,0)
18 bc3<-0
19
20 # Constraint 4 : 0.06 * x1 + 0.03 * x2 - 0.01 * x3
    +0.01 * x4 -0.02 * x5 <=0
21 C4 <- c(0.06,0.03,-0.01,0.01,-0.02)
22 bc4<-0

```

```

23
24 library("boot")
25 ##Simplex is a function which uses dual simplex
    method
26 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

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-2 : Page 41
3
4 # Objective function :Max (30 * x1 + 40 * x2 + 20 *
    x3 + 10 * x4)
5 #                               - (15 * s1 + 20 * s2 + 10 * s3
    + 8 * s4)
6 a <- c(30,40,20,10,-15,-20,-10,-8)
7
8 # Constraint 1 : 0.3 * x1 + 0.3 * x2 + 0.25 * x3 +
    0.15 * x4 <= 1000
9 C1 <- c(0.3,0.3,0.25,0.15,0,0,0,0)
10 bc1<- 1000
11
12 # Constraint 2 : 0.25 * x1 + 0.35 * x2 + 0.3 * x3 +
    0.1 * x4 <= 1000
13 C2 <- c(0.25,0.35,0.3,0.1,0,0,0,0)
14 bc2<-1000
15
16 # Constraint 3 : 0.45 * x1 + 0.5 * x2 + 0.4 * x3 +
    0.22 * x4 <=1000
17 C3 <- c(0.45,0.5,0.4,0.22,0,0,0,0)
18 bc3<-1000
19
20 # Constraint 4 : 0.15 * x1 + 0.15 * x2 + 0.1 * x3 +
    0.05 * x4 <=0

```

```

21 C4 <- c(0.15,0.15,0.1,0.05,0,0,0,0)
22 bc4<-1000
23
24 # Constraint 5 : x1 + s1 = 800
25 C5 <- c(1,0,0,0,1,0,0,0)
26 bc5<-800
27
28 # Constraint 6 : x2 + s2 = 750
29 C6 <- c(0,1,0,0,0,1,0,0)
30 bc6<-750
31
32 # Constraint 7: x3 + s3 = 600
33 C7 <- c(0,0,1,0,0,0,1,0)
34 bc7<-600
35
36 # Constraint 8 : x4 + s4 = 500
37 C8 <- c(0,0,0,1,0,0,0,1)
38 bc8<-500
39
40 library("boot")
41
42 simplex(a , A1 = rbind(C1, C2, C3, C4), b1 = c(bc1,
      bc2, bc3, bc4),
43       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

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-3 : Page 41
3
4 # Objective function :Min (50 * x1 + 45 * x2 + 55 *
      x3 + 48 * x4 + 52 * x5 + 50 * x6)
5 #                               + 8(I1 +I2 +I3 +I4 + I5 + I6)
6 a <- c(50,45,55,48,52,50,8,8,8,8,8,8)

```



```

7
8 # Constraint 1 :  $x_1 - I_1 = 100$ 
9 C1 <- c(1,0,0,0,0,0,-1,0,0,0,0,0)
10 bc1<- 100
11
12 # Constraint 2 :  $I_1 + x_2 - I_2 = 250$ 
13 C2 <- c(0,1,0,0,0,0,1,-1,0,0,0,0)
14 bc2<-250
15
16 # Constraint 3 :  $I_2 + x_3 - I_3 = 190$ 
17 C3 <- c(0,0,1,0,0,0,0,1,-1,0,0,0)
18 bc3<-190
19
20 # Constraint 4 :  $I_3 + x_4 - I_4 = 140$ 
21 C4 <- c(0,0,0,1,0,0,0,0,1,-1,0,0)
22 bc4<-140
23
24 # Constraint 5 :  $I_4 + x_5 - I_5 = 220$ 
25 C5 <- c(0,0,0,0,1,0,0,0,0,1,-1,0)
26 bc5<-220
27
28 # Constraint 6 :  $I_5 + x_6 = 110$ 
29 C6 <- c(0,0,0,0,0,1,0,0,0,0,1,0)
30 bc6<-110
31
32 library("boot")
33
34 simplex(a , A1=NULL,b1=NULL , A2=NULL , b2=NULL , A3 =
      rbind(C1, C2, C3, C4,C5,C6),
35          b3 = c(bc1, bc2, bc3, bc4,bc5,bc6), maxi = F
      )

```

---

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

1 **##Chapter 2 : Modelling with Linear Programming**

```

2  ##Example 4-4 : Page 43
3
4  # Objective function :Min 50(I1 + I2 + I3 + I4) +
    200(Sm1 +Sm2 +Sm3 +Sm4) + 400(Sp1 +Sp2 +Sp3 +Sp4)
5  a <- c
    (0,0,0,0,50,50,50,50,200,200,200,200,400,400,400,400)

6
7  # Constraint 1 : 10*x1 -I1 = 400
8  C1 <- c(10,0,0,0,-1,0,0,0,0,0,0,0,0,0,0,0)
9  bc1<- 400
10
11 # Constraint 2 : I1 + 10 * x2 - I2 = 600
12 C2 <- c(0,10,0,0,1,-1,0,0,0,0,0,0,0,0,0,0)
13 bc2<-600
14
15 # Constraint 3 : I2 + 10 * x3 - I3 = 400
16 C3 <- c(0,0,10,0,0,0,1,-1,0,0,0,0,0,0,0,0)
17 bc3<-400
18
19 # Constraint 4 : I3 + 10 * x4 = 500
20 C4 <- c(0,0,0,10,0,0,0,1,0,0,0,0,0,0,0,0)
21 bc4<-500
22
23 # Constraint 5 : x1 -Sm1 + Sp1 = 0
24 C5 <- c(1,0,0,0,0,0,0,0,-1,0,0,0,1,0,0,0)
25 bc5<-0
26
27 # Constraint 6 : x2 -x1 -Sm2 +Sp2 = 0
28 C6 <- c(-1,1,0,0,0,0,0,0,0,-1,0,0,0,1,0,0)
29 bc6<-0
30
31 # Constraint 7 : x3 -x2 -Sm3 +Sp3 = 0
32 C7 <- c(0,-1,1,0,0,0,0,0,0,0,-1,0,0,0,1,0)
33 bc7<-0
34
35 # Constraint 8 : x4 -x3 -Sm4 +Sp4 = 0
36 C8 <- c(0,0,-1,1,0,0,0,0,0,0,0,-1,0,0,0,1)

```

```

37 bc8<-0
38
39 library("boot")
40
41 simplex(a , A1=NULL,b1=NULL , A2=NULL , b2=NULL , A3 =
      rbind(C1, C2, C3, C4, C5, C6, C7, C8),
42         b3 = c(bc1, bc2, bc3, bc4, bc5, bc6, bc7,
                  bc8), maxi = F)

```

---

### R code Exa 2.4.5 LP solution for Bus Scheduling Problem

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-5 : Page 50
3
4 # Objective function :Min x1 + x2 + x3 + x4 + x5 +
  x6
5 a <- c(1,1,1,1,1,1)
6
7 # Constraint 1 : x1 + x6 >= 4
8 C1 <- c(1,0,0,0,0,1)
9 bc1<- 4
10
11 # Constraint 2 : x1 + x2 >= 8
12 C2 <- c(1,1,0,0,0,0)
13 bc2<-8
14
15 # Constraint 3 : x2 + x3 >= 10
16 C3 <- c(0,1,1,0,0,0)
17 bc3<-10
18
19 # Constraint 4 : x3 + x4 >= 7
20 C4 <- c(0,0,1,1,0,0)
21 bc4<-7
22
23 # Constraint 5 : x4 + x5 >= 12

```

```

24 C5 <- c(0,0,0,1,1,0)
25 bc5<-12
26
27 # Constraint 6 : x5 + x6 >= 4
28 C6 <- c(0,0,0,0,1,1)
29 bc6<-4
30
31 library("lpSolve")
32
33 solution=lp("min",a, rbind(C1, C2, C3, C4, C5, C6),
    rep(">=",6),c(bc1, bc2, bc3, bc4, bc5, bc6))
34
35 solution
36 solution$solution

```

---

#### R code Exa 2.4.6 LP solution for Urban Renewal Model Problem

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-6 : Page 52
3
4 # Objective function :Max 1000 * x1 + 1900 * x2 +
    2700 * x3 + 3400 * x4
5 a <- c(1000, 1900, 2700, 3400, 0)
6
7 # Constraint 1 : 0.18 * x1 + 0.28 * x2 + 0.4 * x3 +
    0.5 * x4 -0.2125 * x5 <= 0
8 C1 <- c(0.18, 0.28, 0.4, 0.5, -0.2125)
9 bc1<- 0
10
11 # Constraint 2 : x5 <= 300
12 C2 <- c(0,0,0,0,1)
13 bc2<-300
14
15 # Constraint 3 : -0.8 * x1 + 0.2 * x2 + 0.2 * x3 +
    0.2 * x4 <= 0

```

```

16 C3 <- c(-0.8, 0.2, 0.2, 0.2, 0)
17 bc3<-0
18
19 # Constraint 4 : 0.1 * x1 - 0.9 * x2 + 0.1 * x3 +
    0.1 * x4 <= 0
20 C4 <- c(0.1, -0.9, 0.1, 0.1, 0)
21 bc4<-7
22
23 # Constraint 5 : 0.25 * x1 + 0.25 * x2 - 0.75 * x3 -
    0.75 * x4 <= 0
24 C5 <- c(0.25,0.25,-0.75,-0.75,0)
25 bc5<-0
26
27 # Constraint 6 : 50 * x1 + 70 * x2 + 130 * x3 + 160
    * x4 + 2* x5 <= 15000
28 C6 <- c(50, 70, 130, 160, 2)
29 bc6<-15000
30
31 #to install the lpSolve package,run the following
    command
32 #install.packages("lpSolve")
33 library("lpSolve")
34
35 solution = lp("max", a, rbind(C1, C2, C3, C4, C5, C6
    ), rep("<=" ,6),
36           c(bc1, bc2, bc3, bc4, bc5, bc6))
37 solution
38 solution$solution

```

---

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

```

1 ##Chapter 2 : Modelling with Linear Programming
2 ##Example 4-7 : Page 57
3

```

```

4 # Objective function :Max 6.7 * (x11 +x21) + 7.2 * (
    x12 + x22) + 8.1 * (x13 + x23)
5 a <- c(6.7, 7.2, 8.1, 6.7, 7.2, 8.1)
6
7 # Constraint 1 : 5 *(x11 + x12 + x13) + 10 * (x21 +
    x22 + x23) <= 1500000
8 C1 <- c(5, 5, 5, 10, 10, 10)
9 bc1<- 1500000
10
11 # Constraint 2 : 2 * (x21 + x22 + x23) <=200000
12 C2 <- c(0, 0, 0, 2, 2, 2)
13 bc2<-200000
14
15 # Constraint 3 : (x11 + x21) <= 50000
16 C3 <- c(1,0,0,1,0,0)
17 bc3<-50000
18
19 # Constraint 4 : (x12 + x22) <= 30000
20 C4 <- c(0, 1, 0, 0, 1, 0)
21 bc4<-30000
22
23 # Constraint 5 : (x13 + x23) <= 40000
24 C5 <- c(0, 0, 1, 0, 0, 1)
25 bc5<-40000
26
27 # Constraint 6 : 5 * x11 - 11 * x21 <= 0
28 C6 <- c(5,0,0,-11,0,0)
29 bc6<-0
30
31 # Constraint 7 : 7* x12 - 9 * x22 <= 0
32 C7 <- c(0, 7, 0, 0, -9, 0)
33 bc7<-0
34 # Constraint 8 : 10 * x12 - 6 * x22 <= 0
35 C8 <- c(0, 0, 10, 0, 0, -6)
36 bc8<-0
37
38 #to install the lpSolve package ,run the following
    command

```

```
39 #install.packages("lpSolve")
40 library("lpSolve")
41
42 solution = lp("max", a, rbind(C1, C2, C3, C4, C5, C6
    , C7, C8), rep("<=",8),
43               c(bc1, bc2, bc3, bc4, bc5, bc6, bc7,
                  bc8))
44 solution
45 solution$solution
```

---

## Chapter 3

# The Simplex Method and Sensitivity Analysis

**R code Exa 3.2.1** Graphical solution of LP with 2 variables

```
1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 2-1 : Page 83
3
4 #To plot the line , we have to consider them as
  equation instead of inequality and express/
5 #them in terms of x2 :
6
7 #Constraint 1 :  $2 * x1 + x2 \leq 4$ 
8 #Con1          :  $x2 = (4 - 2 * x1)$ 
9 con1 <- function(x1) (4 - 2 * x1)
10 plot (con1, xlab = "x1", ylab = "x2", xlim = c(0,6),
      ylim = c(0,5), col = "red",
11       main = "Example 2-1", yaxs= "i", xaxs = "i")
12 #xlab & ylab   : x and y label respectively
13 #xlim          : limits of x value on the plot
14 #ylim          : limits of y value on the plot
15 #col           : color of the line
16 #main          : Title of the plot
```



```

17 #yaxis & xaxis : the style of axis interval
    calculation to be used by R. The default
18 #value is a 4% gap at each end of axis
19
20 #Constraint 2 :  $x_1 + 2 * x_2 \leq 5$ 
21 #Con2 :  $x_2 = (5 - x_1)/2$ 
22 con2 <- function(x1) (5 - x1)/2
23 plot (con2, add=T, xlim = c(0,6), ylim = c(0,5), col
    = "blue")
24 #add : adds to an existing plot
25
26 #Points of intersections of constraints and the axis
    : (0,0),(0,2.5),(1,2),(2,0),(5,0),(0,4)
27 points(c(0,0,1,2,5,0),c(0,2.5,2,0,0,4))
28
29 #Inserting texts to name the points
30 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)
31 #First two arguments are x(+0.1) and y(+0.2)
    coordinates
32 #LETTERS : a stored array variable of all
    alphabets
33 #cex : ratio of modification to font size
34
35 #Add a shaded area
36 polygon(c(0,0,1,2),c(0,2.5,2,0), col = rgb(0.48,
    0.46, 0.46, 0.5),
37         border=NA)
38 #border : option to add border to the shaded
    area
39
40 #Adding "solution space" text to the shaded area at
    (0.7,1)
41 text(0.7,1,"Solution \nSpace",cex = 0.9)
42
43 ##Objective function :  $\text{Max } 2 * x_1 + 3 * x_2$ 
44 # maximum objective is 8, Therefore  $2 * x_1 + 3 * x_2 =$ 
    8

```

```

45 # Obj          :  $x_2 = (8 - 2*x_1)/3$ 
46 Obj <- function(x1) (8 - 2*x1)/3
47 plot (Obj, add=T, xlim = c(0,6), ylim = c(0,5), lty
      =2 )
48 #lty           : option to set the type of line ,2 for
      dashed line
49
50 #Adding text at (3,3)
51 text(3,3,"x1 = 1 \nx2 = 2 \n z = 8")

```

---

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

```

1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 3-1 : Page 89
3
4 # Objective function :Max  $5*x_1 + 4*x_2 + 0*s_1 + 0*s_2$ 
  +  $0*s_3 + 0*s_4$ 
5 a <- c(5,4,0,0,0,0)
6
7 # Constraint 1 :  $6*x_1 + 4*x_2 + s_1 = 24$ 
8 C1 <- c(6,4,1,0,0,0)
9 bc1<- 24
10
11 # Constraint 2 :  $x_1 + 2*x_2 + s_2 = 6$ 
12 C2 <- c(1,2,0,1,0,0)
13 bc2<-6
14
15 # Constraint 3 :  $-x_1 + x_2 + s_3 = 1$ 
16 C3 <- c(-1,1,0,0,1,0)
17 bc3<-1
18
19 # Constraint 4 :  $1*x_2 + s_4 = 2$ 
20 C4 <- c(0,1,0,0,0,1)
21 bc4<-2

```

```

22
23 library("lpSolve")
24
25 solution=lp("max",a, rbind(C1, C2, C3, C4), rep("=",
      ,4),c(bc1, bc2, bc3, bc4))
26
27 solution
28 solution$solution

```

---

### R code Exa 3.4.1 LP solution using M Method

```

1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 4-1 : Page 100
3
4 # Objective function :Min 4 * x1 + x2 + M * R1 + M *
  R2
5 BigM <- 1000
6 a <- c(4,1,0,0,BigM,BigM)
7
8 # Constraint 1 : 3*x1 + x2 + R1 = 3
9 C1 <- c(3,1,0,0,1,0)
10 bc1<- 3
11
12 # Constraint 2 : 4*x1 + 3*x2 -x3 + R2 >= 6
13 C2 <- c(4,3,-1,0,0,1)
14 bc2<-6
15
16 # Constraint 3 : x1 + 2*x2 +x4 <=4
17 C3 <- c(1,2,0,1,0,0)
18 bc3<-4
19
20 library("boot")
21
22 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

```
1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 4-2 : Page 105
3
4 # Objective function :Min 4 * x1 + x2
5 a <- c(4,1)
6
7 # Constraint 1 : 3*x1 + x2 = 3
8 C1 <- c(3,1)
9 bc1<- 3
10
11 # Constraint 2 : 4*x1 + 3*x2 >= 6
12 C2 <- c(4,3)
13 bc2<-6
14
15 # Constraint 3 : x1 + 2*x2 <=4
16 C3 <- c(1,2)
17 bc3<-4
18
19 library("boot")
20
21 simplex(a,C3,bc3,C2,bc2, C1,bc1 ,maxi = F)
22
23 #The method employed by this function is the two
  phase tableau simplex method.
24 #If there are >= or equality constraints an initial
  feasible solution is not
25 #easy to find. To find a feasible solution an
  artificial variable is introduced
26 #into each >= or equality constraint and an
  auxiliary objective function is defined
```

```

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

```

---

### R code Exa 3.5.1 Degenerate Optimal Solution

```

1 ##Chapter 3 : The Simplex Method and Sensitivity
    Analysis
2 ##Example 5-1 : Page 108
3
4 # Objective function :Max 3 * x1 + 9 * x2
5 a <- c(3,9)
6
7 # Constraint 1 : x1 + 4 * x2 <= 8
8 C1 <- c(1,4)
9 bc1<- 8
10
11 # Constraint 2 : x1 + 2*x2 <= 4
12 C2 <- c(1,2)
13 bc2<-4
14
15
16 library("boot")
17
18 solution=simplex(a,rbind(C1,C2),c(bc1,bc2) ,maxi = T
    )

```

```
19 ##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

```
1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 5-2 : Page 112
3
4 # Objective function :Max 2 * x1 + 4 * x2
5 a <- c(2,4)
6
7 # Constraint 1 : 1 * x1 + 2 * x2 <= 5
8 C1 <- c(1,2)
9 bc1<- 5
10
11 # Constraint 2 : x1 + x2 <= 4
12 C2 <- c(1,1)
13 bc2<-4
14
15 library("boot")
16
17 simplex(a,rbind(C1,C2),c(bc1,bc2) ,maxi = T)
18 #The simplex function as well as lpsolve stops as
    soon as a feasible optima is obtained and doesn't
    evaluate
19 #alternate optima
```

---

### R code Exa 3.5.3 Unbounded objective value

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

```

3
4 # Objective function :Max 2 * x1 + x2
5 a <- c(2,1)
6
7 # Constraint 1 : x1 - x2 <= 10
8 C1 <- c(1,-1)
9 bc1<- 10
10
11 # Constraint 2 : 2 * x1 <= 40
12 C2 <- c(2,0)
13 bc2<-40
14
15 library("lpSolve")
16
17 solution = lp("max", a, rbind(C1, C2), rep("<=" ,2),
18             c(bc1, bc2))
19 #Error: status 3 implies that the model is unbounded

```

---

#### R code Exa 3.5.4 Infeasible Solution Space

```

1 ##Chapter 3 : The Simplex Method and Sensitivity
  Analysis
2 ##Example 5-4 : Page 116
3
4 # Objective function :Max 3 * x1 + 2 * x2
5 a <- c(3,2)
6
7 # Constraint 1 : 2 * x1 + x2 <= 2
8 C1 <- c(2,1)
9 bc1<- 2
10
11 # Constraint 2 : 3 * x1 + 4 * x2 >= 12
12 C2 <- c(3,4)
13 bc2<-12
14

```

```

15 library("lpSolve")
16
17 lp("max", a, rbind(C1, C2), c("<=",">="), c(bc1, bc2
   ))
18 #Error: no feasible solution found

```

---

### R code Exa 3.6.1 Changing RHS for Sensitivity analysis

```

1 ##Chapter 3 : The Simplex Method and Sensitivity
   Analysis
2 ##Example 6-1 : Page 118
3
4 # Objective function :Max 30 * x1 + 20 * x2
5 a <- c(30,20)
6
7 # Constraint 1 : 2 * x1 + x2 <= 8
8 C1 <- c(2,1)
9 bc1<- 8
10
11 # Constraint 2 : x1 + 3 * x2 <= 8
12 C2 <- c(1,3)
13 bc2<-8
14
15 library("lpSolve")
16
17 solution=lp("max", a, rbind(C1, C2), rep("<=" ,2), c(
   bc1, bc2),compute.sens = 1)
18
19 ##Unit worth of resources in $/hr
20 solution$duals[1:2]
21
22 ##Lower limit of hours resource for respective
   machines for which dual prices are valid
23 solution$duals.from[1:2]
24

```



```

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

```

---

### R code Exa 3.6.2 Changing objective coefficient for Sensitivity analysis

```

1  ##Chapter 3 : The Simplex Method and Sensitivity
    Analysis
2  ##Example 6-2 : Page 121
3
4  # Objective function :Max 30 * x1 + 20 * x2
5  a <- c(30,20)
6
7  # Constraint 1 : 2 * x1 + x2 <= 8
8  C1 <- c(2,1)
9  bc1<- 8
10
11 # Constraint 2 : x1 + 3 * x2 <= 8
12 C2 <- c(1,3)
13 bc2<-8
14
15 library("lpSolve")
16
17 solution=lp("max", a, rbind(C1, C2), rep("<=" ,2), c(
    bc1, bc2),compute.sens = 1)
18
19 ##Lower limit of respective Objective coefficient
    for which the objective value will not change
20 solution$sens.coef.from
21 ##Upper limit of respective Objective coefficient
    for which the objective value will not change
22 solution$sens.coef.to

```

---

### R code Exa 3.6.3 Sensitivity Analysis using TOYCO Model

```
1 ##Chapter 3 : The Simplex Method and Sensitivity
   Analysis
2 ##Example 6-3 : Page 124
3
4 # Objective function :Max 3 * x1 + 2 * x2 + 5 * x3
5 a <- c(3,2,5)
6
7 # Constraint 1 :    x1 + 2 * x2 + x3 <= 430
8 C1 <- c(1,2,1)
9 bc1<- 430
10
11 # Constraint 2 : 3 * x1 + 2 * x3 <= 460
12 C2 <- c(3,0,2)
13 bc2<-460
14
15 # Constraint 3 : 1 * x1 + 4 * x2 <= 420
16 C3 <- c(1,4,0)
17 bc3<-420
18
19 library("lpSolve")
20 solution=lp("max", a, rbind(C1, C2, C3), rep("<=" ,3)
   , c(bc1, bc2, bc3),compute.sens = 1)
21
22 ##Lower limit of hours resource for respective
   machines for which dual prices are valid
23 solution$duals.from
24
25 ##Upper limit of hours resource for respective
   machines for which dual prices are valid
26 solution$duals.to
27
28 ##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 package

```
1 ##Chapter 4 : Duality and Post-optimality Analysis
2 ##Example 2-1 : Page 160
3
4 # Objective function :Max 5 * x1 + 12 * x2 + 4 * x3
5 a <- c(5,12,4)
6
7 # Constraint 1 : x1 + 2 * x2 + x3 <= 10
8 C1 <- c(1,2,1)
9 bc1<- 10
10
11 # Constraint 2 : 2 * x1 - x2 + 3 * x3 = 8
12 C2 <- c(2,-1,3)
13 bc2<-8
14
15 library("linprog")
16
17 solveLP(a,c(bc1, bc2), rbind(C1, C2), c("<=","="),
18       maximum = T,lpSolve = T)
```

```

19 ## At the moment the dual problem can not be solved
    with equality constraints in the function solveLP
20 ##Nevertheless we change the equality constraint to
    two inequality constraints. i.e.
21 ##  $f_1(x)=b \implies f_1(x) \leq b$  ,  $f_1(x) \geq b$ 
22 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

```

1 ##Chapter 4 : Duality and Post-optimality Analysis
2 ##Example 2-2 : Page 162
3
4 ##Returns primal objective value
5 PrimalObj<-function(x1,x2,x3){
6     return(5*x1+12*x2+4*x3)
7 }
8
9 #Returns dual objective value
10 DualObj<-function(y1,y2){
11     return(10*y1+8*y2)
12 }
13
14 ##Calling primal and dual functions with the
    arbitrary feasible solutions
15 PrimalObj(0,0,8/3)
16 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

```

1 ##Chapter 4 : Duality and Post-optimality Analysis
2 ##Example 2-3 : Page 166
3
4 #Optimal Inverse
5 OptimalInv=matrix(c(2/5,-1/5,1/5,2/5),nrow=2,byrow=T
6 )
7 #Original X1 column
8 OrigX1=c(1,2)
9
10 ##Optimal X1 column using formula 1
11 X1Optimal=OptimalInv%*%OrigX1
12 X1Optimal
13
14 ##Functions to calculate Z coefficients of X1 and R
15 ZCoefX1<-function(y1,y2) {return(y1+2*y2-5)}
16 ZCoefR<-function(y1,y2) {return(paste(y2,"+ M"))}
17
18 ## Calling the function with optimal dual values
19 ZCoefX1(29/5,-2/5)
20 ZCoefR(29/5,-2/5)

```

---

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

```

1 ##Chapter 4 : Duality and Post-optimality Analysis
2 ##Example 2-1 : Page 171
3
4 # Objective function :Max 5 * x1 + 4 * x2
5 a <- c(5,4)
6
7 # Constraint 1 : 6 * x1 + 4 * x2 <= 24
8 C1 <- c(6,4)
9 bc1<- 24
10

```

```

11 # Constraint 2 :  $x_1 + 2 * x_2 \leq 6$ 
12 C2 <- c(1,2)
13 bc2<-6
14
15 # Constraint 3 :  $-x_1 + x_2 \leq 1$ 
16 C3 <- c(-1,1)
17 bc3<-1
18
19 # Constraint 4 :  $x_2 \leq 2$ 
20 C4 <- c(0,1)
21 bc4<-2
22
23 library("linprog")
24
25 solveLP(a,c(bc1, bc2, bc3, bc4), rbind(C1, C2, C3,
      C4), rep("<=",4), maximum = T,lpSolve = T)
26
27 ##solve.dual arguement can be passed to solveLP
      function to solve the dual of the problem
28 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

```

1 ##Chapter 4 : Duality and Post-optimality Analysis
2 ##Example 3-2 : Page 173
3
4 # Objective function :Max  $3 * x_1 + 2 * x_2 + 5 * x_3$ 
5 a <- c(3,2,5)
6
7 # Constraint 1 :  $x_1 + 2 * x_2 + x_3 \leq 430$ 
8 C1 <- c(1,2,1)
9 bc1<- 430

```

```

10
11 # Constraint 2 : 3 * x1 + 2 * x3 <= 460
12 C2 <- c(3,0,2)
13 bc2<-460
14
15 # Constraint 3 : 1 * x1 + 4 * x2 <= 420
16 C3 <- c(1,4,0)
17 bc3<-420
18
19 library("linprog")
20 solveLP(a,c(bc1, bc2, bc3), rbind(C1, C2, C3), rep("
    <=",3), maximum = T,lpSolve = T)
21
22 ##solve.dual arguement can be passed to solveLP
    function to solve the dual of the problem
23 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

```

1 ##Chapter 4 : Duality and Post-optimal Analysis
2 ##Example 2-1 : Page 175
3
4 ##Please note that the constraints are not given in
    the example(printing error) and
5 ##it has been deduced from simplex tableau
6
7 ##We will be using glpkAPI library. If you are using
    Debian system, run the following command
8 ##before running the R script :: sudo apt-get
    install libglpk-dev
9 ##For other linux and windows systems, install GLPK
    which is available at https://www.gnu.org/
    software/glpk/
10 ##or from their respective repositories

```

```

11 library(glpkAPI)
12
13 # Objective function :Min 3 * x1 + 2 * x2 + 1 * x3
14 a <- c(3,2,1)
15
16 # Constraint 1 : 3 * x1 + 1 * x2 + x3 >= 3
17 # Standardizing constraint by multiplying by -1
18 C1 <- c(-3,-1,-1)
19 bc1<- -3
20
21 # Constraint 2 : -3 * x1 + 3 * x2 + x3 >= 6
22 # Standardizing constraint by multiplying by -1
23 C2 <- c(3,-3,-1)
24 bc2<- -6
25
26 # Constraint 3 : x1 + x2 + x3 <= 3
27 C3 <- c(1,1,1)
28 bc3<-3
29
30 #upper bound vector
31 bc <- c(bc1,bc2,bc3)
32
33 #Constraint matrix
34 ConstraintMatrix <- rbind(a,C1,C2,C3)
35
36 #Initiating row and coloumn index variable as well
   as constraint coefficient value variable
37 rowindex <- numeric()
38 colindex <- numeric()
39 value <- numeric()
40
41 #initiate GLPK object and name
42 dualSimplex <- initProbGLPK()
43 setProbNameGLPK(dualSimplex, "Example 4-1")
44 setObjNameGLPK(dualSimplex, "Minimize using Dual
   Simplex")
45
46 #Setting objective direction and number of coloumns

```



```

47 setObjDirGLPK(dualSimplex, GLP_MIN)
48 addColsGLPK(dualSimplex, 3)
49
50 #setting decision variable names,bounds and
    coefficients
51 for (i in 1:3) {
52     setColsNamesGLPK(dualSimplex,i,toString(c("x",i)))
53     setColBndGLPK(dualSimplex, i, GLP_LO, 0.0, 0.0)
54     setObjCoefsGLPK(dualSimplex, i, a[i])
55 }
56
57 #add 4 rows (including the objective)
58 addRowsGLPK(dualSimplex, 4)
59
60 #set row name as objective name itself
61 setRowsNamesGLPK(dualSimplex, 1, getObjNameGLPK(
    dualSimplex))
62
63 #set row names and bounds for constraint
64 for (i in 1:3){
65     setRowsNamesGLPK(dualSimplex, i+1, toString(c("
        Constraint", i)))
66     setRowBndGLPK(dualSimplex, i+1, GLP_UP, 0, bc[i])
67 }
68
69 #initiating row and coloumn index and the values
70 counter=1
71 for (i in 1:4) {
72     for (j in 1:3) {
73         rowindex[counter] <- i
74         colindex[counter] <- j
75         value[counter] <- ConstraintMatrix[i,j]
76         counter=counter+1
77     }
78 }
79
80 #change the soving algorithm to dual simplex
81 setSimplexParmGLPK(METH, GLP_DUAL)

```

```

82
83 #shows the current solver parameters
84 getSimplexParmGLPK()
85
86 #load and initiate all the data
87 loadMatrixGLPK(dualSimplex, 12, rowindex, colindex,
    value)
88
89 #Solve
90 solveSimplexGLPK(dualSimplex)
91
92 #Prints the status, optimal objective value and
    decision variable value
93 getSolStatGLPK(dualSimplex)
94 getObjValGLPK(dualSimplex)
95 getColsPrimGLPK(dualSimplex)
96
97 #prints the summary of the optimization to your
    working directory
98 printSolGLPK(dualSimplex, 'textfile.txt')
99
100 #deletes the glpk object
101 delProbGLPK(dualSimplex)

```

---

#### R code Exa 4.5.1 Changes in RHS

```

1 ##Chapter 4 : Duality and Post-optimal Analysis
2 ##Example 5-1 : Page 182
3
4 ##Situation 1
5 Inverse <- rbind(c(0.5, -0.25, 0), c(0, 0.5, 0), c(-2, 1, 1)
    )
6 ConstraintNNewRightHandSide <- rbind(600, 640, 590)
7 TableauNewRightHandSide <- Inverse %*%
    ConstraintNNewRightHandSide

```

```

8 TableauNewRightHandSide
9
10 ##Situation 2
11 ConstraintNNewRightHandSide <-rbind(450,460,400)
12 TableauNewRightHandSide <- Inverse %*%
    ConstraintNNewRightHandSide
13 TableauNewRightHandSide

```

---

#### R code Exa 4.5.2 Addition of a new constraint

```

1 ##Chapter 4 : Duality and Post-optimal Analysis
2 ##Example 5-2 : Page 185
3
4 ##Situation 1 is a theoretical explanation
5
6 ##Situation 2
7 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 )
8
9 OldX7Row <- tableau[4,]
10 CoeffX2 <-OldX7Row[2]
11 CoeffX3 <-OldX7Row[3]
12 X2Row <- c(-0.25,1,0,0.5,-0.25,0,0,100)
13 X3Row <- c(1.5,0,1,0,1.5,0,0,230)
14 tableau[4,] <- OldX7Row - (CoeffX2 %*% tableau[1,] +
    CoeffX3 %*% tableau[2,])
15 tableau

```

---

#### R code Exa 4.5.3 Changes in objective coefficient

```

1 ##Chapter 4 : Duality and Post-optimal Analysis
2 ##Example 5-3 : Page 187

```

```

3
4 ##Situation 1
5 NewObjCoeffBasic <- c(3,4,0)
6 Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
7 )
8 NewDualVariables <-NewObjCoeffBasic %*% Inverse
9 NewDualVariables
10
11 ##Element-wise multiplication to get Reduced Costs
12 ReducedCostX1<-sum(c(1,3,1)*NewDualVariables) -2
13 ReducedCostX4<-sum(c(1,0,0)*NewDualVariables) -0
14 ReducedCostX5<-sum(c(0,1,0)*NewDualVariables) -0
15 ReducedCostX1
16 ReducedCostX4
17 ReducedCostX5
18
19
20 CurrentOptimal <- c(0,100,230)
21 NewObjCoeff<-c(2,3,4)
22
23 #Optimal Objective value
24 NewRevenue <- sum(NewObjCoeff*CurrentOptimal)
25 NewRevenue
26
27 ##Situation 2
28 NewObjCoeffBasic <- c(3,4,0)
29 Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
30 )
31 NewDualVariables <-NewObjCoeffBasic %*% Inverse
32 NewDualVariables
33
34 ##Element-wise multiplication to get Reduced Costs
35 ReducedCostX1<-sum(c(1,3,1)*NewDualVariables) -6
36 ReducedCostX4<-sum(c(1,0,0)*NewDualVariables) -0
37 ReducedCostX5<-sum(c(0,1,0)*NewDualVariables) -0
38 ReducedCostX1

```

```

39 ReducedCostX4
40 ReducedCostX5
41
42 NewObjCoeff<-c(-0.75,0,0,1.5,1.25,0)
43 C1<-c(-0.25,1,0,0.5,-0.25,0)
44 bc1<-100
45 C2<-c(1.5,0,1,0,0.5,0)
46 bc2<-230
47 C3<-c(2,0,0,-2,1,1)
48 bc3<-20
49 library("lpSolve")
50 solution=lp("max", NewObjCoeff, rbind(C1, C2, C3),
           rep("=",3), c(bc1, bc2, bc3))
51 solution

```

---

#### R code Exa 4.5.4 Addition of a new activity

```

1 ##Chapter 4 : Duality and Post-optimal Analysis
2 ##Example 5-4 : Page 189
3 OptimalDualValues<-c(1,2,0)
4
5 ReducedCostX7 <-sum(c(1,1,2)*OptimalDualValues)-4
6 ReducedCostX7
7
8 Inverse <- rbind(c(0.5,-0.25,0),c(0,0.5,0),c(-2,1,1)
9 )
10 OldX7Col<-rbind(1,1,2)
11 NewX7Col<-Inverse%*%OldX7Col
12 NewX7Col

```

---

## Chapter 5

# Transportation Model and its Variant

R code Exa 5.1.1 A general transportation model

```
1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 1-1 : Page 195
3 ##Refer to the transportation tableau in Page 197
4 costs <- matrix (c(80,215,100,108,102,68), 3, 2,
5                 byrow = T)
6
7 #Constraints 1,2 & 3 are row constraints as they
8   corresponds to rows of transportation tableau
9
10 #Constraints 4 & 5 are column constraints as they
11   corresponds to columns of transportation tableau
12
13 col.signs <-rep("=",2)
14 col.rhs <-c(2300,1400)
15
16 library(lpSolve)
17 ##lpSolve library has lp.transport function
18   specially for problems which can be formulated as
```

```

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

```

---

### R code Exa 5.1.2 Balancing transportation models

```

1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 1-2 : Page 197
3
4 ##Part1- Adding a dummy origin
5 costs <- matrix (c(80,215,100,108,102,68,0,0), 4, 2,
        byrow = T)
6
7 #Constraints 1 to 4 are row constraints as they
        corresponds to rows of transportation tableau
8 row.signs <-rep("=",4)
9 row.rhs <-c(1000,1300,1200,200)
10
11 #Constraints 5 & 6 are column constraints as they
        corresponds to columns of transportation tableau
12 col.signs <-rep("=",2)
13 col.rhs <-c(2300,1400)
14
15 library(lpSolve)
16 solution <- lp.transport (costs, "min", row.signs,
        row.rhs, col.signs, col.rhs)
17 solution$solution
18
19
20 ##Part2 - Adding a dummy destination
21 costs <- matrix (c(80,215,0,100,108,0,102,68,0), 3,
        3,byrow = T)
22

```

```

23 #Constraints 1,2 & 3 are row constraints as they
    corresponds to rows of transportation tableau
24 row.signs <-rep("=",3)
25 row.rhs <-c(1000,1500,1200)
26
27 #Constraints 4 & 5 are column constraints as they
    corresponds to columns of transportation tableau
28 col.signs <-rep("=",3)
29 col.rhs <-c(1900,1400,400)
30
31 library(lpSolve)
32 solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
33 solution$solution

```

---

#### R code Exa 5.2.1 Production inventory control

```

1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 2-1 : Page 202
3
4 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)
5
6 #Constraints 1,2,3 & 4 are row constraints as they
    corresponds to rows of transportation tableau
7 row.signs <-rep("=",4)
8 row.rhs <-c(50,180,280,270)
9
10 #Constraints 4 to 8 are column constraints as they
    corresponds to columns of transportation tableau
11 col.signs <-rep("=",4)
12 col.rhs <-c(100,200,180,300)
13
14 library(lpSolve)

```



```

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

```

---

### R code Exa 5.2.2 Tool sharpening

```

1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 2-2 : Page 203
3
4 #BigM is Big
5 BigM=1000
6
7 #Initializing the matrix with all values as M
8 costs<-matrix(BigM,8,8,byrow = T)
9 #All values of first row are 12
10 costs[1,]=c(rep(12,8))
11 #All values of 8th column is 0
12 costs[,8]=rbind(rep(0,8))
13
14 ##Adding the rest of the values
15 for (i in 2:7) {
16   costs[i,]=c(rep(BigM,i-1),c(6,5,rep(3,4))[1:(8-i)
17     ],0)
18 }
19 #Constraints 1 to 8 are row constraints as they
20   corresponds to rows of transportation tableau
21 row.signs <-rep("=",8)
22 row.rhs <-c(124,24,12,14,20,18,14,22)
23
24 #Constraints 8 to 16 are coloumn constraints as they
25   corresponds to coloumns of transportation
26   tableau
27 col.signs <-rep("=",8)

```

```

25 col.rhs <-c(24,12,14,20,18,14,22,124)
26
27 library(lpSolve)
28 solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
29 solution$solution

```

---

### R code Exa 5.3.1 Sunray Transport

```

1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 3-1 : Page 207
3 costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
    3, 4,byrow = T)
4
5 #Constraints 1,2 & 3 are row constraints as they
    corresponds to rows of transportation tableau
6 row.signs <-rep("=",3)
7 row.rhs <-c(15,25,10)
8
9 #Constraints 4 & 5 are coloumn constraints as they
    corresponds to coloumns of transportation tableau
10 col.signs <-rep("=",4)
11 col.rhs <-c(5,15,15,15)
12
13 library(lpSolve)
14 solution <- lp.transport (costs, "min", row.signs,
    row.rhs, col.signs, col.rhs)
15 solution$objval
16 solution$solution

```

---

### R code Exa 5.3.2 Northwest corner method

```

1 ##Chapter 5 : Transportation Model and its Variant

```

```

2 ##Example 3-2 : Page 208
3 costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
4   3, 4,byrow = T)
5 row.rhs <-c(15,25,10)
6 col.rhs <-c(5,15,15,15)
7 i=1;j=1
8 allotment <- matrix (rep(0,12), 3, 4,byrow = T)
9
10 while (sum(row.rhs) & sum(col.rhs)) {
11   ##Till we reach the last row and coloumn
12   while (i<=3 & j<=4) {
13     ##if demand is >= supply
14     if (row.rhs[i] >= col.rhs[j]) {
15       ##assign the demand to that cell
16       allotment[i,j]=col.rhs[j]
17       ##deduct supply from demand
18       row.rhs[i]=row.rhs[i]-col.rhs[j]
19       ##assign zero to supply
20       col.rhs[j]=0
21     }else {
22       ##assign the supply to that cell
23       allotment[i,j]=row.rhs[i]
24       ##deduct demand from supply
25       col.rhs[j]=col.rhs[j]-row.rhs[i]
26       ##assign zero to demand
27       row.rhs[i]=0
28     }
29     #if demand=0, go to the next demand,else go to
30     next supply
31     ifelse(row.rhs[i] == 0,(i=i+1),(j=j+1))
32   }
33 }
allotment

```

---

### R code Exa 5.3.3 Least cost method

```
1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 3-3 : Page 209
3 costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
4   3, 4,byrow = T)
5 row.rhs <-c(15,25,10)
6 col.rhs <-c(5,15,15,15)
7 costsdup <- costs
8
9 #until there are supply and demand
10 while (sum(row.rhs) & sum(col.rhs)) {
11   #index of min cost
12   index=which.min(costsdup)
13   #get the row index
14   rowindex=index %% length(row.rhs)
15   #get the coloumn index
16   colindex=ceiling(index/length(row.rhs))
17   #if row index=0 ,assign 3(since we are takinf
18     modulus)
19   if (rowindex==0) {rowindex=3}
20
21   #if demand > supply
22   if(row.rhs[rowindex]>=col.rhs[colindex]){
23     #allocate supply to that cell
24     allotment[index] <- col.rhs[colindex]
25     ##deduct supply from demand
26     row.rhs[rowindex] <- row.rhs[rowindex]-col.rhs[
27       colindex]
28     ##assign zero to supply
29     col.rhs[colindex]=0
30   }else{
31     #allocate demand to that cell
32     allotment[index] <- row.rhs[rowindex]
33     ##deduct demand from supply
34     col.rhs[colindex] <- col.rhs[colindex]-row.rhs[
35       rowindex]
```

```

33     ##assign zero to demand
34     row.rhs[rowindex]=0
35 }
36 costsdup[index]=1000
37
38 }
39
40
41 allotment

```

---

### R code Exa 5.3.5 Method of multipliers

```

1  ##Chapter 5 : Transportation Model and its Variant
2  ##Example 3-5 : Page 212
3  #u-v incidence matrix with columns – u1,v1,v2,u2,v3,
   v4,u3 and rows as uv equations
4  a=matrix(c
   (1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,1,0,0,0,0,0,0,1,0,
   ,6,7,byrow = T)
5
6  #cost matrix
7  costs <- matrix (c(10,2,20,11,12,7,9,20,4,14,16,18),
   3, 4,byrow = T)
8  #rhs of uv equations
9  n=c(10,2,7,9,20,18)
10
11 library(limSolve)
12 ##Least Squares with Equality and Inequality
   Constraints
13 Sol=lsei(E=a, F=n, A = diag(7), B = rep(0, 7),G=diag
   (7),H=rep(0, 7), verbose = FALSE)$X
14
15 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

```
1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 4-1 : Page 221
3 costs <- matrix (c(15,10,9,9,15,10,10,12,8), 3, 3,
4   byrow = T)
5 rowmin<-numeric()
5 colmin<-numeric()
6
7 ##Subtracting minimum cost element of row from all
   the elements of rows
8 for(i in 1:nrow(costs)){
9   costs[i,]<-costs[i,]-min(costs[i,])
10 }
11
12 ##Subtracting minimum cost element of column from
   all the elements of column
13 for(i in 1:ncol(costs)){
14   costs[,i]<-costs[,i]-min(costs[,i])
15 }
16
17 ##logic1 is a boolean matrix which contains true if
   cost matrix after the above operations is 0 and 0
   otherwise
18 logic1<-costs==0
19 eqnrow<-numeric()
20
21 ##We formulate it as a transportation tableau such
   that only one zero is selected from every row
   and every column
22 row.signs <-rep("=",nrow(costs))
23 row.rhs <-rep(1,nrow(costs))
24
25 col.signs <-rep("=",ncol(costs))
```

```

26 col.rhs <-rep(1,ncol(costs))
27
28 library(lpSolve)
29 Solution<-lp.transport (logic1, "max", row.signs,
    row.rhs, col.signs, col.rhs)
30 if (Solution$objval==nrow(costs)){
31   Solution$solution
32 }

```

---

#### R code Exa 5.4.2 Hungarian method 2

```

1 ##Chapter 5 : Transportation Model and its Variant
2 ##Example 4-2 : Page 222
3 costs <- matrix (c
    (1,4,6,3,9,7,10,9,4,5,11,7,8,7,8,5), 4, 4,byrow =
    T)
4 rowmin<-numeric()
5 colmin<-numeric()
6 costsdup<-costs
7
8 ##Subtracting minimum cost element of row from all
    the elements of rows
9 for(i in 1:nrow(costsdup)){
10   costsdup[i,]<-costsdup[i,]-min(costsdup[i,])
11 }
12
13 ##Subtracting minimum cost element of column from
    all the elements of column
14 for(i in 1:ncol(costsdup)){
15   costsdup[,i]<-costsdup[,i]-min(costsdup[,i])
16 }
17
18
19 ##We formulate it as a transportation tableau such
    that only one zero is selected from every row

```

```

    and every column
20 ##maketable function returns the constraint matrix
21 eqnrow<-numeric()
22 maketable<-function(costsdup){
23     logic1<-costsdup==0
24     return(logic1)
25 }
26
27 matr<-maketable(costsdup)
28
29 row.signs <-rep("=",nrow(costsdup))
30 row.rhs  <-rep(1,nrow(costsdup))
31
32 col.signs <-rep("=",ncol(costsdup))
33 col.rhs  <-rep(1,ncol(costsdup))
34 library(lpSolve)
35
36 Solution<-lp.transport (matr, "max", row.signs, row.
    rhs, col.signs, col.rhs)
37 while(Solution$objval!=nrow(costsdup)){
38     costsdup<-costsdup-min(costsdup[costsdup>0])
39     for (i in 1:length(costsdup)){
40         ifelse(costsdup[i]<0, (costsdup[i]=0),0)
41     }
42     matr<-maketable(costsdup)
43     Solution<-lp.transport (matr, "max", row.signs,
        row.rhs, col.signs, col.rhs)
44 }
45 Solution$solution

```

---



# Chapter 6

## Network Model

**R code Exa 6.2.1** Minimal spanning tree algorithm

```
1 ##Chapter 6 : Network Model
2 ##Example 2-1 : Page 239
3
4 #If you have trouble installing the package/library ,
   please reinstall R form the following link:https
   ://cran.r-project.org/bin/
5 library(igraph)
6 #creating the undirected graph with 6 nodes
7 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 )
8 #mst function generates the minimum spanning tree
9 MST<-mst(A,weights = c(1,9,5,7,3,6,4,5,10,8,3))
10
11 #The Fruchterman-Reingold Algorithm is a force-
   directed layout algorithm. The idea of a force
   directed layout algorithm is to consider
12 #a force between any two nodes. In this algorithm ,
   the nodes are represented by steel rings and the
   edges are springs between them.
13 #The attractive force is analogous to the spring
```

```

    force and the repulsive force is analogous to the
    electrical force. The basic idea is
14 #to minimize the energy of the system by moving the
    nodes and changing the forces between them.
15 #Here we create the coordinates for our graph using
    layout.fruchterman.reingold function
16 lay <- layout.fruchterman.reingold(A)
17
18 #Assigning the coordinates to the nodes of A
19 V(A)$x <- lay[, 1]
20 V(A)$y <- lay[, 2]
21
22 #assigning range of x and y
23 xlim <- range(lay[,1])
24 ylim <- range(lay[,2])
25
26 #plot graph A
27 plot.igraph(A, layout = lay,vertex.size=20,
28             xlim = xlim, ylim = ylim, rescale =
                FALSE)
29 #plot MST with red edges and nodes over the previous
    graph
30 plot.igraph(MST, layout = lay,vertex.color="red",
    edge.color="red",vertex.size=20,
31             add = TRUE, rescale = FALSE)

```

---

### R code Exa 6.3.1 Equipment replacement as shortest route problem

```

1 ##Chapter 6 : Network Model
2 ##Example 3-1 : Page 243
3
4 #If you have trouble installing the package/library ,
    please reinstall R form the following link:https
    ://cran.r-project.org/bin/
5 library(igraph)

```

```

6
7 #creating the directed graph with 5 nodes
8 A=graph(edges=c(1,2,1,3,1,4,2,3,2,4,2,5,3,4,3,5,4,5)
9         , n=5, directed=T )
9 #creating weights vector for each edges of the graph
10 weightsg<-c
    (4000,5400,9800,4300,6200,8700,4800,7100,4900)
11 #get shortest path from 1 to 5
12 sP<-get.shortest.paths(A, 1, to=5,weights =weightsg)
    $vpath
13 #get shortest path cost from 1 to 5
14 sPCost<-shortest.paths(A, 1, to=5,weights =weightsg)
15 sP
16 sPCost
17
18 #creating coordinates for layout
19 l<-cbind(seq(0,9,2),0)
20
21 #plot the graph with straight edges(edge 1,4,7 & 9
    are straight edges.The order of edges is taken
    from graph initialization in line 6)
22 plot(delete.edges(A,c(2,3,5,6,8)),layout=1*2,vertex.
    size=15,edge.arrow.width = 0.2, asp = 0.5,edge.
    label=weightsg[c(1,4,7,9)])
23 #plot the graph with straight edges over the
    previous graph
24 plot(delete.edges(A,c(1,4,7,9)),edge.curved=.8,
    layout=1*2,vertex.size=15,edge.arrow.width = 0.2,
    asp = 0.5,edge.label=weightsg[c(2,3,5,6,8)],add=T
    )
25 A[] <- 0
26 #assign color red to each node of the graph in the
    shortest path
27 for (ed in 1:(length(sP[[1]])-1)){
28   A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red"
    )
29   V(A)[sP[[1]][ed]]$color<-"red"
30 }

```

```

31 V(A)[5]$color<-"red"
32
33 #plot the shortest path over the previous graph
34 plot(A,layout=1*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

```

1 ##Chapter 6 : Network Model
2 ##Example 3-2 : Page 244
3
4 #If you have trouble installing the package/library ,
  please reinstall R form the following link:https
  ://cran.r-project.org/bin/
5 library(igraph)
6 #creating the directed graph with 7 nodes
7 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)
8 #creating probability vector
9 prob<-c(.2,.9,.8,.6,.1,.3,.4,.35,.25,.5)
10 #creating weight vector from probability vector
11 weightsg<-round(log10(prob),digits = 5)
12 ##calculating shortest path and its cost
13 sP<-get.shortest.paths(A, 1, to=7,weights =weightsg)
  $vpath
14 sPCost<-shortest.paths(A, 1, to=7,weights =weightsg,
  mode = "out")
15 sP
16 10^(-sPCost)
17 #plotting the graph A
18 l<-layout.auto(A)
19 plot(A,vertex.size=15,layout=1,edge.arrow.width =
  0.2, asp = 0.5,edge.label=weightsg)
20

```

```

21 #plotting the shortest path over graph A
22 A[] <- 0
23 for (ed in 1:(length(sP[[1]])-1)){
24   A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red"
25             )
26   V(A)[sP[[1]][ed]]$color<-"red"
27 }
28 V(A)[7]$color<-"red"
29 plot(A,vertex.size=15,layout=1, edge.arrow.width =
30       0.2,edge.color="orange", asp = 0.5,add=T)

```

---

### R code Exa 6.3.3 Three jub puzzle

```

1 ##Chapter 6 : Network Model
2 ##Example 3-3 : Page 245
3 library(igraph)
4
5 #creating the directed graph with 15 nodes
6 A=make_directed_graph(edges=c
7   (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,
8     , n=15)
9   #weights vector
10  weightsg<-rep(1,21)
11  #shortest path
12  sP<-get.shortest.paths(A, 1, to=9,weights =weightsg)
13    $vpath
14  #shortest path cost
15  sPCost<-shortest.paths(A, 1, to=9,weights =weightsg,
16    mode = "out")
17  sP
18  sPCost
19  #circle layout of graph
20  l<-layout.circle (A)
21  #plot graph A
22  plot(A,vertex.size=15,layout=1,edge.arrow.size =

```

```

    0.2, asp = 0.5)
19
20 #make an empty graph and add edges of the shortest
    path to the graph with red color nodes
21 A[] <- 0
22 for (ed in 1:(length(sP[[1]])-1)){
23     A<-A+edge(c(sP[[1]][ed],sP[[1]][ed+1]),color="red"
        )
24     V(A)[sP[[1]][ed]]$color<-"red"
25 }
26 V(A)[9]$color<-"red"
27 #plot the shortest path over the previous graph
28 plot(A,vertex.size=15,layout=1, edge.arrow.size =
    0.2,edge.color="orange", asp = 0.5,add=T)

```

---

#### R code Exa 6.3.4 Dijkstra's algorithm

```

1 ##Chapter 6 : Network Model
2 ##Example 3-4 : Page 248
3
4 # Initializing no of nodes and the distance matrix
5 n = 5
6 d<-matrix(Inf,nrow=n,ncol = n)
7 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
8
9 #initializing the djiktra's algorithm table as shown
    in the book
10 djiktable<-matrix(c(1:n,rep(0,3*n)),nrow=n)
11
12 now=1
13 djiktable[1,4]=1
14
15 #if any of the node doesn't have any outgoing edge,
    make it permanent. Ex: node 5

```

```

16 for (i in 1:5){
17   if (length(which((d[i,]!=Inf) %in% TRUE))==0){
18     djiktable[i,4]=1
19   }
20 }
21 #while there are nodes with temporary status
22 while(sum(djiktable[,4])!=n){
23   #find all the possible nodes form current node
24   possibles<-which((d[now,]!=Inf) %in% TRUE)
25   #for each node in possible nodes
26   for (i in possibles){
27     #assign current node to temp
28     temp<-djiktable[i,2]
29     #if current node is not assigned
30     if(djiktable[i,2]!=0){
31       #the minimum of distance is added
32       djiktable[i,2]=min(djiktable[i,2],djiktable[
         now,2]+ d[now,i])
33     }else{
34       djiktable[i,2]=djiktable[now,2]+ d[now,i]
35     }
36     #if there is no change in the next node
37     if(djiktable[i,2]!=temp){
38       #backtrack
39       djiktable[i,3]=now
40     }
41   }
42
43   #assign permanent status to the minimum index
44   min.indx<-which.min(djiktable[possibles,2])
45   djiktable[possibles[min.indx],4]=1
46   now<- possibles[min.indx]
47 }
48
49 #prints out the shortest route
50 djiktable[now,2]
51 path<-character()
52 while (now!=0){

```

```

53   path<-paste(">",now,path)
54   now=djiktable[now,3]
55 }
56 path

```

---

### R code Exa 6.3.5 Floyds algorithm

```

1  ##Chapter 6 : Network Model
2  ##Example 3-5 : Page 252
3  #Initializing the nodes and floyd's D matrix and S
   matrix
4  n = 5
5  floydD<-array(Inf,dim= c(5,5))
6  diag(floydD) <- 0
7  floydD[1,2]=3;floydD[1,3]=10;floydD[2,4]=5;floydD
   [3,4]=6;floydD[4,5]=4;
8
9  #symmetric matrix
10 for(i in 1:5){
11   j=1
12   while(j<i){
13     floydD[i,j]=floydD[j,i]
14     j=j+1
15   }
16 }
17 floydseq<-matrix(1:n,nrow=n,ncol=n,byrow = T)
18 diag(floydseq) <- 0
19
20 ##Floyd's Algorithm
21 for(k in 1:n){
22   for(i in 1:n){
23     for(j in 1:n){
24       if(i!=k & j!=k & i!=j){
25         if(floydD[i,k]+ floydD[k,j]< floydD[i,j]){
26           floydD[i,j] <- floydD[i,k]+ floydD[k,j]

```



```

27         floydseq[i,j] <- k
28     }
29 }
30 }
31 }
32 }
33 floydD
34 floydseq
35
36 ##Printing the shortest route
37 path<-character()
38 i=1;j=5
39 while(floydseq[i,j]!=j){
40     path<-paste("-",j,path)
41     j=floydseq[i,j]
42 }
43 path<-paste(i,"->",j,path)
44 path

```

---

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

```

1  ##Chapter 6 : Network Model
2  ##Example 3-6 : Page 256
3
4  # Objective function :Min 100 * x12 + 30 * x13 +20*
    x23 + 10*x34 + 60 * x35 + 15 * x42 + 50 * x45
5  a <- c(100,30,20,10,60,15,50)
6
7  # Constraint 1 : x12 + x13 = 1
8  C1 <- c(1,1,0,0,0,0,0)
9  bc1<- 1
10
11 # Constraint 2 : x12 + x42 - x23 = 1
12 C2 <- c(1,0,-1,0,0,1,0)
13 bc2<-1

```

```

14
15 # Constraint 3 :  $x_{13} + x_{23} - x_{34} - x_{35} = 0$ 
16 C3 <- c(0,1,1,-1,-1,0,0)
17 bc3<-0
18
19 # Constraint 4 :  $x_{34} - x_{42} - x_{45} = 0$ 
20 C4 <- c(0,0,0,1,0,-1,-1)
21 bc4<-0
22
23 # Constraint 5 :  $x_{35} + x_{45} = 0$ 
24 C5 <- c(0,0,0,0,1,0,1)
25 bc5<-0
26
27 library("lpSolve")
28
29 solution<-lp("min", a, rbind(C1, C2,C3,C4,C5), rep("
    =",5), c(bc1, bc2,bc3,bc4,bc5))
30 solution$objval
31 solution$solution

```

---

#### R code Exa 6.4.2 Maximal flow algorithm

```

1 ##Chapter 6 : Network Model
2 ##Example 4-2 : Page 263
3
4 #If you have trouble installing the package/library ,
   please reinstall R form the following link:https://cran.r-project.org/bin/
5 library(optrees)
6 #edge matrix with weights
7 arcs<-matrix(c(1,2,20,
8     1,3,30,
9     1,4,10,
10    1,3,30,
11    1,4,10,

```

```

12     2,3,40,
13     2,5,30,
14     2,5,30,
15     3,5,20,
16     4,5,20),
17 byrow=T,ncol=3)
18 #Finds the min cut for maximal flow
19 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

```

1 ##Chapter 6 : Network Model
2 ##Example 4-3 : Page 271
3
4 # Objective function 1 :Max x12 + x13 + x14
5 a <- c(1,1,1,0,0,0,0,0,0)
6
7 # Objective function 2 :Max x25 + x35 + x45
8 b <- c(0,0,0,0,1,0,1,0,1)
9
10 # Constraint 1 : x12 - x23 - x25 = 0
11 C1 <- c(1,0,0,-1,-1,0,0,0,0)
12 bc1<- 0
13
14 # Constraint 2 : x13 + x23 - x34 -x35 + x43 = 0
15 C2 <- c(0,1,0,1,0,-1,-1,1,0)
16 bc2<-0
17
18 # Constraint 3 : x14 + x34 - x43 -x45 = 0
19 C3 <- c(0,0,1,0,0,1,0,-1,-1)
20 bc3<-0
21
22 # Constraint 4 : x12 <= 20

```

```

23 C4 <- c(1,0,0,0,0,0,0,0,0)
24 bc4<-20
25
26 # Constraint 5 : x13 <= 30
27 C5 <- c(0,1,0,0,0,0,0,0,0)
28 bc5<-30
29
30 # Constraint 6 : x14 <= 10
31 C6 <- c(0,0,1,0,0,0,0,0,0)
32 bc6<-10
33
34 # Constraint 7 : x23 <= 40
35 C7 <- c(0,0,0,1,0,0,0,0,0)
36 bc7<-40
37
38 # Constraint 8 : x25 <= 30
39 C8 <- c(0,0,0,0,1,0,0,0,0)
40 bc8<-30
41
42 # Constraint 9 : x34 <= 10
43 C9 <- c(0,0,0,0,0,1,0,0,0)
44 bc9<-10
45
46 # Constraint 10 : x35 <= 20
47 C10 <- c(0,0,0,0,0,0,1,0,0)
48 bc10<-20
49
50 # Constraint 11 : x43 <= 5
51 C11 <- c(0,0,0,0,0,0,0,1,0)
52 bc11<-5
53
54 # Constraint 12 : x45 <= 20
55 C12 <- c(0,0,0,0,0,0,0,0,1)
56 bc12<-20
57
58 library("lpSolve")
59
60 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))
61 solution$objval
62 solution$solution
63
64 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))
65 solution$objval
66 solution$solution

```

---

### R code Exa 6.5.1 Network representation for PERT and CPM

```

1 ##Chapter 6 : Network Model
2 ##Example 4-3 : Page 275
3 library(igraph)
4 #create a directed graph
5 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)
6
7 #creating layout for plot
8 l<-cbind(c(1,1.5,3,5,4.5,7,9,11,13),c
        (1,3,3,3,2,3,2,1,1))
9
10 #creating plot
11 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"))
12 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

```
1 ##Chapter 6 : Network Model
2 ##Example 5-2 : Page 280
3
4 #create distance matrix with -1 meaning no edge and
   0 meaning no cost edge
5 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)
6
7 ##Forward pass
8 ECT<-numeric()
9 ECT[1]<-0
10 for(i in 2:6){
11   index<-which((D[,i]>=0) %in% TRUE)
12   ECT[i]<-max(ECT[index]+D[index,i])
13 }
14
15 ##Backward pass
16 BP<-numeric()
17 BP[6]<-ECT[6]
18 for(i in 5:1){
19   index<-which((D[i,]>=0) %in% TRUE)
20   BP[i]<-min(BP[index]-D[i,index])
21 }
22
23 ##Finding critical nodes
24 critical<-character()
25 criti<-numeric()
26 critj<-numeric()
27 for(i in 1:6){
28   for(j in 1:6){
29     if(BP[i]==ECT[i] & BP[j]==ECT[j] & BP[j]-ECT[i]
```

```

        ]==D[i,j]){
30         critical<-cbind(critical,paste(i,"-",j))
31         criti=cbind(criti,i)
32         critj=cbind(critj,j)
33     }
34 }
35 }
36 critical
37
38 ##Duration of the project
39 TotalDays<-0
40 for (i in 1:length(criti)){
41     TotalDays<-TotalDays+D[criti[i],critj[i]]
42 }
43 TotalDays

```

---

### R code Exa 6.5.3 Preliminary schedule

```

1 ##Chapter 6 : Network Model
2 ##Example 5-3 : Page 282
3 #Add an empty plot
4 plot(1, type="n", axes=T, xlab="Days", ylab="", yaxt
      = 'n',xlim=c(0,25),ylim=c(0,10), yaxs= "i", xaxs =
        "i")
5
6 #Add line segment for each segment
7 segments(0, 9, x1 = 5, y1 = 9)
8 segments(5, 8, x1 = 13, y1 = 8)
9 segments(13, 7, x1 = 25, y1 = 7)
10 segments(0, 6, x1 = 11, y1 = 6,lty = 2)
11 segments(5, 5, x1 = 11, y1 = 5,lty = 2)
12 segments(8, 4, x1 = 13, y1 = 4,lty = 2)
13 segments(8, 3, x1 = 25, y1 = 3,lty = 2)
14 segments(13, 2, x1 = 25, y1 = 2,lty = 2)
15 #Add text at specific coordinates

```

```

16 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)
17 #Add legend
18 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

```

1 ##Chapter 6 : Network Model
2 ##Example 5-4 : Page 284
3
4 #create distance matrix with -1 meaning no edge and
   0 meaning no cost edge
5 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)
6
7 ##Forward pass
8 ECT<-numeric()
9 ECT[1]<-0
10 for(i in 2:6){
11   index<-which((D[,i]>=0) %in% TRUE)
12   ECT[i]<-max(ECT[index]+D[index,i])
13 }
14
15 ##Backward pass
16 BP<-numeric()
17 BP[6]<-ECT[6]
18 for(i in 5:1){
19   index<-which((D[i,]>=0) %in% TRUE)
20   BP[i]<-min(BP[index]-D[i,index])
21 }
22
23 ##Finding critical nodes

```



```

24 critical<-character()
25 criti<-numeric()
26 critj<-numeric()
27 for(i in 1:6){
28   for(j in 1:6){
29     if(BP[i]==ECT[i] & BP[j]==ECT[j] & BP[j]-ECT[i]
        ]==D[i,j]){
30       critical<-cbind(critical,paste(i,"-",j))
31       criti=cbind(criti,i)
32       critj=cbind(critj,j)
33     }
34   }
35 }
36 critical
37
38 ##Duration of the project
39 TotalDays<-0
40 for (i in 1:length(criti)){
41   TotalDays<-TotalDays+D[criti[i],critj[i]]
42 }
43 TotalDays
44
45 #Calculating total float and free float for non-
    critical activities
46 NonCritical<-matrix(c(1,3,2,3,3,5,3,6,4,6),ncol=2,
    byrow = T)
47 NCA<-character()
48 duration<-numeric()
49 TotalF<-numeric()
50 FreeF<-numeric()
51 for(i in 1:length(NonCritical[,1])){
52   j<-NonCritical[i,1]
53   k<-NonCritical[i,2]
54   NCA[i]<-paste(j,"—>",k)
55   duration[i]<- D[j,k]
56   TotalF[i]<- BP[k]-ECT[j]-D[j,k]
57   FreeF[i]<-ECT[k]-BP[j]-D[j,k]
58 }

```

```

59
60 #Displaying df
61 df<-data.frame(NCA,duration>TotalF,FreeF)
62 names(df)<-c("Non-Critical Activity","Duration","
    Total Float","Free float")
63 df

```

---

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

```

1 ##Chapter 6 : Network Model
2 ##Example 5-5 : Page 288
3
4 # Objective function :Max 6*x12 + 6*x13 + 3*x23 + 8*
    x24 + 2*x35 + 11*x36 +1 *x46 + 12*x56
5 a <- c(6,6,3,8,2,11,0,1,12)
6
7 # Constraint 1 : -x12 - x13 = -1
8 C1 <- c(-1,-1,0,0,0,0,0,0,0)
9 bc1<- -1
10
11 # Constraint 2 : x12 - x23 - x24 = 0
12 C2 <- c(1,0,-1,-1,0,0,0,0,0)
13 bc2<-0
14
15 # Constraint 3 : x13 + x23 - x35 - x36 = 0
16 C3 <- c(0,1,1,0,-1,-1,0,0,0)
17 bc3<-0
18
19 # Constraint 4 : x24 -x45 -x46 = 0
20 C4 <- c(0,0,0,1,0,0,-1,-1,0)
21 bc4<-0
22
23 # Constraint 5 : x35 + x45 - x56 = 0
24 C5 <- c(0,0,0,0,1,0,1,0,-1)

```

```

25 bc5<-0
26
27 # Constraint 6 : x35 + x46 + x56 = 1
28 C6 <- c(0,0,0,0,0,1,0,1,1)
29 bc6<-1
30 library("lpSolve")
31 solution<-lp("max", a, rbind(C1, C2,C3,C4,C5,C6),
    rep("=",6), c(bc1, bc2,bc3,bc4,bc5,bc6))
32 solution$objval
33 solution$solution

```

---

# Chapter 7

## Advanced Linear Programming

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

```
1 ##Chapter 7 : Advanced Linear Programming
2 ##Example 1-2 : Page 300
3
4 #Creating the A and b matrix
5 A=matrix(c(1,3,-1,2,-2,-2),nrow=2,byrow=T)
6 b=matrix(c(4,2),nrow=2)
7
8
9 s<-character()
10 basic<-character()
11 Type<-character()
12
13 for(i in 3:1){
14   for(j in i:3){
15     ##if i!=j and ith and jth column are not
16     linearly dependent
17     if(i!=j & !all(abs(A[,i]/A[,j])==c(1,1))){
18       ##Solve for X with the ith and jth column
19       a<-solve(A[,c(i,j)])%*%b
20       s<-rbind(s,toString(a))
21       basic<-rbind(basic,c(paste(i,"&",j)))
```

```

21         #if all values in a>=0,then feasible ,
           infeasible otherwise
22         Type<-rbind(Type,ifelse(all(a >= 0),"Feasible"
           ,"Infeasible"))
23     }
24 }
25 }
26
27 cbind(basic,s,Type)

```

---

### R code Exa 7.1.3 Simmplex tableau in matrix form

```

1  ##Chapter 7 : Advanced Linear Programming
2  ##Example 1-3 : Page 303
3
4  # Objective function :Max x1 + 4*x2 + 2*x3 + 5*x4
5  a <- c(1,4,7,5)
6
7  # Constraint 1 : 2*x1 + x2 + 2*x3 + 4*x4 = 10
8  C1 <- c(2,1,2,4)
9  bc1<- 10
10
11 # Constraint 2 : 3*x1 - x2 - 2*x3 + 6*x4 = 5
12 C2 <- c(3,-1,-2,6)
13 bc2<-5
14
15 A<-rbind(C1,C2)
16 b<-rbind(bc1,bc2)
17
18 Binv<-solve(A[,c(1,2)])
19 X<-Binv%*%b
20
21 Binv
22 X

```

---

### R code Exa 7.2.1 Revised simplex algorithmr

```
1 ##Chapter 7 : Advanced Linear Programming
2 ##Example 2-1 : Page 309
3
4 # Objective function
5 a <- c(5,4,0,0,0,0)
6
7 #Constraints
8 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))
9 b<-rbind(24,6,1,2)
10 library("lpSolve")
11
12 solution<-lp("max", a, C, rep("=",4), b)
13 solution$objval
14 solution$solution
15
16 ##This solver is based on the revised simplex method
```

---

### R code Exa 7.3.1 Bounded variables algorithm

```
1 ##Chapter 7 : Advanced Linear Programming
2 ##Example 3-1 : Page 315
3 # Objective function
4 a <- c(3,5,2)
5
6 #Constraints
7 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))
8 b<-rbind(14,43,4,10,-7,3)
9 library("lpSolve")
```

```

10
11 solution<-lp("max", a, C, rep("<=" ,6), b)
12 solution$objval
13 solution$solution

```

---

#### R code Exa 7.4.1 Dual simplex algorithm

```

1 ##Chapter 7 : Advanced Linear Programming
2 ##Example 4-1 : Page 322
3 # Objective function
4 w <- c(3,5,0,0)
5
6 #Constraints
7 C<-rbind(c(1,2,1,0),c(-1,3,0,1))
8 b<-rbind(5,2)
9
10 B<-C[,c(1,4)]
11 Binv<-solve(B)
12
13 #Associated primal and dual variables are
14   assocprimal and assocdual
15 assocprimal<-Binv%*%b
16 assocdual<-w[c(1,4)]%*%Binv
17
18 ##the objective values are
19 primalobj<-c(3,0)%*%assocprimal
20 dualobj<-assocdual%*%c(5,0)
21
22 primalobj
23 dualobj

```

---

# Chapter 8

## Integer Linear Programming

R code Exa 8.1.1 Project selection

```
1 ##Chapter 8 : Integer Linear Programming
2 ##Example 1-1 : Page 336
3
4 # Objective function
5 a <- c(20,40,20,15,30)
6
7 # Constraint 1 :
8 C1 <- c(5,4,3,7,8)
9 bc1<- 25
10
11 # Constraint 2
12 C2 <- c(1,7,9,4,6)
13 bc2<-25
14
15 # Constraint 3
16 C3 <- c(8,10,2,1,10)
17 bc3<-25
18
19 library("lpSolve")
20 solution<-lp("max", a, rbind(C1, C2,C3), rep("<=" ,3)
    , c(bc1, bc2,bc3),all.bin=T)
```



```
21 solution$objval
22 solution$solution
```

---

### R code Exa 8.1.2 Installing security phones

```
1 ##Chapter 8 : Integer Linear Programming
2 ##Example 1-2 : Page 340
3
4 # Objective function
5 a <- rep(1,8)
6
7 # Constraint 1 :
8 C1 <- c(1,1,rep(0,6))
9 bc1<- 1
10
11 # Constraint 2
12 C2 <- c(0,1,1,rep(0,5))
13 bc2<-1
14
15 # Constraint 3
16 C3 <- c(0,0,0,1,1,0,0,0)
17 bc3<-1
18
19 # Constraint 4
20 C4 <- c(rep(0,6),1,1)
21 bc4<-1
22
23 # Constraint 5
24 C5 <- c(rep(0,5),1,1,0)
25 bc5<-1
26
27 # Constraint 6
28 C6 <- c(0,1,0,0,0,1,0,0)
29 bc6<-1
30
```

```

31 # Constraint 7
32 C7 <- c(1,0,0,0,0,1,0,0)
33 bc7<-1
34
35 # Constraint 8
36 C8 <- c(0,0,0,1,0,0,1,0)
37 bc8<-1
38
39 # Constraint 9
40 C9 <- c(0,1,0,1,0,0,0,0)
41 bc9<-1
42
43 # Constraint 10
44 C10 <- c(0,0,0,0,1,0,0,1)
45 bc10<-1
46
47 # Constraint 11
48 C11 <- c(0,0,1,0,1,0,0,0)
49 bc11<-1
50
51 library("lpSolve")
52 solution<-lp("min", a, rbind(C1, C2,C3,C4,C5,C6,C7,
53                               C8,C9,C10,C11), rep(">=",11), c(bc1, bc2,bc3,bc4,
54                               bc5,bc6,bc7,bc8,bc9,bc10,bc11),all.bin=T)
53 solution$objval
54 solution$solution

```

---

### R code Exa 8.1.3 Choosing a telephone company

```

1 ##Chapter 8 : Integer Linear Programming
2 ##Example 1-3 : Page 346
3
4 # Objective function
5 a <- c(0.25,0.21,0.22,16,25,18)
6

```

```

7 # Constraint 1 :
8 C1 <- c(1,1,1,0,0,0)
9 bc1<- 200
10
11 # Constraint 2
12 C2 <- c(1,0,0,-200,0,0)
13 bc2<-0
14
15 # Constraint 3
16 C3 <- c(0,1,0,0,-200,0)
17 bc3<-0
18
19 # Constraint 4
20 C4 <- c(0,0,1,0,0,-200)
21 bc4<-0
22
23 library("lpSolve")
24 solution<-lp("min", a, rbind(C1, C2,C3,C4), c("=", "
    <=", "<=", "<="), c(bc1, bc2,bc3,bc4),int.vec=c
    (4,5,6))
25 solution$objval
26 solution$solution

```

---

#### R code Exa 8.1.4 Job scheeduling model

```

1 ##Chapter 8 : Integer Linear Programming
2 ##Example 1-4 : Page 350
3
4 M<-1000
5 # Objective function
6 a <- c(0,0,0,0,0,0,19,0,12,0,34,0)
7
8 # Constraint 1 :
9 C1 <- c(1,-1,0,M,0,0,0,0,0,0,0,0)
10 bc1<- 20

```

```

11
12 # Constraint 2
13 C2 <-c(-1,1,0,-M,0,0,0,0,0,0,0,0)
14 bc2<-5-M
15
16 # Constraint 3
17 C3 <- c(1,0,-1,0,M,0,0,0,0,0,0,0)
18 bc3<-15
19
20 # Constraint 4
21 C4 <- c(-1,0,1,0,-M,0,0,0,0,0,0,0)
22 bc4<-5-M
23
24 # Constraint 5
25 C5 <- c(0,1,-1,0,M,0,0,0,0,0,0,0)
26 bc5<-15
27
28 # Constraint 6
29 C6 <- c(0,-1,1,0,-M,0,0,0,0,0,0,0)
30 bc6<-20-M
31
32
33 # Constraint 7
34 C7 <- c(1,0,0,0,0,0,-1,1,0,0,0,0)
35 bc7<-20
36
37 # Constraint 8
38 C8 <- c(0,1,0,0,0,0,0,0,-1,1,0,0)
39 bc8<-2
40
41 # Constraint 9
42 C9 <- c(0,0,1,0,0,0,0,0,0,0,-1,1)
43 bc9<-20
44
45 library("lpSolve")
46 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))

```

```
47 solution$objval
48 solution$solution
```

---

#### R code Exa 8.2.1 Branch and bound algorithm

```
1 ##Chapter 8 : Integer Linear Programming
2 ##Example 2-1 : Page 356
3
4 # Objective function
5 a <- c(5,4)
6
7 # Constraint 1 :
8 C1 <- c(1,1)
9 bc1<- 5
10
11 # Constraint 2
12 C2 <-c(10,6)
13 bc2<-45
14
15 library("lpSolve")
16 solution<-lp("max", a, rbind(C1, C2), rep("<=" ,2), c
    (bc1, bc2),int.vec=c(1,2))
17 solution$objval
18 solution$solution
19
20 ## 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

```
1 ##Chapter 8 : Integer Linear Programming
2 ##Example 2-2 : Page 364
3
```

```

4 # Objective function
5 a <- c(7,10)
6
7 # Constraint 1 :
8 C1 <- c(-1,3)
9 bc1<- 6
10
11 # Constraint 2
12 C2 <-c(7,1)
13 bc2<-35
14
15 library("lpSolve")
16 solution<-lp("max", a, rbind(C1, C2), rep("<=" ,2), c
    (bc1, bc2),int.vec=c(1,2))
17 solution$objval
18 solution$solution
19
20 ## 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

```
1 ##Chapter 8 : Heuristic Programming
2 ##Example 2-1 : Page 382
3
4 ##Initializing function F, Random number R and index
5 F=c(90,60,50,80,100,40,20,70)
6 R=0.1002
7 index=ceiling(R*8)
8 #while any of neighbouring values are lower,keep
  looping
9 while (F[index]>=F[index+1] | F[index]>=F[index-1]){
  index==1,1,index-1)]]{
10 #if the next index is less than the current index
11 if(F[index+1]<F[index]){
12   index=index+1
13 }
14 #if the previous index is less than the current
  index
15 if(F[index-1]<F[index]){
16   index=index+1
17 }
18 }
```

```
19 index
20 F[index]
```

---

### R code Exa 9.2.2 Random walk heuristic

```
1 ##Chapter 8 : Heuristic Programming
2 ##Example 2-2 : Page 383
3
4 ##Initializing function F, Random number R,current
   index and index
5 F=c(90,60,50,80,100,40,20,70)
6 R=0.1002
7 count=0
8 current=ceiling(R*8)
9 index<-ceiling(runif(1,min = 0,max = 8))
10 #while count < 3,keep looping
11 while (count<3){
12
13     count=count+1
14     if(F[index]<F[current]){
15         #set current to index and a new random value is
           allotted to index
16         current=index
17         index<-ceiling(runif(1,min = 0,max = 8))
18         #count is set to 0 when another index lesser
           than current is found
19         count=0
20     }
21 }
22 current
23 F[current]
```

---

### R code Exa 9.2.3 Random walk heuristic for continuous variables



```

1 ##Chapter 8 : Heuristic Programming
2 ##Example 2-3 : Page 385
3
4 #Function F returns the value of F for a given x
5 F <- function(x){return(x**5-10*x**4+35*x**3-50*x**
      2+24*x)}
6
7 #Intial values
8 x=0.5
9 newx=5
10 count=0
11 #while count<3
12 while(count<3){
13   #generate newx random uniform sampling
14   while(newx>4 | newx<0){newx<-x+(runif(1)-0.5)*
      (4-0)}
15   #if the new value is better than the old
16   if (F(newx)<F(x)){
17     #assign newx as x,newx as 5(so that it enters
      newx while loop in next iteration) and count
      to 0
18     x=newx
19     newx=5
20     count=0
21   }else{
22     #else increment count
23     count=count+1
24   }
25 }
26
27 print(paste("Optimal using uniform distribution = ",
      x))
28
29 x=0.5
30 newx=5
31 count=0
32 #while count<3
33 while(count<3){

```

```

34 #generate newx random uniform sampling
35 while(newx>4 | newx<0){newx<-x+(4*rnorm(1)/6)}
36 if (F(newx)<F(x)){
37     #assign newx as x,newx as 5(so that it enters
        newx while loop in next iteration) and count
        to 0
38     x=newx
39     newx=5
40     count=0
41 }else{
42     #else increment count
43     count=count+1
44     newx=5
45 }
46 }
47
48 print(paste("Optimal using normal distribution = ",x
    ))

```

---

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

```

1 ##Chapter 8 : Heuristic Programming
2 ##Example 3-1 : Page 388
3
4 G<-rep(0,8)
5 F=c(90,60,50,80,100,40,20,70)
6 ##function to return the neighbourhood of an index
7 N<-function(index){
8     A<-numeric()
9     for(j in 1:2){
10         for(i in 1:4){
11             if(index+((-1)**j)*i<=8 & index+((-1)**j)*i>0)
12                 {
13                     A<-c(A, index+((-1)**j)*i)
14                 }
15         }
16     }
17 }

```

```

13     }
14 }
15 }
16 return(A)
17 }
18
19 #for 100 repetitions
20 for(m in 1:100){
21   #take a random uniform index
22   index=ceiling(runif(1)*8)
23   Tenure=3
24   L=c(0)
25   #a random index from the neighbourhood not in tabu
      list
26   i=ceiling(runif(1)*sum(sum(!(N(index) %in% L))))
27
28   #while new index is lesser than old
29   while(F[index]>F[N(index)[!(N(index) %in% L)]] [i])
      {
30     #add current node to tabu list
31     L<-c(L, index)
32     #pick new node
33     index=N(index)[!(N(index) %in% L) ] [i]
34     #remove the node after their tenure
35     if(i>2){
36       L<-L[-1]
37     }
38     ##pick the next index
39     i=ceiling(runif(1)*sum(sum(!(N(index) %in% L))))
40   }
41   #counter for bar plot
42   G[index]=G[index]+1
43 }
44
45 barplot(G/sum(G), names.arg = 1:8)

```

---

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

```

1 ##Chapter 8 : Heuristic Programming
2 ##Example 3-3 : Page 396
3 library(GenSA)
4 #function that returns the function value
5 Fx<-function(x){
6   F=c(90,60,50,80,100,40,20,70)
7   return(F[ceiling(x)])
8 }
9 #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
10 Solution=GenSA(fn=Fx,lower=c(1),upper=c(8),control =
   list(smooth =F))
11 Solution$value
12 ceiling(Solution$par)

```

---

**R code Exa 9.3.4** Job scheduling using simulated annealing algorithm

```

1 ##Chapter 8 : Heuristic Programming
2 ##Example 3-1 : Page 388
3
4 #function to return the cost of a sequence
5 Fx<-function(index1){
6   cost<-0
7   date<-0
8   ctable=matrix(c
        (10,15,3,10,8,20,2,22,6,10,5,10,7,30,4,8),nrow
        = 4,byrow=T)
9

```

```

10   for(i in 1:4){
11       date<-date+ctable[index1[i],1]
12
13       if(date > ctable[index1[i],2]){
14           cost<-cost+(date-ctable[index1[i],2])*ctable[
15               index1[i],4]
16       }else{
17           cost<-cost+(ctable[index1[i],2]-date)*ctable[
18               index1[i],3]
19       }
20   }
21   return(cost)
22 }
23 #function to get the neighbourhoods of a sequence
24 N<-function(index1){
25     swap <- function(x,i) {x[c(i,i+1)] <- x[c(i+1,i)];
26         x}
27     A<-array(numeric(),c(0,4))
28     for(i in 1:3){
29         A<-rbind(A,swap(index1,i))
30     }
31     return(A)
32 }
33 #Initializations
34 index1=sample(c(1,2,3,4))
35 i=1;Ti<-Fx(index1)
36 #for 60 repetitions
37 while(i<=50){
38     Ti=0.5*Ti
39     p=0
40     ##Step 2 of the algorithm
41     if(p<3){
42         Neighbours<-N(index1)
43         RandomN<-Neighbours[ceiling(runif(1,0,3)),]
44         if(Fx(index1)<Fx(RandomN)){

```

```

45         if(runif(1)<exp(-abs(Fx(index1)-Fx(RandomN))/
46             Ti)){
47             index1<-RandomN
48             p=p+1
49             i=i+1
50         }else{
51             i=i+1
52         }
53     }else{
54         index1<-RandomN
55         p=p+1
56         i=i+1
57     }
58 }else{
59     i=i+1
60 }
61 Fx(index1)

```

---

## Chapter 10

# Travelling Salesman Problem

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

```
1 ##Chapter 10 : Travelling Salesman Problem
2 ##Example 3-1 : Page 438
3
4 #TSP library
5 library("TSP")
6 #Distance matrix
7 dij<-matrix(c(Inf,10,3,6,9,5,Inf,5,4,2,4,9,Inf,
8             ,7,8,7,1,3,Inf,4,3,2,6,5,Inf),nrow=5,byrow = T)
9 #making a asymmetric TSP instance
10 atsp<-ATSP(dij)
11 #Solve TSP
12 d=solve_TSP(atsp,method = "nearest_insertion",
13             control = list(start,"1"))
14 d[1:5]
```

---

**R code Exa 10.3.2** Travelling Salesman Problem using cutting plane algorithm

```

1 ##Chapter 10 : Travelling Salesman Problem
2 ##Example 3-2 : Page 441
3
4 #TSP library
5 library("TSP")
6 #Distance matrix
7 dij<-matrix(c(Inf,13,21,26,10,Inf,29,20,30,20,Inf
8             ,5,12,30,7,Inf),nrow=4,byrow = T)
9 #making a asymeric TSP instance
10 atsp<-ATSP(dij)
11 #Solve TSP
12 d=solve_TSP(atsp,method = "nearest_insertion",
13             control = list(start,"1"))
14 d
15 d[1:4]

```

---

**R code Exa 10.4.1** Travelling Salesman Problem using nearest neighbour heuristic

```

1 ##Chapter 10 : Travelling Salesman Problem
2 ##Example 3-1 : Page 443
3
4 #Distance matrix
5 dij<-matrix(c(Inf,120,220,150,210,120,Inf
6             ,100,110,130,220,80,Inf,160,185,150,Inf,160,Inf
7             ,190,210,130,185,Inf,Inf),nrow=5,byrow = T)
8 #Source node
9 source=3
10 #All edges leading to 3 has infinite length
11 dij[,3]=Inf
12 tour<-source
13 ##Nearest neighbour heuristic
14 for(i in 1:4){
15     #choose nearest neighbour to the node
16     mini<-which.min(dij[source,])
17 }

```



```
15  #add it to the tour
16  tour<-c(tour,mini)
17  #make it as the source for next iteration
18  source=mini
19  #set all edges leading to current node as Inf
20  dij[,mini]=Inf
21 }
```

---

# Chapter 11

## Deterministic Dynamic Programming

**R code Exa 11.1.1** Shortest route problem using dynamic programming

```
1 ##Chapter 11 : Deterministic Dynamic Programming
2 ##Example 1-1 : Page 461
3
4 #Create a matrix with all Inf
5 d<-matrix(Inf,7,7)
6 #Add edge weights/lengths
7 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)
8 #Dynamic algorithm for Shortest path algorithm
9 ShortestDistance<-function(node,stage){
10
11   index<-which(d[1:7,node]!=Inf)
12   #if it is node 1 at stage 2, return 1
13   if(c(1) %in% index & stage==2){
14     return(d[1,node])
15   #else return the minimum distance of all possible
     nodes from current node
16   }else{
17     dist<-numeric()
```

```
18     for(i in 1:length(index)){
19         dist[i]<-(min(d[index[i],node]+
20             ShortestDistance(index[i],stage-1)))
21     }
22     return(min(dist))
23 }
24 ShortestDistance(7,4)
```

---

# Chapter 12

## Deterministic Inventory Modelling

**R code Exa 12.2.1** Analyzing the nature of demand

```
1 ##Chapter 12 : Deterministic Inventory Modelling
2 ##Example 2-1 : Page 493
3
4 ##Natural gas consumption details
5 d<-matrix(c
      (100,110,90,70,65,50,40,42,56,68,88,95,110,125,98,80,60,53,44,45,6
6          88,79,56,57,38,39,60,70,82,90,121,130,95,90,70,58,41,44,6
7          68,55,43,41,65,79,88,94,130,122,100,85,73,58,42,43,64,75
8          55,45,40,67,78,98,97,130,115,100,95,80,60,49,48,64,85,96
9          39,69,90,100,110,87,80,78,75,69,48,39,41,50,70,88,93)
      ,byrow=T,ncol=12)
10 #row and coloumn names
11 rownames(d)<-1990:1999
12 colnames(d)<-month.abb
13
```

```

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

```

---

### R code Exa 12.3.1 Classic EOQ model problem

```

1 ##Chapter 12 : Deterministic Inventory Modelling
2 ##Example 3-1 : Page 495
3
4 #input  parametres
5 D=100
6 K=100
7 h=0.02
8 L=12
9 #optimal quantity
10 y=sqrt(2*K*D/h)
11
12 print(paste("EOQ =",y," units"))
13 #cycle length
14 t=y/D
15 print(paste("Cycle length =",t," days"))
16 #number of cycles
17 L0=L- floor(L/t)*t
18
19 print(paste("Reorder point =",L0*D," units"))
20 print(paste("Daily inventory cost = $",(K*D/y)+h*y/
    2))

```

---

### R code Exa 12.3.2 Optimal order policy

```

1 ##Chapter 12 : Deterministic Inventory Modelling
2 ##Example 3-2 : Page 501
3
4 #input parameters
5 D=187.5
6 h=0.02
7 K=20
8 L=2
9 c1=3
10 c2=2.5
11 q=1000
12 #optimal quantity
13 ym=sqrt(2*K*D/h)
14 #function to calculate the least cost
15 if(q<ym){
16   Y=ym
17 }else{
18   x=polyroot(c(2*K*D/h,(2*(c2*D-(c1*D+(K*D/ym)+(h*ym
19     /2))/h),1))
20   if(Re(x[2])>q){
21     Y=q
22   }else{
23     Y=ym
24   }
25 }

```

---

#### R code Exa 12.4.1 Dynamic EOQ models with no setup

```

1 ##Chapter 12 : Deterministic Inventory Modelling
2 ##Example 4-1 : Page 508
3
4 #Setting up initial cost matrix and other values
5 cost<-matrix(Inf,8,5)
6 for(i in 1:4){

```

```

7   for (j in i:4){
8       cost[2*i-1,j]=6 + (j-i)*0.1
9       cost[2*i,j]=9 + (j-i)*0.1
10  }
11  }
12  cost[,5]=0
13  totalcost=0
14  Regular=c(90,100,120,110)
15  Overtime=c(50,60,80,70)
16  Demand=c(100,190,210,160,20)
17  Supply= c(rbind(Regular,Overtime))
18
19  #Setting up the resultant table
20  rown=numeric()
21  for(i in 1:4){rown=c(rown,paste("R",i,sep = " "),
22      paste("O",i,sep = " "))}
22  allocation=matrix(0,8,5,dimnames = list(rown,c(1:4,"
23      Surplus"))))
24
25  for(i in 1:5){
26
27      while(Demand[i]>0){
28          minindex=which.min(cost[,i])
29          #if it can still meet the demand
30          if(Supply[minindex]>0){
31              #allocate min of demand or supply
32              allocation[minindex,i]=min(Demand[i],Supply[
33                  minindex])
34              #calculate cost
35              totalcost=totalcost+allocation[minindex,i]*
36                  cost[minindex,i]
37              ##Subtract the allocated form supply and
38                  demand
39              tmp=Demand[i]
40              Demand[i]=Demand[i]-min(Demand[i],Supply[
41                  minindex])
42              Supply[minindex]=Supply[minindex]-min(tmp,

```

```
        Supply[minindex])
39     cost[minindex,i]=Inf
40     #else set all cost of that week to Inf so that
        it doesnt get chosen again
41     }else{
42         cost[minindex,]=Inf
43     }
44
45
46 }
47 }
48 allocation
49 totalcost
```

---



# Chapter 13

## Decision Analysis and Games

R code Exa 13.1.1 Overall Idea of AHP

```
1 ##Chapter 13 : Decision Analysis and Games
2 ##Example 1-1 : Page 527
3
4 #Setting up input values
5 LocationC=c(12.9,27.2,59.4)
6 ReputationC=c(54.5,27.3,18.2)
7 LocationP=0.17
8 ReputationP=0.83
9 Criteria=c(LocationP,ReputationP)
10 #composite weights
11 Choice=Criteria%*%rbind(LocationC,ReputationC)
12
13 colnames(Choice)<-c("UofA","UofB","UofC")
14
15
16 #Setting up the data tree for the particular problem
17 #If you have trouble installing the package/library,
    please reinstall R from the following link:https://cran.r-project.org/bin/
18 library(data.tree)
19 SelectAUniversity <- Node$new("Select a University")
```

```

20   Location<-SelectAUniversity$AddChild(paste("
      Location",LocationP))
21   for(child in 1:3){
22     Location$AddChild(paste(" Uof",LETTERS[child],
      LocationC[child]))
23   }
24   Reputation<-SelectAUniversity$AddChild(paste("
      Reputation",ReputationP))
25   for(child in 1:3){
26     Reputation$AddChild(paste(" Uof",LETTERS[child
      ],LocationC[child]))
27   }
28   print(SelectAUniversity)
29   print(Choice)

```

---

#### R code Exa 13.1.2 Weights and consistency for AHP

```

1  ##Chapter 13 : Decision Analysis and Games
2  ##Example 1-2 : Page 530
3
4  A=matrix(c(1,0.2,5,1),nrow = 2,byrow = T)
5  rowMeans(sweep(A,2,colSums(A),'/'))
6  AL=matrix(c(1,0.5,0.2,2,1,0.5,5,2,1),nrow = 3,byrow
      = T)
7  rowMeans(sweep(AL,2,colSums(AL),'/'))
8  AR=matrix(c(1,2,3,0.5,1,1.5,1/3,2/3,1),nrow = 3,
      byrow = T)
9  rowMeans(sweep(AR,2,colSums(AR),'/'))

```

---

#### R code Exa 13.1.3 Consistency ratio

```

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

```

```

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

```

---

#### R code Exa 13.2.1 Decision tree

```

1 ##Chapter 13 : Decision Analysis and Games
2 ##Example 2-1 : Page 538
3
4 CompanyA=c(5000,-2000)
5 CompanyB=c(1500,500)
6 #Probability of occurrence
7 POfOccurance=c(0.6,0.4)
8 #Expected return of stock A and B
9 ExpectedStockA=POfOccurance*%CompanyA
10 ExpectedStockB=POfOccurance*%CompanyB
11 ExpectedStockA
12 ExpectedStockB

```

---

#### R code Exa 13.2.2 Expected value criterion

```

1 ##Chapter 13 : Decision Analysis and Games
2 ##Example 2-2 : Page 544
3

```

```

4 #Probability matrix
5 CondP=matrix(c(0.9,0.1,0.5,0.5),nrow=2,byrow = T)
6
7 PriorP=c(0.6,0.4)
8 JointP=CondP*PriorP
9 AbsP=colSums(JointP)
10 BayesP=sweep(JointP,2,AbsP,'/')
11
12 CompanyA=c(5000,-2000)
13 CompanyB=c(1500,500)
14
15 ExpectedStockAat4=BayesP[,1]*%CompanyA
16 ExpectedStockBat5=BayesP[,1]*%CompanyB
17 ExpectedStockAat6=BayesP[,2]*%CompanyA
18 ExpectedStockBat7=BayesP[,2]*%CompanyB
19
20 ExpectedStockAat4
21 ExpectedStockBat5
22 ExpectedStockAat6
23 ExpectedStockBat7

```

---

### R code Exa 13.3.1 Decision under uncertainty

```

1 ##Chapter 13 : Decision Analysis and Games
2 ##Example 3-1 : Page 553
3
4 #Cost matrix
5 costmatrix<-matrix(c
      (5,10,18,25,8,7,12,23,21,18,12,21,30,22,19,15),
      nrow = 4,byrow = T)
6
7 #Laplace criterion
8 print("Laplace")
9 P=1/4
10 E=rowSums(costmatrix)*P

```

```

11 min(E)*1000
12
13 #Minimax criterion
14 print("Minimax")
15 E=apply(costmatrix,1,max)
16 min(E)*1000
17
18 #Savage criterion
19 print("Savage")
20 r=sweep(costmatrix,2,apply(costmatrix,2,min))
21 E=apply(r,1,max)
22 min(E)

```

---

#### R code Exa 13.4.1 Two person zero sum game

```

1 ##Chapter 13 : Decision Analysis and Games
2 ##Example 4-1 : Page 556
3
4 #payoff matrix
5 payoffmatrix=matrix(c(8,-2,9,-3,6,5,6,8,-2,4,-9,5),
6   nrow = 3,byrow = T)
7
8 RowMin=apply(payoffmatrix,1,min)
9 print("Maximin")
10 max(RowMin)
11
12 ColMax=apply(payoffmatrix,2,max)
13 print("Maximin")
14 min(ColMax)

```

---

#### R code Exa 13.4.3 Mixed strategy games

```

1 ##Chapter 13 : Decision Analysis and Games

```

```

2 ##Example 4-3 : Page 559
3
4 #plot an empty graph
5 plot(1, type="n", axes=F,xlab = "x1", ylab = "x2",
      xlim = c(-0.5,1.5), ylim = c(-2,7), col = "red",
6       yaxs= "i", xaxs = "i")
7
8 #Add custom axis on sides 1 and 2
9 axis(side = 1,pos = 0,at = seq(0,1,0.25))
10 axis(side = 2,pos = 1,at = seq(-2,9,2),padj=3.5)
11 axis(side = 2,pos = 0,at = seq(-2,9,2))
12
13 #Add line segments for the constraints
14 segments (0,6,1,-1,col = "red")
15 segments (0,4,1,2,col = "yellow")
16 segments (0,3,1,2,col = "green")
17 segments (0,2,1,3,col = "blue")
18 #adding a line segment to indicate the maximum
19 segments (0.5,0,0.5,2.5,col = "black",lwd = 1.5)
20 #Adding the name of the constraints on the graph
21 text(0,4,"B1",pos=4,cex=0.5)
22 text(0,3,"B2",pos=4,cex=0.5)
23 text(0,2,"B3",pos=4,cex=0.5)
24 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

```
1 ##Chapter 14 : Probabilistic Inventory Models
2 ##Example 1-1 : Page 575
3
4 #Daily demand
5 D=100
6 #standard deviation
7 dev=10
8 #probability of running out of stock
9 alpha=0.05
10 #lead time
11 L=2
12 #average demand during lead time
13 muL=D*L
14 #standard deviation of demand during lead time
15 devL=sqrt((dev**2)*L)
16 #optimal inventory level for reordering
17 x=qnorm(0.05,100,10,lower.tail = F)
18 z=(x-D)/dev
19 B=devL*z
20 B
```

---

**R code Exa 14.2.1** Newsvendor problem

```
1 ##Chapter 14 : Probabilistic Inventory Models
2 ##Example 2-1 : Page 582
3
4 #holding cost
5 h=25
6 #penalty cost
7 p=45
8 #critical ratio
9 CR=p/(p+h)
10
11 D=matrix(c(200,220,300,320,340,0.1,0.2,0.4,0.2,0.1),
12          nrow = 2,byrow = T)
13
14 #The demand is a normal distribuion
15 print("Case A")
16 ystar=qnorm(0.643,300,20)
17
18 #The demand is a discrete PDF
19 print("Case B")
20 CDF=numeric()
21 for(i in 1:5){CDF[i]=sum(D[2,1:i])}
22 rbind(D,CDF)
```

---



# Chapter 15

## Markov Chain

**R code Exa 15.2.1** Absolute probabilities after transitions

```
1 ##Chapter 15 : Markov Chain
2 ##Example 2-1 : Page 596
3
4 #Library expm is needed to get element-wise power of
  a matrix
5 library(expm)
6 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)
7 P8=P%^8
8 P16=P%^16
9
10 #Steady-state probabilities
11 a1=c(1,0,0)%*%P
12 a8=c(1,0,0)%*%P8
13 a16=c(1,0,0)%*%P16
14
15 a1
16 a8
17 a16
```

---

### R code Exa 15.3.1 Absorbing and transient states

```
1 ##Chapter 15 : Markov Chain
2 ##Example 3-1 : Page 598
3
4 ##Editing mistake in textbook
5 ##The last element of P should be one as (sum over j
   )(p)=1 for all i
6 P=matrix(c(0.2,0.5,0.3,0,0.5,0.5,0,0,1),nrow = 3,
   byrow = T)
7 library(expm)
8 P%%100
```

---

### R code Exa 15.3.2 Periodic states

```
1 ##Chapter 15 : Markov Chain
2 ##Example 3-2 : Page 599
3
4 #transition matrix
5 P=matrix(c(0,0.6,0.4,0,1,0,0.6,0.4,0),nrow = 3,byrow
   = T)
6 library(expm)
7 #n=2
8 P%%2
9 #n=3
10 P%%3
11 #n=4
12 P%%4
13 #n=5
14 P%%5
```

---

### R code Exa 15.4.1 Steady state probabilities

```
1 ##Chapter 15 : Markov Chain
2 ##Example 4-1 : Page 601
3
4 #transition matrix
5 A=matrix(c(0.6,-0.4,0.4,0.1,0.3,-0.45,1,1,1),nrow =
      3,byrow = T)
6 b=c(rep(0,2),1)
7 #solve the linear equations
8 pi=solve(A,b)
9 pi
10 #mean recurrence time
11 mu=1/pi
12 mu
```

---

### R code Exa 15.4.2 Cost model

```
1 ##Chapter 15 : Markov Chain
2 ##Example 4-2 : Page 602
3
4 #transition matrix
5 A=matrix(c(0.6,-0.4,0.4,0.1,0.3,-0.45,1,1,1),nrow =
      3,byrow = T)
6 b=c(rep(0,2),1)
7
8 #solve the linear equations
9 pi=solve(A,b)
10 #mean recurrence time
11 mu=1/pi
12
13 #expected abbual cost of fertilizer
```

```

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

```

---

#### R code Exa 15.5.1 Mean first passage time

```

1 ##Chapter 15 : Markov Chain
2 ##Example 5-1 : Page 606
3
4 #transition matrix
5 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)
6 N1=P[-1,-1]
7 #mean first passage time
8 mu=solve(diag(2)- N1)%*%rep(1,2)

```

---

#### R code Exa 15.6.1 Analysis of absorbing states

```

1 ##Chapter 15 : Markov Chain
2 ##Example 6-1 : Page 610
3
4 #transition matrix
5 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)
6 N=P[1:4,1:4]
7 A=P[1:4,5:6]
8 #Espected time of absorption
9 ExpT=solve(diag(4)-N)
10 #Pobability of absorption
11 POfAbsorption=ExpT %*% A
12 ExpT
13 POfAbsorption

```

---

# Chapter 16

## Queuing Systems

**R code Exa 16.4.1** Pure birth model

```
1 ##Chapter 16 : Queuing Systems
2 ##Example 4-1 : Page 629
3
4 #birth-rate per day
5 lambdaday=24*60/12
6 #birth-rate per year
7 lambdayear=lambdaday*365
8 #probability of no births during a day
9 P=((lambdaday*1)^0 * exp(-120*1))/factorial(0)
10 P
11 #probability of issuing 50 birth certificate in 3
    hrs given that 40 certificates were issued during
    the first 2 hrs off 3 hr period
12 P1=((60/12 *1)^10 *exp(-5*1))/factorial(10)
13 P1
```

---

**R code Exa 16.4.2** Pure death model

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 4-1 : Page 632
3
4 #probability of placing an order in any one day of
  the week
5 Pfunction <- function(mu,t,n,Slimit){
6   #empty matrix for the table
7   A<-matrix(0,nrow = 3,ncol = 7)
8   p0=0
9   for(t in 1:7){
10    P=p0
11    for(n in 1:5){
12      P=P+((mu*t)^(Slimit-n) *exp(-mu*t))/factorial(
        Slimit-n)
13    }
14    #appending to the table
15    A[1,t]=t
16    A[2,t]=mu*t
17    A[3,t]=round(P,digits = 4)
18  }
19  return(A)
20 }
21 Pfunction(3,7,5,18)
22
23 #average number of dozen roses discarded at the end
  of the week
24 discardedRoses<-function(N,mu,n,t){
25   P=0
26   for(i in 1:n){
27     P=P+i*((mu*t)^(N-i) * exp(-mu*t))/factorial(N-i)
28   }
29   return(P)
30 }
31 discardedRoses(18,3,18,7)

```

---

### R code Exa 16.6.1 Measures of performance

```
1 ##Chapter 16 : Queuing Systems
2 ##Example 6-1 : Page 642
3
4 #Function to calculate p0
5 P0<-function(n,servers,temp){
6   x=1
7   for(i in 1:n){
8     if (i<6){
9       x=x+(temp^i)/factorial(i)
10    }else{
11      x=x+(temp^i)/(factorial(servers)*servers^(i-
12        servers))
13    }
14    return(1/x)
15  }
16  p0=P0(8,5,3)
17  p0
18
19 #calculate Pn
20 temp=3
21 servers=5
22 A=matrix(0,2,8)
23 for(i in 1:8){
24   if (i<6){
25     A[1,i]=i
26     A[2,i]=p0*(3^i)/factorial(i)
27   }else{
28     A[1,i]=i
29     A[2,i]=p0*(temp^i)/(factorial(servers)*servers^(
30       i-servers))
31   }
32 }
33 A
34 #arrival rate
35 lambda=6
```

```

35 #arrivals lost
36 lambdalost=lambda*A[2,8]
37 lambdalost
38 #effective arrivals
39 lamdaeff=lambda-lambdalost
40 lamdaeff
41 #average lengths in the systems
42 Ls=0*p0+sum(1:8*A[2,])
43 Ls
44 #waiting time in the systems
45 Ws=Ls/lamdaeff
46 Ws
47 #average lengths in the queue
48 Wq=Ws-1/2
49 Wq
50 #average number of occupied spaces
51 cbar=lamdaeff/2
52 cbar
53 #parking lot utilization
54 utilization=cbar/servers
55 utilization

```

---

#### R code Exa 16.6.2 MM1 GDInfInf model

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-2 : Page 645
3
4 #Queueing library to process different queueing
  models
5 library(queueing)
6 #creating a MM1 instance with the following
  parameters
7 x=NewInput.MM1(lambda=4, mu=6,n=25)
8 #Solving the model which returns a list
9 y=QueueingModel(x)

```



```

10 #probability of each of the n customers in the
    system
11 P=y$Pn
12
13 #number of parking spaces such that an arriving car
    finds a place atleast 90% of times
14 cummP=numeric()
15 for(i in 1:length(P)){cummP[i]=sum(P[1:i])}
16 min(which(cummP > 0.9))-1

```

---

#### R code Exa 16.6.4 MM1 GDNInf model

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-4 : Page 649
3
4 #Queueing library to process different queueing
    models
5 library(queueing)
6 #creating a MMK instance with the following
    parameters
7 x=NewInput.MM1K(lambda=4, mu=6,k=5)
8 #Solving the model which returns a list
9 y=QueueingModel(x)
10 summary(y)

```

---

#### R code Exa 16.6.5 MMc GDInfInf model

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-5 : Page 653
3
4 #Queueing library to process different queueing
    models
5 library(queueing)

```

```

6
7 #creating a MMc instance with the following
  parameters
8 x=NewInput.MMC(lambda=8, mu=5, c=2)
9 #Solving the model which returns a list
10 y=QueueingModel(x)
11 summary(y)
12
13 #creating a MMc instance with the following
  parameters
14 x=NewInput.MMC(lambda=16, mu=5, c=4)
15 #Solving the model which returns a list
16 y=QueueingModel(x)
17 summary(y)

```

---

#### R code Exa 16.6.6 MMc GDNInf model

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-6 : Page 657
3
4 #Queueing library to process different queueing
  models
5 library(queueing)
6
7 #creating a MMc instance with the following
  parameters
8 x=NewInput.MMCK(lambda=16, mu=5, c=4, k=10)
9 #Solving the model which returns a list
10 y=QueueingModel(x)
11 summary(y)

```

---

#### R code Exa 16.6.7 MMInf GDInfNEInf or Slef service models

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-7 : Page 660
3
4 #Queueing library to process different queueing
  models
5 library(queueing)
6 #creating a MMc instance with the following
  parameters
7 x=NewInput.MMInf(lambda=12, mu=0.333)
8 #Solving the model which returns a list
9 y=QueueingModel(x)
10 summary(y)
11 estimate=(0.25*round(y$L)*1000)*(1-0.2)+(0.75*round(
  y$L)*1000)*(1+0.12)
12 estimate
13 ##The answer for the estimate is given wrong in the
  book

```

---

#### **R code Exa 16.6.8 MMR GDKK or Machine servicing model**

```

1 ##Chapter 16 : Queuing Systems
2 ##Example 6-8 : Page 662
3
4 #Queueing library to process different queueing
  models
5 library(queueing)
6
7 #Empty matrix table for Machine productivity
8 McProductivity=matrix(0,2,4)
9 #Making a matrix for different number of
  repairpersons
10 for(i in 1:4){
11   #creating a MMc instance with the following
     parameters
12   x=NewInput.MMCKK(lambda=0.5, mu=5, c=i, k=22)

```

```

13  #Solving the model which returns a list
14  y=QueueingModel(x)
15  McProductivity[1,i]=(22-y$L)*100/22
16  if(i>1){
17      McProductivity[2,i]=McProductivity[1,i]-
          McProductivity[1,i-1]
18  }
19 }
20 McProductivity

```

---

**R code Exa 16.7.1** MG1 GDInfInf model or Pollaczek Khintchine formula

```

1  ##Chapter 16 : Queuing Systems
2  ##Example 7-1 : Page 664
3
4  #arrival rate
5  lambda=4
6  #Expectation of arrival
7  EOfT=1/6
8  #variance of arrival
9  VarOfT=0
10 #Length of the system
11 Ls=lambda*EOfT+((lambda^2 *(EOfT^2 + VarOfT))/(2*(1-
    lambda*EOfT)))
12 #Length of the queue
13 Lq=Ls-lambda*EOfT
14 #Waiting time in the system
15 Ws=Ls/lambda
16 #Waiting time in the queue
17 Wq=Lq/lambda
18 Ls
19 Lq
20 Ws
21 Wq

```

---

### R code Exa 16.9.1 Cost models

```
1 ##Chapter 16 : Queuing Systems
2 ##Example 9-1 : Page 667
3
4 #Queueing library to process different queueing
  models
5 library(queueing)
6 #inputs
7 Speed=c(30,36,50,66)
8 OperatingCost=c(15,20,24,27)
9
10 #service rate for each model
11 ServiceRate=numeric()
12 for(i in 1:4){
13   ServiceRate[i]=24/(10000/(Speed[i]*60))
14 }
15 ServiceRate
16 Ls=numeric()
17
18 #length of system for each model
19 for(i in 1:4){
20   #creating a MMc instance with the following
    parameters
21   y=NewInput.MM1(lambda=4, mu=ServiceRate[i])
22   #Solving the model which returns a list
23   z=QueueingModel(y)
24   Ls[i]=z$L
25 }
26 Ls
27 #cost for the 4 models
28 cost=matrix(0,4,3)
29 colnames(cost)=c("EOC", "EWC", "ETC")
30 for(i in 1:4){
```

```

31   cost[i,1]=24*OperatingCost[i]
32   cost[i,2]=80*Ls[i]
33   cost[i,3]=cost[i,1]+cost[i,2]
34 }
35 cost

```

---

#### **R code Exa 16.9.2** Practical problem for MMc GDInfInf model

```

1  ##Chapter 16 : Queuing Systems
2  ##Example 9-2 : Page 670
3
4  #Queueing library to process different queueing
   models
5  library(queueing)
6
7  Ls=numeric()
8  ETC=numeric()
9  for (c in 2:6){
10   #creating a MMc instance with the following
       parameters
11   y=NewInput.MMC(lambda=17.5, mu=10,c)
12   #Solving the model which returns a list
13   z=QueueingModel(y)
14   Ls[c]=z$L
15   ETC[c]=12*c + 50*Ls[c]
16 }
17 ETC[2:6]

```

---

#### **R code Exa 16.9.3** Aspiration level model

```

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

```

```

4 #Queueing library to process different queueing
  models
5 library(queueing)
6 Ws=numeric()
7 R0=numeric()
8
9
10 for (c in 2:8){
11   #creating a MMc instance with the following
    parameters
12   y=NewInput.MMC(lambda=17.5, mu=10,c)
13   #Solving the model which returns a list
14   z=QueueingModel(y)
15   #waiting time in the system
16   Ws[c]=z$W*60
17   #idleness percentage of the system
18   R0[c]=(1-z$R0)*100
19 }
20 rbind(Ws,R0)[,2:8]

```

---

# Chapter 17

## Simulation Modelling

**R code Exa 17.1.1** Monte carlo sampling

```
1 ##Chapter 17 : Simulation Modelling
2 ##Example 1-1 : Page 681
3
4 #number of trials
5 n=1000
6 #Setting up parameters for the plot
7 par(pty="s")
8 #plot empty graph
9 plot(1, axes=T, asp = 1, xlim = c(-4, 6), ylim = c(-3,
10      7))
11 #getting a sample of 1000 random uniform numbers
12     within the given range
13 x=-4+(10)*runif(n)
14 y=-3+10*runif(n)
15 #counting the number of true values for the give
16     expression
17 m=sum((x-1)^2+(y-2)^2<25)
18 #plotting the points on the graph
19 points(x, y)
20 #plotting the circle
21 symbols(x=1,y=2,circles = 5,add = T,inches = F)
```



```

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

```

---

### R code Exa 17.3.3 Erlang distribution

```

1 ##Chapter 17 : Simulation Modelling
2 ##Example 3-3 : Page 691
3
4 #Function which returns an erlang distribution
5 erlang=function(m,lambda){
6   R=c(0.0589,0.6733,0.4799)
7   y=-log(prod(R))/lambda
8   return(y)
9 }
10 erlang(3,4)

```

---

### R code Exa 17.4.1 Multiplicative congruential method

```

1 ##Chapter 17 : Simulation Modelling
2 ##Example 3-3 : Page 691
3
4 #inputs for the random number generator
5 b=9
6 c=5
7 m=12
8
9
10 u=numeric()
11 #initial random number
12 u[1]=11

```

```
13
14 R=numeric()
15 for(i in 2:4){
16     u[i]=(b*u[i-1]+c)%m
17     R[i-1]=u[i]/m
18 }
19 R
```

---

# Chapter 18

## Classical Optimization theory

**R code Exa 18.1.1** Necessary and sufficient conditions

```
1 ##Chapter 18 : Classical Optimization theory
2 ##Example 1-1 : Page 714
3
4 #Function to be given as input to optim function
5 minimize<-function(x){
6   x1=x[1]
7   x2=x[2]
8   x3=x[3]
9   return(-(x1+2*x2+x2*x3-x1^2-x2^2-x3^2))
10 }
11
12 #Calculates the optimal value
13 optim(c(0,0,0),minimize,hessian = T )
14 #the Jacobian for each
15 f=expression(x1+2*x2+x2*x3-x1^2-x2^2-x3^2)
16 for(i in 1:3){
17   x=D(f,paste("x",i,sep = " "))
18   print(x)
19 }
```

---

**R code Exa 18.1.3** Newton Raphson method

```
1 ##Chapter 18 : Classical Optimization theory
2 ##Example 1-3 : Page 717
3
4 #The function expression has to be given as function
  to nlm function
5 g<-function(x){
6   return((3*x-2)^2 * (2*x-3)^2)
7 }
8 #calling nlm function with function g and initial
  guess 0
9 nlm(g,0)
```

---