R Textbook Companion for Numerical Methods for Engineers by S. C. Chapra and R. P. Canale[1]

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

June 2, 2020

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

Book Description

Title: 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

```
g=9.8
#m/s^2; acceleration due to gravity
m=68.1
#kg
c=12.5
#kg/sec; drag coefficient
count=1
v = matrix(0,1)
for (i in (seq(0,12,2))){
  v[count]=g*m*(1-exp(-c*i/m))/c
  cat("v(m/s)=",v[count],"Time(s)=",i)
  count=count+1;
}
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

```
lbm=9999
#cm, measured length of bridge
lrm=9
#cm, measured length of rivet
lbt=10000
#cm, true length of bridge
lrt=10
#cm,true length of rivet

#calculating true error below;
Etb=lbt-lbm
#cm, true error in bridge
Etr=lrt-lrm
#cm, true error in rivet

#calculating percent relative error below
etb=Etb*100/lbt
#percent relative error for bridge
etr=Etb*100/lrt
```

```
#percent relative error for rivet
cat("a. The true error is")
cat(Etb,"cm"," for the bridge")
cat(Etr,"cm"," for the rivet")
cat("b. The percent relative error is")
cat(etb,"%"," for the bridge")
cat(etr,"%"," for the rivet")
```

R code Exa 3.2 Iterative error estimation

```
n=3
#number of significant figures
es=0.5*(10^(2-n))
#percent, specified error criterion
x=0.5;
f = matrix(0,1)
f[1]=1
#first estimate f=e^x = 1
ft=1.648721
#true value of e^0.5=f
et = matrix(0,1)
et[1]=(ft-f[1])*100/ft
ea = matrix(0,1)
ea[1]=100;
i=2
while (ea[i-1]>=es){
  f[i]=f[i-1]+(x^(i-1))/(factorial(i-1))
  et[i]=(ft-f[i])*100/ft
  ea[i]=(f[i]-f[i-1])*100/f[i]
  i=i+1
}
for (j in 1:i-1){
  cat("term number=",j,"\n","Result=",f[j],"\n","
     True % relative error=",et[j],"\n","Approximate
      estimate of error(%)=",ea[j],"\n")
```

```
  cat("
    ————————————————————————————————————————————————————————————————
    n")
}
```

R code Exa 3.3 Range of Integers

```
n=16
#no of bits
num=0
for (i in 0:(n-2)){
  num=num+(1*(2^i))
}
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

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

R code Exa 3.5 Machine Epsilon

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

R code Exa 3.6 Interdependent Computations

```
readinteger <- function()
{
  n <- readline(prompt="Input a number: ")
  return(as.integer(n))
}

num<-readinteger()

sum=0
for (i in 1:100000){
  sum=sum+num
}
cat("The number summed up 100,000 times is=",sum)
```

R code Exa 3.7 Subtractive Cancellation

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

```
x2=(-b-(D^0.5))/(2*a)
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

```
f <- function(x) {
  exp(x)
}

sum=1
test=0
i=0
term=1
x1= 10
x2= -10
while (sum~=test){
  cat("sum:",sum,"\n","term:",term,"\n","i:",i,"\n",
     "----------------------------------------\n")
  i=i+1
  term=term*x/i
  test=sum
  sum=sum+term
}
cat("Exact Value",f(x1))
cat("Exact Value",f(x2))
```

Chapter 4

Truncation Errors and the Taylor Series

R code Exa 4.1 Polynomial Taylor Series

```
DD <- function(expr, name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}

f <- function(x) {
  return(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2)
}

xi=0
xf=1
h=xf-xi
fi=f(xi)
#function value at xi
ffa=f(xf)
#actual function value at xf

#for n=0, i.e, zero order approximation
```

```
ff=fi
Et = matrix(0,5)
Et[1]=ffa-ff
#truncation error at x=1
cat("The value of f at x=0 :",fi,"\n",
    "The value of f at x=1 due to zero order approximation :",ff,"\n",
    "Truncation error :",Et[1],"\n",
    "--------------------------------------------------\n")

#for n=1, i.e, first order approximation
f1 <- function(x) {
  return(eval(DD(expr = expression(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2),"x",1)))
}

f1i=f1(xi)
#value of first derivative of function at xi
f1f=fi+f1i*h
#value of first derivative of function at xf
Et[2]=ffa-f1f
#truncation error at x=1
cat("The value of first derivative of f at x=0 :",f1i,"\n",
    "The value of f at x=1 due to first order approximation :",f1f,"\n",
    "Truncation error :",Et[2],"\n",
    "--------------------------------------------------\n")


#for n=2, i.e, second order approximation
f2 <- function(x) {
  return(eval(DD(expr = expression(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2),"x",2)))
}

```

```
f2i=f2(xi)
#value of second derivative of function at xi
f2f=f1f+f2i*(h^2)/factorial(2)
#value of second derivative of function at xf
Et[3]=ffa-f2f
#truncation error at x=1
cat("The value of first derivative of f at x=0 :",f2i,"\n",
    "The value of f at x=1 due to first order approximation :",f2f,"\n",
    "Truncation error :",Et[3],"\n",
    "-----------------------------------------------------\n")

#for n=3, i.e, third order approximation
f3 <- function(x) {
  return(eval(DD(expr = expression(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2),"x",3)))
}
f3i=f3(xi)
#value of third derivative of function at xi
f3f=f2f+f3i*(h^3)/factorial(3)
#value of third derivative of function at xf
Et[4]=ffa-f3f
#truncation error at x=1
cat("The value of first derivative of f at x=0 :",f3i,"\n",
    "The value of f at x=1 due to first order approximation :",f3f,"\n",
    "Truncation error :",Et[4],"\n",
    "-----------------------------------------------------\n")

#for n=4, i.e, fourth order approximation
f4 <- function(x) {
  return(eval(DD(expr = expression(-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2),"x",4)))
```

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

R code Exa 4.2 Taylor Series Expansion

```
DD <- function(expr, name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}

f <- function(x) {
  return(cos(x))
}

pi = 3.1415927
et = matrix(0,7)

xi=pi/4
xf=pi/3
h=xf-xi
fi=f(xi)
#function value at xi
```

```
ffa=f(xf)
#actual function value at xf

#for n=0, i.e, zero order approximation
ff=fi;
et[1]=(ffa-ff)*100/ffa
#percent relative error at x=1
cat("The value of f at x=1 due to zero order approximation :",ff,"\n",
    "% relative error :",et[1],"\n",
    "--------------------------------------------------\n")


#for n=1, i.e, first order approximation
f1 <- function(x) {
  return(eval(DD(expr = expression(cos(x)),"x",1)))
}
f1i=f1(xi)
#value of first derivative of function at xi
f1f=fi+f1i*h
#value of first derivative of function at xf
et[2]=(ffa-f1f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to first order approximation :",f1f,"\n",
    "% relative error :",et[2],"\n",
    "--------------------------------------------------\n")


#for n=2, i.e, second order approximation
f2 <- function(x) {
  return(eval(DD(expr = expression(cos(x)),"x",2)))
}
f2i=f2(xi)
#value of second derivative of function at xi
f2f=f1f+f2i*(h^2)/factorial(2)
```

```
#value of second derivative of function at xf
et[3]=(ffa-f2f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to second order approximation :",f2f,"\n",
    "% relative error :",et[3],"\n",
    "--------------------------------------------------\n")

#for n=3, i.e, third order approximation
f3 <- function(x) {
  return(eval(DD(expr = expression(cos(x)),"x",3)))
}
f3i=f3(xi)
#value of third derivative of function at xi
f3f=f2f+f3i*(h^3)/factorial(3)
#value of third derivative of function at xf
et[4]=(ffa-f3f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to third order approximation :",f3f,"\n",
    "% relative error :",et[4],"\n",
    "--------------------------------------------------\n")

#for n=4, i.e, fourth order approximation
f4 <- function(x) {
  return(eval(DD(expr = expression(cos(x)),"x",4)))
}
f4i=f4(xi)
#value of fourth derivative of function at xi
f4f=f3f+f4i*(h^4)/factorial(4)
#value of fourth derivative of function at xf
et[5]=(ffa-f4f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to fourth order approximation :",f4f,"\n",
    "% relative error :",et[5],"\n",
```

```
    "------------------------------------------------------------\n")


#for n=5, i.e, fifth order approximation
f5i=(f4(1.1*xi)-f4(0.9*xi))/(2*0.1)
#value of fifth derivative of function at xi (central difference method)
f5f=f4f+f5i*(h^5)/factorial(5)
#value of fifth derivative of function at xf
et[6]=(ffa-f5f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to fifth order approximation :",f5f,"\n",
    "% relative error :",et[6],"\n",
    "------------------------------------------------------------\n")


#for n=6, i.e, sixth order approximation
f6 <- function(x) {
  return(eval(DD(expr = expression(cos(x)),"x",4)))
}
f6i=(f4(1.1*xi)-2*f4(xi)+f4(0.9*xi))/(0.1^2)
#value of sixth derivative of function at xi (central difference method)
f6f=f5f+f6i*(h^6)/factorial(6)
#value of sixth derivative of function at xf
et[7]=(ffa-f6f)*100/ffa
#% relative error at x=1
cat("The value of f at x=1 due to sixth order approximation :",f6f,"\n",
    "% relative error :",et[7],"\n",
    "------------------------------------------------------------\n")
```

R code Exa 4.4 Finite divided difference approximation of derivatives

```
DD <- function(expr, name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}

f <- function(x) {
  return(-0.1*(x^4)-0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2)
}

x=0.5
h= 0.5
x1=x-h
x2=x+h
#forward difference method
fdx1=(f(x2)-f(x))/h
#derivative at x
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
#backward difference method
fdx2=(f(x)-f(x1))/h
#derivative at x
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
#central difference method
```

```
fdx3=(f(x2)-f(x1))/(2*h)
#derivative at x
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
cat("For h=",h,"\n",
    "Derivative at x by forward difference method=",fdx1,"and percent error=",et1,"\n",
    "Derivative at x by backward difference method=",fdx2,"and percent error=",et2,"\n",
    "Derivative at x by central difference method=",fdx3,"and percent error=",et3,"\n")


h= 0.25
x1=x-h
x2=x+h
#forward difference method
fdx1=(f(x2)-f(x))/h
#derivative at x
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
#backward difference method
fdx2=(f(x)-f(x1))/h
#derivative at x
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
#central difference method
fdx3=(f(x2)-f(x1))/(2*h)
#derivative at x
```

```
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
cat("For h=",h,"\n",
    "Derivative at x by forward difference method=",fdx1,"and percent error=",et1,"\n",
    "Derivative at x by backward difference method=",fdx2,"and percent error=",et2,"\n",
    "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

```
DD <- function(expr, name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}

x=2.5
delta=0.01
deltafx=abs(eval(DD(expr = expression(x^3),name = "x",order = 1)))*delta
fx=f(x)
cat("true value is between",fx-deltafx,"and",fx+deltafx)
```

R code Exa 4.6 Error propagation in multivariable function

```
library(Deriv)

```

```
DD <- function(expr, name, order = 1) {
  if(order < 1) stop("'order' must be >= 1")
  if(order == 1) D(expr, name)
  else DD(D(expr, name), name, order - 1)
}

f <- function(F,L,E,I) {
  (F*(L^4))/(8*E*I)
}

Fbar=50
#lb/ft
Lbar=30
#ft
Ebar=1.5*(10^8)
#lb/ft^2
Ibar=0.06
#ft^4
deltaF=2
#lb/ft
deltaL=0.1
#ft
deltaE=0.01*(10^8)
#lb/ft^2
deltaI=0.0006
#ft^4
ybar=(Fbar*(Lbar^4))/(8*Ebar*Ibar)

f1 <- function(F) {
  (F*(Lbar^4))/(8*Ebar*Ibar)
}
f_1<-Deriv(f1)
f2 <- function(L) {
  (Fbar*(L^4))/(8*Ebar*Ibar)
}
f_2<-Deriv(f2)
f3 <- function(E) {
  (Fbar*(Lbar^4))/(8*E*Ibar)
```

```r
}
f_3<-Deriv(f3)
f4 <- function(I) {
  (Fbar*(Lbar^4))/(8*Ebar*I)
}
f_4<-Deriv(f4)
deltay=abs(f_1(Fbar))*deltaF+
  abs(f_2(Lbar))*deltaL+
  abs(f_3(Ebar))*deltaE+
  abs(f_4(Ibar))*deltaI;

cat("The value of y is between:",ybar-deltay,"and",ybar+deltay)
ymin=((Fbar-deltaF)*((Lbar-deltaL)^4))/(8*(Ebar+deltaE)*(Ibar+deltaI));
ymax=((Fbar+deltaF)*((Lbar+deltaL)^4))/(8*(Ebar-deltaE)*(Ibar-deltaI));
cat("ymin is calculated at lower extremes of F, L, E, I values as =",ymin)
cat("ymax is calculated at higher extremes of F, L, E, I values as =",ymax)
```

R code Exa 4.7 Condition Number

```r
library(Deriv)

f <- function(x) {
  tan(x)
}

f_ = Deriv(f)

pi = 3.1415927
x1bar=(pi/2)+0.1*(pi/2)
x2bar=(pi/2)+0.01*(pi/2)
```

```
#computing condition number for x1bar
condnum1=x1bar*f_(x1bar)/f(x1bar)
cat("The condition number of function for x=",x1bar, "is:",condnum1)
if (abs(condnum1)>1){
  cat("Function is ill-conditioned for x=",x1bar)
}
#computing condition number for x2bar
condnum2=x2bar*f_(x2bar)/f(x2bar)
cat("The condition number of function for x=",x2bar, "is:",condnum2)
if (abs(condnum2)>1){
  cat("Function is ill-conditioned for x=",x2bar)
}
```

Chapter 5

Bracketing Methods

R code Exa 5.1 Graphical Approach

```
m=68.1
#kg
v=40
#m/s
t=10
#s
g=9.8
#m/s^2

f <- function(c) {
  g*m*(1-exp(-c*t/m))/c - v
}

cat("For various values of c and f(c) is found as:")
i=0
fc = matrix(0,5)
for (c in seq(4,20,4)){
  i=i+1
  bracket=c(c, f(c))
  cat(bracket)
  fc[i]=f(c)
```

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

R code Exa 5.2 Computer Graphics to Locate Roots

```
f <- function(x) {
  sin(10*x)+cos(3*x)
}

count=1
val = matrix(0,100)
func = matrix(0,100)
for (i in seq(1,5,0.05)){
  val[count]=i
  func[count]=f(i)
  count=count+1
}
plot(val,func,main = "x vs f(x)",xlab = 'x',ylab = '
   f(x)')
lines(val,func)
```

R code Exa 5.3 Bisection

```
m=68.1
#kg
v=40
#m/s
t=10
#s
g=9.8
```

```
#m/s^2

f <- function(c) {
  g*m*(1-exp(-c*t/m))/c - v
}

x1=12
x2=16
xt=14.7802
#true value
#"enter the tolerable true percent error="
e=2
xr=(x1+x2)/2
etemp=abs(xr-xt)/xt*100
#error
while (etemp>e){
  if (f(x1)*f(xr)>0){
    x1=xr
    xr=(x1+x2)/2
    etemp=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)<0){
    x2=xr
    xr=(x1+x2)/2
    etemp=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)==0) {
    break
  }
}
cat("The result is=",xr)
```

R code Exa 5.4 Error Estimates for Bisection

```
m=68.1
```

```
#kg
v=40
#m/s
t=10
#s
g=9.8
#m/s^2

f <- function(c) {
  g*m*(1-exp(-c*t/m))/c - v
}

x1=12
x2=16
xt=14.7802
#true value
#"enter the tolerable approximate error="
e=0.5
xr=(x1+x2)/2
i=1
et=abs(xr-xt)/xt*100
#error
cat("Iteration:",i)
cat("xl:",x1)
cat("xu:",x2)
cat("xr:",xr)
cat("et(%):",et)
cat("----------------------------------------")
etemp=100

while (etemp>e){
  if (f(x1)*f(xr)>0){
    x1=xr
    xr=(x1+x2)/2
    etemp=abs(xr-x1)/xr*100
    et=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)<0){
```

```
    x2=xr
    xr=(x1+x2)/2
    etemp=abs(xr-x2)/xr*100
    et=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)==0){
    break
  }
  i=i+1
  cat("Iteration:",i)
  cat("xl:",x1)
  cat("xu:",x2)
  cat("xr:",xr)
  cat("et(%):",et)
  cat("ea(%)",etemp)
  cat("-----------------------------------------------")
}
cat("The result is=",xr)
```

R code Exa 5.5 False Position

```
m=68.1
#kg
v=40
#m/s
t=10
#s
g=9.8
#m/s^2

f <- function(c) {
  g*m*(1-exp(-c*t/m))/c - v
}

x1=12
```

```
x2=16
xt=14.7802
#true value
#"enter the tolerable true percent error="
e=
xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
etemp=abs(xr-xt)/xt*100
#error
while (etemp>e){
  if (f(x1)*f(xr)>0){
    x1=xr
    xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
    etemp=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)<0){
    x2=xr
    xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
    etemp=abs(xr-xt)/xt*100
  }
  if (f(x1)*f(xr)==0){
    break
  }
}
cat("The result is=",xr)
```

R code Exa 5.6 Bracketing and False Position Methods

```
f <- function(x) {
  x^10 - 1
}

x1=0
x2=1.3
xt=1

```

```
#using bisection method
cat("BISECTION METHOD:")
xr=(x1+x2)/2
et=abs(xr-xt)/xt*100
#error
cat("Iteration:",1,"\n","xl:",x1,"\n","xu:",x2,"\n","xr:",xr,"\n","et(%):",et,"\n","----------------------------------------------\n")

for (i in 2:5){
  if (f(x1)*f(xr)>0){
    x1=xr
    xr=(x1+x2)/2
    ea=abs(xr-x1)/xr*100
    et=abs(xr-xt)/xt*100
    } else if (f(x1)*f(xr)<0){
      x2=xr
      xr=(x1+x2)/2
      ea=abs(xr-x2)/xr*100
      et=abs(xr-xt)/xt*100
    }

  if (f(x1)*f(xr)==0){
    break
  }
cat("Iteration:",i,"\n")
cat("xl:",x1,"\n")
cat("xu:",x2,"\n")
cat("xr:",xr,"\n")
cat("et(%):",et,"\n")
cat("ea(%)",ea,"\n")
cat("----------------------------------------------\n")
}

#using false position method
cat("FALSE POSITION METHOD:")
x1=0
x2=1.3
```

```
xt=1
xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
et=abs(xr-xt)/xt*100
#error
cat("Iteration:",1,"\n","xl:",x1,"\n","xu:",x2,"\n","xr:",xr,"\n","et(%):",et,"\n","----------------------------------------------------")

for (i in 2:5){
  if (f(x1)*f(xr)>0){
    x1=xr
    xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
    ea=abs(xr-x1)/xr*100
    et=abs(xr-xt)/xt*100
  }
  else if (f(x1)*f(xr)<0){
      x2=xr
      xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1))
      ea=abs(xr-x2)/xr*100
      et=abs(xr-xt)/xt*100
    }
  if (f(x1)*f(xr)==0){
    break
  }
cat("Iteration:",i,"\n")
cat("xl:",x1,"\n")
cat("xu:",x2,"\n")
cat("xr:",xr,"\n")
cat("et(%):",et,"\n")
cat("ea(%)",ea,"\n")
cat("----------------------------------------------------\n")
}
```

Chapter 6

Open Methods

R code Exa 6.11 Newton Raphson for a nonlinear Problem

```
u <- function(x,y) {
  x^2+x*y-10
}

v <- function(x,y) {
  y+3*x*y^2-57
}

x=1.5
y=3.5
e<-c(100, 100)
while  (e[1]>0.0001 & e[2]>0.0001){
  J=matrix(data = c(2*x+y, x, 3*y^2, 1+6*x*y),nrow =
      2,ncol = 2,byrow = TRUE)
  deter=det(J)
  u1=u(x,y)
  v1=v(x,y)
  x=x-((u1*J[2,2]-v1*J[1,2])/deter)
  y=y-((v1*J[1,1]-u1*J[2,1])/deter)
  e[1]=abs(2-x)
  e[2]=abs(3-y)
```

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

Chapter 7

Roots of Polynomials

R code Exa 7.1 Polynomial Deflation

```
f <- function(x) {
  (x-4)*(x+6)
}

n=2
a = matrix(0,3)
a[1]=-24
a[2]=2
a[3]=1
t=4
r=a[3]
a[3]=0
for (i in seq(n,1,-1)){
  s=a[i]
  a[i]=r
  r=s+r*t
  }
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

```r
f <- function(x) {
  x^3 - 13*x - 12
}

x1t=-3
x2t=-1
x3t=4
x0=4.5
x1=5.5
x2=5

cat("iteration:",0,"\n","xr:",x2,"
  ————————————————————————————————————————————————\n")

for (i in 1:4){
  h0=x1-x0
  h1=x2-x1
  d0=(f(x1)-f(x0))/(x1-x0)
  d1=(f(x2)-f(x1))/(x2-x1)
  a=(d1-d0)/(h1+h0)
  b=a*h1+d1
  c=f(x2)
  d=(b^2 - 4*a*c)^0.5
  if (abs(b+d)>abs(b-d)){
  x3=x2+((-2*c)/(b+d))
}else {
      x3=x2+((-2*c)/(b-d))
    }
  ea=abs(x3-x2)*100/x3
  x0=x1
  x1=x2
  x2=x3
```

```
  cat("iteration:",i,"\n")
  cat("xr:",x2,"\n")
  cat("ea(%):",ea,"\n")
  cat("
    ——————————————————————————————————————————————————————\n
    ")
}
```

R code Exa 7.3 Bairstows Method

```
f <- function(x) {
  x^5-3.5*x^4+2.75*x^3+2.125*x^2-3.875*x+1.25
}

r=-1
s=-1
es=1
#%
n=6
count=1
ear=100
eas=100
a<-c(1.25, -3.875, 2.125, 2.75, -3.5, 1)
b<-matrix(0,n)
c<-matrix(0,n)
while ((ear>es) & (eas>es)){
  b[n]=a[n]
  b[n-1]=a[n-1]+r*b[n]
  for (i in seq(n-2,1,-1)){
    b[i]=a[i]+r*b[i+1]+s*b[i+2]
  }
  c[n]=b[n]
  c[n-1]=b[n-1]+r*c[n]
  for (i in seq((n-2),2,-1)){
    c[i]=b[i]+r*c[i+1]+s*c[i+2]
```

```
  }
#c(3)*dr+c(4)*ds=-b(2)
#c(2)*dr+c(3)*ds=-b(1)
ds=((-b[1])+(b[2]*c[2]/c[3]))/(c[3]-(c[4]*c[2]/c[3]))
dr=(-b[2]-c[4]*ds)/c[3]
r=r+dr
s=s+ds
ear=abs(dr/r)*100
eas=abs(ds/s)*100
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")
count=count+1;
}
x1=(r+(r^2 + 4*s)^0.5)/2
x2=(r-(r^2 + 4*s)^0.5)/2
bracket<-c(x1, x2)
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

```
f <- function(x) {
  x-cos(x)
}

x1=0

if (f(x1)<0){
  x2=x1+0.001
```

```
  while (f(x2)<0){
    x2=x2+0.001
  }
  } else if(f(x1)>0){
    x2=x1+0.001
    while (f(x2)>0){
      x2=x2+0.001
    }
  } else{
    cat("The root is=",x1)
  }

x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1))
cat("The root is=",x)
```

R code Exa 7.5 Solving nonlinear system

```
u <- function(x,y) {
  x^2+x*y-10
}

v <- function(x,y) {
  y+3*x*y^2-57
}

x=1
y=3.5
while (u(x,y)!=v(x,y)){
  x=x+1
  y=y-0.5
}
cat("x=",x)
cat("y=",y)
```

R code Exa 7.6 Root Location

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

R code Exa 7.7 Roots of Polynomials

```r
library(pracma)
library(polynom)

fun <- function (x) (x^5 - (3.5*x^4) +(2.75*x^3) +(2.125*x^2) - (3.875*x) + 1.25)
fzero(f = fun,x =1)
Deriv::Deriv(f = fun,x = "x")

b<-c(1,0.5,-0.5)
a<-c(1,-3.5,2.75,2.125,-3.875,1.25)
answer = deconv(a,b)
d = answer$q
e = answer$r
polyroot(a)
polyroot(d)
conv(d,b)
a<-conv(d,b)
polyroot(a)
```

R code Exa 7.8 Root Location

```
f <- function(x) {
  x-cos(x)
}

x1=0
if (f(x1)<0){
  x2=x1+0.00001
  while (f(x2)<0){
    x2=x2+0.00001
  }
} else if (f(x1)>0){
    x2=x1+0.00001
    while (f(x2)>0){
      x2=x2+0.00001
    }
} else {
  cat("The root is=",x1)
}


x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1))
cat("The root is=",x)
```

Chapter 9

Gauss Elimination

R code Exa 9.2 Determinants

```r
#For fig9.1
a= matrix(data = c(3, 2,-1, 2),nrow = 2,ncol = 2, byrow = TRUE)
cat("The value of determinant for system repesented in fig 9.1 =",det(a))
#For fig9.2 (a)
a= matrix(data = c(-0.5, 1,-0.5, 1),nrow = 2,ncol = 2,byrow = TRUE)
cat("The value of determinant for system repesented in fig 9.2 (a) =",det(a))
#For fig9.2 (b)
a= matrix(data = c(-0.5, 1,-1, 2),nrow = 2,ncol = 2, byrow = TRUE)
cat("The value of determinant for system repesented in fig 9.2 (b) =",det(a))
#For fig9.2 (c)
a= matrix(data = c(-0.5, 1,-2.3/5, 1),nrow = 2,ncol = 2,byrow = TRUE)
cat("The value of determinant for system repesented in fig 9.2 (c) =",det(a))
```

R code Exa 9.3 Cramers Rule

```
#the matrix or the system
b1=-0.01
b2=0.67
b3=-0.44
a<-matrix(data = c(0.3, 0.52, 1,0.5, 1, 1.9,0.1,
   0.3, 0.5),nrow = 3,ncol = 3,byrow = TRUE)
a1<-matrix(data = c(a[2,2], a[2,3],a[3,2], a[3,3]),
   nrow = 2,ncol = 2,byrow = TRUE)
A1=det(a1)
a2<-matrix(data = c(a[2,1], a[2,3],a[3,1], a[3,3]),
   nrow = 2,ncol = 2,byrow = TRUE)
A2=det(a2)
a3<-matrix(data = c(a[2,1], a[2,2],a[3,1], a[3,2]),
   nrow = 2,ncol = 2,byrow = TRUE)
A3=det(a3)
D=a[1,1]*A1-a[1,2]*A2+a[1,3]*A3
p<-matrix(data = c(b1, 0.52, 1,b2, 1, 1.9,b3, 0.3,
   0.5),nrow = 3,ncol = 3,byrow = TRUE)
q<-matrix(data = c(0.3, b1, 1,0.5, b2, 1.9,0.1, b3,
   0.5),nrow = 3,ncol = 3,byrow = TRUE)
r<-matrix(data = c(0.3, 0.52, b1,0.5, 1 ,b2,0.1,
   0.3, b3),nrow = 3,ncol = 3,byrow = TRUE)

x1=det(p)/D
x2=det(q)/D
x3=det(r)/D
cat("The values are:","x1=",x1,",x2=",x2,",x3=",x3)
```

R code Exa 9.4 Elimination of Unknowns

```
#the equations are:
#3*x1+2*x2=18
#-x1+2*x2=2
a11=3
a12=2
b1=18
a21=-1
a22=2
b2=2
x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
cat("x1=",x1)
cat("x2=",x2)
```

R code Exa 9.5 Naive Gauss Elimination

```
n=3
b<-matrix(c(7.85,-19.3,71.4), nrow = 1, ncol = 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)
for (k in 1:1){
  for (i in 2:3){
    fact=a[i,k]/a[k,k]
    for (j in 2:3){
      a[i,j]=a[i,j]-fact*a[k,j]
    }
    b[i]=b[i]-fact*b[k]
    print(b)
  }
}
x<-matrix(0,3)
x[3]=b[3]/a[3,3]
for (i in seq(2,1,-1)){
  s=b[i]
  for (j in (i+1):3){
```

```
    s=s-a[i,j]*x[j]
    print(s)
  }
  x[i]=b[i]/a[i,i]
}
cat("x1=",x[1],",x2=",x[2],",x3=",x[3])
```

R code Exa 9.6 ill conditioned systems

```
a11=1
a12=2
b1=10
a21=1.1
a22=2
b2=10.4
x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
cat("For the original system:","x1=",x1,",x2=",x2)
a21=1.05
x1=(b1*a22-a12*b2)/(a11*a22-a12*a21)
x2=(b2*a11-a21*b1)/(a11*a22-a12*a21)
cat("For the new system:","x1=",x1,",x2=",x2)
```

R code Exa 9.7 Effect of Scale on Determinant

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

```
a<-matrix(c(1, 2,1.1, 2), nrow = 2, ncol = 2,byrow = TRUE)
b1=10
b2=10.4
cat("The determinant for part(b)=",det(a))
#part c
a1=a*10
b1=100
b2=104
cat("The determinant for part(c)=",det(a1))
```

R code Exa 9.8 Scaling

```
#part a
a<-matrix(c(1, 0.667,-0.5, 1), nrow = 2, ncol = 2, byrow = TRUE)
b1=6
b2=1
cat("The determinant for part(a)=",det(a))
#part b
a<-matrix(c(0.5, 1,0.55, 1), nrow = 2, ncol = 2, byrow = TRUE)
b1=5
b2=5.2
cat("The determinant for part(b)=",det(a))
#part c
b1=5
b2=5.2
cat("The determinant for part(c)=",det(a))
```

R code Exa 9.11 Solution of Linear Algebraic Equations

```
a<-matrix(c(70, 1, 0,60, -1, 1,40, 0 ,-1),nrow = 3, ncol = 3,byrow = TRUE)
b<-matrix(c(636,518,307),nrow = 3,ncol = 1,byrow = TRUE)
x=abs(solve(a,b))
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

```
maxf = -1e+09

n=10000
for (j in 1:n){
  Rnd=runif(2)
  x = -2 + 4 * Rnd[1]
  y = 1 + 2 * Rnd[2]
  fn = y - x - (2 * (x ^ 2)) - (2 * x * y) - (y ^ 2)
  if (fn > maxf){
    maxf = fn
    maxx = x
    maxy = y
  }
  if (mod(j,1000)==0){
    cat("Iteration:",j,"\n")
    cat("x:",x,"\n")
    cat("y:",y,"\n")
    cat("function value:",fn,"\n")
    cat("-------------------------------------------\
```

```
        n")
    }
}
```

R code Exa 14.2 Path of Steepest Descent

```
f <- function(x,y) {
  x*y*y
}

p1<-c(2, 2)
elevation=f(p1[1],p1[2])
dfx=p1[1]*p1[1]
dfy=2*p1[1]*p1[2]
theta=atan(dfy/dfx)
slope=(dfx^2 + dfy^2)^0.5
cat("Elevation:",elevation,"Theta:",theta,"slope:",
   slope)
```

R code Exa 14.3 1 D function along Gradient

```
f <- function(x,y) {
  2*x*y + 2*x - x^2 - 2*y^2
}

x=-1
y=1
dfx=2*y+2-2*x
dfy=2*x-4*y
#the function can thus be expressed along h axis as
#f((x+dfx*h),(y+dfy*h))
cat("The final equation is=","180*h^2 + 72*h - 7")
```

R code Exa 14.4 Optimal Steepest Descent

```
f <- function(x,y) {
  2*x*y + 2*x - x^2 - 2*y^2
}

x=-1
y=1
d2fx=-2
d2fy=-4
d2fxy=2

modH=d2fx*d2fy-(d2fxy)^2

for (i in 1:25){
  dfx=2*y+2-2*x
  dfy=2*x - 4*y
  #the function can thus be expressed along h axis as
  #f((x+dfx*h),(y+dfy*h))
  g <- function(h) {
    2*(x+dfx*h)*(y+dfy*h) + 2*(x+dfx*h) - (x+dfx*h)^2 - 2*(y+dfy*h)^2
    }
  #2*dfx*(y+dfy*h)+2*dfy*(x+dfx*h)+2*dfx-2*(x+dfx*h)*dfx-4*(y+dfy*h)*dfy=g'(h)=0
  #2*dfx*y + 2*dfx*dfy*h + 2*dfy*x + 2*dfy*dfx*h + 2*dfx - 2*x*dfx - 2*dfx*dfx*h - 4*y*dfy - 4*dfy*dfy*h=0
  #h(2*dfx*dfy+2*dfy*dfx-2*dfx*dfx-4*dfy*dfy)=-(2*dfx*y+2*dfy*x-2*x*dfx-4*y*dfy)
  h=(2*dfx*y+2*dfy*x-2*x*dfx-4*y*dfy+2*dfx)/(-1*(2*dfx*dfy+2*dfy*dfx-2*dfx*dfx-4*dfy*dfy))
  x=x+dfx*h
```

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

Chapter 15

Constrained Optimization

R code Exa 15.1 Setting up LP problem

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

R code Exa 15.2 Graphical Solution

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

```
x22<-matrix(0,8)
x23<-matrix(0,8)
x24<-matrix(0,8)
x25<-matrix(0,8)
x26<-matrix(0,8)
for (x1 in 0:8){
  x21[x1+1]=-(7/11)*x1+7
  x22[x1+1]=(80-10*x1)/8
  x23[x1+1]=6
  x24[x1+1]=-150*x1/175
  x25[x1+1]=(600-150*x1)/175
  x26[x1+1]=(1400-150*x1)/175
}
x1=0:8


plot(x1,x24,main ='Z=0')
lines(x1,x25,main = 'Z=600')
lines(x1,x26,main = 'Z=1400')
plot(x1,x21,main = 'x2 vs x1')
plot(x1,x22,xlab = 'x1 (tonnes)')
plot(x1,x23,ylab = 'x2 (tonnes)')
```

R code Exa 15.3 Linear Programming Problem

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

```

```
while (e>total[5]){
  if (total[1]<=x5[1]){
    if (total[2]<=x5[2]){
      if (total[3]<=x5[3]){
        if (total[4]<=x5[4]){
          l=1
        }
      }
    }
  }
if (l==1){
  x1[1]=x1[1]+4.888889
  x1[2]=x1[2]+3.888889
  profit<-c(x1[1]*x4[1], x1[2]*x4[2])
  total[5]=profit[1]+profit[2]
}
}
cat("The maximized profit is=",total[5])
```

R code Exa 15.4 Nonlinear constrained optimization

```
Mt=2000
#kg
g=9.8
#m/s^2
c0=200
#$
c1=56
#$/m
c2=0.1
#$/m^2
vc=20
#m/s
kc=3
#kg/(s*m^2)
```

```
z0=500
#m
t=27
r=2.943652
n=6
pi = 3.1415927
A=2*pi*r*r
l=(2^0.5)*r
c=3*A
m=Mt/n

f <- function(t) {
  (z0+g*m*m/(c*c)*(1-exp(-c*t/m)))*c/(g*m)
}

while (abs(f(t)-t)>0.00001){
  t=t+0.00001
}
v=g*m*(1-exp(-c*t/m))/c
cat("The final value of velocity=",v,"\n")
cat("The final no. of load parcels=",n,"\n")
cat("The chute radius=",r,"m","\n")
cat("The minimum cost($)=",(c0+c1*l+c2*A*A)*n)
```

R code Exa 15.5 One dimensional Optimization

```
library(neldermead)

fx <- function(x) {
  -(2*sin(x))+x^2/10
}

x=fminsearch(fx,0)
x$output$algorithm
x = x$optbase$xopt
```

```
cat("After maximization:\n")
cat("x=",x)
cat("f(x)=",fx(x),"\n")
```

R code Exa 15.6 Multidimensional Optimization

```
library(neldermead)

fx <- function(x) {
  -(2*x[1]*x[2]+2*x[1]-x[1]^2-2*x[2]^2)
}

x=fminsearch(fun = fx,x0 = c(-1,1))
x = x$optbase$xopt
cat("After maximization:","\n","x=",x[1],",",x[2],"\n","f(x)=",fx(x),"\n")
```

R code Exa 15.7 Locate Single Optimum

```
fx <- function(x) {
  -(2*sin(x)-x^2/10)
}

x=fminsearch(fx,0)
x = x$optbase$xopt
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

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

g = 9.8
#m/s^2
m = 68.1
#kg
c = 12.5
#kg/s
v1<-matrix(0,15)
v2<-matrix(0,15)
for (i in 1:15){
  v1[i] = g*m*(1 - exp(-c*s[i]/m))/c
  v2[i] = g*m*s[i]/(c*(3.75+s[i]))
}
cat("time = ",s,"\n","measured v =",v,"\n","using
  equation(1.10) v1 = ","\n",v1,"\n","using
  equation((17.3)) v2 =","\n",v2)
plot(v,v1)
lines(v,v1,main = 'v vs v1',xlab = 'v',ylab = 'v1')
```

R code Exa 17.3.b linear regression using computer

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

g = 9.8
#m/s^2
m = 68.1
#kg
c = 12.5
#kg/s
v1<-matrix(0,15)
v2<-matrix(0,15)
for (i in 1:15){
  v1[i] = g*m*(1 - exp(-c*s[i]/m))/c
  v2[i] = g*m*s[i]/(c*(3.75+s[i]))
}
cat("time = ",s,"\n","measured v =",v,"\n","using
  equation(1.10) v1 = ","\n",v1,"\n","using
  equation((17.3)) v2 =","\n",v2)
plot(v,v2)
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

```
x<-c(1, 4, 6, 5, 3, 1.5, 2.5, 3.5)
y<-c(0, 1.3862944, 1.7917595, 1.6094379, 1.0986123,
    0.4054641, 0.9162907, 1.2527630)
n=8
fdd = matrix(0,nrow =n,ncol = n)
for (i in 1:n){
  fdd[i,1]=y[i]
}

for (j in 2:n){
  for (i in 1:(n-j+1)){
    fdd[i,j]=(fdd[i+1,j-1]-fdd[i,j-1])/(x[i+j-1]-x[i
        ])
  }
}
xterm=1
yint<-matrix(0,1)
yint[1]=fdd[1,1]

order<-matrix(0,n)
Ea<-matrix(0,n)
```

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

Chapter 19

Fourier Approximation

R code Exa 19.1 Least Square Fit

```
f <- function(t) {
  1.7+cos(4.189*t+1.0472)
}

deltat=0.15
t1=0
t2=1.35
omega=4.189
del=(t2-t1)/9
t<-matrix(0,10)
for (i in 1:10){
  t[i]=t1+del*(i-1)
}
sumy=0
suma=0
sumb=0
y<-matrix(0,10)
a<-matrix(0,10)
b<-matrix(0,10)
for (i in 1:10){
  y[i]=f(t[i])
```

```
  a[i]=y[i]*cos(omega*t[i])
  b[i]=y[i]*sin(omega*t[i])
  sumy=sumy+y[i]
  suma=suma+a[i]
  sumb=sumb+b[i]
}
A0=sumy/10
A1=2*suma/10
B1=2*sumb/10
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")
theta=atan(-B1/A1)
C1=(A1^2 + B1^2)^0.5
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

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

```
#ak=-4/(k*%pi) for k=3,7,11,.....
#ak=0 for k=even integers
#similarly we find the b's.
#all the b's=0
cat("The fourier approximtion 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

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

R code Exa 19.5 Curve Fitting

```
#install.packages("signal",dependencies = TRUE)
library(signal)
x=0:10
y=sin(x)
xi=seq(0,10,.25)
#part a
```

```
yi=interp1(x,y,xi)
plot(xi,yi,main = "y vs x (part a)",xlab = "x",ylab ="y" )

#part b
#fitting x and y in a fifth order polynomial
p<-c(0.0008, -0.0290, 0.3542, -1.6854, 2.586, -0.0915)

for (i in 1:41){
  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]
}
plot(xi,yi,main = "y vs x (part b)",xlab = "x",ylab = "y")

#part c
d = spline(x,y,method = "fmm",n = length(x))
plot(x,d$y,main = "y vs x (part c)",xlab = "x",ylab = "y")
lines(x,d$y)
```

R code Exa 19.6 Polynomial Regression

```
x<-c(0.05, 0.12, 0.15, 0.3, 0.45, 0.7, 0.84, 1.05)
y<-c(0.957, 0.851, 0.832, 0.72, 0.583, 0.378, 0.295, 0.156)
sx=sum(x)
sxx=sum(x*x)
sx3=sum(x*x*x)
sx4=sum(x*x*x*x)
sx5=sum(x*x*x*x*x)
sx6=sum(x*x*x*x*x*x)
n=8
sy=sum(y)
```

```
sxy=sum(x*y)
sx2y=sum(x*x*y)
sx3y=sum(x*x*x*y)
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)
p<-matrix(data = c(sy,sxy,sx2y,sx3y),nrow = 4,ncol = 1,byrow = TRUE)
a=solve(m,p)
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

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

tval=1.640533
a=0
b=0.8
fa=f(a)
fb=f(b)
l=(b-a)*((fa+fb)/2)
Et=tval-l
#error
et=Et*100/tval
#percent relative error

#by using approximate error estimate

#the second derivative of f

```

```
g <- function(x) {-400+4050*x-10800*x^2+8000*x^3}
ans = integrate(f = g,lower = 0,upper = 0.8)

f2x = ans$value/(b-a)
#average value of second derivative

Ea=-(1/12)*(f2x)*(b-a)^3

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

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

a=0
b=0.8
tval=1.640533
n=2
h=(b-a)/n
fa=f(a)
fb=f(b)
fh=f(h)
l=(b-a)*(fa+2*fh+fb)/(2*n)
Et=tval-l
#error
et=Et*100/tval
#percent relative error

#by using approximate error estimate

#the second derivative of f
```

```
g <- function(x) {
  -400+4050*x-10800*x^2+8000*x^3
}
ans = integrate(f = g,lower = 0,upper = 0.8)

f2x = ans$value/(b-a)
#average value of second derivative

Ea=-(1/12)*(f2x)*(b-a)^3/(n^2);
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

```
g=9.8
#m/s^2; acceleration due to gravity

m=68.1
#kg

c=12.5
#kg/sec; drag coefficient

f <- function(t) {
  g*m*(1-exp(-c*t/m))/c
}

tval=289.43515
#m

a=0
b=10
fa=f(a)
fb=f(b)
```

```

for (i in seq(10,20,10)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}

for (i in seq(50,100,50)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
```

```
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}

for (i in seq(100,200,100)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}

for (i in seq(200,500,300)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
```

```
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}
for (i in seq(1000,2000,1000)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}

for (i in seq(2000,5000,3000)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
```

```
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}

for (i in seq(5000,10000,5000)){
  n=i
  h=(b-a)/n
  cat("No. of segments=",i,"\n","Segment size=",h,"\n")
  j=a+h
  s=0
  while (j<b){
    s=s+f(j)
    j=j+h
  }
  l=(b-a)*(fa+2*s+fb)/(2*n)
  Et=tval-l
  #error
  et=Et*100/tval
  #percent relative error
  cat("Estimated d=",l,"m","\n","et(%)",et,"\n","------------------------------------------------------------\n")
}
```

R code Exa 21.4 Single Simpsons 1 by 3 rule

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

a=0
b=0.8
tval=1.640533
n=2
h=(b-a)/n
fa=f(a)
fb=f(b)
fh=f(h)

l=(b-a)*(fa+4*fh+fb)/(3*n)
cat("l=",l)
Et=tval-l
#error
et=Et*100/tval
#percent relative error

#by using approximate error estimate

#the fourth derivative of f
g <- function(x) {
  -21600+48000*x
}
ans = integrate(f = g,0,0.8)
f4x=ans$value/(b-a)
#average value of fourth derivative
Ea=-(1/2880)*(f4x)*(b-a)^5
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

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

a=0
b=0.8
tval=1.640533
n=4
h=(b-a)/n
fa=f(a)
fb=f(b)
j=a+h
s=0
count=1
while (j<b){
  if ((-1)^count==-1){
    s=s+4*f(j)
  } else {
    s=s+2*f(j)
  }
  count=count+1
  j=j+h
}

l=(b-a)*(fa+s+fb)/(3*n)
cat("l=",l,"\n")
Et=tval-l
#error
et=Et*100/tval
#percent relative error

#by using approximate error estimate

#the fourth derivative of f

g <- function(x) {
  -21600+48000*x
}
```

```
ans = integrate(f = g,0,0.8)

f4x=ans$value/(b-a)
#average value of fourth derivative
Ea=-(1/(180*4^4))*(f4x)*(b-a)^5
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

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

a=0
b=0.8
tval=1.640533
#part a
n=3
h=(b-a)/n
fa=f(a)
fb=f(b)
j=a+h
s=0
count=1
while (j<b){
  s=s+3*f(j)
  count=count+1
  j=j+h
}
l=(b-a)*(fa+s+fb)/(8)
cat("Part A:","\n","l=",l,"\n")
Et=tval-l
#error
```

```
et=Et*100/tval
#percent relative error

#by using approximate error estimate

#the fourth derivative of f
g <- function(x) {
  -21600+48000*x
}

ans= integrate(f = g,0,0.8)

f4x=ans$value /(b-a)
#average value of fourth derivative
Ea=-(1/6480)*(f4x)*(b-a)^5
cat("The Error Et=",Et,"\n","The percent relative error et=",et,"%","\n","The approximate error estimate without using the true value=",Ea,"\n")

#part b
n=5
h=(b-a)/n
l1=(a+2*h-a)*(fa+4*f(a+h)+f(a+2*h))/6
l2=(a+5*h-a-2*h)*(f(a+2*h)+3*(f(a+3*h)+f(a+4*h))+fb)/8
l=l1+l2
cat("------------------------------------------------------------\n")
cat("Part B:","\n","l=",l,"\n")
Et=tval-l
#error
et=Et*100/tval
#percent relative error
cat("The Error Et=",Et,"\n","The percent relative error et=",et,"%")
```

R code Exa 21.7 Unequal Trapezoidal segments

```
f <- function(x) {
  (0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
}
func<-matrix(0,11)
tval=1.640533
x<-c(0, 0.12, 0.22, 0.32, 0.36, 0.4, 0.44, 0.54, 0.64, 0.7, 0.8)
for (i in 1:11){
  func[i]=f(x[i])
}
l=0
for (i in 1:10){
  l=l+(x[i+1]-x[i])*(func[i]+func[i+1])/2
}

cat("l=",l)
Et=tval-l
#error
et=Et*100/tval
#percent relative error
cat("The Error Et=",Et,"\n","The percent relative error et=",et,"%")
```

R code Exa 21.8 Simpsons Uneven data

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

tval=1.640533
```

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

R code Exa 21.9 Average Temperature Determination

```r
f <- function(x,y) {
  2*x*y+2*x-x^2-2*y^2+72
}

len=8
#m, length
wid=6
#m, width
a=0
b=len
n=2
```

```
h=(b-a)/n
a1=0
b1=wid
h1=(b1-a1)/n

fa=f(a,0)
fb=f(b,0)
fh=f(h,0)
lx1=(b-a)*(fa+2*fh+fb)/(2*n)

fa=f(a,h1)
fb=f(b,h1)
fh=f(h,h1)
lx2=(b-a)*(fa+2*fh+fb)/(2*n)

fa=f(a,b1)
fb=f(b,b1)
fh=f(h,b1)
lx3=(b-a)*(fa+2*fh+fb)/(2*n)

l=(b1-a1)*(lx1+2*lx2+lx3)/(2*n)

avg_temp=l/(len*wid)
cat("The average termperature is=",avg_temp)
```

Chapter 23

Numerical differentiation

R code Exa 23.4 Integration and Differentiation

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

a=0
b=0.8
Q=integrate(f,0,0.8)
cat("Q=",Q,"\n")
x<-c(0, 0.12, 0.22, 0.32, 0.36, 0.4 ,0.44, 0.54
   ,0.64, 0.7, 0.8)
y=f(x)

#This algorithm uses
#the formula for the area of a trapezoid: area =
   width  average of the lengths of the parallel
   sides.

UseTrapezoidRule <- function(xmin, xmax, num_
   intervals) {
  #Calculate the width of a trapezoid.
  dx = (xmax - xmin) / num_intervals
```

```
  #Add up the trapezoids' areas.
  total_area = 0
  x = xmin
  for (i in 1:num_intervals){
    total_area = total_area + dx * (f(x) + f(x + dx)) / 2
    x = x + dx
  }
  return(total_area)
}

integral = UseTrapezoidRule(0,0.8,10000)

cat("Trapezoid intergral=",integral,"\n","diff(x)=",diff(x),"\n")
d=diff(y)/diff(x)
cat("d=",d)
```

R code Exa 23.5 Integrate a function

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

a=0
b=0.8
Qt=1.640533
Q=integrate(f,0,0.8)
cat("Computed=",Q$value,"\n","Error estimate=",abs(Q$value-Qt)*100/Qt,"\n")
```

Chapter 25

Runga Kutta methods

R code Exa 25.4 Solving ODEs

```
m=68.1
g=9.8
c=12.5
a=8.3
b=2.2
vmax=46

f <- function(t,v,parms) {
  list(c(g-c*v/m))
}

v0=0
t=0:15
sol<-ode(y = v0,times = t,func = f,parms = NULL)
sol <-data.frame(sol)
plot(t,sol$X1,main ="velocity vs time", xlab = "t (s
    )",ylab = "v (m/s)")
lines(t,sol$X1,col = "red")

f1 <- function(t,v,parms) {
  list(c(g-(c/m)*(v+a*(v/vmax)^b)))
```

```
}

sol<-ode(y = v0,times = t,func = f1,parms = NULL)
sol <-data.frame(sol)
lines(t,sol$X1,col="blue")
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

```
library(deSolve)
f <- function(x,y,parms) {
  a = y[2]
  b = -16.1*y[1]
  list(c(a,b))
}

x=seq(0,4,0.1)
y0<-c(0.1, 0)
sol<-ode(y = y0,times = x,func = f,parms = NULL)
sol <-data.frame(sol)
plot(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab
  = "y",type = "n")
lines(x,sol$X2,col = "blue")
lines(x,sol$X1,col = "red")
#legend(x = 3,y = 0.3,legend = c("y1,y3","y2,y4"),
  lty=c(1,1))

g <- function(x,y,parms) {
  a = y[2]
  b = -16.1*sin(y[1])
  list(c(a,b))
}
sol<-ode(y = y0,times = x,func = g,parms = NULL)
sol <-data.frame(sol)
```

```
lines(c(0,4),c(-.5,.5),main = "y vs x",xlab = "x",ylab = "y",type = "n")
lines(x,sol$X2,col = "blue")
lines(x,sol$X1,col = "red")
#legend(x = 3,y = 0.3,legend = c("y1,y3","y2,y4"),lty=c(1,1))

pi = 3.1415927

y0<-c(pi/4, 0)
sol<-ode(y = y0,times = x,func = f,parms = NULL)
sol <-data.frame(sol)
lines(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab = "y",type = "n")
lines(x,sol$X2,col = "blue")
lines(x,sol$X1,col = "red")
legend(x = 3,y = 3,legend = c("y1,y3","y2,y4"),lty=c(1,1))

sol<-ode(y = y0,times = x,func = g,parms = NULL)
sol <-data.frame(sol)
lines(c(0,4),c(-4,4),main = "y vs x",xlab = "x",ylab = "y",type = "n")
lines(x,sol$X2,col = "blue")
lines(x,sol$X1,col = "red")
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

```
f <- function(x,y,parms) {
  list(c(10*exp(-(x-2)^2/(2*(0.075^2)))-0.6*y))
}

x=seq(0,4,0.1)
```

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

Chapter 26

Stiffness and multistep methods

R code Exa 26.1 Explicit and Implicit Euler

```
f <- function(t,y) {
  -1000*y+3000-2000*exp(-t)
}

y0=0
#explicit euler
h1=0.0005
y1 = matrix(0,60)
y1[1]=y0
count=2
t=seq(0,0.006,0.0001)
for (i in seq(0,0.0059,0.0001)){
  y1[count]=y1[count-1]+f(i,y1[count-1])*h1
  count=count+1
}
h2=0.0015
y2 = matrix(0,60)
y2[1]=y0
count=2
t=seq(0,0.006,0.0001)
for (i in seq(0,0.0059,0.0001)){
```

```
  y2[count]=y2[count-1]+f(i,y2[count-1])*h2
  count=count+1
}
plot(t,y2,main = "y vs t",xlab = "t",ylab = "y")
lines(t,y2,col="red")
lines(t,y1,col = "blue")
legend(x = 0.004,y = 0.5,legend = c("h=0.0005","h=0.0015"),lty = c(1,1))

#implicit order
h3=0.05
y3 = matrix(0,39)
y3[1]=y0
count=2;
t=seq(0,0.4,0.01)
for (j in seq(0,0.39,0.01)){
  y3[count]=(y3[count-1]+3000*h3-2000*h3*exp(-(j+0.01)))/(1+1000*h3)
  count=count+1
}
plot(t,y3,main = "y vs t",xlab = "t",ylab = "y")
lines(t,y3)
```

Chapter 27

Boundary Value and Eigenvalue problems

R code Exa 27.3 Finite Difference Approximation

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

R code Exa 27.4 Mass Spring System

```
library(rootSolve)
m1=40
#kg
m2=40
```

```
#kg
k=200
#N/m
fun <- function (sqw) sqw^2-20*sqw+75
p <- uniroot.all(fun, c(0,100))
p<-round(data.frame(p))
r<-matrix(0,2)
r[1]<-p$p[1]
r[2]<-p$p[2]
f1=(r[1])^0.5
f2=(r[2])^0.5
pi = 3.1415927
Tp1=(2*pi)/f1
Tp2=(2*pi)/f2

#for first mode
cat("For first mode:","\n","Period of oscillation:",
   Tp1,"\n","A1=-A2","\n","
   ------------------------------------------------------------\
   n")

#for first mode
cat("For second mode:","\n","Period of oscillation:"
   ,Tp2,"\n","A1=A2")
```

R code Exa 27.5 Axially Loaded column

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

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

R code Exa 27.6 Polynomial Method

```
library(rootSolve)
E=10*10^9
#Pa
I=1.25*10^-5
#m^4
L=3
#m
true<-c(1.0472, 2.0944, 3.1416, 4.1888)

#part a
h1=3/2
fun <- function (p) -h1^2*p^2+2
p <- uniroot.all(fun, c(-100,100))
p<-data.frame(p)
x<-matrix(0,2)
x[1]<-p$p[1]
x[2]<-p$p[2]
e=abs(abs(x[1])-true[1])*100/true[1];
cat("p=",x,"\n","error=",e,"----------------------------------------------------------------------\n")

#part b
h2=3/3
fun <- function (p) (3-(4*p^2)+p^4) # = (2 - p^2)^2 - 1
```

```
p <- uniroot.all(fun, c(-10,10))
p<-data.frame(p)
x<-matrix(0,2)
e<-matrix(0,2)
x[1]<-p$p[3]
x[2]<-p$p[1]
e[1]=abs(abs(x[1])-true[2])*100/true[2]
e[2]=abs(abs(x[2])-true[1])*100/true[1]
cat("p=",x,"\n","error=",e,"
    ------------------------------------------------------------\
    n")

#part c
h3=3/4;
fun <- function (p) (2-h3^2*p^2)^3 - 2*(2-h3^2*p^2)
#a= #= (2 - 0.5625*p^2)^3 - 2 *(2 - 0.5625*p^2)
p <- uniroot.all(fun, c(-10,10))
p<-data.frame(p)
x<-matrix(0,3)
e<-matrix(0,3)
x[1]<-p$p[1]
x[2]<-p$p[2]
x[3]<-p$p[3]
e[1]=abs(abs(x[1])-true[3])*100/true[3]
e[2]=abs(abs(x[2])-true[2])*100/true[2]
e[3]=abs(abs(x[3])-true[1])*100/true[1]
cat("p=",x,"\n","error=",e,"
    ------------------------------------------------------------\
    n")


#part d
h4=3/5;
fun <- function (p) (2-h4^2*p^2)^4 - 3*(2-h4^2*p^2)
    ^2 + 1
p <- uniroot.all(fun, c(-10,10))
p<-data.frame(p)
x<-matrix(0,4)
```

```
e<-matrix(0,4)
x[1]<-p$p[1]
x[2]<-p$p[2]
x[3]<-p$p[3]
x[4]<-p$p[4]
e[1]=abs(abs(x[1])-true[4])*100/true[4]
e[2]=abs(abs(x[2])-true[3])*100/true[3]
e[3]=abs(abs(x[3])-true[2])*100/true[2]
e[4]=abs(abs(x[4])-true[1])*100/true[1]
cat("p=",x,"\n","error=",e,"
    ——————————————————————————————————————————————————\
    n")
```

R code Exa 27.7 Power Method Highest Eigenvalue

```
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
    )
b<-matrix(c(1.778,0,1.778),nrow = 3,ncol = 1,byrow =
     TRUE)
ea=100
count=1
eigen<-matrix(0,1000)
while (ea>0.1){
  maxim=b[1]
  for (i in 2:3){
    if (abs(b[i])>abs(maxim)){
      maxim=b[i]
    }
  }
  eigen[count]=maxim
  b=a %*%(b/maxim)
  if (count==1){
    ea=20
    count=count+1
```

```
  } else {
    ea=abs(eigen[count]-eigen[count-1])*100/abs(
       eigen[count])
    count=count+1
  }
}
cat("The largest eigen value",eigen[count-1])
```

R code Exa 27.8 Power Method Lowest Eigenvalue

```
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
   )
b<-matrix(c(1.778,0,1.778),nrow = 3,ncol = 1,byrow =
    TRUE)
ea=100
count=1
eigen<-matrix(0,100)
ai=solve(a)
while (ea>4){
  maxim=b[1]
  for (i in 2:3){
    if (abs(b[i])>abs(maxim)){
      maxim=b[i]
    }
  }
  eigen[count]=maxim
  b=ai%*%(b/maxim)
  if (count==1){
    ea=20
    count=count+1
  } else {
    ea=abs(eigen[count]-eigen[count-1])*100/abs(
       eigen[count])
    count=count+1
```

```
  }
}
cat(”The smallest eigen value”,(1/eigen[count-1])^0.5)
```

R code Exa 27.9 Eigenvalues and ODEs

```
library(deSolve)

predprey <- function(t,y,parms) {
  a = 1.2*y[1]-0.6*y[1]*y[2]
  b = -0.8*y[2]+0.3*y[1]*y[2]
  list(c(a,b))
}
t=seq(0,20,0.1)
y0<-c(2, 1)
sol=ode(y = y0,parms = NULL,times = t,func = predprey)
sol<-data.frame(sol)
plot(t,sol$X1,main = ”y vs t”, xlab = ”t”,ylab = ”y”)
lines(t,sol$X1)
lines(t,sol$X2)

plot(sol$X1,sol$X2,main = ”space-space plot (y1 vs y2)”, xlab = ”y1”,ylab = ”y2”)
lines(sol$X1,sol$X2)
```

R code Exa 27.11 Solving ODEs

```
library(deSolve)

predprey <- function(t,y,parms) {
```

```
  a = 1.2*y[1]-0.6*y[1]*y[2]
  b = -0.8*y[2]+0.3*y[1]*y[2]
  list(c(a,b))
}
t=0:10
y0<-c(2, 1)
sol=ode(y = y0,parms = NULL,times = t,func = predprey)
sol<-data.frame(sol)

count=0;
for (i in 1:11){
  cat("istep=",count+1,"\n","time=",count,"\n","y1=",sol$X1[i],"\n","y2=",sol$X2[i],"\n","----------------------------------------------------------------\n")
count=count+1
}
```

Chapter 31

Finite Element Method

R code Exa 31.1 Analytical Solution for Heated Rod

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

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

R code Exa 31.2 Element Equation for Heated Rod

```
xf=10
#cm
xe=2.5
#cm
#T(0,t)=40; boundary condition
#T(10,t)=200; boundary condition
#f(x)=10; uniform heat source
f <- function(x) {
  10*(xe-x)/xe
}
int1=integrate(f = f,lower = 0,upper = xe)

g <- function(x) {
  10*(x-0)/xe
}
int2=integrate(f = g,lower = 0,upper = xe)

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")
```