

R Textbook Companion for
Numerical Methods for Engineers
by S. C. Chapra and R. P. Canale¹

Created by
Bhushan Manjarekar
B.E.
Electronics Engineering
Mumbai University
Cross-Checked by
R TBC Team

August 11, 2020

¹Funded 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: Numerical Methods for Engineers

Author: S. C. Chapra and R. P. Canale

Publisher: McGraw Hill, New York

Edition: 5

Year: 2006

ISBN: 0071244298

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
1 Mathematical Modelling and Engineering Problem Solving	5
3 Approximations and Round off Errors	6
4 Truncation Errors and the Taylor Series	11
5 Bracketing Methods	24
6 Open Methods	32
7 Roots of Polynomials	34
9 Gauss Elimination	41
14 Multidimensional Unconstrained Optimization	47
15 Constrained Optimization	51
17 Least squares regression	56
18 Interpolation	58
19 Fourier Approximation	60
21 Newton Cotes Integration Formulas	65
23 Numerical differentiation	79

25	Runga Kutta methods	81
26	Stiffness and multistep methods	85
27	Boundary Value and Eigenvalue problems	87
31	Finite Element Method	95

List of R Codes

Exa 1.1	Analytical Solution to Falling Parachutist Problem . .	5
Exa 3.1	Calculations of Errors	6
Exa 3.2	Iterative error estimation	7
Exa 3.3	Range of Integers	8
Exa 3.4	Floating Point Numbers	8
Exa 3.5	Machine Epsilon	8
Exa 3.6	Interdependent Computations	9
Exa 3.7	Subtractive Cancellation	9
Exa 3.8	Infinite Series Evaluation	10
Exa 4.1	Polynomial Taylor Series	11
Exa 4.2	Taylor Series Expansion	14
Exa 4.4	Finite divided difference approximation of derivatives .	18
Exa 4.5	Error propagation in function of single variable	20
Exa 4.6	Error propagation in multivariable function	20
Exa 4.7	Condition Number	22
Exa 5.1	Graphical Approach	24
Exa 5.2	Computer Graphics to Locate Roots	25
Exa 5.3	Bisection	25
Exa 5.4	Error Estimates for Bisection	26
Exa 5.5	False Position	28
Exa 5.6	Bracketing and False Position Methods	29
Exa 6.11	Newton Raphson for a nonlinear Problem	32
Exa 7.1	Polynomial Deflation	34
Exa 7.2	Mullers Method	35
Exa 7.3	Bairstows Method	36
Exa 7.4	Locate single root	37
Exa 7.5	Solving nonlinear system	38
Exa 7.6	Root Location	39

Exa 7.7	Roots of Polynomials	39
Exa 7.8	Root Location	40
Exa 9.2	Determinants	41
Exa 9.3	Cramers Rule	42
Exa 9.4	Elimination of Unknowns	42
Exa 9.5	Naive Gauss Elimination	43
Exa 9.6	ill conditioned systems	44
Exa 9.7	Effect of Scale on Determinant	44
Exa 9.8	Scaling	45
Exa 9.11	Solution of Linear Algebraic Equations	45
Exa 14.1	Random Search Method	47
Exa 14.2	Path of Steepest Descent	48
Exa 14.3	1 D function along Gradient	48
Exa 14.4	Optimal Steepest Descent	49
Exa 15.1	Setting up LP problem	51
Exa 15.2	Graphical Solution	51
Exa 15.3	Linear Programming Problem	52
Exa 15.4	Nonlinear constrained optimization	53
Exa 15.5	One dimensional Optimization	54
Exa 15.6	Multidimensional Optimization	55
Exa 15.7	Locate Single Optimum	55
Exa 17.3.a	linear regression using computer	56
Exa 17.3.b	linear regression using computer	57
Exa 18.5	Error Estimates for Order of Interpolation	58
Exa 19.1	Least Square Fit	60
Exa 19.2	Continuous Fourier Series Approximation	61
Exa 19.4	Data Analysis	62
Exa 19.5	Curve Fitting	62
Exa 19.6	Polynomial Regression	63
Exa 21.1	Single trapezoidal rule	65
Exa 21.2	Multiple trapezoidal rule	66
Exa 21.3	Evaluating Integrals	67
Exa 21.4	Single Simpsons 1 by 3 rule	71
Exa 21.5	Multiple Simpsons 1 by 3 rule	72
Exa 21.6	Simpsons 3 by 8 rule	74
Exa 21.7	Unequal Trapezoidal segments	76
Exa 21.8	Simpsons Uneven data	76
Exa 21.9	Average Temperature Determination	77

Exa 23.4	Integration and Differentiation	79
Exa 23.5	Integrate a function	80
Exa 25.4	Solving ODEs	81
Exa 25.11	Solving systems of ODEs	82
Exa 25.14	Adaptive Fourth order RK scheme	83
Exa 26.1	Explicit and Implicit Euler	85
Exa 27.3	Finite Difference Approximation	87
Exa 27.4	Mass Spring System	87
Exa 27.5	Axially Loaded column	88
Exa 27.6	Polynomial Method	89
Exa 27.7	Power Method Highest Eigenvalue	91
Exa 27.8	Power Method Lowest Eigenvalue	92
Exa 27.9	Eigenvalues and ODEs	93
Exa 27.11	Solving ODEs	93
Exa 31.1	Analytical Solution for Heated Rod	95
Exa 31.2	Element Equation for Heated Rod	96

Chapter 1

Mathematical Modelling and Engineering Problem Solving

R code Exa 1.1 Analytical Solution to Falling Parachutist Problem

```
1 g=9.8
2 #m/s^2; acceleration due to gravity
3 m=68.1
4 #kg
5 c=12.5
6 #kg/sec; drag coefficient
7 count=1
8 v = matrix(0,1)
9 for (i in (seq(0,12,2))){
10   v[count]=g*m*(1-exp(-c*i/m))/c
11   cat("v(m/s)=",v[count],"Time(s)=",i)
12   count=count+1;
13 }
14 cat("v(m/s)=",g*m/c,"Time(s)=", "infinity")
```

Chapter 3

Approximations and Round off Errors

R code Exa 3.1 Calculations of Errors

```
1  lbm=9999
2  #cm, measured length of bridge
3  lrm=9
4  #cm, measured length of rivet
5  lbt=10000
6  #cm, true length of bridge
7  lrt=10
8  #cm, true length of rivet
9
10 #calculating true error below;
11 Etb=lbt-lbm
12 #cm, true error in bridge
13 Etr=lrt-lrm
14 #cm, true error in rivet
15
16 #calculating percent relative error below
17 etb=Etb*100/lbt
18 #percent relative error for bridge
19 etr=Etr*100/lrt
```

```

20 #percent relative error for rivet
21 cat("a. The true error is")
22 cat(Etb,"cm","for the bridge")
23 cat(Etr,"cm","for the rivet")
24 cat("b. The percent relative error is")
25 cat(etb,"%","for the bridge")
26 cat(etr,"%","for the rivet")

```

R code Exa 3.2 Iterative error estimation

```

1  n=3
2  #number of significant figures
3  es=0.5*(10^(2-n))
4  #percent, specified error criterion
5  x=0.5;
6  f = matrix(0,1)
7  f[1]=1
8  #first estimate f=e^x = 1
9  ft=1.648721
10 #true value of e^0.5=f
11 et = matrix(0,1)
12 et[1]=(ft-f[1])*100/ft
13 ea = matrix(0,1)
14 ea[1]=100;
15 i=2
16 while (ea[i-1]>=es){
17   f[i]=f[i-1]+(x^(i-1))/(factorial(i-1))
18   et[i]=(ft-f[i])*100/ft
19   ea[i]=(f[i]-f[i-1])*100/f[i]
20   i=i+1
21 }
22 for (j in 1:i-1){
23   cat("term number=",j,"\n","Result=",f[j],"\n","
      True % relative error=",et[j],"\n","Approximate
      estimate of error(%)=",ea[j],"\n")

```

```

24   cat("
      n")
25 }

```

R code Exa 3.3 Range of Integers

```

1  n=16
2  #no of bits
3  num=0
4  for (i in 0:(n-2)){
5    num=num+(1*(2^i))
6  }
7  cat("Thus a 16-bit computer word can store decimal
      integers ranging from",(-1*num),"to",num)

```

R code Exa 3.4 Floating Point Numbers

```

1  n=7
2  #no. of bits
3  #the maximum value of exponents is given by
4  Max=1*(2^1)+1*(2^0)
5  #mantissa is found by
6  mantissa=1*(2^-1)+0*(2^-3)+0*(2^-3)
7  num=mantissa*(2^(Max*-1))
8  #smallest possible positive number for this system
9  cat("The smallest possible positive number for this
      system is",num)

```

R code Exa 3.5 Machine Epsilon

```

1 b=2
2 #base
3 t=3
4 #number of mantissa bits
5 E=2^(1-t)
6 #epsilon
7 cat("value of epsilon=",E)

```

R code Exa 3.6 Interdependent Computations

```

1 readinteger <- function()
2 {
3   n <- readline(prompt="Input a number: ")
4   return(as.integer(n))
5 }
6
7 num<-readinteger()
8
9 sum=0
10 for (i in 1:100000){
11   sum=sum+num
12 }
13 cat("The number summed up 100,000 times is=",sum)

```

R code Exa 3.7 Subtractive Cancellation

```

1 a=1
2 b=3000.001
3 c=3
4 #the roots of the quadratic equation x^2+3000.001*x
   +3=0 are found as
5 D=(b^2)-4*a*c
6 x1=(-b+(D^0.5))/(2*a)

```

```

7 x2=(-b-(D^0.5))/(2*a)
8 cat("The roots of the quadratic equation (x^2)
    +(3000.001*x)+3=0 are = ",x1,"and",x2)

```

R code Exa 3.8 Infinite Series Evaluation

```

1 f <- function(x) {
2   exp(x)
3 }
4
5 sum=1
6 test=0
7 i=0
8 term=1
9 x1= 10
10 x2= -10
11 while (sum~=test){
12   cat("sum:",sum,"\n","term:",term,"\n","i:",i,"\n",
    "-----\n")
13   i=i+1
14   term=term*x/i
15   test=sum
16   sum=sum+term
17 }
18 cat("Exact Value",f(x1))
19 cat("Exact Value",f(x2))

```

Chapter 4

Truncation Errors and the Taylor Series

R code Exa 4.1 Polynomial Taylor Series

```
1 DD <- function(expr, name, order = 1) {
2   if(order < 1) stop("'order' must be >= 1")
3   if(order == 1) D(expr, name)
4   else DD(D(expr, name), name, order - 1)
5 }
6
7 f <- function(x) {
8   return(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2)
9 }
10
11 xi=0
12 xf=1
13 h=xf-xi
14 fi=f(xi)
15 #function value at xi
16 ffa=f(xf)
17 #actual function value at xf
18
19 #for n=0, i.e, zero order approximation
```

```

20 ff=fi
21 Et = matrix(0,5)
22 Et[1]=ffa-ff
23 #truncation error at x=1
24 cat("The value of f at x=0 :",fi,"\n",
25     "The value of f at x=1 due to zero order
      approximation :",ff,"\n",
26     "Truncation error :",Et[1],"\n",
27     "-----\n")
28
29 #for n=1, i.e, first order approximation
30 f1 <- function(x) {
31     return(eval(DD(expr = expression(-0.1*x^4-0.15*x
      ^3-0.5*x^2-0.25*x+1.2),"x",1)))
32 }
33
34 f1i=f1(xi)
35 #value of first derivative of function at xi
36 f1f=fi+f1i*h
37 #value of first derivative of function at xf
38 Et[2]=ffa-f1f
39 #truncation error at x=1
40 cat("The value of first derivative of f at x=0 :",
      f1i,"\n",
41     "The value of f at x=1 due to first order
      approximation :",f1f,"\n",
42     "Truncation error :",Et[2],"\n",
43     "-----\n")
44
45
46 #for n=2, i.e, second order approximation
47 f2 <- function(x) {
48     return(eval(DD(expr = expression(-0.1*x^4-0.15*x
      ^3-0.5*x^2-0.25*x+1.2),"x",2)))
49 }
50

```



```

51
52 f2i=f2(xi)
53 #value of second derivative of function at xi
54 f2f=f1f+f2i*(h^2)/factorial(2)
55 #value of second derivative of function at xf
56 Et[3]=ffa-f2f
57 #truncation error at x=1
58 cat("The value of first derivative of f at x=0 :",
      f2i,"\n",
59      "The value of f at x=1 due to first order
      approximation :",f2f,"\n",
60      "Truncation error :",Et[3],"\n",
61      "-----\n")
62
63 #for n=3, i.e, third order approximation
64 f3 <- function(x) {
65   return(eval(DD(expr = expression(-0.1*x^4-0.15*x
      ^3-0.5*x^2-0.25*x+1.2),"x",3)))
66 }
67 f3i=f3(xi)
68 #value of third derivative of function at xi
69 f3f=f2f+f3i*(h^3)/factorial(3)
70 #value of third derivative of function at xf
71 Et[4]=ffa-f3f
72 #truncation error at x=1
73 cat("The value of first derivative of f at x=0 :",
      f3i,"\n",
74      "The value of f at x=1 due to first order
      approximation :",f3f,"\n",
75      "Truncation error :",Et[4],"\n",
76      "-----\n")
77
78 #for n=4, i.e, fourth order approximation
79 f4 <- function(x) {
80   return(eval(DD(expr = expression(-0.1*x^4-0.15*x
      ^3-0.5*x^2-0.25*x+1.2),"x",4)))

```

```

81 }
82 f4i=f4(xi)
83 #value of fourth derivative of function at xi
84 f4f=f3f+f4i*(h^4)/factorial(4)
85 #value of fourth derivative of function at xf
86 Et[5]=ffa-f4f
87 #truncation error at x=1
88 cat("The value of first derivative of f at x=0 :",
      f4i,"\n",
89      "The value of f at x=1 due to first order
      approximation :",f4f,"\n",
90      "Truncation error :",Et[5],"\n",
91      "-----\n")

```

R code Exa 4.2 Taylor Series Expansion

```

1 DD <- function(expr, name, order = 1) {
2   if(order < 1) stop("'order' must be >= 1")
3   if(order == 1) D(expr, name)
4   else DD(D(expr, name), name, order - 1)
5 }
6
7 f <- function(x) {
8   return(cos(x))
9 }
10
11 pi = 3.1415927
12 et = matrix(0,7)
13
14 xi=pi/4
15 xf=pi/3
16 h=xf-xi
17 fi=f(xi)
18 #function value at xi

```

```

19 ffa=f(xf)
20 #actual function value at xf
21
22 #for n=0, i.e, zero order approximation
23 ff=fi;
24 et[1]=(ffa-ff)*100/ffa
25 #percent relative error at x=1
26 cat("The value of f at x=1 due to zero order
    approximation :",ff,"\n",
27      "% relative error :",et[1],"\n",
28      "-----\n"
    n")
29
30
31 #for n=1, i.e, first order approximation
32 f1 <- function(x) {
33     return(eval(DD(expr = expression(cos(x)), "x", 1)))
34 }
35 f1i=f1(xi)
36 #value of first derivative of function at xi
37 f1f=fi+f1i*h
38 #value of first derivative of function at xf
39 et[2]=(ffa-f1f)*100/ffa
40 #% relative error at x=1
41 cat("The value of f at x=1 due to first order
    approximation :",f1f,"\n",
42      "% relative error :",et[2],"\n",
43      "-----\n"
    n")
44
45
46 #for n=2, i.e, second order approximation
47 f2 <- function(x) {
48     return(eval(DD(expr = expression(cos(x)), "x", 2)))
49 }
50 f2i=f2(xi)
51 #value of second derivative of function at xi
52 f2f=f1f+f2i*(h^2)/factorial(2)

```

```

53 #value of second derivative of function at xf
54 et[3]=(ffa-f2f)*100/ffa
55 #% relative error at x=1
56 cat("The value of f at x=1 due to second order
    approximation :",f2f,"\n",
57     "% relative error :",et[3],"\n",
58     "-----\n")
59
60 #for n=3, i.e, third order approximation
61 f3 <- function(x) {
62     return(eval(DD(expr = expression(cos(x)),"x",3)))
63 }
64 f3i=f3(xi)
65 #value of third derivative of function at xi
66 f3f=f2f+f3i*(h^3)/factorial(3)
67 #value of third derivative of function at xf
68 et[4]=(ffa-f3f)*100/ffa
69 #% relative error at x=1
70 cat("The value of f at x=1 due to third order
    approximation :",f3f,"\n",
71     "% relative error :",et[4],"\n",
72     "-----\n")
73
74 #for n=4, i.e, fourth order approximation
75 f4 <- function(x) {
76     return(eval(DD(expr = expression(cos(x)),"x",4)))
77 }
78 f4i=f4(xi)
79 #value of fourth derivative of function at xi
80 f4f=f3f+f4i*(h^4)/factorial(4)
81 #value of fourth derivative of function at xf
82 et[5]=(ffa-f4f)*100/ffa
83 #% relative error at x=1
84 cat("The value of f at x=1 due to fourth order
    approximation :",f4f,"\n",
85     "% relative error :",et[5],"\n",

```

```

86     "-----\
      n")
87
88
89 #for n=5, i.e, fifth order approximation
90 f5i=(f4(1.1*xi)-f4(0.9*xi))/(2*0.1)
91 #value of fifth derivative of function at xi (
    central difference method)
92 f5f=f4f+f5i*(h^5)/factorial(5)
93 #value of fifth derivative of function at xf
94 et[6]=(ffa-f5f)*100/ffa
95 #% relative error at x=1
96 cat("The value of f at x=1 due to fifth order
    approximation :",f5f,"\n",
97     "% relative error :",et[6],"\n",
98     "-----\
      n")
99
100
101 #for n=6, i.e, sixth order approximation
102 f6 <- function(x) {
103     return(eval(DD(expr = expression(cos(x)), "x", 4)))
104 }
105 f6i=(f4(1.1*xi)-2*f4(xi)+f4(0.9*xi))/(0.1^2)
106 #value of sixth derivative of function at xi (
    central difference method)
107 f6f=f5f+f6i*(h^6)/factorial(6)
108 #value of sixth derivative of function at xf
109 et[7]=(ffa-f6f)*100/ffa
110 #% relative error at x=1
111 cat("The value of f at x=1 due to sixth order
    approximation :",f6f,"\n",
112     "% relative error :",et[7],"\n",
113     "-----\
      n")

```

R code Exa 4.4 Finite divided difference approximation of derivatives

```
1 DD <- function(expr, name, order = 1) {
2   if(order < 1) stop("'order' must be >= 1")
3   if(order == 1) D(expr, name)
4   else DD(D(expr, name), name, order - 1)
5 }
6
7 f <- function(x) {
8   return(-0.1*(x^4)-0.15*(x^3)-0.5*(x^2)-0.25*(x)
9         +1.2)
10 }
11 x=0.5
12 h= 0.5
13 x1=x-h
14 x2=x+h
15 #forward difference method
16 fdx1=(f(x2)-f(x))/h
17 #derivative at x
18 et1=abs((fdx1-eval(DD(expr = expression(-0.1*(x^4)
19   -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2), name = "x",
20   order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
21   -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2), name = "x",
22   order = 1)))*100
23 #backward difference method
24 fdx2=(f(x)-f(x1))/h
25 #derivative at x
26 et2=abs((fdx2-eval(DD(expr = expression(-0.1*(x^4)
27   -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2), name = "x",
28   order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
29   -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2), name = "x",
30   order = 1)))*100
31 #central difference method
```

```

24 fdx3=(f(x2)-f(x1))/(2*h)
25 #derivative at x
26 et3=abs((fdx3-eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))*100
27 cat("For h=",h,"\n",
28      "Derivative at x by forward difference method=",
      fdx1,"and percent error=",et1,"\n",
29      "Derivative at x by backward difference method=",
      fdx2,"and percent error=",et2,"\n",
30      "Derivative at x by central difference method=",
      fdx3,"and percent error=",et3,"\n")
31
32
33 h= 0.25
34 x1=x-h
35 x2=x+h
36 #forward difference method
37 fdx1=(f(x2)-f(x))/h
38 #derivative at x
39 et1=abs((fdx1-eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))*100
40 #backward difference method
41 fdx2=(f(x)-f(x1))/h
42 #derivative at x
43 et2=abs((fdx2-eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
      -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
      order = 1)))*100
44 #central difference method
45 fdx3=(f(x2)-f(x1))/(2*h)
46 #derivative at x

```

```

47 et3=abs((fdx3-eval(DD(expr = expression(-0.1*(x^4)
    -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
    order = 1)))/eval(DD(expr = expression(-0.1*(x^4)
    -0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2),name = "x",
    order = 1)))*100
48 cat("For h=",h,"\n",
49     "Derivative at x by forward difference method=",
    fdx1,"and percent error=",et1,"\n",
50     "Derivative at x by backward difference method=",
    fdx2,"and percent error=",et2,"\n",
51     "Derivative at x by central difference method=",
    fdx3,"and percent error=",et3,"\n")

```

R code Exa 4.5 Error propagation in function of single variable

```

1 DD <- function(expr, name, order = 1) {
2   if(order < 1) stop("'order' must be >= 1")
3   if(order == 1) D(expr, name)
4   else DD(D(expr, name), name, order - 1)
5 }
6
7 x=2.5
8 delta=0.01
9 deltafx=abs(eval(DD(expr = expression(x^3),name = "x",
    order = 1)))*delta
10 fx=f(x)
11 cat("true value is between",fx-deltafx,"and",fx+
    deltafx)

```

R code Exa 4.6 Error propagation in multivariable function

```

1 library(Deriv)
2

```



```

3 DD <- function(expr, name, order = 1) {
4   if(order < 1) stop("'order' must be >= 1")
5   if(order == 1) D(expr, name)
6   else DD(D(expr, name), name, order - 1)
7 }
8
9 f <- function(F,L,E,I) {
10  (F*(L^4))/(8*E*I)
11 }
12
13 Fbar=50
14 #lb/ft
15 Lbar=30
16 #ft
17 Ebar=1.5*(10^8)
18 #lb/ft^2
19 Ibar=0.06
20 #ft^4
21 deltaF=2
22 #lb/ft
23 deltaL=0.1
24 #ft
25 deltaE=0.01*(10^8)
26 #lb/ft^2
27 deltaI=0.0006
28 #ft^4
29 ybar=(Fbar*(Lbar^4))/(8*Ebar*Ibar)
30
31 f1 <- function(F) {
32  (F*(Lbar^4))/(8*Ebar*Ibar)
33 }
34 f_1<-Deriv(f1)
35 f2 <- function(L) {
36  (Fbar*(L^4))/(8*Ebar*Ibar)
37 }
38 f_2<-Deriv(f2)
39 f3 <- function(E) {
40  (Fbar*(Lbar^4))/(8*E*Ibar)

```

```

41 }
42 f_3<-Deriv(f3)
43 f4 <- function(I) {
44   (Fbar*(Lbar^4))/(8*Ebar*I)
45 }
46 f_4<-Deriv(f4)
47 deltay=abs(f_1(Fbar))*deltaF+
48   abs(f_2(Lbar))*deltaL+
49   abs(f_3(Ebar))*deltaE+
50   abs(f_4(Ibar))*deltaI;
51
52 cat("The value of y is between:",ybar-deltay,"and",
     ybar+deltay)
53 ymin=((Fbar-deltaF)*((Lbar-deltaL)^4))/(8*(Ebar+
     deltaE)*(Ibar+deltaI));
54 ymax=((Fbar+deltaF)*((Lbar+deltaL)^4))/(8*(Ebar-
     deltaE)*(Ibar-deltaI));
55 cat("ymin is calculated at lower extremes of F, L, E
     , I values as =",ymin)
56 cat("ymax is calculated at higher extremes of F, L,
     E, I values as =",ymax)

```

R code Exa 4.7 Condition Number

```

1 library(Deriv)
2
3 f <- function(x) {
4   tan(x)
5 }
6
7 f_ = Deriv(f)
8
9 pi = 3.1415927
10 x1bar=(pi/2)+0.1*(pi/2)
11 x2bar=(pi/2)+0.01*(pi/2)

```

```

12 #computing condition number for x1bar
13 condnum1=x1bar*f_(x1bar)/f(x1bar)
14 cat("The condition number of function for x=",x1bar,
      " is:",condnum1)
15 if (abs(condnum1)>1){
16     cat("Function is ill-conditioned for x=",x1bar)
17 }
18 #computing condition number for x2bar
19 condnum2=x2bar*f_(x2bar)/f(x2bar)
20 cat("The condition number of function for x=",x2bar,
      " is:",condnum2)
21 if (abs(condnum2)>1){
22     cat("Function is ill-conditioned for x=",x2bar)
23 }

```

Chapter 5

Bracketing Methods

R code Exa 5.1 Graphical Approach

```
1  m=68.1
2  #kg
3  v=40
4  #m/s
5  t=10
6  #s
7  g=9.8
8  #m/s^2
9
10 f <- function(c) {
11   g*m*(1-exp(-c*t/m))/c - v
12 }
13
14 cat("For various values of c and f(c) is found as:")
15 i=0
16 fc = matrix(0,5)
17 for (c in seq(4,20,4)){
18   i=i+1
19   bracket=c(c, f(c))
20   cat(bracket)
21   fc[i]=f(c)
```

```

22 }
23 c<-c(4, 8, 12, 16, 20)
24 plot(c,fc,main = 'f(c) vs c',xlab = 'c',ylab = 'f(c)
    (m/s)')
25 lines(c,fc)

```

R code Exa 5.2 Computer Graphics to Locate Roots

```

1 f <- function(x) {
2   sin(10*x)+cos(3*x)
3 }
4
5 count=1
6 val = matrix(0,100)
7 func = matrix(0,100)
8 for (i in seq(1,5,0.05)){
9   val[count]=i
10  func[count]=f(i)
11  count=count+1
12 }
13 plot(val,func,main = "x vs f(x)",xlab = 'x',ylab = '
    f(x)')
14 lines(val,func)

```

R code Exa 5.3 Bisection

```

1 m=68.1
2 #kg
3 v=40
4 #m/s
5 t=10
6 #s
7 g=9.8

```

```

8 #m/s^2
9
10 f <- function(c) {
11   g*m*(1-exp(-c*t/m))/c - v
12 }
13
14 x1=12
15 x2=16
16 xt=14.7802
17 #true value
18 #”enter the tolerable true percent error=”
19 e=2
20 xr=(x1+x2)/2
21 etemp=abs(xr-xt)/xt*100
22 #error
23 while (etemp>e){
24   if (f(x1)*f(xr)>0){
25     x1=xr
26     xr=(x1+x2)/2
27     etemp=abs(xr-xt)/xt*100
28   }
29   if (f(x1)*f(xr)<0){
30     x2=xr
31     xr=(x1+x2)/2
32     etemp=abs(xr-xt)/xt*100
33   }
34   if (f(x1)*f(xr)==0) {
35     break
36   }
37 }
38 cat("The result is",xr)

```

R code Exa 5.4 Error Estimates for Bisection

```
1 m=68.1
```

```

2 #kg
3 v=40
4 #m/s
5 t=10
6 #s
7 g=9.8
8 #m/s^2
9
10 f <- function(c) {
11   g*m*(1-exp(-c*t/m))/c - v
12 }
13
14 x1=12
15 x2=16
16 xt=14.7802
17 #true value
18 #”enter the tolerable approximate error=”
19 e=0.5
20 xr=(x1+x2)/2
21 i=1
22 et=abs(xr-xt)/xt*100
23 #error
24 cat(" Iteration :",i)
25 cat(" x1:",x1)
26 cat(" x2:",x2)
27 cat(" xr:",xr)
28 cat(" et(%):" ,et)
29 cat("_____")
30 etemp=100
31
32 while (etemp>e){
33   if (f(x1)*f(xr)>0){
34     x1=xr
35     xr=(x1+x2)/2
36     etemp=abs(xr-x1)/xr*100
37     et=abs(xr-xt)/xt*100
38   }
39   if (f(x1)*f(xr)<0){

```

```

40     x2=xr
41     xr=(x1+x2)/2
42     etemp=abs(xr-x2)/xr*100
43     et=abs(xr-xt)/xt*100
44 }
45 if (f(x1)*f(xr)==0){
46     break
47 }
48 i=i+1
49 cat(" Iteration:",i)
50 cat(" x1:",x1)
51 cat(" xu:",x2)
52 cat(" xr:",xr)
53 cat(" et(%):" ,et)
54 cat(" ea(%)",etemp)
55 cat("-----")
56 }
57 cat("The result is=",xr)

```

R code Exa 5.5 False Position

```

1  m=68.1
2  #kg
3  v=40
4  #m/s
5  t=10
6  #s
7  g=9.8
8  #m/s^2
9
10 f <- function(c) {
11     g*m*(1-exp(-c*t/m))/c - v
12 }
13
14 x1=12

```



```

15 x2=16
16 xt=14.7802
17 #true value
18 #”enter the tolerable true percent error=”
19 e=
20 xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
21 etemp=abs(xr-xt)/xt*100
22 #error
23 while (etemp>e){
24   if (f(x1)*f(xr)>0){
25     x1=xr
26     xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
27     etemp=abs(xr-xt)/xt*100
28   }
29   if (f(x1)*f(xr)<0){
30     x2=xr
31     xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
32     etemp=abs(xr-xt)/xt*100
33   }
34   if (f(x1)*f(xr)==0){
35     break
36   }
37 }
38 cat("The result is=",xr)

```

R code Exa 5.6 Bracketing and False Position Methods

```

1 f <- function(x) {
2   x^10 - 1
3 }
4
5 x1=0
6 x2=1.3
7 xt=1
8

```

```

9 #using bisection method
10 cat("BISECTION METHOD:")
11 xr=(x1+x2)/2
12 et=abs(xr-xt)/xt*100
13 #error
14 cat("Iteration:",1,"\n", "xl:",x1,"\n", "xu:",x2,"\n",
    "xr:",xr,"\n", "et(%):",et,"\n",
    "-----\n")
15
16 for (i in 2:5){
17     if (f(x1)*f(xr)>0){
18         x1=xr
19         xr=(x1+x2)/2
20         ea=abs(xr-x1)/xr*100
21         et=abs(xr-xt)/xt*100
22     } else if (f(x1)*f(xr)<0){
23         x2=xr
24         xr=(x1+x2)/2
25         ea=abs(xr-x2)/xr*100
26         et=abs(xr-xt)/xt*100
27     }
28
29     if (f(x1)*f(xr)==0){
30         break
31     }
32 cat("Iteration:",i,"\n")
33 cat("xl:",x1,"\n")
34 cat("xu:",x2,"\n")
35 cat("xr:",xr,"\n")
36 cat("et(%):",et,"\n")
37 cat("ea(%)",ea,"\n")
38 cat("-----\n")
39 }
40
41 #using false position method
42 cat("FALSE POSITION METHOD:")
43 x1=0
44 x2=1.3

```

```

45 xt=1
46 xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
47 et=abs(xr-xt)/xt*100
48 #error
49 cat("Iteration:",1,"\n","xl:",x1,"\n","xu:",x2,"\n",
    "xr:",xr,"\n","et(%):",et,"\n",
    "-----")
50
51 for (i in 2:5){
52     if (f(x1)*f(xr)>0){
53         x1=xr
54         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
55         ea=abs(xr-x1)/xr*100
56         et=abs(xr-xt)/xt*100
57     }
58     else if (f(x1)*f(xr)<0){
59         x2=xr
60         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
61         ea=abs(xr-x2)/xr*100
62         et=abs(xr-xt)/xt*100
63     }
64     if (f(x1)*f(xr)==0){
65         break
66     }
67 cat("Iteration:",i,"\n")
68 cat("xl:",x1,"\n")
69 cat("xu:",x2,"\n")
70 cat("xr:",xr,"\n")
71 cat("et(%):",et,"\n")
72 cat("ea(%)",ea,"\n")
73 cat("-----\n")
74 }

```

Chapter 6

Open Methods

R code Exa 6.11 Newton Raphson for a nonlinear Problem

```
1 u <- function(x,y) {
2   x^2+x*y-10
3 }
4
5 v <- function(x,y) {
6   y+3*x*y^2-57
7 }
8
9 x=1.5
10 y=3.5
11 e<-c(100, 100)
12 while (e[1]>0.0001 & e[2]>0.0001){
13   J=matrix(data = c(2*x+y, x, 3*y^2, 1+6*x*y),nrow =
14     2,ncol = 2,byrow = TRUE)
15   deter=det(J)
16   u1=u(x,y)
17   v1=v(x,y)
18   x=x-((u1*J[2,2]-v1*J[1,2])/deter)
19   y=y-((v1*J[1,1]-u1*J[2,1])/deter)
20   e[1]=abs(2-x)
21   e[2]=abs(3-y)
```

```
21 }  
22 bracket<-c(x, y)  
23 cat(bracket)
```

Chapter 7

Roots of Polynomials

R code Exa 7.1 Polynomial Deflation

```
1 f <- function(x) {  
2   (x-4)*(x+6)  
3 }  
4  
5 n=2  
6 a = matrix(0,3)  
7 a[1]=-24  
8 a[2]=2  
9 a[3]=1  
10 t=4  
11 r=a[3]  
12 a[3]=0  
13 for (i in seq(n,1,-1)){  
14   s=a[i]  
15   a[i]=r  
16   r=s+r*t  
17 }  
18 cat("The quptient is a(1)+a(2)*x where :", "a(1)=", a  
    [1], "a(2)=", a[2], "remainder=", r)
```

R code Exa 7.2 Mullers Method

```
1 f <- function(x) {
2   x^3 - 13*x - 12
3 }
4
5 x1t=-3
6 x2t=-1
7 x3t=4
8 x0=4.5
9 x1=5.5
10 x2=5
11
12 cat("iteration:",0,"\n","xr:",x2,"
13
14 for (i in 1:4){
15   h0=x1-x0
16   h1=x2-x1
17   d0=(f(x1)-f(x0))/(x1-x0)
18   d1=(f(x2)-f(x1))/(x2-x1)
19   a=(d1-d0)/(h1+h0)
20   b=a*h1+d1
21   c=f(x2)
22   d=(b^2 - 4*a*c)^0.5
23   if (abs(b+d)>abs(b-d)){
24     x3=x2+((-2*c)/(b+d))
25 }else {
26     x3=x2+((-2*c)/(b-d))
27   }
28   ea=abs(x3-x2)*100/x3
29   x0=x1
30   x1=x2
31   x2=x3
```

```

32   cat(" iteration:",i,"\n")
33   cat(" xr:",x2,"\n")
34   cat(" ea(%):" ,ea,"\n")
35   cat("
                                     \n
      ")
36 }

```

R code Exa 7.3 Bairstows Method

```

1  f <- function(x) {
2    x^5-3.5*x^4+2.75*x^3+2.125*x^2-3.875*x+1.25
3  }
4
5  r=-1
6  s=-1
7  es=1
8  #%
9  n=6
10 count=1
11 ear=100
12 eas=100
13 a<-c(1.25, -3.875, 2.125, 2.75, -3.5, 1)
14 b<-matrix(0,n)
15 c<-matrix(0,n)
16 while ((ear>es) & (eas>es)){
17   b[n]=a[n]
18   b[n-1]=a[n-1]+r*b[n]
19   for (i in seq(n-2,1,-1)){
20     b[i]=a[i]+r*b[i+1]+s*b[i+2]
21   }
22   c[n]=b[n]
23   c[n-1]=b[n-1]+r*c[n]
24   for (i in seq((n-2),2,-1)){
25     c[i]=b[i]+r*c[i+1]+s*c[i+2]

```



```

26   }
27   #c(3)*dr+c(4)*ds=-b(2)
28   #c(2)*dr+c(3)*ds=-b(1)
29   ds=((-b[1])+(b[2]*c[2]/c[3]))/(c[3]-(c[4]*c[2]/c[3])
      )
30   dr=(-b[2]-c[4]*ds)/c[3]
31   r=r+dr
32   s=s+ds
33   ear=abs(dr/r)*100
34   eas=abs(ds/s)*100
35   cat("Iteration:",count,"\n","delata r:",dr,"\n","
      delata s:",ds,"\n","r:",r,"\n","s:",s,"\n","Error
      in r:",ear,"\n","Error in s:",eas,"\n","
      _____\
      n")
36   count=count+1;
37   }
38   x1=(r+(r^2 + 4*s)^0.5)/2
39   x2=(r-(r^2 + 4*s)^0.5)/2
40   bracket<-c(x1, x2)
41   cat("The roots are:",bracket,"The quotient is:",x^3
      - 4*x^2 + 5.25*x - 2.5","\n","
      _____\
      n")
_____

```

R code Exa 7.4 Locate single root

```

1 f <- function(x) {
2   x-cos(x)
3 }
4
5 x1=0
6
7 if (f(x1)<0){
8   x2=x1+0.001

```

```

9   while (f(x2)<0){
10     x2=x2+0.001
11   }
12 } else if(f(x1)>0){
13   x2=x1+0.001
14   while (f(x2)>0){
15     x2=x2+0.001
16   }
17 } else{
18   cat("The root is=",x1)
19 }
20
21 x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1))
22 cat("The root is=",x)

```

R code Exa 7.5 Solving nonlinear system

```

1 u <- function(x,y) {
2   x^2+x*y-10
3 }
4
5 v <- function(x,y) {
6   y+3*x*y^2-57
7 }
8
9 x=1
10 y=3.5
11 while (u(x,y)!=v(x,y)){
12   x=x+1
13   y=y-0.5
14 }
15 cat("x=",x)
16 cat("y=",y)

```

R code Exa 7.6 Root Location

```
1 library(pracma)
2 fun <- function (x) x^10 -1
3 fzero(f = fun,x = c(0,4))
4 fzero(f = fun,x = c(0,1.3))
5 fzero(f = fun,x = c(-1.3,0))
6 fzero(f = fun,x = c(-1.28, 0.9051))
```

R code Exa 7.7 Roots of Polynomials

```
1 library(pracma)
2 library(polynom)
3
4 fun <- function (x) (x^5 - (3.5*x^4) +(2.75*x^3)
   +(2.125*x^2) - (3.875*x) + 1.25)
5 fzero(f = fun,x =1)
6 Deriv::Deriv(f = fun,x = "x")
7
8 b<-c(1,0.5,-0.5)
9 a<-c(1,-3.5,2.75,2.125,-3.875,1.25)
10 answer = deconv(a,b)
11 d = answer$q
12 e = answer$r
13 polyroot(a)
14 polyroot(d)
15 conv(d,b)
16 a<-conv(d,b)
17 polyroot(a)
```

R code Exa 7.8 Root Location

```
1 f <- function(x) {  
2   x-cos(x)  
3 }  
4  
5 x1=0  
6 if (f(x1)<0){  
7   x2=x1+0.00001  
8   while (f(x2)<0){  
9     x2=x2+0.00001  
10  }  
11 } else if (f(x1)>0){  
12   x2=x1+0.00001  
13   while (f(x2)>0){  
14     x2=x2+0.00001  
15   }  
16 } else {  
17   cat("The root is=",x1)  
18 }  
19  
20  
21 x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1))  
22 cat("The root is=",x)
```

Chapter 9

Gauss Elimination

R code Exa 9.2 Determinants

```
1 #For fig9.1
2 a= matrix(data = c(3, 2,-1, 2),nrow = 2,ncol = 2,
  byrow = TRUE)
3 cat("The value of determinant for system represented
  in fig 9.1 =",det(a))
4 #For fig9.2 (a)
5 a= matrix(data = c(-0.5, 1,-0.5, 1),nrow = 2,ncol =
  2,byrow = TRUE)
6 cat("The value of determinant for system represented
  in fig 9.2 (a) =",det(a))
7 #For fig9.2 (b)
8 a= matrix(data = c(-0.5, 1,-1, 2),nrow = 2,ncol = 2,
  byrow = TRUE)
9 cat("The value of determinant for system represented
  in fig 9.2 (b) =",det(a))
10 #For fig9.2 (c)
11 a= matrix(data = c(-0.5, 1,-2.3/5, 1),nrow = 2,ncol
  = 2,byrow = TRUE)
12 cat("The value of determinant for system represented
  in fig 9.2 (c) =",det(a))
```

R code Exa 9.3 Cramers Rule

```
1 #the matrix or the system
2 b1=-0.01
3 b2=0.67
4 b3=-0.44
5 a<-matrix(data = c(0.3, 0.52, 1,0.5, 1, 1.9,0.1,
6   0.3, 0.5),nrow = 3,ncol = 3,byrow = TRUE)
7 a1<-matrix(data = c(a[2,2], a[2,3],a[3,2], a[3,3]),
8   nrow = 2,ncol = 2,byrow = TRUE)
9 A1=det(a1)
10 a2<-matrix(data = c(a[2,1], a[2,3],a[3,1], a[3,3]),
11   nrow = 2,ncol = 2,byrow = TRUE)
12 A2=det(a2)
13 a3<-matrix(data = c(a[2,1], a[2,2],a[3,1], a[3,2]),
14   nrow = 2,ncol = 2,byrow = TRUE)
15 A3=det(a3)
16 D=a[1,1]*A1-a[1,2]*A2+a[1,3]*A3
17 p<-matrix(data = c(b1, 0.52, 1,b2, 1, 1.9,b3, 0.3,
18   0.5),nrow = 3,ncol = 3,byrow = TRUE)
19 q<-matrix(data = c(0.3, b1, 1,0.5, b2, 1.9,0.1, b3,
20   0.5),nrow = 3,ncol = 3,byrow = TRUE)
21 r<-matrix(data = c(0.3, 0.52, b1,0.5, 1 ,b2,0.1,
22   0.3, b3),nrow = 3,ncol = 3,byrow = TRUE)
23 x1=det(p)/D
24 x2=det(q)/D
25 x3=det(r)/D
26 cat("The values are:", "x1=",x1," ,x2=",x2," ,x3=",x3)
```

R code Exa 9.4 Elimination of Unknowns

```

1 #the equations are:
2 #3*x1+2*x2=18
3 #-x1+2*x2=2
4 a11=3
5 a12=2
6 b1=18
7 a21=-1
8 a22=2
9 b2=2
10 x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
11 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
12 cat("x1=",x1)
13 cat("x2=",x2)

```

R code Exa 9.5 Naive Gauss Elimination

```

1 n=3
2 b<-matrix(c(7.85,-19.3,71.4), nrow = 1, ncol = 3)
3 a<-matrix(data = c(3, -0.1, -0.2,0.1, 7 , -0.3,0.3,
  -0.2, 10),nrow = 3,ncol = 3,byrow = TRUE)
4 for (k in 1:1){
5   for (i in 2:3){
6     fact=a[i,k]/a[k,k]
7     for (j in 2:3){
8       a[i,j]=a[i,j]-fact*a[k,j]
9     }
10    b[i]=b[i]-fact*b[k]
11    print(b)
12  }
13 }
14 x<-matrix(0,3)
15 x[3]=b[3]/a[3,3]
16 for (i in seq(2,1,-1)){
17   s=b[i]
18   for (j in (i+1):3){

```

```

19     s=s-a[i,j]*x[j]
20     print(s)
21 }
22 x[i]=b[i]/a[i,i]
23 }
24 cat("x1=",x[1]," ,x2=",x[2]," ,x3=",x[3])

```

R code Exa 9.6 ill conditioned systems

```

1 a11=1
2 a12=2
3 b1=10
4 a21=1.1
5 a22=2
6 b2=10.4
7 x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
8 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
9 cat("For the original system:", "x1=",x1," ,x2=",x2)
10 a21=1.05
11 x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
12 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
13 cat("For the new system:", "x1=",x1," ,x2=",x2)

```

R code Exa 9.7 Effect of Scale on Determinant

```

1 #part a
2 a<-matrix(c(3, 2,-1, 2), nrow = 2, ncol = 2,byrow =
      TRUE)
3 b1=18
4 b2=2
5 cat("The determinant for part(a)=",det(a))
6 #part b

```



```

7 a<-matrix(c(1, 2,1.1, 2), nrow = 2, ncol = 2,byrow =
  TRUE)
8 b1=10
9 b2=10.4
10 cat("The determinant for part(b)=",det(a))
11 #part c
12 a1=a*10
13 b1=100
14 b2=104
15 cat("The determinant for part(c)=",det(a1))

```

R code Exa 9.8 Scaling

```

1 #part a
2 a<-matrix(c(1, 0.667,-0.5, 1), nrow = 2, ncol = 2,
  byrow = TRUE)
3 b1=6
4 b2=1
5 cat("The determinant for part(a)=",det(a))
6 #part b
7 a<-matrix(c(0.5, 1,0.55, 1), nrow = 2, ncol = 2,
  byrow = TRUE)
8 b1=5
9 b2=5.2
10 cat("The determinant for part(b)=",det(a))
11 #part c
12 b1=5
13 b2=5.2
14 cat("The determinant for part(c)=",det(a))

```

R code Exa 9.11 Solution of Linear Algebraic Equations

```

1 a<-matrix(c(70, 1, 0,60, -1, 1,40, 0 ,-1),nrow = 3,
            ncol = 3,byrow = TRUE)
2 b<-matrix(c(636,518,307),nrow = 3,ncol = 1,byrow =
            TRUE)
3 x=abs(solve(a,b))
4 cat("a=",x[1],"m/s ^2", "\n", "T=", x[2], "N", "\n", "R=", x
      [3], "N", "\n")

```

Chapter 14

Multidimensional Unconstrained Optimization

R code Exa 14.1 Random Search Method

```
1 maxf = -1e+09
2
3 n=10000
4 for (j in 1:n){
5   Rnd=runif(2)
6   x = -2 + 4 * Rnd[1]
7   y = 1 + 2 * Rnd[2]
8   fn = y - x - (2 * (x ^ 2)) - (2 * x * y) - (y ^ 2)
9   if (fn > maxf){
10     maxf = fn
11     maxx = x
12     maxy = y
13   }
14   if (mod(j,1000)==0){
15     cat(" Iteration:",j,"\n")
16     cat("x:",x,"\n")
17     cat("y:",y,"\n")
18     cat("function value:",fn,"\n")
19     cat("-----\n")
```

```

        n")
20   }
21 }

```

R code Exa 14.2 Path of Steepest Descent

```

1  f <- function(x,y) {
2    x*y*y
3  }
4
5  p1<-c(2, 2)
6  elevation=f(p1[1],p1[2])
7  dfx=p1[1]*p1[1]
8  dfy=2*p1[1]*p1[2]
9  theta=atan(dfy/dfx)
10 slope=(dfx^2 + dfy^2)^0.5
11 cat("Elevation:",elevation,"Theta:",theta,"slope:",
      slope)

```

R code Exa 14.3 1 D function along Gradient

```

1  f <- function(x,y) {
2    2*x*y + 2*x - x^2 - 2*y^2
3  }
4
5  x=-1
6  y=1
7  dfx=2*y+2-2*x
8  dfy=2*x-4*y
9  #the function can thus be expressed along h axis as
10 #f((x+dfx*h),(y+dfy*h))
11 cat("The final equation is=", "180*h^2 + 72*h - 7")

```

R code Exa 14.4 Optimal Steepest Descent

```

1  f <- function(x,y) {
2    2*x*y + 2*x - x^2 - 2*y^2
3  }
4
5  x=-1
6  y=1
7  d2fx=-2
8  d2fy=-4
9  d2fxy=2
10
11 modH=d2fx*d2fy-(d2fxy)^2
12
13 for (i in 1:25){
14   dfx=2*y+2-2*x
15   dfy=2*x - 4*y
16   #the function can thus be expressed along h axis
17   as
18   #f((x+dfx*h),(y+dfy*h))
19   g <- function(h) {
20     2*(x+dfx*h)*(y+dfy*h) + 2*(x+dfx*h) - (x+dfx*h)
21     ^2 - 2*(y+dfy*h)^2
22   }
23   #2*dfx*(y+dfy*h)+2*dfy*(x+dfx*h)+2*dfx-2*(x+dfx*h)
24   #*dfx-4*(y+dfy*h)*dfy=g'(h)=0
25   #2*dfx*y + 2*dfx*dfy*h + 2*dfy*x + 2*dfy*dfx*h + 2
26   #*dfx - 2*x*dfx - 2*dfx*dfx*h - 4*y*dfy - 4*dfy*
27   #dfy*h=0
28   #h(2*dfx*dfy+2*dfy*dfx-2*dfx*dfx-4*dfy*dfy)=-(2*
29   #dfx*y+2*dfy*x-2*x*dfx-4*y*dfy)
30   h=(2*dfx*y+2*dfy*x-2*x*dfx-4*y*dfy+2*dfx)/(-1*(2*
31   dfx*dfy+2*dfy*dfx-2*dfx*dfx-4*dfy*dfy))
32   x=x+dfx*h

```

```
26     y=y+dfy*h
27 }
28 cat("The final values are:",x,"", y)
```

Chapter 15

Constrained Optimization

R code Exa 15.1 Setting up LP problem

```
1 regular<-c(7, 10, 9 ,150)
2 premium<-c(11, 8 ,6 ,175)
3 res_avail<-c(77, 80)
4 #total profit(to be maximized)=z=150*x1+175*x2
5 #total gas used=7*x1+11*x2 (has to be less than 77 m
  ^3/week)
6 #similarly other constraints are developed
7 cat("Maximize z=150*x1+175*x2")
8 cat("subject to")
9 cat("7*x1+11*x2<=77 (Material constraint)")
10 cat("10*x1+8*x2<=80 (Time constraint)")
11 cat("x1<=9 (Regular storage constraint)")
12 cat("x2<=6 (Premium storage constraint)")
13 cat("x1,x2>=0 (Positivity constraint)")
```

R code Exa 15.2 Graphical Solution

```
1 x21<-matrix(0,8)
```

```

2 x22<-matrix(0,8)
3 x23<-matrix(0,8)
4 x24<-matrix(0,8)
5 x25<-matrix(0,8)
6 x26<-matrix(0,8)
7 for (x1 in 0:8){
8   x21[x1+1]=-(7/11)*x1+7
9   x22[x1+1]=(80-10*x1)/8
10  x23[x1+1]=6
11  x24[x1+1]=-150*x1/175
12  x25[x1+1]=(600-150*x1)/175
13  x26[x1+1]=(1400-150*x1)/175
14 }
15 x1=0:8
16
17
18 plot(x1,x24,main = 'Z=0')
19 lines(x1,x25,main = 'Z=600')
20 lines(x1,x26,main = 'Z=1400')
21 plot(x1,x21,main = 'x2 vs x1')
22 plot(x1,x22,xlab = 'x1 (tonnes)')
23 plot(x1,x23,ylab = 'x2 (tonnes)')

```

R code Exa 15.3 Linear Programming Problem

```

1 x1<-c(4.888889, 3.888889)
2 x2<-c(7, 11)
3 x3<-c(10, 8)
4 x4<-c(150, 175)
5 x5<-c(77, 80, 9, 6)
6 profit<-c(x1[1]*x4[1], x1[2]*x4[2])
7 total<-c(x1[1]*x3[1]+x1[2]*x3[2], x1[1]*x3[1]+x1[2]*
8           x3[2], x1[1], x1[2], profit[1]+profit[2])
9 e=1000

```



```

10 while (e>total[5]){
11   if (total[1]<=x5[1]){
12     if (total[2]<=x5[2]){
13       if (total[3]<=x5[3]){
14         if (total[4]<=x5[4]){
15           l=1
16         }
17       }
18     }
19   }
20   if (l==1){
21     x1[1]=x1[1]+4.888889
22     x1[2]=x1[2]+3.888889
23     profit<-c(x1[1]*x4[1], x1[2]*x4[2])
24     total[5]=profit[1]+profit[2]
25   }
26 }
27 cat("The maximized profit is=",total[5])

```

R code Exa 15.4 Nonlinear constrained optimization

```

1 Mt=2000
2 #kg
3 g=9.8
4 #m/s^2
5 c0=200
6 # $
7 c1=56
8 #$/m
9 c2=0.1
10 #$/m^2
11 vc=20
12 #m/s
13 kc=3
14 #kg/(s*m^2)

```

```

15 z0=500
16 #m
17 t=27
18 r=2.943652
19 n=6
20 pi = 3.1415927
21 A=2*pi*r*r
22 l=(2^0.5)*r
23 c=3*A
24 m=Mt/n
25
26 f <- function(t) {
27   (z0+g*m*m/(c*c)*(1-exp(-c*t/m)))*c/(g*m)
28 }
29
30 while (abs(f(t)-t)>0.00001){
31   t=t+0.00001
32 }
33 v=g*m*(1-exp(-c*t/m))/c
34 cat("The final value of velocity=",v,"\n")
35 cat("The final no. of load parcels=",n,"\n")
36 cat("The chute radius=",r,"m","\n")
37 cat("The minimum cost ($)=", (c0+c1*l+c2*A*A)*n)

```

R code Exa 15.5 One dimensional Optimization

```

1 library(neldermead)
2
3 fx <- function(x) {
4   -(2*sin(x))+x^2/10
5 }
6
7 x=fminsearch(fx,0)
8 x$output$algorithm
9 x = x$optbase$xopt

```

```

10 cat("After maximization:\n")
11 cat("x=",x)
12 cat("f(x)=",fx(x)," \n")

```

R code Exa 15.6 Multidimensional Optimization

```

1 library(neldermead)
2
3 fx <- function(x) {
4   -(2*x[1]*x[2]+2*x[1]-x[1]^2-2*x[2]^2)
5 }
6
7 x=fminsearch(fun = fx,x0 = c(-1,1))
8 x = x$optbase$xopt
9 cat("After maximization:", "\n", "x=", x[1], ", ", x[2], "\n",
    "f(x)=", fx(x), "\n")

```

R code Exa 15.7 Locate Single Optimum

```

1 fx <- function(x) {
2   -(2*sin(x)-x^2/10)
3 }
4
5 x=fminsearch(fx,0)
6 x = x$optbase$xopt
7 cat("After maximization:", "\n", "x=", x, "\n", "f(x)=",
    fx(x), "\n")

```

Chapter 17

Least squares regression

R code Exa 17.3.a linear regression using computer

```
1 s<-c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
2 v<-c
    (10,16.3,23,27.5,31,35.6,39,41.5,42.9,45,46,45.5,46,49,50)

3 g = 9.8
4 #m/s^2
5 m = 68.1
6 #kg
7 c = 12.5
8 #kg/s
9 v1<-matrix(0,15)
10 v2<-matrix(0,15)
11 for (i in 1:15){
12   v1[i] = g*m*(1 - exp(-c*s[i]/m))/c
13   v2[i] = g*m*s[i]/(c*(3.75+s[i]))
14 }
15 cat("time = ",s,"\n","measured v =",v,"\n"," using
    equation (1.10) v1 = ", "\n",v1,"\n"," using
    equation ((17.3)) v2 =", "\n",v2)
16 plot(v,v1)
17 lines(v,v1,main = 'v vs v1',xlab = 'v',ylab = 'v1')
```

R code Exa 17.3.b linear regression using computer

```
1 s<-c(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15)
2 v<-c
    (10,16.3,23,27.5,31,35.6,39,41.5,42.9,45,46,45.5,46,49,50)

3 g = 9.8
4 #m/s^2
5 m = 68.1
6 #kg
7 c = 12.5
8 #kg/s
9 v1<-matrix(0,15)
10 v2<-matrix(0,15)
11 for (i in 1:15){
12   v1[i] = g*m*(1 - exp(-c*s[i]/m))/c
13   v2[i] = g*m*s[i]/(c*(3.75+s[i]))
14 }
15 cat("time = ",s,"\n","measured v =",v,"\n","using
    equation(1.10) v1 = ","\n",v1,"\n","using
    equation((17.3)) v2 =","\n",v2)
16 plot(v,v2)
17 lines(v,v2,main = 'v vs v2',xlab = 'v',ylab = 'v2')
```

Chapter 18

Interpolation

R code Exa 18.5 Error Estimates for Order of Interpolation

```
1 x<-c(1, 4, 6, 5, 3, 1.5, 2.5, 3.5)
2 y<-c(0, 1.3862944, 1.7917595, 1.6094379, 1.0986123,
      0.4054641, 0.9162907, 1.2527630)
3 n=8
4 fdd = matrix(0,nrow =n,ncol = n)
5 for (i in 1:n){
6   fdd[i,1]=y[i]
7 }
8
9 for (j in 2:n){
10   for (i in 1:(n-j+1)){
11     fdd[i,j]=(fdd[i+1,j-1]-fdd[i,j-1])/(x[i+j-1]-x[i
12       ])
13   }
14   xterm=1
15   yint<-matrix(0,1)
16   yint[1]=fdd[1,1]
17
18   order<-matrix(0,n)
19   Ea<-matrix(0,n)
```

```
20 for (order in 2:n){
21   xterm=xterm*(2-x[order-1])
22   yint2=yint[order-1]+fdd[1,order]*xterm
23   Ea[order-1]=yint2-yint[order-1]
24   yint[order]=yint2
25 }
26 cat("F(x)=",yint,"\n","Ea=",Ea)
```

Chapter 19

Fourier Approximation

R code Exa 19.1 Least Square Fit

```
1 f <- function(t) {  
2   1.7+cos(4.189*t+1.0472)  
3 }  
4  
5 deltat=0.15  
6 t1=0  
7 t2=1.35  
8 omega=4.189  
9 del=(t2-t1)/9  
10 t<-matrix(0,10)  
11 for (i in 1:10){  
12   t[i]=t1+del*(i-1)  
13 }  
14 sumy=0  
15 suma=0  
16 sumb=0  
17 y<-matrix(0,10)  
18 a<-matrix(0,10)  
19 b<-matrix(0,10)  
20 for (i in 1:10){  
21   y[i]=f(t[i])
```



```

22  a[i]=y[i]*cos(omega*t[i])
23  b[i]=y[i]*sin(omega*t[i])
24  sumy=sumy+y[i]
25  suma=suma+a[i]
26  sumb=sumb+b[i]
27  }
28  A0=sumy/10
29  A1=2*suma/10
30  B1=2*sumb/10
31  cat("The least square fit is y=A0+A1*cos(w0*t)+A2*
      sin(w0*t), where", "\n", "A0=", A0, "\n", "A1=", A1, "\n
      ", "B1=", B1, "\n")
32  theta=atan(-B1/A1)
33  C1=(A1^2 + B1^2)^0.5
34  cat("Alternatively, the least square fit can be
      expressed as", "\n", "y=A0+C1*cos(w0*t + theta),
      where", "\n", "A0=", A0, "\n", "Theta=", theta, "\n", "C1
      =", C1, "\n", "Or", "\n", "y=A0+C1*sin(w0*t + theta +
      pi/2), where", "\n", "A0=", A0, "\n", "Theta=", theta, "
      \n", "C1=", C1, "\n")

```

R code Exa 19.2 Continuous Fourier Series Approximation

```

1  a0=0
2  #f(t)=-1 for -T/2 to -T/4
3  #f(t)=1 for -T/4 to T/4
4  #f(t)=-1 for T/4 to T/2
5  #ak=2/T* (integration of f(t)*cos(w0*t) from -T/2 to
      T/2)
6  #ak=2/T*((integration of f(t)*cos(w0*t) from -T/2 to
      -T/4) + (integration of f(t)*cos(w0*t) from -T/4
      to T/4) + (integration of f(t)*cos(w0*t) from T/
      4 to T/2))
7  #Therefore,
8  #ak=4/(k*pi) for k=1,5,9,.....

```

```

9 #ak=-4/(k*%pi) for k=3,7,11,.....
10 #ak=0 for k=even integers
11 #similarly we find the b's.
12 #all the b's=0
13 cat("The fourier approximtation is:", "\n", "4/(%pi)*cos
      (w)*t) - 4/(3*%pi)*cos(3*(w)*t) + 4/(5*%pi)*cos(5
      *(w)*t) - 4/(7*%pi)*cos(7*(w)*t) + .....")

```

R code Exa 19.4 Data Analysis

```

1 s<-c(0.0002, 0.0002, 0.0005, 0.0005, 0.001, 0.001)
2 r<-c(0.2, 0.5, 0.2, 0.5, 0.2, 0.5)
3 u<-c(0.25, 0.5, 0.4, 0.75, 0.5, 1)
4 logs=log10(s)
5 logr=log10(r)
6 logu=log10(u)
7 m<-matrix(0,nrow = 6,ncol = 3)
8 for (i in 1:6){
9   m[i,1]=1
10   m[i,2]=logs[i]
11   m[i,3]=logr[i]
12 }
13 a=qr.solve(m,transpose(logu))
14 cat(" alpha=",10^a[1], " sigma=",a[2], " rho=",a[3])

```

R code Exa 19.5 Curve Fitting

```

1 #install.packages(" signal",dependencies = TRUE)
2 library(signal)
3 x=0:10
4 y=sin(x)
5 xi=seq(0,10,.25)
6 #part a

```

```

7 yi=interp1(x,y,xi)
8 plot(xi,yi,main = "y vs x (part a)",xlab = "x",ylab
  ="y" )
9
10 #part b
11 #fitting x and y in a fifth order polynomial
12 p<-c(0.0008, -0.0290, 0.3542, -1.6854, 2.586,
  -0.0915)
13
14 for (i in 1:41){
15   yi[i]=p[1]*(xi[i]^5)+p[2]*(xi[i]^4)+p[3]*(xi[i]^3)
  +p[4]*(xi[i]^2)+p[5]*(xi[i])+p[6]
16 }
17 plot(xi,yi,main = "y vs x (part b)",xlab = "x",ylab
  = "y")
18
19 #part c
20 d = spline(x,y,method = "fmm",n = length(x))
21 plot(x,d$y,main = "y vs x (part c)",xlab = "x",ylab
  = "y")
22 lines(x,d$y)

```

R code Exa 19.6 Polynomial Regression

```

1 x<-c(0.05, 0.12, 0.15, 0.3, 0.45, 0.7, 0.84, 1.05)
2 y<-c(0.957, 0.851, 0.832, 0.72, 0.583, 0.378, 0.295,
  0.156)
3 sx=sum(x)
4 sxx=sum(x*x)
5 sx3=sum(x*x*x)
6 sx4=sum(x*x*x*x)
7 sx5=sum(x*x*x*x*x)
8 sx6=sum(x*x*x*x*x*x)
9 n=8
10 sy=sum(y)

```

```

11 sxy=sum(x*y)
12 sx2y=sum(x*x*y)
13 sx3y=sum(x*x*x*y)
14 m<-matrix(data = c(n, sx, sxx, sx3,sx, sxx, sx3, sx4
    ,sxx, sx3, sx4, sx5,sx3, sx4, sx5, sx6),nrow = 4,
    ncol = 4,byrow = TRUE)
15 p<-matrix(data = c(sy,sxy,sx2y,sx3y),nrow = 4,ncol =
    1,byrow = TRUE)
16 a=solve(m,p)
17 cat("The cubic polynomial is y=a0 + a1*x + a2*x^2 +
    a3*x^3, where a0, a1, a2 and a3 are", "\n",a[1], "\n",
    a[2], "\n",a[3], "\n",a[4], "\n")

```

Chapter 21

Newton Cotes Integration Formulas

R code Exa 21.1 Single trapezoidal rule

```
1 f <- function(x) {  
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)  
3 }  
4  
5 tval=1.640533  
6 a=0  
7 b=0.8  
8 fa=f(a)  
9 fb=f(b)  
10 l=(b-a)*((fa+fb)/2)  
11 Et=tval-l  
12 #error  
13 et=Et*100/tval  
14 #percent relative error  
15  
16 #by using approximate error estimate  
17  
18 #the second derivative of f  
19
```

```

20 g <- function(x) {-400+4050*x-10800*x^2+8000*x^3}
21 ans = integrate(f = g, lower = 0, upper = 0.8)
22
23 f2x = ans$value/(b-a)
24 #average value of second derivative
25
26 Ea=-(1/12)*(f2x)*(b-a)^3
27
28 cat("The Error Et=",Et,"\n","The percent relative
      error et=",et,"%","\n","The approximate error
      estimate without using the true value=",Ea)

```

R code Exa 21.2 Multiple trapezoidal rule

```

1 f <- function(x) {
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
3 }
4
5 a=0
6 b=0.8
7 tval=1.640533
8 n=2
9 h=(b-a)/n
10 fa=f(a)
11 fb=f(b)
12 fh=f(h)
13 l=(b-a)*(fa+2*fh+fb)/(2*n)
14 Et=tval-l
15 #error
16 et=Et*100/tval
17 #percent relative error
18
19 #by using approximate error estimate
20
21 #the second derivative of f

```

```

22 g <- function(x) {
23   -400+4050*x-10800*x^2+8000*x^3
24 }
25 ans = integrate(f = g, lower = 0, upper = 0.8)
26
27 f2x = ans$value/(b-a)
28 #average value of second derivative
29
30 Ea=-(1/12)*(f2x)*(b-a)^3/(n^2);
31 cat("The Error Et=",Et,"\n","The percent relative
    error et=",et,"%","\n","The approximate error
    estimate without using the true value=",Ea)

```

R code Exa 21.3 Evaluating Integrals

```

1 g=9.8
2 #m/s^2; acceleration due to gravity
3
4 m=68.1
5 #kg
6
7 c=12.5
8 #kg/sec; drag coefficient
9
10 f <- function(t) {
11   g*m*(1-exp(-c*t/m))/c
12 }
13
14 tval=289.43515
15 #m
16
17 a=0
18 b=10
19 fa=f(a)
20 fb=f(b)

```

```

21
22 for (i in seq(10,20,10)){
23     n=i
24     h=(b-a)/n
25     cat("No. of segments=",i,"\n","Segment size=",h,"\n")
26     j=a+h
27     s=0
28     while (j<b){
29         s=s+f(j)
30         j=j+h
31     }
32     l=(b-a)*(fa+2*s+fb)/(2*n)
33     Et=tval-l
34     #error
35     et=Et*100/tval
36     #percent relative error
37     cat("Estimated d=",l,"m","\n","et(%)",et,"\n",
        _____\
        n")
38 }
39
40 for (i in seq(50,100,50)){
41     n=i
42     h=(b-a)/n
43     cat("No. of segments=",i,"\n","Segment size=",h,"\n")
44     j=a+h
45     s=0
46     while (j<b){
47         s=s+f(j)
48         j=j+h
49     }
50     l=(b-a)*(fa+2*s+fb)/(2*n)
51     Et=tval-l
52     #error
53     et=Et*100/tval
54     #percent relative error

```



```

55     cat(" Estimated d=",l,"m", "\n", " et (%)", et, "\n", "
        -----\n"
        n")
56 }
57
58 for (i in seq(100,200,100)){
59     n=i
60     h=(b-a)/n
61     cat("No. of segments=",i," \n", "Segment size=",h," \n"
        n")
62     j=a+h
63     s=0
64     while (j<b){
65         s=s+f(j)
66         j=j+h
67     }
68     l=(b-a)*(fa+2*s+fb)/(2*n)
69     Et=tval-l
70     #error
71     et=Et*100/tval
72     #percent relative error
73     cat(" Estimated d=",l,"m", "\n", " et (%)", et, "\n", "
        -----\n"
        n")
74 }
75
76 for (i in seq(200,500,300)){
77     n=i
78     h=(b-a)/n
79     cat("No. of segments=",i," \n", "Segment size=",h," \n"
        n")
80     j=a+h
81     s=0
82     while (j<b){
83         s=s+f(j)
84         j=j+h
85     }
86     l=(b-a)*(fa+2*s+fb)/(2*n)

```

```

87     Et=tval-1
88     #error
89     et=Et*100/tval
90     #percent relative error
91     cat(" Estimated d=",l,"m", "\n", " et (%)",et, "\n", "
      _____\
            n" )
92 }
93 for (i in seq(1000,2000,1000)){
94     n=i
95     h=(b-a)/n
96     cat("No. of segments=",i, "\n", "Segment size=",h, "\
            n" )
97     j=a+h
98     s=0
99     while (j<b){
100         s=s+f(j)
101         j=j+h
102     }
103     l=(b-a)*(fa+2*s+fb)/(2*n)
104     Et=tval-1
105     #error
106     et=Et*100/tval
107     #percent relative error
108     cat(" Estimated d=",l,"m", "\n", " et (%)",et, "\n", "
      _____\
            n" )
109 }
110
111 for (i in seq(2000,5000,3000)){
112     n=i
113     h=(b-a)/n
114     cat("No. of segments=",i, "\n", "Segment size=",h, "\
            n" )
115     j=a+h
116     s=0
117     while (j<b){
118         s=s+f(j)

```

```

119     j=j+h
120 }
121 l=(b-a)*(fa+2*s+fb)/(2*n)
122 Et=tval-l
123 #error
124 et=Et*100/tval
125 #percent relative error
126 cat(" Estimated d=",l,"m","\n"," et(%)",et,"\n",
      "n")
127 }
128
129 for (i in seq(5000,10000,5000)){
130     n=i
131     h=(b-a)/n
132     cat("No. of segments=",i,"\n"," Segment size=",h,"\n")
133     j=a+h
134     s=0
135     while (j<b){
136         s=s+f(j)
137         j=j+h
138     }
139     l=(b-a)*(fa+2*s+fb)/(2*n)
140     Et=tval-l
141     #error
142     et=Et*100/tval
143     #percent relative error
144     cat(" Estimated d=",l,"m","\n"," et(%)",et,"\n",
      "n")
145 }

```

R code Exa 21.4 Single Simpsons 1 by 3 rule

```

1 f <- function(x) {
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
3 }
4
5 a=0
6 b=0.8
7 tval=1.640533
8 n=2
9 h=(b-a)/n
10 fa=f(a)
11 fb=f(b)
12 fh=f(h)
13
14 l=(b-a)*(fa+4*fh+fb)/(3*n)
15 cat("l=",l)
16 Et=tval-l
17 #error
18 et=Et*100/tval
19 #percent relative error
20
21 #by using approximate error estimate
22
23 #the fourth derivative of f
24 g <- function(x) {
25   -21600+48000*x
26 }
27 ans = integrate(f = g,0,0.8)
28 f4x=ans$value/(b-a)
29 #average value of fourth derivative
30 Ea=-(1/2880)*(f4x)*(b-a)^5
31 cat("The Error Et=",Et,"\n","The percent relative
    error et=",et,"%","\n","The approximate error
    estimate without using the true value=",Ea)

```

R code Exa 21.5 Multiple Simpsons 1 by 3 rule

```

1 f <- function(x) {
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
3 }
4
5 a=0
6 b=0.8
7 tval=1.640533
8 n=4
9 h=(b-a)/n
10 fa=f(a)
11 fb=f(b)
12 j=a+h
13 s=0
14 count=1
15 while (j<b){
16   if ((-1)^count==-1){
17     s=s+4*f(j)
18   } else {
19     s=s+2*f(j)
20   }
21   count=count+1
22   j=j+h
23 }
24
25 l=(b-a)*(fa+s+fb)/(3*n)
26 cat(" l=",l,"\n")
27 Et=tval-l
28 #error
29 et=Et*100/tval
30 #percent relative error
31
32 #by using approximate error estimate
33
34 #the fourth derivative of f
35
36 g <- function(x) {
37   -21600+48000*x
38 }

```

```

39 ans = integrate(f = g,0,0.8)
40
41 f4x=ans$value/(b-a)
42 #average value of fourth derivative
43 Ea=-(1/(180*4^4))*(f4x)*(b-a)^5
44 cat("The Error Et=",Et,"\n","The percent relative
      error et=",et,"%","\n","The approximate error
      estimate without using the true value=",Ea,"\n")

```

R code Exa 21.6 Simpsons 3 by 8 rule

```

1 f <- function(x) {
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
3 }
4
5 a=0
6 b=0.8
7 tval=1.640533
8 #part a
9 n=3
10 h=(b-a)/n
11 fa=f(a)
12 fb=f(b)
13 j=a+h
14 s=0
15 count=1
16 while (j<b){
17   s=s+3*f(j)
18   count=count+1
19   j=j+h
20 }
21 l=(b-a)*(fa+s+fb)/(8)
22 cat("Part A:", "\n", "l=", l, "\n")
23 Et=tval-l
24 #error

```

```

25 et=Et*100/tval
26 #percent relative error
27
28 #by using approximate error estimate
29
30 #the fourth derivative of f
31 g <- function(x) {
32     -21600+48000*x
33 }
34
35 ans= integrate(f = g,0,0.8)
36
37 f4x=ans$value / (b-a)
38 #average value of fourth derivative
39 Ea=-(1/6480)*(f4x)*(b-a)^5
40 cat("The Error Et=",Et,"\n","The percent relative
      error et=",et,"%","\n","The approximate error
      estimate without using the true value=",Ea,"\n")
41
42 #part b
43 n=5
44 h=(b-a)/n
45 l1=(a+2*h-a)*(fa+4*f(a+h)+f(a+2*h))/6
46 l2=(a+5*h-a-2*h)*(f(a+2*h)+3*(f(a+3*h)+f(a+4*h))+fb)
      /8
47 l=l1+l2
48 cat("
      _____\n")
49 cat("Part B:","\n","l=",l,"\n")
50 Et=tval-l
51 #error
52 et=Et*100/tval
53 #percent relative error
54 cat("The Error Et=",Et,"\n","The percent relative
      error et=",et,"%")
      _____

```

R code Exa 21.7 Unequal Trapezoidal segments

```
1 f <- function(x) {  
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)  
3 }  
4 func<-matrix(0,11)  
5 tval=1.640533  
6 x<-c(0, 0.12, 0.22, 0.32, 0.36, 0.4, 0.44, 0.54,  
       0.64, 0.7, 0.8)  
7 for (i in 1:11){  
8   func[i]=f(x[i])  
9 }  
10 l=0  
11 for (i in 1:10){  
12   l=l+(x[i+1]-x[i])*(func[i]+func[i+1])/2  
13 }  
14  
15 cat("l=",l)  
16 Et=tval-l  
17 #error  
18 et=Et*100/tval  
19 #percent relative error  
20 cat("The Error Et=",Et,"\n","The percent relative  
    error et=",et,"%")
```

R code Exa 21.8 Simpsons Uneven data

```
1 f <- function(x) {  
2   (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)  
3 }  
4  
5 tval=1.640533
```



```

6 x<-c(0, 0.12, 0.22, 0.32, 0.36, 0.4, 0.44, 0.54,
      0.64, 0.7 ,0.8)
7 func<-matrix(0,11)
8 for (i in 1:11){
9   func[i]=f(x[i])
10 }
11 l1=(x[2]-x[1])*((f(x[1])+f(x[2]))/2)
12 l2=(x[4]-x[2])*((f(x[4])+4*f(x[3])+f(x[2]))/6)
13 l3=(x[7]-x[4])*((f(x[4])+3*(f(x[5])+f(x[6]))+f(x[7]))
    /8)
14 l4=(x[9]-x[7])*((f(x[7])+4*f(x[8])+f(x[9]))/6)
15 l5=(x[10]-x[9])*((f(x[10])+f(x[9]))/2)
16 l6=(x[11]-x[10])*((f(x[11])+f(x[10]))/2)
17 l=l1+l2+l3+l4+l5+l6
18 cat("l=",l,"\n")
19 Et=tval-l
20 #error
21 et=Et*100/tval
22 #percent relative error
23 cat("The Error Et=",Et,"\n","The percent relative
    error et=",et,"%")

```

R code Exa 21.9 Average Temperature Determination

```

1 f <- function(x,y) {
2   2*x*y+2*x-x^2-2*y^2+72
3 }
4
5 len=8
6 #m,length
7 wid=6
8 #m,width
9 a=0
10 b=len
11 n=2

```

```

12 h=(b-a)/n
13 a1=0
14 b1=wid
15 h1=(b1-a1)/n
16
17 fa=f(a,0)
18 fb=f(b,0)
19 fh=f(h,0)
20 lx1=(b-a)*(fa+2*fh+fb)/(2*n)
21
22 fa=f(a,h1)
23 fb=f(b,h1)
24 fh=f(h,h1)
25 lx2=(b-a)*(fa+2*fh+fb)/(2*n)
26
27 fa=f(a,b1)
28 fb=f(b,b1)
29 fh=f(h,b1)
30 lx3=(b-a)*(fa+2*fh+fb)/(2*n)
31
32 l=(b1-a1)*(lx1+2*lx2+lx3)/(2*n)
33
34 avg_temp=l/(len*wid)
35 cat("The average temperature is=",avg_temp)

```

Chapter 23

Numerical differentiation

R code Exa 23.4 Integration and Differentiation

```
1 f <- function(x) {  
2   0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5  
3 }  
4  
5 a=0  
6 b=0.8  
7 Q=integrate(f,0,0.8)  
8 cat("Q=",Q,"\n")  
9 x<-c(0, 0.12, 0.22, 0.32, 0.36, 0.4 ,0.44, 0.54  
    ,0.64, 0.7, 0.8)  
10 y=f(x)  
11  
12 #This algorithm uses  
13 #the formula for the area of a trapezoid: area =  
    width average of the lengths of the parallel  
    sides.  
14  
15 UseTrapezoidRule <- function(xmin, xmax, num_  
    intervals) {  
16   #Calculate the width of a trapezoid.  
17   dx = (xmax - xmin) / num_intervals
```

```

18  #Add up the trapezoids' areas.
19  total_area = 0
20  x = xmin
21  for (i in 1:num_intervals){
22      total_area = total_area + dx * (f(x) + f(x + dx)
23          ) / 2
24      x = x + dx
25  }
26  return(total_area)
27 }
28 integral = UseTrapezoidRule(0,0.8,10000)
29
30 cat("Trapezoid intergral=",integral,"\n","diff(x)=",
31     diff(x),"\n")
32 d=diff(y)/diff(x)
33 cat("d=",d)

```

R code Exa 23.5 Integrate a function

```

1  f <- function(x) {
2      0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5
3  }
4
5  a=0
6  b=0.8
7  Qt=1.640533
8  Q=integrate(f,0,0.8)
9  cat("Computed=",Q$value,"\n","Error estimate=",abs(Q
10     $value-Qt)*100/Qt,"\n")

```

Chapter 25

Runga Kutta methods

R code Exa 25.4 Solving ODEs

```
1 m=68.1
2 g=9.8
3 c=12.5
4 a=8.3
5 b=2.2
6 vmax=46
7
8 f <- function(t,v,parms) {
9   list(c(g-c*v/m))
10 }
11
12 v0=0
13 t=0:15
14 sol<-ode(y = v0,times = t,func = f,parms = NULL)
15 sol <-data.frame(sol)
16 plot(t,sol$X1,main =" velocity vs time", xlab = "t (s
   )",ylab = "v (m/s)")
17 lines(t,sol$X1,col = "red")
18
19 f1 <- function(t,v,parms) {
20   list(c(g-(c/m)*(v+a*(v/vmax)^b)))
```

```

21 }
22
23 sol<-ode(y = v0,times = t,func = f1,parms = NULL)
24 sol <-data.frame(sol)
25 lines(t,sol$X1,col="blue")
26 legend(x = 10,y = 20,legend = c("Linear","Nonlinear"
    ),lty=c(1,1),col=c("red","blue"))

```

R code Exa 25.11 Solving systems of ODEs

```

1 library(deSolve)
2 f <- function(x,y,parms) {
3   a = y[2]
4   b = -16.1*y[1]
5   list(c(a,b))
6 }
7
8 x=seq(0,4,0.1)
9 y0<-c(0.1, 0)
10 sol<-ode(y = y0,times = x,func = f,parms = NULL)
11 sol <-data.frame(sol)
12 plot(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab
    = "y",type = "n")
13 lines(x,sol$X2,col = "blue")
14 lines(x,sol$X1,col = "red")
15 #legend(x = 3,y = 0.3,legend = c("y1,y3","y2,y4"),
    lty=c(1,1))
16
17 g <- function(x,y,parms) {
18   a = y[2]
19   b = -16.1*sin(y[1])
20   list(c(a,b))
21 }
22 sol<-ode(y = y0,times = x,func = g,parms = NULL)
23 sol <-data.frame(sol)

```

```

24 lines(c(0,4),c(-.5,.5),main = "y vs x",xlab = "x",
      ylab = "y",type = "n")
25 lines(x,sol$X2,col = "blue")
26 lines(x,sol$X1,col = "red")
27 #legend(x = 3,y = 0.3,legend = c("y1,y3","y2,y4"),
      lty=c(1,1))
28
29 pi = 3.1415927
30
31 y0<-c(pi/4, 0)
32 sol<-ode(y = y0,times = x,func = f,parms = NULL)
33 sol <-data.frame(sol)
34 lines(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab
      = "y",type = "n")
35 lines(x,sol$X2,col = "blue")
36 lines(x,sol$X1,col = "red")
37 legend(x = 3,y = 3,legend = c("y1,y3","y2,y4"),lty=c
      (1,1))
38
39 sol<-ode(y = y0,times = x,func = g,parms = NULL)
40 sol <-data.frame(sol)
41 lines(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab
      = "y",type = "n")
42 lines(x,sol$X2,col = "blue")
43 lines(x,sol$X1,col = "red")
44 legend(x = 3,y = 3,legend = c("y1,y3","y2,y4"),lty=c
      (1,1))

```

R code Exa 25.14 Adaptive Fourth order RK scheme

```

1 f <- function(x,y,parms) {
2   list(c(10*exp(-(x-2)^2/(2*(0.075^2)))-0.6*y))
3 }
4
5 x=seq(0,4,0.1)

```

```
6 y0=0.5
7 sol<-ode(y = y0,times = x,func = f,parms = NULL)
8 sol <-data.frame(sol)
9 plot(x,sol$X1,main ="y vs x",xlab = "x",ylab = "y")
10 lines(x,sol$X1)
```

Chapter 26

Stiffness and multistep methods

R code Exa 26.1 Explicit and Implicit Euler

```
1 f <- function(t,y) {
2   -1000*y+3000-2000*exp(-t)
3 }
4
5 y0=0
6 #explicit euler
7 h1=0.0005
8 y1 = matrix(0,60)
9 y1[1]=y0
10 count=2
11 t=seq(0,0.006,0.0001)
12 for (i in seq(0,0.0059,0.0001)){
13   y1[count]=y1[count-1]+f(i,y1[count-1])*h1
14   count=count+1
15 }
16 h2=0.0015
17 y2 = matrix(0,60)
18 y2[1]=y0
19 count=2
20 t=seq(0,0.006,0.0001)
21 for (i in seq(0,0.0059,0.0001)){
```

```

22     y2[count]=y2[count-1]+f(i,y2[count-1])*h2
23     count=count+1
24 }
25 plot(t,y2,main = "y vs t",xlab = "t",ylab = "y")
26 lines(t,y2,col="red")
27 lines(t,y1,col = "blue")
28 legend(x = 0.004,y = 0.5,legend = c("h=0.0005","h
    =0.0015"),lty = c(1,1))
29
30 #implicit order
31 h3=0.05
32 y3 = matrix(0,39)
33 y3[1]=y0
34 count=2;
35 t=seq(0,0.4,0.01)
36 for (j in seq(0,0.39,0.01)){
37     y3[count]=(y3[count-1]+3000*h3-2000*h3*exp(-(j
        +0.01)))/(1+1000*h3)
38     count=count+1
39 }
40 plot(t,y3,main = "y vs t",xlab = "t",ylab = "y")
41 lines(t,y3)

```

Chapter 27

Boundary Value and Eigenvalue problems

R code Exa 27.3 Finite Difference Approximation

```
1 h=0.01
2 delx=2
3 x=2+h*delx^2
4 a<-matrix(c(x, -1, 0, 0,-1, x, -1, 0, 0, -1, x, -1,
              0, 0, -1, x),nrow = 4,ncol = 4,byrow = TRUE)
5 b<-matrix(c(40.8, 0.8, 0.8, 200.8),nrow = 4,ncol =
              1,byrow = TRUE)
6 T=solve(a,b)
7 cat("The temperature at the interior nodes:",abs(T))
```

R code Exa 27.4 Mass Spring System

```
1 library(rootSolve)
2 m1=40
3 #kg
4 m2=40
```

```

5 #kg
6 k=200
7 #N/m
8 fun <- function (sqw) sqw^2-20*sqw+75
9 p <- uniroot.all(fun, c(0,100))
10 p<-round(data.frame(p))
11 r<-matrix(0,2)
12 r[1]<-p$p[1]
13 r[2]<-p$p[2]
14 f1=(r[1])^0.5
15 f2=(r[2])^0.5
16 pi = 3.1415927
17 Tp1=(2*pi)/f1
18 Tp2=(2*pi)/f2
19
20 #for first mode
21 cat("For first mode:", "\n", "Period of oscillation:",
      Tp1, "\n", "A1=-A2", "\n", "
      _____\n
      n")
22
23 #for first mode
24 cat("For second mode:", "\n", "Period of oscillation:"
      ,Tp2, "\n", "A1=A2")
      _____

```

R code Exa 27.5 Axially Loaded column

```

1 E=10*10^9
2 #Pa
3 I=1.25*10^-5
4 #m^4
5 L=3
6 #m
7 pi = 3.1415927
8 for (i in 1:8){

```

```

9   p=i*pi/L
10  P=i^2*(pi)^2*E*I/(L^2*1000)
11  cat("n=",i,"\n","p=",p,"m^-2","\n","P=",P,"kN","
      n")
12 }

```

R code Exa 27.6 Polynomial Method

```

1  library(rootSolve)
2  E=10*10^9
3  #Pa
4  I=1.25*10^-5
5  #m^4
6  L=3
7  #m
8  true<-c(1.0472, 2.0944, 3.1416, 4.1888)
9
10 #part a
11 h1=3/2
12 fun <- function (p) -h1^2*p^2+2
13 p <- uniroot.all(fun, c(-100,100))
14 p<-data.frame(p)
15 x<-matrix(0,2)
16 x[1]<-p$p[1]
17 x[2]<-p$p[2]
18 e=abs(abs(x[1])-true[1])*100/true[1];
19 cat("p=",x,"\n","error=",e,"
      n")
20
21 #part b
22 h2=3/3
23 fun <- function (p) (3-(4*p^2)+p^4) # = (2 - p^2)^2
      - 1

```

```

24 p <- uniroot.all(fun, c(-10,10))
25 p<-data.frame(p)
26 x<-matrix(0,2)
27 e<-matrix(0,2)
28 x[1]<-p$p[3]
29 x[2]<-p$p[1]
30 e[1]=abs(abs(x[1])-true[2])*100/true[2]
31 e[2]=abs(abs(x[2])-true[1])*100/true[1]
32 cat("p=",x,"\n", "error=",e,"
n")
33
34 #part c
35 h3=3/4;
36 fun <- function (p) (2-h3^2*p^2)^3 - 2*(2-h3^2*p^2)
37 #a= #= (2 - 0.5625*p^2)^3 - 2 *(2 - 0.5625*p^2)
38 p <- uniroot.all(fun, c(-10,10))
39 p<-data.frame(p)
40 x<-matrix(0,3)
41 e<-matrix(0,3)
42 x[1]<-p$p[1]
43 x[2]<-p$p[2]
44 x[3]<-p$p[3]
45 e[1]=abs(abs(x[1])-true[3])*100/true[3]
46 e[2]=abs(abs(x[2])-true[2])*100/true[2]
47 e[3]=abs(abs(x[3])-true[1])*100/true[1]
48 cat("p=",x,"\n", "error=",e,"
n")
49
50
51 #part d
52 h4=3/5;
53 fun <- function (p) (2-h4^2*p^2)^4 - 3*(2-h4^2*p^2)
^2 + 1
54 p <- uniroot.all(fun, c(-10,10))
55 p<-data.frame(p)
56 x<-matrix(0,4)

```

```

57 e<-matrix(0,4)
58 x[1]<-p$p[1]
59 x[2]<-p$p[2]
60 x[3]<-p$p[3]
61 x[4]<-p$p[4]
62 e[1]=abs(abs(x[1])-true[4])*100/true[4]
63 e[2]=abs(abs(x[2])-true[3])*100/true[3]
64 e[3]=abs(abs(x[3])-true[2])*100/true[2]
65 e[4]=abs(abs(x[4])-true[1])*100/true[1]
66 cat("p=",x,"\n", "error=",e,"

```

```

n")

```

R code Exa 27.7 Power Method Highest Eigenvalue

```

1 a<-matrix(c(3.556, -1.668, 0, -1.778, 3.556, -1.778,
    0, -1.778, 3.556),nrow = 3,ncol = 3,byrow = TRUE
    )
2 b<-matrix(c(1.778,0,1.778),nrow = 3,ncol = 1,byrow =
    TRUE)
3 ea=100
4 count=1
5 eigen<-matrix(0,1000)
6 while (ea>0.1){
7     maxim=b[1]
8     for (i in 2:3){
9         if (abs(b[i])>abs(maxim)){
10             maxim=b[i]
11         }
12     }
13     eigen[count]=maxim
14     b=a %*%(b/maxim)
15     if (count==1){
16         ea=20
17         count=count+1

```

```

18   } else {
19     ea=abs(eigen[count]-eigen[count-1])*100/abs(
        eigen[count])
20     count=count+1
21   }
22 }
23 cat("The largest eigen value",eigen[count-1])

```

R code Exa 27.8 Power Method Lowest Eigenvalue

```

1  a<-matrix(c(3.556, -1.668, 0, -1.778, 3.556, -1.778,
        0, -1.778, 3.556),nrow = 3,ncol = 3,byrow = TRUE
        )
2  b<-matrix(c(1.778,0,1.778),nrow = 3,ncol = 1,byrow =
        TRUE)
3  ea=100
4  count=1
5  eigen<-matrix(0,100)
6  ai=solve(a)
7  while (ea>4){
8    maxim=b[1]
9    for (i in 2:3){
10     if (abs(b[i])>abs(maxim)){
11       maxim=b[i]
12     }
13   }
14   eigen[count]=maxim
15   b=ai%*(b/maxim)
16   if (count==1){
17     ea=20
18     count=count+1
19   } else {
20     ea=abs(eigen[count]-eigen[count-1])*100/abs(
        eigen[count])
21     count=count+1

```



```

22 }
23 }
24 cat("The smallest eigen value", (1/eigen[count-1])
    ^0.5)

```

R code Exa 27.9 Eigenvalues and ODEs

```

1 library(deSolve)
2
3 predprey <- function(t,y,parms) {
4   a = 1.2*y[1]-0.6*y[1]*y[2]
5   b = -0.8*y[2]+0.3*y[1]*y[2]
6   list(c(a,b))
7 }
8 t=seq(0,20,0.1)
9 y0<-c(2, 1)
10 sol=ode(y = y0,parms = NULL,times = t,func =
    predprey)
11 sol<-data.frame(sol)
12 plot(t,sol$X1,main = "y vs t", xlab = "t",ylab = "y"
    )
13 lines(t,sol$X1)
14 lines(t,sol$X2)
15
16 plot(sol$X1,sol$X2,main = "space-space plot (y1 vs
    y2)", xlab = "y1",ylab = "y2")
17 lines(sol$X1,sol$X2)

```

R code Exa 27.11 Solving ODEs

```

1 library(deSolve)
2
3 predprey <- function(t,y,parms) {

```

```

4   a = 1.2*y[1]-0.6*y[1]*y[2]
5   b = -0.8*y[2]+0.3*y[1]*y[2]
6   list(c(a,b))
7 }
8 t=0:10
9 y0<-c(2, 1)
10 sol=ode(y = y0,parms = NULL,times = t,func =
    predprey)
11 sol<-data.frame(sol)
12
13 count=0;
14 for (i in 1:11){
15   cat(" istep=",count+1,"\n","time=",count,"\n","y1="
      ,sol$X1[i],"\n","y2=",sol$X2[i],"\n","
      -----\
      n")
16   count=count+1
17 }
-----

```

Chapter 31

Finite Element Method

R code Exa 31.1 Analytical Solution for Heated Rod

```
1 #d2T/dx2=-10; equation to be solved
2 #T(0,t)=40; boundary condition
3 #T(10,t)=200; boundary condition
4 #f(x)=10; uniform heat source
5 #we assume a solution T=a*X^2 + b*x +c
6 #differentiating twice we get d2T/dx2=2*a
7 a=-10/2
8 #using first boundary condition
9 c=40
10 #using second boundary condtion
11 b=66
12 #hence final solution T=-5*x^2 + 66*x + 40
13 f <- function(x) {
14   -5*x^2 + 66*x + 40
15 }
16 T<-matrix(0,110)
17 count=1
18 for (i in seq(0,11,0.1)){
19   T[count]=f(i)
20   count=count+1
21 }
```

```

22 x<-seq(0,11,0.1)
23 plot(x,T,main = "Temperature(T) vs distance(x)",xlab
      = "x (cm)",ylab = "T (units)")
24 lines(x,T)

```

R code Exa 31.2 Element Equation for Heated Rod

```

1  xf=10
2  #cm
3  xe=2.5
4  #cm
5  #T(0,t)=40; boundary condition
6  #T(10,t)=200; boundary condition
7  #f(x)=10; uniform heat source
8  f <- function(x) {
9    10*(xe-x)/xe
10 }
11 int1=integrate(f = f,lower = 0,upper = xe)
12
13 g <- function(x) {
14   10*(x-0)/xe
15 }
16 int2=integrate(f = g,lower = 0,upper = xe)
17
18 cat("The results are:", "\n", "0.4*T1-0.4*T2=-(dT/dx)*",
      "x1 + c1", "\n", "where c1=", int1$value, "\n", "and", "\n",
      "-0.4*T1+0.4*T2=-(dT/dx)*x2 + c2", "\n", "where",
      "c2=", int2$value, "\n")

```
