

Scilab Textbook Companion for  
Numerical Methods For Engineers  
by S. C. Chapra And R. P. Canale<sup>1</sup>

Created by  
Meghana Sundaresan  
Numerical Methods  
Chemical Engineering  
VNIT  
College Teacher  
Kailas L. Wasewar  
Cross-Checked by

July 31, 2019

<sup>1</sup>Funded by a grant from the National Mission on Education through ICT, <http://spoken-tutorial.org/NMEICT-Intro>. This Textbook Companion and Scilab codes written in it can be downloaded from the "Textbook Companion Project" section at the website <http://scilab.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

Scilab 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)

**AP** Appendix to Example(Scilab Code that is an Appednix to a particular Example of the above book)

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

# Contents

List of Scilab Codes	4
1 Mathematical Modelling and Engineering Problem Solving	6
2 Programming and Software	8
3 Approximations and Round off Errors	10
4 Truncation Errors and the Taylor Series	15
5 Bracketing Methods	25
6 Open Methods	34
7 Roots of Polynomials	44
9 Gauss Elimination	51
10 LU Decomposition and matrix inverse	61
11 Special Matrices and gauss seidel	65
13 One dimensional unconstrained optimization	69
14 Multidimensional Unconstrained Optimization	72

15 Constrained Optimization	76
17 Least squares regression	82
18 Interpolation	96
19 Fourier Approximation	108
21 Newton Cotes Integration Formulas	116
22 Integration of equations	129
23 Numerical differentiation	135
25 Runga Kutta methods	139
26 Stiffness and multistep methods	156
27 Boundary Value and Eigen Value problems	165
29 Finite Difference Elliptic Equations	180
30 Finite Difference Parabolic Equations	185
31 Finite Element Method	191

# List of Scilab Codes

Exa 1.1	Analytical Solution to Falling Parachutist Problem . . . . .	6
Exa 1.2	Numerical Solution to Falling Parachutist Problem . . . . .	6
Exa 2.1	roots of quadratic . . . . .	8
Exa 3.1	Calculations of Errors . . . . .	10
Exa 3.2	Iterative error estimation . . . . .	11
Exa 3.3	Range of Integers . . . . .	11
Exa 3.4	Floating Point Numbers . . . . .	12
Exa 3.5	Machine Epsilon . . . . .	12
Exa 3.6	Interdependent Computations . . . . .	12
Exa 3.7	Subtractive Cancellation . . . . .	13
Exa 3.8	Infinite Series Evaluation . . . . .	13
Exa 4.1	Polynomial Taylor Series . . . . .	15
Exa 4.2	Taylor Series Expansion . . . . .	17
Exa 4.3	Effect of Nonlinearity and Step size on Taylor expansion . . . . .	20
Exa 4.4	Finite divided difference approximation of derivatives . . . . .	21
Exa 4.5	Error propagation in function of single variable . . . . .	22
Exa 4.6	Error propagation in multivariable function . . . . .	22
Exa 4.7	Condition Number . . . . .	24
Exa 5.1	Graphical Approach . . . . .	25
Exa 5.2	Computer Graphics to Locate Roots . . . . .	27
Exa 5.3	Bisection . . . . .	28
Exa 5.4	Error Estimates for Bisection . . . . .	29
Exa 5.5	False Position . . . . .	30
Exa 5.6	Bracketing and False Position Methods . . . . .	31

Exa 6.1	simple fixed point iteration . . . . .	34
Exa 6.2	The Two curve graphical method . . . . .	34
Exa 6.3	Newton Raphson Method . . . . .	36
Exa 6.4	Error analysis of Newton Raphson method .	37
Exa 6.5	slowly converging function with Newton Raph- son method . . . . .	38
Exa 6.6	The secant method . . . . .	38
Exa 6.7	secant and false position techniques . . . . .	39
Exa 6.8	Modified secant method . . . . .	40
Exa 6.9	modified newton raphson method . . . . .	41
Exa 6.10	fixed point iteration for nonlinear system . .	42
Exa 6.11	Newton Raphson for a nonlinear Problem .	42
Exa 7.1	Polynomial Deflation . . . . .	44
Exa 7.2	Mullers Method . . . . .	45
Exa 7.3	Bairstows Method . . . . .	46
Exa 7.4	Locate single root . . . . .	47
Exa 7.5	Solving nonlinear system . . . . .	48
Exa 7.6	Root Location . . . . .	49
Exa 7.7	Roots of Polynomials . . . . .	49
Exa 7.8	Root Location . . . . .	49
Exa 9.1	Graphical Method for two Equations . . . . .	51
Exa 9.2	Deterinants . . . . .	53
Exa 9.3	Cramers Rule . . . . .	53
Exa 9.4	Elimination of Unknowns . . . . .	54
Exa 9.5	Naive Gauss Elimination . . . . .	55
Exa 9.6	ill conditioned systems . . . . .	56
Exa 9.7	Effect of Scale on Determinant . . . . .	56
Exa 9.8	Scaling . . . . .	57
Exa 9.9	Partial Pivoting . . . . .	57
Exa 9.10	Effect of scaling on Pivoting and round off .	58
Exa 9.11	Solution of Linear Algebraic Equations . . . .	59
Exa 9.12	Gauss Jordan method . . . . .	59
Exa 10.1	LU decomposition with gauss elimination . .	61
Exa 10.2	The substitution steps . . . . .	62
Exa 10.3	Matrix inversion . . . . .	62
Exa 10.4	Matrix condition evaluation . . . . .	63
Exa 11.1	Tridiagonal solution with Thomas algorithm	65
Exa 11.2	Cholesky Decomposition . . . . .	66

Exa 11.3	Gauss Seidel method . . . . .	66
Exa 11.4	Linear systems . . . . .	67
Exa 11.5	Manipulate linear algebraic equations . . . . .	68
Exa 11.6	Analyze and solve Hilbert matrix . . . . .	68
Exa 13.1	Golden section method . . . . .	69
Exa 13.2	Quadratic interpolation . . . . .	70
Exa 13.3	Newtons method . . . . .	71
Exa 14.1	Random Search Method . . . . .	72
Exa 14.2	Path of Steepest Descent . . . . .	73
Exa 14.3	1 D function along Gradient . . . . .	73
Exa 14.4	Optimal Steepest Descent . . . . .	74
Exa 15.1	Setting up LP problem . . . . .	76
Exa 15.2	Graphical Solution . . . . .	78
Exa 15.3	Linear Programming Problem . . . . .	78
Exa 15.4	Nonlinear constrained optimization . . . . .	79
Exa 15.5	One dimensional Optimization . . . . .	80
Exa 15.6	Multidimensional Optimization . . . . .	80
Exa 15.7	Locate Single Optimum . . . . .	81
Exa 17.1	Linear regression . . . . .	82
Exa 17.2	Estimation of errors for the linear least square fit . . . . .	83
Exa 17.3.a	linear regression using computer . . . . .	84
Exa 17.3.b	linear regression using computer . . . . .	85
Exa 17.4	Linearization of a power function . . . . .	87
Exa 17.5	polynomial regression . . . . .	88
Exa 17.6	Multiple linear regression . . . . .	92
Exa 17.7	Confidence interval for linear regression . . . . .	93
Exa 17.8	Gauss Newton method . . . . .	94
Exa 18.1	Linear interpolation . . . . .	96
Exa 18.2	Quadratic interpolation . . . . .	97
Exa 18.3	Newtons divided difference Interpolating polynomials . . . . .	97
Exa 18.4	Error estimation for Newtons polynomial . . . . .	98
Exa 18.5	Error Estimates for Order of Interpolation . . . . .	99
Exa 18.6	Lagrange interpolating polynomials . . . . .	100
Exa 18.7	Lagrange interpolation using computer . . . . .	100
Exa 18.8	First order splines . . . . .	103
Exa 18.9	Quadratic splines . . . . .	103



Exa 18.10	Cubic splines . . . . .	106
Exa 19.1	Least Square Fit . . . . .	108
Exa 19.2	Continuous Fourier Series Approximation . .	109
Exa 19.3	Trendline . . . . .	110
Exa 19.4	Data Analysis . . . . .	110
Exa 19.5	Curve Fitting . . . . .	112
Exa 19.6	Polynomial Regression . . . . .	115
Exa 21.1	Single trapezoidal rule . . . . .	116
Exa 21.2	Multiple trapezoidal rule . . . . .	117
Exa 21.3	Evaluating Integrals . . . . .	118
Exa 21.4	Single Simpsons 1 by 3 rule . . . . .	122
Exa 21.5	Multiple Simpsons 1 by 3 rule . . . . .	123
Exa 21.6	Simpsons 3 by 8 rule . . . . .	124
Exa 21.7	Unequal Trapezoidal segments . . . . .	126
Exa 21.8	Simpsons Uneven data . . . . .	127
Exa 21.9	Average Temperature Determination . . . .	127
Exa 22.1	Error corrections of the trapezoidal rule . .	129
Exa 22.2	Higher order error correction of integral esti- mates . . . . .	129
Exa 22.3	Two point gauss legendre formulae . . . . .	130
Exa 22.4	Three point gauss legendre method . . . . .	131
Exa 22.5	Applying Gauss Quadrature to the falling Parachutist problem . . . . .	132
Exa 22.6	Evaluation of improper integral . . . . .	133
Exa 23.1	High accuracy numerical differentiation for- mulas . . . . .	135
Exa 23.2	Richardson extrapolation . . . . .	136
Exa 23.3	Differentiating unequally spaced data . . . .	137
Exa 23.4	Integration and Differentiation . . . . .	137
Exa 23.5	Integrate a function . . . . .	138
Exa 25.1	Eulers method . . . . .	139
Exa 25.2	Taylor series estimate for error by eulers method	140
Exa 25.3	Effect of reduced step size on Eulers method	140
Exa 25.4	Solving ODEs . . . . .	141
Exa 25.5	Heuns method . . . . .	142
Exa 25.6	Comparison of various second order RK 4 method	143
Exa 25.7	Classical fourth order RK method . . . . .	144
Exa 25.8	Comparison of Runge Kutta methods . . . .	145

Exa 25.9	Solving systems of ODE using Eulers method	147
Exa 25.10	Solving systems of ODE using RK 4 method	147
Exa 25.11	Solving systems of ODEs . . . . .	148
Exa 25.12	Adaptive fourth order RK method . . . . .	150
Exa 25.13	Runga kutta fehlberg method . . . . .	153
Exa 25.14	Adaptive Fourth order RK scheme . . . . .	154
Exa 26.1	Explicit and Implicit Euler . . . . .	156
Exa 26.2	Non self starting Heun method . . . . .	157
Exa 26.3	Estimate of per step truncation error . . . . .	160
Exa 26.4	Effect of modifier on Predictor Corrector re- sults . . . . .	161
Exa 26.5	Milnes Method . . . . .	161
Exa 26.6	Fourth order Adams method . . . . .	162
Exa 26.7	stability of Milnes and Fourth order Adams method . . . . .	163
Exa 27.1	The shooting method . . . . .	165
Exa 27.2	The shooting method for non linear problems	166
Exa 27.3	Finite Difference Approximation . . . . .	167
Exa 27.4	Mass Spring System . . . . .	168
Exa 27.5	Axially Loaded column . . . . .	168
Exa 27.6	Polynomial Method . . . . .	169
Exa 27.7	Power Method Highest Eigenvalue . . . . .	171
Exa 27.8	Power Method Lowest Eigenvalue . . . . .	171
Exa 27.9	Eigenvalues and ODEs . . . . .	172
Exa 27.10.a	Stiff ODEs . . . . .	173
Exa 27.10.b	Stiff ODEs . . . . .	177
Exa 27.11	Solving ODEs . . . . .	178
Exa 29.1	Temperature of a heated plate with fixed bound- ary conditions . . . . .	180
Exa 29.2	Flux distribution for a heated plate . . . . .	181
Exa 29.3	Heated plate with a insulated edge . . . . .	182
Exa 29.4	Heated plate with an irregular boundary . . . . .	183
Exa 30.1	Explicit solution of a one dimensional heat conduction equation . . . . .	185
Exa 30.2	Simple implicit solution of a heat conduction equation . . . . .	186
Exa 30.3	Crank Nicolson solution to the heat conduc- tion equation . . . . .	187

Exa 30.4	Comparison of Analytical and Numerical solution . . . . .	188
Exa 30.5	ADI Method . . . . .	188
Exa 31.1	Analytical Solution for Heated Rod . . . . .	191
Exa 31.2	Element Equation for Heated Rod . . . . .	193
Exa 31.3	Temperature of a heated plate . . . . .	193

# List of Figures

5.1	Graphical Approach . . . . .	26
5.2	Computer Graphics to Locate Roots . . . . .	27
6.1	The Two curve graphical method . . . . .	35
9.1	Graphical Method for two Equations . . . . .	52
15.1	Graphical Solution . . . . .	77
17.1	linear regression using computer . . . . .	85
17.2	linear regression using computer . . . . .	86
17.3	Linearization of a power function . . . . .	88
17.4	Linearization of a power function . . . . .	89
17.5	polynomial regression . . . . .	90
18.1	First order splines . . . . .	102
18.2	Quadratic splines . . . . .	104
18.3	Cubic splines . . . . .	106
19.1	Trendline . . . . .	111
19.2	Curve Fitting . . . . .	113
19.3	Curve Fitting . . . . .	114
25.1	Solving ODEs . . . . .	141
25.2	Solving systems of ODEs . . . . .	150

25.3 Solving systems of ODEs . . . . .	151
25.4 Adaptive Fourth order RK scheme . . . . .	154
26.1 Explicit and Implicit Euler . . . . .	158
26.2 Explicit and Implicit Euler . . . . .	159
27.1 The shooting method for non linear problems . . . . .	166
27.2 Eigenvalues and ODEs . . . . .	174
27.3 Eigenvalues and ODEs . . . . .	175
27.4 Stiff ODEs . . . . .	176
27.5 Stiff ODEs . . . . .	177
31.1 Analytical Solution for Heated Rod . . . . .	192

# Chapter 1

## Mathematical Modelling and Engineering Problem Solving

Scilab code Exa 1.1 Analytical Solution to Falling Parachutist Problem

```
1 clc;
2 clear;
3 g=9.8; //m/s^2; acceleration due to gravity
4 m=68.1; //kg
5 c=12.5; //kg/sec; drag coefficient
6 count=1;
7 for i=0:2:12
8     v(count)=g*m*(1-exp(-c*i/m))/c;
9     disp(v(count), "v(m/s)=" , i , "Time(s)=" )
10    count=count+1;
11 end
12 disp(g*m/c, "v(m/s)=" , "infinity" , "Time(s)=" )
```

---

Scilab code Exa 1.2 Numerical Solution to Falling Parachutist Problem

```
1 clc;
```

```
2 clear;
3 g=9.8; //m/s^2; acceleration due to gravity
4 m=68.1; //kg
5 c=12.5; //kg/sec; drag coefficient
6 count=2;
7 v(1)=0;
8 disp(v(1), "v(m/s)=" ,0, "Time(s)=")
9 for i=2:2:12
10     v(count)=v(count-1)+(g-c*v(count-1)/m)*(2);
11     disp(v(count), "v(m/s)=" ,i, "Time(s)=")
12     count=count+1;
13 end
14 disp(g*m/c, "v(m/s)=" , "infinity" , "Time(s)=")
```

---

## Chapter 2

# Programming and Software

Scilab code Exa 2.1 roots of quadratic

```
1  clc;
2  clear;
3  response=1;
4  while response==1
5  a=input("Input the value of a:")
6  b=input("Input the value of b:")
7  c=input("Input the value of c:")
8  if a==0 then
9      if b~=0 then
10         r1=-c/b;
11         disp(r1,"The root:")
12     else disp("Trivial Solution.")
13     end
14 else
15     discr=b^2-4*a*c;
16     if discr>=0 then
17         r1=(-b+sqrt(discr))/(2*a);
18         r2=(-b-sqrt(discr))/(2*a);
19         disp(r2,"and",r1,"The roots are:")
20     else
21         r1=-b/(2*a);
```



```
22         r2=r1;
23         i1=sqrt(abs(discr))/(2*a);
24         i2=-i1;
25         disp(r2+i2*sqrt(-1),r1+i1*sqrt(-1),"The
           roots are:")
26     end
27 end
28 response=input("Do you want to continue (press 1 for
           yes and 2 for no)?")
29 if response=2 then exit;
30 end
31 end
32 end
```

---

## Chapter 3

# Approximations and Round off Errors

Scilab code Exa 3.1 Calculations of Errors

```
1  clc;
2  clear;
3  lbm=9999; //cm, measured length of bridge
4  lrm=9; //cm, measured length of rivet
5  lbt=10000; //cm, true length of bridge
6  lrt=10; //cm, true length of rivet
7  //calculating true error below;
8  Etb=lbt-lbm; //cm, true error in bridge
9  Etr=lrt-lrm; //cm, true error in rivet
10 //calculating percent relative error below
11 etb=Etb*100/lbt; //percent relative error for bridge
12 etr=Etr*100/lrt; //percent relative error for rivet
13 disp("a. The true error is")
14 disp("for the bridge", "cm", Etb)
15 disp("for the rivet", "cm", Etr)
16 disp("b. The percent relative error is")
17 disp("for the bridge", "%", etb)
18 disp("for the rivet", "%", etr)
```

---

### Scilab code Exa 3.2 Iterative error estimation

```
1  clc;
2  clear;
3  n=3; //number of significant figures
4  es=0.5*(10^(2-n)); //percent, specified error
   criterion
5  x=0.5;
6  f(1)=1; //first estimate f=e^x = 1
7  ft=1.648721; //true value of e^0.5=f
8  et(1)=(ft-f(1))*100/ft;
9  ea(1)=100;
10 i=2;
11 while ea(i-1)>=es
12     f(i)=f(i-1)+(x^(i-1))/(factorial(i-1));
13     et(i)=(ft-f(i))*100/ft;
14     ea(i)=(f(i)-f(i-1))*100/f(i);
15     i=i+1;
16 end
17 for j=1:i-1
18     disp(ea(j)," Approximate estimate of error (%)=" ,et
           (j)," True % relative error=" ,f(j)," Result=" ,j,
           " term number=")
19     disp("
           _____
           ")
20 end
```

---

### Scilab code Exa 3.3 Range of Integers

```
1  n=16; //no of bits
2  num=0;
```

```

3 for i=0:(n-2)
4     num=num+(1*(2^i));
5 end
6 disp((-1*num)," to",num,"Thus a 16-bit computer word
    can store decimal integers ranging from")

```

---

### Scilab code Exa 3.4 Floating Point Numbers

```

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

```

---

### Scilab code Exa 3.5 Machine Epsilon

```

1 clc;
2 clear;
3 b=2; //base
4 t=3; //number of mantissa bits
5 E=2^(1-t); //epsilon
6 disp(E,"value of epsilon=")

```

---

### Scilab code Exa 3.6 Interdependent Computations

```

1  clc;
2  clear;
3  num=input("Input a number: ")
4  sum=0;
5  for i=1:100000
6      sum=sum+num;
7  end
8  disp(sum,"The number summed up 100,000 times is=")

```

---

### Scilab code Exa 3.7 Subtractive Cancellation

```

1  clc;
2  clear;
3  a=1;
4  b=3000.001;
5  c=3;
6  //the roots of the quadratic equation  $x^2+3000.001*x$ 
   //  $+3=0$  are found as
7  D=(b^2)-4*a*c;
8  x1=(-b+(D^0.5))/(2*a);
9  x2=(-b-(D^0.5))/(2*a);
10 disp(x2,and,x1,"The roots of the quadratic equation
    (x^2)+(3000.001*x)+3=0 are = ")

```

---

### Scilab code Exa 3.8 Infinite Series Evaluation

```

1  clc;
2  clear;
3  function y=f(x)
4      y=exp(x)
5  endfunction
6  sum=1;
7  test=0;

```

```
8 i=0;
9 term=1;
10 x=input("Input value of x:")
11 while sum~=test
12     disp(sum,"sum:",term,"term:",i,"i:")
13     disp("-----")
14     i=i+1;
15     term=term*x/i;
16     test=sum;
17     sum=sum+term;
18 end
19 disp(f(x),"Exact Value")
```

---

# Chapter 4

## Truncation Errors and the Taylor Series

Scilab code Exa 4.1 Polynomial Taylor Series

```
1  clc;
2  clear;
3  function y=f(x)
4      y=-0.1*x^4-0.15*x^3-0.5*x^2-0.25*x+1.2;
5  endfunction
6  xi=0;
7  xf=1;
8  h=xf-xi;
9  fi=f(xi); //function value at xi
10 ffa=f(xf); //actual function value at xf
11
12 //for n=0, i.e, zero order approximation
13 ff=fi;
14 Et(1)=ffa-ff; //truncation error at x=1
15 disp(fi,"The value of f at x=0 :")
16 disp(ff,"The value of f at x=1 due to zero order
    approximation :")
17 disp(Et(1),"Truncation error :")
18 disp("-----")
```

```

    ")
19
20 //for n=1, i.e, first order approximation
21 deff('y=f1(x)', 'y=derivative(f,x,order=4)')
22 f1i=f1(xi); //value of first derivative of function
    at xi
23 f1f=f1+f1i*h; //value of first derivative of function
    at xf
24 Et(2)=ffa-f1f; //truncation error at x=1
25 disp(f1i,"The value of first derivative of f at x=0
    :")
26 disp(f1f,"The value of f at x=1 due to first order
    approximation :")
27 disp(Et(2),"Truncation error :")
28 disp("-----")
    ")
29
30
31 //for n=2, i.e, second order approximation
32 deff('y=f2(x)', 'y=derivative(f1,x,order=4)')
33 f2i=f2(xi); //value of second derivative of function
    at xi
34 f2f=f1f+f2i*(h^2)/factorial(2); //value of second
    derivative of function at xf
35 Et(3)=ffa-f2f; //truncation error at x=1
36 disp(f2i,"The value of second derivative of f at x=0
    :")
37 disp(f2f,"The value of f at x=1 due to second order
    approximation :")
38 disp(Et(3),"Truncation error :")
39 disp("-----")
    ")
40
41 //for n=3, i.e, third order approximation
42 deff('y=f3(x)', 'y=derivative(f2,x,order=4)')
43 f3i=f3(xi); //value of third derivative of function
    at xi
44 f3f=f2f+f3i*(h^3)/factorial(3); //value of third

```



```

    derivative of function at xf
45 Et(4)=ffa-f3f;//truncation error at x=1
46 disp(f3i,"The value of third derivative of f at x=0
    :")
47 disp(f3f,"The value of f at x=1 due to third order
    approximation :")
48 disp(Et(4),"Truncation error :")
49 disp("-----")
    ")
50
51 //for n=4, i.e, fourth order approximation
52 deff('y=f4(x)', 'y=derivative(f3,x,order=4)')
53 f4i=f4(xi);//value of fourth derivative of function
    at xi
54 f4f=f3f+f4i*(h^4)/factorial(4);//value of fourth
    derivative of function at xf
55 Et(5)=ffa-f4f;//truncation error at x=1
56 disp(f4i,"The value of fourth derivative of f at x=0
    :")
57 disp(f4f,"The value of f at x=1 due to fourth order
    approximation :")
58 disp(Et(5),"Truncation error :")
59 disp("-----")
    ")

```

---

#### Scilab code Exa 4.2 Taylor Series Expansion

```

1 clc;
2 clear;
3 function y=f(x)
4     y=cos(x)
5 endfunction
6 xi=%pi/4;
7 xf=%pi/3;
8 h=xf-xi;

```

```

 9  fi=f(xi); //function value at xi
10  ffa=f(xf); //actual function value at xf
11
12  //for n=0, i.e, zero order approximation
13  ff=fi;
14  et(1)=(ffa-ff)*100/ffa; //percent relative error at x
    =1
15  disp(ff,"The value of f at x=1 due to zero order
    approximation :")
16  disp(et(1),"% relative error :")
17  disp("-----")
    ")
18
19  //for n=1, i.e, first order approximation
20  deff('y=f1(x)', 'y=derivative(f,x,order=4)')
21  f1i=f1(xi); //value of first derivative of function
    at xi
22  f1f=fi+f1i*h; //value of first derivative of function
    at xf
23  et(2)=(ffa-f1f)*100/ffa; // % relative error at x=1
24  disp(f1f,"The value of f at x=1 due to first order
    approximation :")
25  disp(et(2),"% relative error :")
26  disp("-----")
    ")
27
28
29  //for n=2, i.e, second order approximation
30  deff('y=f2(x)', 'y=derivative(f1,x,order=4)')
31  f2i=f2(xi); //value of second derivative of function
    at xi
32  f2f=f1f+f2i*(h^2)/factorial(2); //value of second
    derivative of function at xf
33  et(3)=(ffa-f2f)*100/ffa; // % relative error at x=1
34  disp(f2f,"The value of f at x=1 due to second order
    approximation :")
35  disp(et(3),"% relative error :")
36  disp("-----")

```

```

    ")
37
38
39 //for n=3, i.e, third order approximation
40 deff('y=f3(x)', 'y=derivative(f2,x,order=4)')
41 f3i=f3(xi); //value of third derivative of function
    at xi
42 f3f=f2f+f3i*(h^3)/factorial(3); //value of third
    derivative of function at xf
43 et(4)=(ffa-f3f)*100/ffa; // % relative error at x=1
44 disp(f3f, "The value of f at x=1 due to third order
    approximation :")
45 disp(et(4), "% relative error :")
46 disp("-----")
    ")
47
48
49 //for n=4, i.e, fourth order approximation
50 deff('y=f4(x)', 'y=derivative(f3,x,order=4)')
51 f4i=f4(xi); //value of fourth derivative of function
    at xi
52 f4f=f3f+f4i*(h^4)/factorial(4); //value of fourth
    derivative of function at xf
53 et(5)=(ffa-f4f)*100/ffa; // % relative error at x=1
54 disp(f4f, "The value of f at x=1 due to fourth order
    approximation :")
55 disp(et(5), "% relative error :")
56 disp("-----")
    ")
57
58
59 //for n=5, i.e, fifth order approximation
60 f5i=(f4(1.1*xi)-f4(0.9*xi))/(2*0.1); //value of fifth
    derivative of function at xi (central difference
    method)
61 f5f=f4f+f5i*(h^5)/factorial(5); //value of fifth
    derivative of function at xf
62 et(6)=(ffa-f5f)*100/ffa; // % relative error at x=1

```

```

63 disp(f5f,"The value of f at x=1 due to fifth order
    approximation :")
64 disp(et(6),"% relative error :")
65 disp("-----")
    ")
66
67
68 //for n=6, i.e, sixth order approximation
69 deff('y=f6(x)', 'y=derivative(f5 ,x,order=4)')
70 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)
71 f6f=f5f+f6i*(h^6)/factorial(6); //value of sixth
    derivative of function at xf
72 et(6)=(ffa-f6f)*100/ffa; // % relative error at x=1
73 disp(f6f,"The value of f at x=1 due to sixth order
    approximation :")
74 disp(et(6),"% relative error :")
75 disp("-----")
    ")

```

---

#### Scilab code Exa 4.3 Effect of Nonlinearity and Step size on Taylor expansion

```

1  clc;
2  clear;
3  m=input("Input value of m:")
4  h=input("Input value of h:")
5  function y=f(x)
6      y=x^m
7  endfunction
8  x1=1;
9  x2=x1+h;
10 fx1=f(x1);
11 fx2=fx1+m*(fx1^(m-1))*h;
12 if m==1 then

```

```

13     R=0;
14     else if m==2 then
15         R=2*(h^2)/factorial(2);
16         end
17         if m==3 then
18             R=(6*(x1)*(h^2)/factorial(2))+(6*(h^3)/
                factorial(3));
19             end
20             if m==4 then
21                 R=(12*(x1^2)*(h^2)/factorial(2))+(24*(x1)*(h
                    ^3)/factorial(3))+(24*(h^4)/factorial(4))
                ;
22             end
23     end
24     disp(R,"Remainder:",fx2,"The value by first order
        approximation:")
25     disp(f(x2),"True Value at x2:")

```

---

#### Scilab code Exa 4.4 Finite divided difference approximation of derivatives

```

1  clc;
2  clear;
3  function y=f(x)
4      y=-0.1*(x^4)-0.15*(x^3)-0.5*(x^2)-0.25*(x)+1.2
5  endfunction
6  x=0.5;
7  h=input("Input h:")
8  x1=x-h;
9  x2=x+h;
10 //forward difference method
11 fdx1=(f(x2)-f(x))/h;//derivative at x
12 et1=abs((fdx1-derivative(f,x))/derivative(f,x))*100;
13 //backward difference method
14 fdx2=(f(x)-f(x1))/h;//derivative at x
15 et2=abs((fdx2-derivative(f,x))/derivative(f,x))*100;

```

```

16 //central difference method
17 fdx3=(f(x2)-f(x1))/(2*h); //derivative at x
18 et3=abs((fdx3-derivative(f,x))/derivative(f,x))*100;
19 disp(h,"For h=")
20 disp(et1,"and percent error=",fdx1," Derivative at x
    by forward difference method=")
21 disp(et2,"and percent error=",fdx2," Derivative at x
    by backward difference method=")
22 disp(et3,"and percent error=",fdx3," Derivative at x
    by central difference method=")

```

---

#### Scilab code Exa 4.5 Error propagation in function of single variable

```

1 clc;
2 clear;
3 function y=f(x)
4     y=x^3
5 endfunction
6 x=2.5;
7 delta=0.01;
8 deltafx=abs(derivative(f,x))*delta;
9 fx=f(x);
10 disp(fx+deltafx,"and",fx-deltafx,"true value is
    between")

```

---

#### Scilab code Exa 4.6 Error propagation in multivariable function

```

1 clc;
2 clear;
3 function y=f(F,L,E,I)
4     y=(F*(L^4))/(8*E*I)
5 endfunction
6 Fbar=50; //lb/ft

```

```

7 Lbar=30; // ft
8 Ebar=1.5*(10^8); // lb/ft^2
9 Ibar=0.06; // ft^4
10 deltaF=2; // lb/ft
11 deltaL=0.1; // ft
12 deltaE=0.01*(10^8); // lb/ft^2
13 deltaI=0.0006; // ft^4
14 ybar=(Fbar*(Lbar^4))/(8*Ebar*Ibar);
15 function y=f1(F)
16     y=(F*(Lbar^4))/(8*Ebar*Ibar)
17 endfunction
18 function y=f2(L)
19     y=(Fbar*(L^4))/(8*Ebar*Ibar)
20 endfunction
21 function y=f3(E)
22     y=(Fbar*(Lbar^4))/(8*E*Ibar)
23 endfunction
24 function y=f4(I)
25     y=(Fbar*(Lbar^4))/(8*Ebar*I)
26 endfunction
27
28 deltay=abs(derivative(f1,Fbar))*deltaF+abs(
    derivative(f2,Lbar))*deltaL+abs(derivative(f3,
    Ebar))*deltaE+abs(derivative(f4,Ibar))*deltaI;
29
30 disp(ybar+deltay,"and",ybar-deltay,"The value of y
    is between:")
31 ymin=((Fbar-deltaF)*((Lbar-deltaL)^4))/(8*(Ebar+
    deltaE)*(Ibar+deltaI));
32 ymax=((Fbar+deltaF)*((Lbar+deltaL)^4))/(8*(Ebar-
    deltaE)*(Ibar-deltaI));
33 disp(ymin,"ymin is calculated at lower extremes of F
    , L, E, I values as =")
34 disp(ymax,"ymax is calculated at higher extremes of
    F, L, E, I values as =")

```

---

### Scilab code Exa 4.7 Condition Number

```
1  clc;
2  clear;
3  function y=f(x)
4      y=tan(x)
5  endfunction
6  x1bar=(%pi/2)+0.1*(%pi/2);
7  x2bar=(%pi/2)+0.01*(%pi/2);
8  //computing condition number for x1bar
9  condnum1=x1bar*derivative(f,x1bar)/f(x1bar);
10 disp(condnum1," is:",x1bar,"The condition number of
    function for x=")
11 if abs(condnum1)>1 then disp(x1bar," Function is ill-
    conditioned for x=")
12 end
13 //computing condition number for x2bar
14 condnum2=x2bar*derivative(f,x2bar)/f(x2bar);
15 disp(condnum2," is:",x2bar,"The condition number of
    function for x=")
16 if abs(condnum2)>1 then disp(x2bar," Function is ill-
    conditioned for x=")
17 end
```

---



# Chapter 5

## Bracketing Methods

Scilab code Exa 5.1 Graphical Approach

```
1  clc ;
2  clear ;
3  m=68.1; //kg
4  v=40; //m/s
5  t=10; //s
6  g=9.8; //m/s^2
7  function y=f(c)
8      y=g*m*(1-exp(-c*t/m))/c - v;
9  endfunction
10 disp("For various values of c and f(c) is found as:")
11 )
12 i=0;
13 for c=4:4:20
14     i=i+1;
15     bracket=[c f(c)];
16     disp(bracket)
17     fc(i)=f(c);
18 end
19 c=[4 8 12 16 20]
```

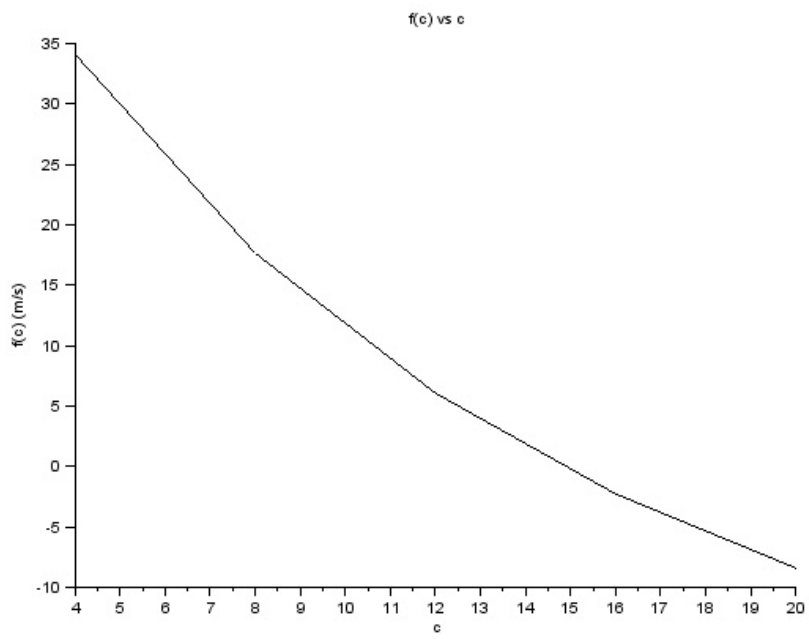


Figure 5.1: Graphical Approach

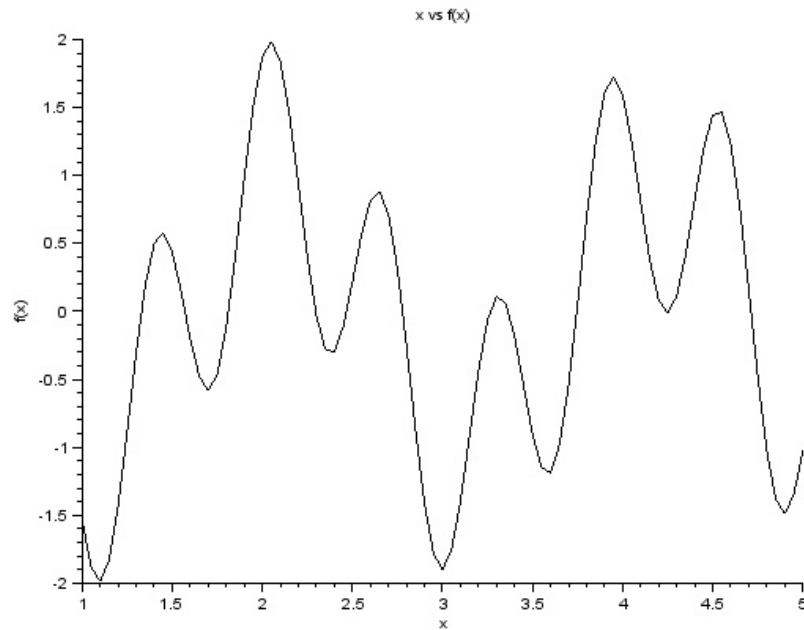


Figure 5.2: Computer Graphics to Locate Roots

```
19 plot2d(c,fc)
20 xtitle('f(c) vs c','c','f(c) (m/s)')
```

---

#### Scilab code Exa 5.2 Computer Graphics to Locate Roots

```
1 clc;
2 clear;
3 function y=f(x)
4     y=sin(10*x)+cos(3*x);
5 endfunction
6 count=1;
```

```

7 for i=1:0.05:5
8     val(count)=i;
9     func(count)=f(i);
10    count=count+1;
11 end
12 plot2d(val,func)
13 xtitle("x vs f(x)", 'x', 'f(x)')

```

---

### Scilab code Exa 5.3 Bisection

```

1 clc;
2 clear;
3 m=68.1; //kg
4 v=40; //m/s
5 t=10; //s
6 g=9.8; //m/s^2
7 function y=f(c)
8     y=g*m*(1-exp(-c*t/m))/c - v;
9 endfunction
10 x1=12;
11 x2=16;
12 xt=14.7802; //true value
13 e=input("enter the tolerable true percent error=")
14 xr=(x1+x2)/2;
15 etemp=abs(xr-xt)/xt*100; //error
16 while etemp>e
17     if f(x1)*f(xr)>0 then
18         x1=xr;
19         xr=(x1+x2)/2;
20         etemp=abs(xr-xt)/xt*100;
21     end
22     if f(x1)*f(xr)<0 then
23         x2=xr;
24         xr=(x1+x2)/2;
25         etemp=abs(xr-xt)/xt*100;

```

```

26     end
27     if f(x1)*f(xr)==0 then break;
28     end
29 end
30
31 disp(xr,"The result is=")

```

---

#### Scilab code Exa 5.4 Error Estimates for Bisection

```

1  clc;
2  clear;
3  m=68.1; //kg
4  v=40; //m/s
5  t=10; //s
6  g=9.8; //m/s^2
7  function y=f(c)
8      y=g*m*(1-exp(-c*t/m))/c - v;
9  endfunction
10 x1=12;
11 x2=16;
12 xt=14.7802; //true value
13 e=input("enter the tolerable approximate error=")
14 xr=(x1+x2)/2;
15 i=1;
16 et=abs(xr-xt)/xt*100; //error
17 disp(i,"Iteration:")
18 disp(x1,"xl:")
19 disp(x2,"xu:")
20 disp(xr,"xr:")
21 disp(et,"et(%):")
22 disp("-----")
23 etemp=100;
24 while etemp>e
25     if f(x1)*f(xr)>0 then
26         x1=xr;

```

```

27         xr=(x1+x2)/2;
28         etemp=abs(xr-x1)/xr*100;
29         et=abs(xr-xt)/xt*100;
30     end
31     if f(x1)*f(xr)<0 then
32         x2=xr;
33         xr=(x1+x2)/2;
34         etemp=abs(xr-x2)/xr*100;
35         et=abs(xr-xt)/xt*100;
36     end
37     if f(x1)*f(xr)==0 then break;
38     end
39     i=i+1;
40     disp(i," Iteration :")
41     disp(x1," x1:")
42     disp(x2," xu:")
43     disp(xr," xr:")
44     disp(et," et (%) :")
45     disp(etemp," ea (%)")
46     disp("-----")
47     end
48
49     disp(xr," The result is=")

```

---

### Scilab code Exa 5.5 False Position

```

1  clc;
2  clear;
3  m=68.1; //kg
4  v=40; //m/s
5  t=10; //s
6  g=9.8; //m/s^2
7  function y=f(c)
8      y=g*m*(1-exp(-c*t/m))/c - v;
9  endfunction

```

```

10 x1=12;
11 x2=16;
12 xt=14.7802; //true value
13 e=input("enter the tolerable true percent error=")
14 xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));
15 etemp=abs(xr-xt)/xt*100; //error
16 while etemp>e
17     if f(x1)*f(xr)>0 then
18         x1=xr;
19         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));
20         etemp=abs(xr-xt)/xt*100;
21     end
22     if f(x1)*f(xr)<0 then
23         x2=xr;
24         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));
25         etemp=abs(xr-xt)/xt*100;
26     end
27     if f(x1)*f(xr)==0 then break;
28 end
29 end
30 disp(xr,"The result is=")

```

---

### Scilab code Exa 5.6 Bracketing and False Position Methods

```

1  clc;
2  clear;
3  function y=f(x)
4      y=x^10 - 1;
5  endfunction
6  x1=0;
7  x2=1.3;
8  xt=1;
9
10 //using bisection method
11 disp("BISECTION METHOD:")

```

```

12 xr=(x1+x2)/2;
13 et=abs(xr-xt)/xt*100; //error
14 disp(1," Iteration :")
15 disp(x1," x1:")
16 disp(x2," xu:")
17 disp(xr," xr:")
18 disp(et," et (%) :")
19 disp("-----")
20 for i=2:5
21     if f(x1)*f(xr)>0 then
22         x1=xr;
23         xr=(x1+x2)/2;
24         ea=abs(xr-x1)/xr*100;
25         et=abs(xr-xt)/xt*100;
26     else if f(x1)*f(xr)<0 then
27         x2=xr;
28         xr=(x1+x2)/2;
29         ea=abs(xr-x2)/xr*100;
30         et=abs(xr-xt)/xt*100;
31     end
32 end
33 if f(x1)*f(xr)==0 then break;
34 end
35 disp(i," Iteration :")
36 disp(x1," x1:")
37 disp(x2," xu:")
38 disp(xr," xr:")
39 disp(et," et (%) :")
40 disp(ea," ea (%)")
41 disp("-----")
42 end
43
44 //using false position method
45 disp("FALSE POSITION METHOD:")
46 x1=0;
47 x2=1.3;
48 xt=1;
49 xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));;

```



```

50 et=abs(xr-xt)/xt*100; //error
51 disp(1," Iteration :")
52 disp(x1," x1:")
53 disp(x2," xu:")
54 disp(xr," xr:")
55 disp(et," et (%) :")
56 disp("-----")
57 for i=2:5
58     if f(x1)*f(xr)>0 then
59         x1=xr;
60         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));
61         ea=abs(xr-x1)/xr*100;
62         et=abs(xr-xt)/xt*100;
63
64     else if f(x1)*f(xr)<0 then
65         x2=xr;
66         xr=x1-(f(x1)*(x2-x1))/(f(x2)-f(x1));
67         ea=abs(xr-x2)/xr*100;
68         et=abs(xr-xt)/xt*100;
69     end
70 end
71 if f(x1)*f(xr)==0 then break;
72 end
73 disp(i," Iteration :")
74 disp(x1," x1:")
75 disp(x2," xu:")
76 disp(xr," xr:")
77 disp(et," et (%) :")
78 disp(ea," ea (%)")
79 disp("-----")
80 end

```

---

# Chapter 6

## Open Methods

Scilab code Exa 6.1 simple fixed point iteration

```
1 //clc()
2 //f(x) = exp(-x) - x;
3 //using simple fixed point iteration ,  $X_{i+1} = \exp(-X_i)$ 
4 x = 0; //initial guess
5 for i = 1:11
6     if i == 1 then
7         y(i) = x;
8     else
9         y(i) = exp(-y(i-1));
10        e(i) = (y(i) - y(i-1)) * 100 / y(i);
11    end
12 end
13 disp(y,"x = ")
14 disp("%",e,"e = ")
```

---

Scilab code Exa 6.2 The Two curve graphical method

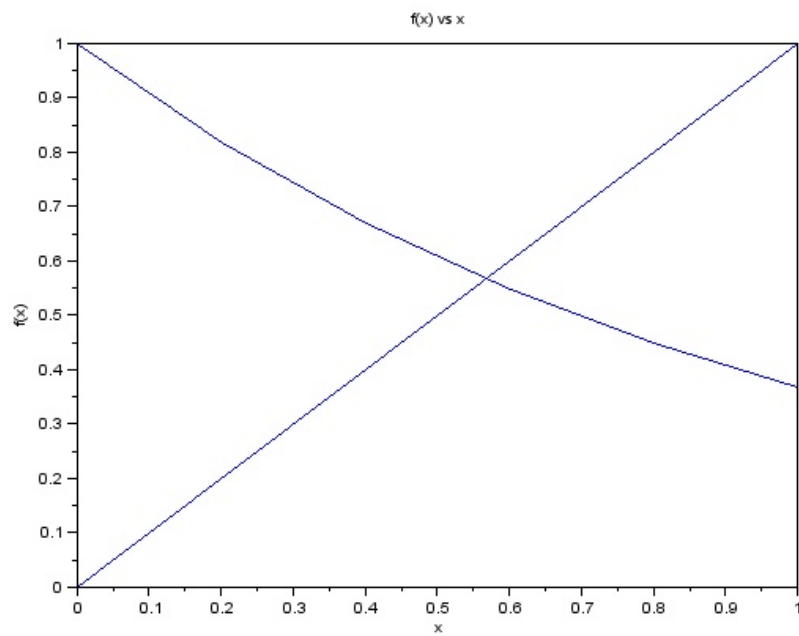


Figure 6.1: The Two curve graphical method

```

1 //clc ()
2 //y1 = x
3 //y2 = exp(-x)
4 for i = 1:6
5     if i == 1 then
6         x(i) = 0;
7         else    x(i) = x(i-1) + 0.2;
8     end
9     y1(i) = x(i);
10    y2(i) = exp(-x(i));
11 end
12 disp(x,"x = ")
13 disp(y1,"y1 = ")
14 disp(y2,"y2 = ")
15 plot(x,y1);
16 plot(x,y2);
17 xtitle("f(x) vs x","x","f(x)")
18 // from the graph, we get
19 x7 = 5.7;
20 disp(x7, "answer using two curve graphical method =
    ");

```

---

### Scilab code Exa 6.3 Newton Raphson Method

```

1 //clc ()
2 //f(x) = exp(-x)-x
3 //f'(x) = -exp(-x)-1
4 for i = 1:5
5     if i == 1 then
6         x(i) = 0;
7     else
8         x(i) = x(i-1) - (exp(-x(i-1))-x(i-1))/(-exp
            (-x(i-1))-1);
9         et(i) = (x(i) - x(i-1)) * 100 / x(i);
10        x(i-1) = x(i);

```

```

11     end
12 end
13 disp(x)
14 disp(et)

```

---

#### Scilab code Exa 6.4 Error analysis of Newton Raphson method

```

1 //clc()
2 //f(x) = exp(-x) - x
3 //f'(x) = -exp(-x) - 1
4 //f''(x) = exp(-x)
5 xr = 0.56714329;
6 //E(ti+1) = -f''(x)* E(ti) / 2 * f'(x)
7 Et0 = 0.56714329;
8 Et1 = -exp(-xr)* ((Et0)^2) / (2 * (-exp(-xr) - 1));
9 disp("which is close to the true error of 0.06714329",Et1,"Et1 = ")
10 Et1true = 0.06714329;
11 Et2 = -exp(-xr)* ((Et1true)^2) / (2 * (-exp(-xr) - 1));
12 disp("which is close to the true error of 0.0008323",Et2,"Et2 = ")
13 Et2true = 0.0008323;
14 Et3 = -exp(-xr)* ((Et2true)^2) / (2 * (-exp(-xr) - 1));
15 disp("which is close to the true error",Et3,"Et3 = ")
16 Et4 = -exp(-xr)* ((Et3)^2) / (2 * (-exp(-xr) - 1));
17 disp("which is close to the true error",Et4,"Et4 = ")
18 disp("Thus it illustrates that the error of newton raphson method for this case is proportional (by a factor of 0.18095) to the square of the error of the previous iteration")

```

---

Scilab code Exa 6.5 slowly converging function with Newton Raphson method

```
1 //clc ()
2 z = 0.5;
3 //f(x) = x^10 - 1
4 //f'(x) = 10*x^9
5 for i = 1:40
6     if i==1 then
7         y(i) = z;
8     else
9         y(i) = y(i-1) - (y(i-1)^10 - 1)/(10*y(i-1)
          ^9)
10    end
11 end
12 disp(y)
13 disp("Thus, after the first poor prediction, the
    technique is converging on to the true root of 1
    but at a very slow rate")
```

---

Scilab code Exa 6.6 The secant method

```
1 //clc ()
2 funcprot(0)
3 //f(x) = exp(-x)-x
4 for i = 1:5
5     if i==1 then
6         x(i) = 0;
7     else
8         if i==2 then
9             x(i) = 1;
10        else
```

```

11         x(i) =x(i-1) - (exp(-x(i-1))-x(i-1))*(x(i-2)
            - x(i-1))/((exp(-x(i-2))-x(i-2))-exp(-x
            (i-1))-x(i-1)))
12         er(i) = (0.56714329 - x(i)) * 100 /
            0.56714329;
13         end
14     end
15 end
16 disp(x(1:5),"x =")
17 disp(er(3:5),"et =")

```

---

#### Scilab code Exa 6.7 secant and false position techniques

```

1 //clc()
2 funcprot(0)
3 //f(x) = log(x)
4 disp("secant method")
5 for i = 1:4
6     if i==1 then
7         x(i) = 0.5;
8     else
9         if i==2 then
10            x(i) = 5;
11        else
12            x(i) =x(i-1) - log(x(i-1)) * (x(i-2) - x(i
                -1))/(log(x(i-2)) - log(x(i-1)));
13        end
14    end
15 end
16 disp(x(1:4),"x =")
17 disp("thus , secant method is divergent")
18 disp("Now, False position method")
19     xl = 0.5;
20     xu = 5;
21 for i = 1:3

```

```

22     m = log(xl);
23     n = log(xu);
24     xr = xu - n*(xl - xu)/(m - n);
25     disp(xr,"xr = ")
26     w = log(xr);
27     if m*w < 0 then
28         xu = xr;
29     else
30         xl = xr;
31     end
32 end
33
34 disp("thus , false position method is convergent")

```

---

#### Scilab code Exa 6.8 Modified secant method

```

1 //clc ()
2 del = 0.01;
3 z = 0.56714329
4 x1 = 1;
5 //f(x) = exp(-x) - x
6 disp(x1,"x1 = ")
7 for i = 1:4
8     if i == 1 then
9         x(i) = 1
10    else
11        w = x(i-1);
12        m = exp(-x(i-1)) - x(i-1);
13        x(i-1) = x(i-1)*(1+del);
14        n = exp(-x(i-1)) - x(i-1);
15        x(i) = w - (x(i-1) - w) * m/(n-m);
16        em = (z - x(i))*100/z;
17        disp(x(i),"x = ")
18        disp("%",em,"error = ")
19    end

```



20 end

---

### Scilab code Exa 6.9 modified newton raphson method

```
1 //clc()
2 //f(x) = x^3 - 5*x^2 + 7*x -3
3 //f'(x) = 3*x^2 - 10*x + 7
4 disp("standard Newton Raphson method")
5 for i = 1:7
6     if i == 1 then
7         x(i) = 0;
8     else
9         x(i) = x(i-1) - ((x(i-1))^3 - 5*(x(i-1))^2
10             + 7*x(i-1) -3)/(3*(x(i-1))^2 - 10*(x(i-1)) + 7);
11         et(i) = (1 - x(i)) * 100 / 1;
12         disp(x(i),"x = ")
13         disp("%",et(i),"error = ")
14         x(i-1) = x(i);
15     end
16 end
17 disp("Modified Newton Raphson method")
18 //f(x) = 6*x - 10
19 for i = 1:4
20     if i == 1 then
21         x(i) = 0;
22     else
23         x(i) = x(i-1) - ((x(i-1))^3 - 5*(x(i-1))^2
24             + 7*x(i-1) -3)*((3*(x(i-1))^2 - 10*(x(i-1))
25             + 7))/((3*(x(i-1))^2 - 10*(x(i-1))
26             + 7)^2 - ((x(i-1))^3 - 5*(x(i-1))^2 + 7*
27             x(i-1) -3) * (6*x(i-1) - 10));
28         et(i) = (1 - x(i)) * 100 / 1;
29         disp(x(i),"x = ")
30         disp("%",et(i),"error = ")
31     end
32 end
```

```

26         x(i-1) = x(i);
27     end
28 end

```

---

**Scilab code Exa 6.10** fixed point iteration for nonlinear system

```

1 //clc()
2 //u(x,y) = x^2 + x*y - 10
3 //v(x,y) = y + 3*x*y^2 -57
4 for i = 1:4
5     if i == 1 then
6         x(i) = 1.5;
7         y(i) = 3.5;
8     else
9         x(i) = sqrt(10 - (x(i-1))*(y(i-1)));
10        y(i) = sqrt((57 - y(i-1))/(3*x(i)));
11        disp(x(i),"x =")
12        disp(y(i),"y =")
13    end
14 end
15 disp("Thus the approaching to the true value of x =
      2 and y = 3")

```

---

**Scilab code Exa 6.11** Newton Raphson for a nonlinear Problem

```

1 clc;
2 clear;
3 function z=u(x,y)
4     z=x^2+x*y-10
5 endfunction
6 function z=v(x,y)
7     z=y+3*x*y^2-57
8 endfunction

```

```
9 x=1.5;
10 y=3.5;
11 e=[100 100];
12 while e(1)>0.0001 & e(2)>0.0001
13     J=[2*x+y x; 3*y^2 1+6*x*y];
14     deter=determ(J);
15     u1=u(x,y);
16     v1=v(x,y);
17     x=x-((u1*J(2,2)-v1*J(1,2))/deter);
18     y=y-((v1*J(1,1)-u1*J(2,1))/deter);
19     e(1)=abs(2-x);
20     e(2)=abs(3-y);
21 end
22 bracket=[x y];
23 disp(bracket)
```

---

# Chapter 7

## Roots of Polynomials

Scilab code Exa 7.1 Polynomial Deflation

```
1  clc;
2  clear;
3  function y=f(x)
4      y=(x-4)*(x+6)
5  endfunction
6  n=2;
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=(n):-1:1
14     s=a(i);
15     a(i)=r;
16     r=s+r*t;
17 end
18 disp("The quptient is a(1)+a(2)*x where :")
19 disp(a(1)," a(1)=")
20 disp(a(2)," a(2)=")
21 disp(r," remainder=")
```

---

Scilab code Exa 7.2 Mullers Method

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

```

32 disp(i," iteration:")
33 disp(x2," xr:")
34 disp(ea," ea(%):")
35 disp("-----")
    )
36 end

```

---

### Scilab code Exa 7.3 Bairstows Method

```

1  clc;
2  clear;
3  function y=f(x)
4      y=x^5-3.5*x^4+2.75*x^3+2.125*x^2-3.875*x+1.25
5  endfunction
6  r=-1;
7  s=-1;
8  es=1; //%
9  n=6;
10 count=1;
11 ear=100;
12 eas=100;
13 a=[1.25 -3.875 2.125 2.75 -3.5 1];
14 while (ear>es) and (eas>es)
15
16     b(n)=a(n);
17     b(n-1)=a(n-1)+r*b(n);
18     for i=n-2:-1:1
19         b(i)=a(i)+r*b(i+1)+s*b(i+2);
20     end
21     c(n)=b(n);
22     c(n-1)=b(n-1)+r*c(n);
23     for i=(n-2):-1:2
24         c(i)=b(i)+r*c(i+1)+s*c(i+2);
25     end
26     //c(3)*dr+c(4)*ds=-b(2)

```

```

27     //c(2)*dr+c(3)*ds=-b(1)
28     ds=((-b(1)+(b(2)*c(2)/c(3)))/(c(3)-(c(4)*c(2)
        )/c(3)));
29     dr=(-b(2)-c(4)*ds)/c(3);
30     r=r+dr;
31     s=s+ds;
32     ear=abs(dr/r)*100;
33     eas=abs(ds/s)*100;
34     disp(count," Iteration:")
35     disp(dr," delata r:")
36     disp(ds," delata s:")
37     disp(r," r:")
38     disp(s," s:")
39     disp(ear," Error in r:")
40     disp(eas," Error in s:")
41     disp("
        _____
        ")
42     count=count+1;
43 end
44 x1=(r+(r^2 + 4*s)^0.5)/2;
45 x2=(r-(r^2 + 4*s)^0.5)/2;
46 bracket=[x1 x2];
47 disp(bracket,"The roots are:")
48 disp("x^3 - 4*x^2 + 5.25*x - 2.5", "The quotient is:")
    )
49 disp("
        _____
        ")
        _____

```

#### Scilab code Exa 7.4 Locate single root

```

1  clc;
2  clear;
3  function y=f(x)

```

```

4     y=x-cos(x)
5 endfunction
6 x1=0;
7 if f(x1)<0 then
8     x2=x1+0.001;
9     while f(x2)<0
10        x2=x2+0.001;
11    end
12 elseif x2=x1+0.001;
13    while f(x2)>0
14        x2=x2+0.001;
15    end
16 else disp(x1,"The root is=")
17 end
18 x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1));
19 disp(x,"The root is=")

```

---

#### Scilab code Exa 7.5 Solving nonlinear system

```

1 clc;
2 clear;
3 function z=u(x,y)
4     z=x^2+x*y-10
5 endfunction
6 function z=v(x,y)
7     z=y+3*x*y^2-57
8 endfunction
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 end
15 disp(x,"x=")
16 disp(y,"y=")

```



---

### Scilab code Exa 7.6 Root Location

```
1 clc;
2 clear;
3 x=poly(0, 's');
4 p=x^10 -1;
5 disp("The roots of the polynomial are:")
6 disp(roots(p))
```

---

### Scilab code Exa 7.7 Roots of Polynomials

```
1 clc;
2 clear;
3 x=poly(0, 's');
4 p=x^5 - 3.5*x^4 +2.75*x^3 +2.125*x^2 - 3.875*x +
   1.25;
5 disp("The roots of the polynomial are:")
6 disp(roots(p))
```

---

### Scilab code Exa 7.8 Root Location

```
1 clc;
2 clear;
3 function y=f(x)
4     y=x-cos(x)
5 endfunction
6 x1=0;
7 if f(x1)<0 then
8     x2=x1+0.00001;
```

```
9     while f(x2)<0
10         x2=x2+0.00001;
11     end
12 elseif x2=x1+0.00001;
13     while f(x2)>0
14         x2=x2+0.00001;
15     end
16 else disp(x1,"The root is=")
17 end
18 x=x2-(x2-x1)*f(x2)/(f(x2)-f(x1));
19 disp(x,"The root is=")
```

---

# Chapter 9

## Gauss Elimination

Scilab code Exa 9.1 Graphical Method for two Equations

```
1  clc ;
2  clear ;
3  //the equations are:
4  //3*x1 + 2*x2=18 and -x1 + 2*x2=2
5
6  //equation 1 becomes ,
7  //x2=-(3/2)*x1 + 9
8  //equation 2 becomes ,
9  //x2=-(1/2)*x1 + 1
10
11 //plotting equation 1
12 for x1=1:6
13     x2(x1)=-(3/2)*x1 + 9;
14 end
15 x1=[1 2 3 4 5 6];
16 //plotting equation 2
17 for x3=1:6
18     x4(x3)=(1/2)*x3 + 1;
19 end
```

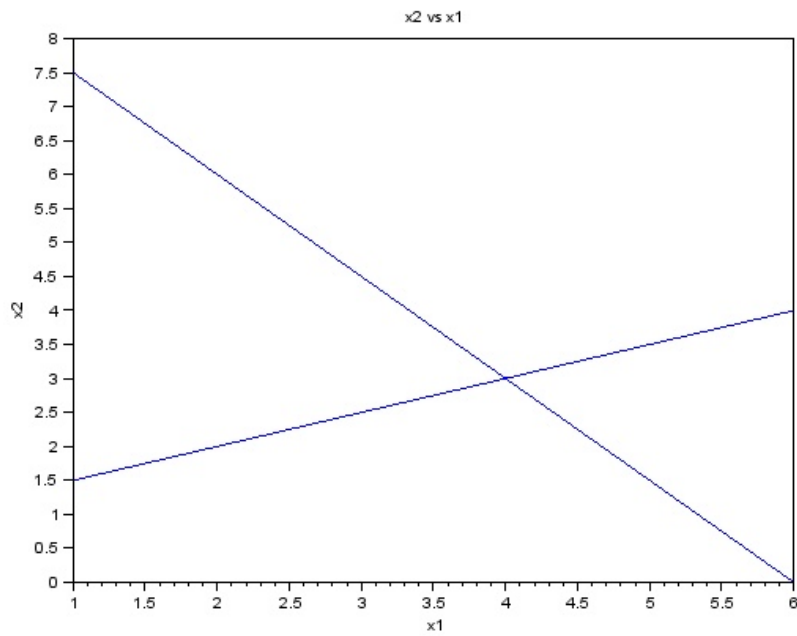


Figure 9.1: Graphical Method for two Equations

```
20 x3=[1 2 3 4 5 6];
21 plot(x1,x2)
22 plot(x3,x4)
23 xtitle("x2 vs x1","x1","x2")
24 //the lines meet at x1=4 amd x2=3
25 disp(3,"x2=","and",4,"x1=","The lines meet at=")
```

---

### Scilab code Exa 9.2 Deterinants

```
1 clc;
2 clear;
3 //For fig9.1
4 a=[3 2;-1 2];
5 disp(determ(a),"The value of determinant for system
    represented in fig 9.1 =")
6 //For fig9.2 (a)
7 a=[-0.5 1;-0.5 1];
8 disp(determ(a),"The value of determinant for system
    represented in fig 9.2 (a) =")
9 //For fig9.2 (b)
10 a=[-0.5 1;-1 2];
11 disp(determ(a),"The value of determinant for system
    represented in fig 9.2 (b) =")
12 //For fig9.2 (c)
13 a=[-0.5 1;-2.3/5 1];
14 disp(determ(a),"The value of determinant for system
    represented in fig 9.2 (c) =")
```

---

### Scilab code Exa 9.3 Cramers Rule

```
1 clc;
2 clear;
3 //the matrix or the system
```

```

4 b1=-0.01;
5 b2=0.67;
6 b3=-0.44;
7 a=[0.3 0.52 1;0.5 1 1.9;0.1 0.3 0.5];
8 a1=[a(2,2) a(2,3);a(3,2) a(3,3)];
9 A1=determ(a1);
10 a2=[a(2,1) a(2,3);a(3,1) a(3,3)];
11 A2=determ(a2);
12 a3=[a(2,1) a(2,2);a(3,1) a(3,2)];
13 A3=determ(a3);
14 D=a(1,1)*A1-a(1,2)*A2+a(1,3)*A3;
15 p=[b1 0.52 1;b2 1 1.9;b3 0.3 0.5];
16 q=[0.3 b1 1;0.5 b2 1.9;0.1 b3 0.5];
17 r=[0.3 0.52 b1;0.5 1 b2;0.1 0.3 b3];
18 x1=det(p)/D;
19 x2=det(q)/D;
20 x3=det(r)/D;
21 disp("The values are:")
22 disp(x1,"x1=")
23 disp(x2,"x2=")
24 disp(x3,"x3=")

```

---

#### Scilab code Exa 9.4 Elimination of Unknowns

```

1 clc;
2 clear;
3 //the equations are:
4 //3*x1+2*x2=18
5 //-x1+2*x2=2
6 a11=3;
7 a12=2;
8 b1=18;
9 a21=-1;
10 a22=2;
11 b2=2;

```

```
12 x1=(b1*a22-a12*b2)/(a11*a22-a12*a21);
13 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21);
14 disp(x1,"x1=")
15 disp(x2,"x2=")
```

---

### Scilab code Exa 9.5 Naive Gauss Elimination

```
1  clc;
2  clear;
3  n=3;
4  b(1)=7.85;
5  b(2)=-19.3;
6  b(3)=71.4;
7  a=[3 -0.1 -0.2;0.1 7 -0.3;0.3 -0.2 10];
8  for k=1:n-1
9      for i=k+1:n
10         fact=a(i,k)/a(k,k);
11         for j=k+1:n
12             a(i,j)=a(i,j)-fact*(a(k,j));
13         end
14         b(i)=b(i)-fact*b(k);
15     end
16 end
17 x(n)=b(n)/a(n,n);
18 for i=n-1:-1:1
19     s=b(i);
20     for j=i+1:n
21         s=s-a(i,j)*x(j)
22     end
23     x(i)=b(i)/a(i,i);
24 end
25 disp(x(1),"x1=")
26 disp(x(2),"x2=")
27 disp(x(3),"x3=")
```

---

### Scilab code Exa 9.6 ill conditioned systems

```
1  clc;
2  clear;
3  a11=1;
4  a12=2;
5  b1=10;
6  a21=1.1;
7  a22=2;
8  b2=10.4;
9  x1=(b1*a22-a12*b2)/(a11*a22-a12*a21);
10 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21);
11 disp("For the original system:")
12 disp(x1,"x1=")
13 disp(x2,"x2=")
14 a21=1.05;
15 x1=(b1*a22-a12*b2)/(a11*a22-a12*a21);
16 x2=(b2*a11-a21*b1)/(a11*a22-a12*a21);
17 disp("For the new system:")
18 disp(x1,"x1=")
19 disp(x2,"x2=")
```

---

### Scilab code Exa 9.7 Effect of Scale on Determinant

```
1  clc;
2  clear;
3  //part a
4  a=[3 2;-1 2];
5  b1=18;
6  b2=2;
7  disp(determ(a),"The determinant for part(a)=")
8  //part b
```



```

9 a=[1 2;1.1 2];
10 b1=10;
11 b2=10.4;
12 disp(determ(a),"The determinant for part(b)=")
13 //part c
14 a1=a*10;
15 b1=100;
16 b2=104;
17 disp(determ(a1),"The determinant for part(c)=")

```

---

### Scilab code Exa 9.8 Scaling

```

1 clc;
2 clear;
3 //part a
4 a=[1 0.667;-0.5 1];
5 b1=6;
6 b2=1;
7 disp(determ(a),"The determinant for part(a)=")
8 //part b
9 a=[0.5 1;0.55 1];
10 b1=5;
11 b2=5.2;
12 disp(determ(a),"The determinant for part(b)=")
13 //part c
14 b1=5;
15 b2=5.2;
16 disp(determ(a),"The determinant for part(c)=")

```

---

### Scilab code Exa 9.9 Partial Pivoting

```

1 //clc()
2 //0.0003*x1 + 3*x2 = 2.0001

```

```

3 //1*x1 + 1*x2 = 1
4 a11 = 0.0003;
5 //multiplying first equation by 1/0.0003, we get, x1
  + 10000*x2 = 6667
6 x2 = (6667-1)/(10000-1);
7 x1 = 6667 - 10000*x2;
8 disp(x2,"x2 = ")
9 disp(x1,"x1 = ")
10 disp("The error varies depending on the no. of
  significant figures used")

```

---

**Scilab code Exa 9.10** Effect of scaling on Pivoting and round off

```

1 //clc()
2 //2*x1 + 10000*x2 = 10000
3 //x1 + x2 = 2
4 x1 = 1;
5 x2 = 1;
6 disp("without scaling , applying forward elimination"
  )
7 //x1 is too small and can be neglected
8 x21 = 10000/10000;
9 x11 = 0;
10 e1 = (x1 - x11)*100/x1;
11 disp(x21,"x2 = ")
12 disp(x11,"x1 = ")
13 disp(e1,"error for x1 = ")
14 disp("with scaling")
15 //0.00002*x1 + x2 = 1
16 //now x1 is neglected because of the co efficient
17 x22 = 1;
18 x12 = 2 - x22;
19 disp(x12,"x1 = ")
20 disp(x22,"x2 = ")
21 //using original co efficient

```

```

22 //x1 can be neglected
23 disp("pivot and retaining original coefficients")
24 x22 = 10000/10000;
25 x12 = 2 - x1;
26 disp(x12,"x1 = ")
27 disp(x22,"x2 = ")

```

---

### Scilab code Exa 9.11 Solution of Linear Algebraic Equations

```

1 clc;
2 clear;
3 a=[70 1 0;60 -1 1;40 0 -1];
4 b=[636;518;307];
5 x=abs(linsolve(a,b));
6 disp("m/s ^2",x(1,1),"a=")
7 disp("N",x(2,1),"T=")
8 disp("N",x(3,1),"R=")

```

---

### Scilab code Exa 9.12 Gauss Jordan method

```

1 //clc()
2 //3*x1 - 0.1*x2 - 0.2*x3 = 7.85
3 //0.1*x1 + 7*x2 - 0.3*x3 = -19.3
4 //0.3*x1 - 0.2*x2 + 10*x3 = 71.4
5 // this can be written in matrix form as
6 A =
    [3,-0.1,-0.2,7.85;0.1,7,-0.3,-19.3;0.3,-0.2,10,71.4];

7 disp(A,"Equation in matrix form can be written as")
8 X = A(1,:) / det(A(1,1));
9 Y = A(2,:) - 0.1*X;
10 Z = A(3,:) - 0.3*X;
11 Y = Y/det(Y(1,2));

```

```
12 X = X - Y * det(X(1,2));
13 Z = Z - Y * det(Z(1,2));
14 Z = Z/det(Z(1,3));
15 X = X - Z*det(X(1,3));
16 Y = Y - Z*det(Y(1,3));
17 A = [X;Y;Z];
18 disp(A,"final matrix = ")
19 disp(det(A(1,4)),"x1 = ")
20 disp(det(A(2,4)),"x2 = ")
21 disp(det(A(3,4)),"x3 = ")
```

---

# Chapter 10

## LU Decomposition and matrix inverse

Scilab code Exa 10.1 LU decomposition with gauss elimination

```
1 //clc()
2 A = [3, -0.1, -0.2; 0.1, 7, -0.3; 0.3, -0.2, 10];
3 U = A;
4 disp(A, "A =")
5 m = det(U(1,1));
6 n = det(U(2,1));
7 a = n/m;
8 U(2,:) = U(2,:) - U(1,:) / (m/n);
9 n = det(U(3,1));
10 b = n/m;
11 U(3,:) = U(3,:) - U(1,:) / (m/n);
12 m = det(U(2,2));
13 n = det(U(3,2));
14 c = n/m;
15 U(3,:) = U(3,:) - U(2,:) / (m/n);
16 disp(U, "U = ")
17 L = [1, 0, 0; a, 1, 0; b, c, 1];
18 disp(L, "L calculated based on gauss elimination
    method = ")
```

---

### Scilab code Exa 10.2 The substitution steps

```
1 //clc()
2 A = [3,-0.1,-0.2;0.1,7,-0.3;0.3,-0.2,10];
3 B = [7.85;-19.3;71.4];
4 X = inv(A) * B;
5 disp(X,"X = ")
```

---

### Scilab code Exa 10.3 Matrix inversion

```
1 //clc()
2 A = [3,-0.1,-0.2;0.1,7,-0.3;0.3,-0.2,10];
3 //B = inv(A)
4 L = [1,0,0;0.033333,1,0;0.1,-0.02713,1];
5 U = [3,-0.1,-0.2;0,7.0033,-0.293333;0,0,10.012];
6 for i = 1:3
7     if i==1 then
8         m = [1;0;0];
9     else
10        if i==2 then
11            m = [0;1;0];
12        else
13            m = [0;0;1];
14        end
15    end
16    d = inv(L) * m;
17    x = inv(U) * d;
18    B(:,i) = x
19 end
20 disp(B)
```

---

#### Scilab code Exa 10.4 Matrix condition evaluation

```
1 //clc()
2 A = [1,1/2,1/3;1/2,1/3,1/4;1/3,1/4,1/5];
3 n = det(A(2,1));
4 A(2,:) = A(2,+)/n;
5 n = det(A(3,1));
6 A(3,:) = A(3,+)/n;
7 B = inv(A);
8 disp(A,"A = ")
9 for j = 1:3
10     a = 0;
11     for i = 1:3
12         m(i) = det(A(j,i));
13         su(j) = a + m(i);
14         a = su(j);
15     end
16 end
17 if su(1) < su(2) then
18     if su(2) < su(3) then
19         z = su(3);
20     else
21         z = su(2);
22     end
23 else
24     if su(1) < su(3) then
25         z = su(3);
26     else
27         z = su(1);
28     end
29 end
30 for j = 1:3
31     a = 0;
32     for i = 1:3
```

```
33     m(i) = det(B(j,i));
34     sm(j) = a + abs(m(i));
35     a = sm(j);
36 end
37 end
38 if sm(1) < sm(2) then
39     if sm(2) < sm(3) then
40         y = sm(3);
41     else
42         y = sm(2);
43     end
44 else
45     if sm(1) < sm(3) then
46         y = sm(3);
47     else
48         y = sm(1);
49     end
50 end
51 C = z*y;
52 disp(C,"Condition number for the matrix =")
```

---



# Chapter 11

## Special Matrices and gauss seidel

Scilab code Exa 11.1 Tridiagonal solution with Thomas algorithm

```
1 //clc()
2 A =
    [2.04,-1,0,0;-1,2.04,-1,0;0,-1,2.04,-1;0,0,-1,2.04];

3 B = [40.8;0.8;0.8;200.8];
4 g = det(A(1,2));
5 f1 = det(A(1,1));
6 e2 = det(A(2,1))/f1;
7 f2 = det(A(1,1)) - e2 * det(A(2,1));
8 e3 = det(A(2,1))/f2;
9 f3 = det(A(1,1)) - e3 * det(A(2,1));
10 e4 = det(A(2,1))/f3;
11 f4 = det(A(1,1)) - e4 * det(A(2,1));
12 M = [f1,g,0,0;e2,f2,g,0;0,e3,f3,g;0,0,e4,f4];
13 L = [1,0,0,0;det(M(2,1)),1,0,0;0,0,0,0;det(M(3,2)),
    det(M(4,3)),1];
14 U = [det(M(1,1)),g,0,0;0,0,0,0;det(M(2,2)),g,0,0;0,0,0,0;det(M(3,3)),g,0,0;0,0,0,0;det(M(4,4))];
15 r1 = det(B(1,1));
```

```

16 r2 = det(B(2,1)) - e2*det(B(1,1));
17 r3 = det(B(3,1)) - e3*r2;
18 r4= det(B(4,1)) - e4*r3;
19 N = [r1;r2;r3;r4];
20 T4 = r4/det(U(4,4));
21 T3 = (r3 - g*T4)/det(U(3,3));
22 T2 = (r2 - g*T3)/det(U(2,2));
23 T1 = (r1 - g*T2)/det(U(1,1));
24 disp(T1,"T1 = ")
25 disp(T2,"T2 = ")
26 disp(T3,"T3 = ")
27 disp(T4,"T4 = ")

```

---

### Scilab code Exa 11.2 Cholesky Decomposition

```

1 //clc()
2 A = [6,15,55;15,55,225;55,225,979];
3 s1 = 0
4 l11 = (det(A(1,1)))^(1/2);
5 //for second row
6 l21 = (det(A(2,1)))/l11;
7 l22 = (det(A(2,2)) - l21^2)^(0.5);
8 //for third row
9 l31 = (det(A(3,1)))/l11;
10 l32 = (det(A(3,2)) - l21*l31)/l22;
11 l33 = (det(A(3,3)) - l31^2 - l32^2)^(0.5);
12 L = [l11,0,0;l21,l22,0;l31,l32,l33];
13 disp(L,"L = ")

```

---

### Scilab code Exa 11.3 Gauss Seidel method

```

1 //clc()
2 //3x - 0.1y - 0.2z = 7.85

```

```

3 //0.1x + 7y - 0.3z = -19.3
4 //0.3x - 0.2y + 10z = 71.4
5 Y = 0;
6 Z = 0;
7 for i = 1:2
8     x(i) = (7.85 + 0.1*Y+0.2*Z)/3;
9     X = x(i);
10    y(i) = (-19.3 - 0.1*X + 0.3*Z)/7;
11    Y = y(i);
12    z(i) = (71.4 - 0.3*X+0.2*Y)/10;
13    Z = z(i);
14    if i==2 then
15        ex = (x(i) - x(i-1))*100/x(i);
16        ey = (y(i) - y(i-1))*100/y(i);
17        ez = (z(i) - z(i-1))*100/z(i);
18    end
19 end
20 disp(x(1:2),"x through two iterations =")
21 disp(y(1:2),"y through two iterations =")
22 disp(z(1:2),"z through two iterations =")
23 disp("%",ex,"error of x = ")
24 disp("%",ey,"error of y = ")
25 disp("%",ez,"error of z = ")

```

---

#### Scilab code Exa 11.4 Linear systems

```

1 //clc()
2 A = [1,0.5,1/3;1,2/3,1/2;1,3/4,3/5];
3 B = [1.8333333;2.166667;2.35];
4 U = inv(A);
5 X = U*B;
6 x = det(X(1,1));
7 y = det(X(2,1));
8 z = det(X(3,1));
9 disp(x,"x = ")

```

```
10 disp(y,"y = ")
11 disp(z,"z = ")
```

---

### Scilab code Exa 11.5 Manipulate linear algebraic equations

```
1 //clc()
2 A = [1,0.5,1/3;1,2/3,1/2;1,3/4,3/5];
3 B = [1.8333333;2.166667;2.35];
4 U = inv(A);
5 X = U*B;
6 x = det(X(1,1));
7 y = det(X(2,1));
8 z = det(X(3,1));
9 disp(x,"x = ")
10 disp(y,"y = ")
11 disp(z,"z = ")
```

---

### Scilab code Exa 11.6 Analyze and solve Hilbert matrix

```
1 //clc()
2 A = [1,0.5,1/3;1/2,1/3,1/4;1/3,1/4,1/5];
3 B = [1.8333333;1.0833333;0.7833333];
4 U = inv(A);
5 X = U*B;
6 x = det(X(1,1));
7 y = det(X(2,1));
8 z = det(X(3,1));
9 disp(x,"x = ")
10 disp(y,"y = ")
11 disp(z,"z = ")
```

---

# Chapter 13

## One dimensional unconstrained optimization

Scilab code Exa 13.1 Golden section method

```
1 //clc()
2 //f(x) = 2sinx - x^2/10
3 x1(1) = 0;
4 xu(1) = 4;
5 for i = 1:10
6     d(i) = ((5)^(0.5) - 1)*(xu(i) - x1(i))/2;
7     x1(i) = x1(i) + d(i);
8     x2(i) = xu(i) - d(i);
9     m(i) = 2*sin(x1(i)) - (x1(i)^2)/10;
10    n(i) = 2*sin(x2(i)) - (x2(i)^2)/10;
11    if n(i) > m(i) then
12        xu(i+1) = x1(i);
13        x1(i+1) = x1(i);
14    else
15        x1(i+1) = x2(i);
16        xu(i+1) = xu(i);
17    end
18 end
19 disp(x1,"x1 = ")
```

```

20 disp(x2,"x2 = ")
21 disp(x1,"x1 = ")
22 disp(xu,"xu = ")

```

---

### Scilab code Exa 13.2 Quadratic interpolation

```

1 //clc()
2 //f(x) = 2sinx - x^2/10
3 x0(1) = 0;
4 x1(1) = 1
5 x2(1) = 4;
6 for i = 1:6
7     m(i) = 2*sin(x0(i)) - (x0(i)^2)/10;
8     n(i) = 2*sin(x1(i)) - (x1(i)^2)/10;
9     r(i) = 2*sin(x2(i)) - (x2(i)^2)/10;
10    x3(i) = ((m(i)*(x1(i) ^ 2 -x2(i) ^ 2)) + (n(i)*(
        x2(i) ^ 2 -x0(i) ^ 2)) + (r(i)*(x0(i) ^ 2 -x1
        (i) ^ 2)))/((2*m(i)*(x1(i) -x2(i)))+(2*n(i)*(
        x2(i) -x0(i)))+(2*r(i)*(x0(i) -x1(i)))));
11    s(i) = 2*sin(x3(i)) - (x3(i)^2)/10;
12    if x1(i) > x3(i) then
13        if n(i)<s(i) then
14            x0(i+1) = x0(i);
15            x1(i+1) = x3(i);
16            x2(i+1) = x1(i);
17        else
18            x0(i+1) = x1(i);
19            x1(i+1) = x3(i);
20            x2(i+1) = x2(i);
21        end
22    else
23        if n(i)>s(i) then
24            x0(i+1) = x0(i);
25            x1(i+1) = x3(i);
26            x2(i+1) = x1(i);

```

```

27         else
28             x0(i+1) = x1(i);
29             x1(i+1) = x3(i);
30             x2(i+1) = x2(i);
31         end
32     end
33 end
34 disp(x0(1:6), "x0 = ")
35 disp(x1(1:6), "x1 = ")
36 disp(x3(1:6), "x3 = ")
37 disp(x2(1:6), "x2 = ")

```

---

### Scilab code Exa 13.3 Newtons method

```

1 //clc()
2 //f(x) = 2sinx - x^2/10
3 x(1) = 2.5;
4 //f'(x) = 2cosx - x/5
5 //f''(x) = -2sinx - 1/5
6 for i = 2:10
7     x(i) = x(i-1) - (2*cos(x(i-1)) - x(i-1)/5)/(-2*sin
            (x(i-1)) - 1/5);
8 end
9 disp(x, "x = ")

```

---

# Chapter 14

## Multidimensional Unconstrained Optimization

Scilab code Exa 14.1 Random Search Method

```
1  clc;
2  clear;
3  function z=f(x,y)
4      z=y-x-(2*(x^2))-(2*x*y)-(y^2);
5  endfunction
6  x1=-2;
7  x2=2;
8  y1=1;
9  y2=3;
10 fmax=-1*10^(-15);
11 n=10000;
12 for j=1:n
13     r=rand(1,2);
14     x=x1+(x2-x1)*r(1,1);
15     y=y1+(y2-y1)*r(1,2);
16     fn=f(x,y);
17     if fn>fmax then
18         fmax=fn;
19         xmax=x;
```



```

20         ymax=y;
21     end
22     if modulo(j,1000)==0 then
23
24         disp(j,"Iteration:")
25         disp(x,"x:")
26         disp(y,"y:")
27         disp(fn,"function value:")
28         disp("-----")
29     end
30 end

```

---

#### Scilab code Exa 14.2 Path of Steepest Descent

```

1  clc;
2  clear;
3  function z=f(x,y)
4      z=x*y*y
5  endfunction
6  p1=[2 2];
7  elevation=f(p1(1),p1(2));
8  dfx=p1(1)*p1(1);
9  dfy=2*p1(1)*p1(2);
10 theta=atan(dfy/dfx);
11 slope=(dfx^2 + dfy^2)^0.5;
12 disp(elevation,"Elevation:")
13 disp(theta,"Theta:")
14 disp(slope,"slope:")

```

---

#### Scilab code Exa 14.3 1 D function along Gradient

```

1  clc;

```

```

2 clear;
3 function z=f(x,y)
4     z=2*x*y + 2*x - x^2 - 2*y^2
5 endfunction
6 x=-1;
7 y=1;
8 dfx=2*y+2-2*x;
9 dfy=2*x-4*y;
10 //the function can thus be expressed along h axis as
11 //f((x+dfx*h),(y+dfy*h))
12 disp("180*h^2 + 72*h - 7","The final equation is=")

```

---

#### Scilab code Exa 14.4 Optimal Steepest Descent

```

1 clc;
2 clear;
3 function z=f(x,y)
4     z=2*x*y + 2*x - x^2 - 2*y^2
5 endfunction
6 x=-1;
7 y=1;
8 d2fx=-2;
9 d2fy=-4;
10 d2fxy=2;
11
12 modH=d2fx*d2fy-(d2fxy)^2;
13
14 for i=1:25
15     dfx=2*y+2-2*x;
16     dfy=2*x - 4*y;
17     //the function can thus be expressed along h axis as
18     //f((x+dfx*h),(y+dfy*h))
19     function d=g(h)
20         d=2*(x+dfx*h)*(y+dfy*h) + 2*(x+dfx*h) - (x+dfx*h
                )^2 - 2*(y+dfy*h)^2

```

```

21 endfunction
22 //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
23 //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
24 //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)
25 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));
26 x=x+dfx*h;
27 y=y+dfy*h;
28 end
29 disp([x y], "The final values are:")

```

---

# Chapter 15

## Constrained Optimization

Scilab code Exa 15.1 Setting up LP problem

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

---

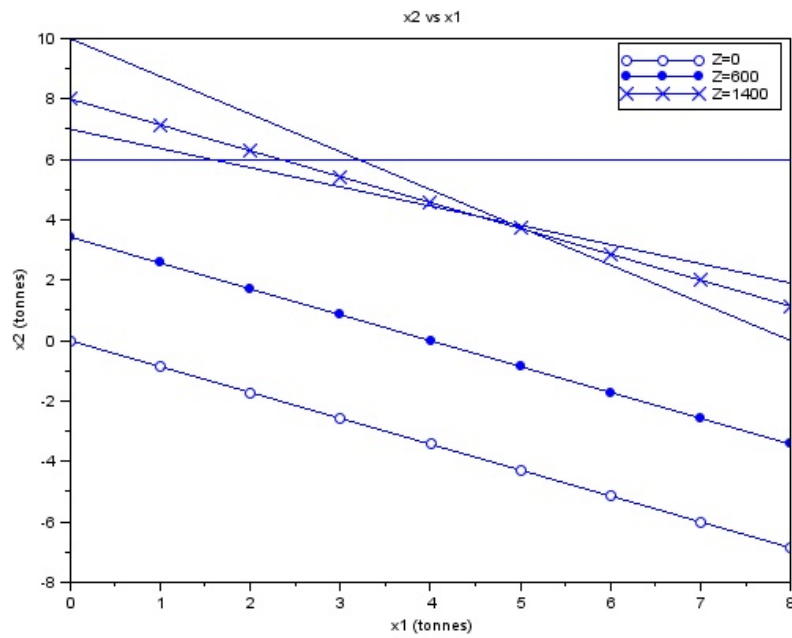


Figure 15.1: Graphical Solution

### Scilab code Exa 15.2 Graphical Solution

```
1  clc;
2  clear;
3  for x1=0:8
4      x21(x1+1)=-(7/11)*x1+7;
5      x22(x1+1)=(80-10*x1)/8;
6      x23(x1+1)=6;
7      x24(x1+1)=-150*x1/175;
8      x25(x1+1)=(600-150*x1)/175;
9      x26(x1+1)=(1400-150*x1)/175;
10 end
11 x1=0:8;
12
13 plot(x1,x24,'o-')
14 plot(x1,x25,'.-')
15 plot(x1,x26,'x-')
16 h1=legend(['Z=0';'Z=600';'Z=1400'])
17 plot(x1,x21);
18 plot(x1,x22);
19 plot(x1,x23);
20 xtitle('x2 vs x1','x1 (tonnes)','x2 (tonnes)')
```

---

### Scilab code Exa 15.3 Linear Programming Problem

```
1  clc;
2  clear;
3  x1=[4.888889 3.888889];
4  x2=[7 11];
5  x3=[10 8];
6  x4=[150 175];
7  x5=[77 80 9 6];
8  profit=[x1(1)*x4(1) x1(2)*x4(2)];
9  total=[x1(1)*x3(1)+x1(2)*x3(2) x1(1)*x3(1)+x1(2)*x3
        (2) x1(1) x1(2) profit(1)+profit(2)];
```

```

10 e=1000;
11
12 while e>total(5)
13     if total(1)<=x5(1) then
14         if total(2)<=x5(2) then
15             if total(3)<=x5(3) then
16                 if total(4)<=x5(4) then
17                     l=1;
18                 end
19             end
20         end
21     end
22     if l==1 then
23         x1(1)=x1(1)+4.888889;
24         x1(2)=x1(2)+3.888889;
25         profit=[x1(1)*x4(1) x1(2)*x4(2)];
26         total(5)=profit(1)+profit(2);
27     end
28 end
29 disp(total(5),"The maximized profit is=")

```

---

#### Scilab code Exa 15.4 Nonlinear constrained optimization

```

1  clc;
2  clear;
3  Mt=2000; //kg
4  g=9.8; //m/s^2
5  c0=200; // $
6  c1=56; // $/m
7  c2=0.1; // $/m^2
8  vc=20; //m/s
9  kc=3; //kg/(s*m^2)
10 z0=500; //m
11 t=27;
12 r=2.943652;

```

```

13 n=6;
14 A=2*%pi*r*r;
15 l=(2^0.5)*r;
16 c=3*A;
17 m=Mt/n;
18 function y=f(t)
19     y=(z0+g*m*m/(c*c)*(1-exp(-c*t/m)))*c/(g*m);
20 endfunction
21
22     while abs(f(t)-t)>0.00001
23         t=t+0.00001;
24     end
25 v=g*m*(1-exp(-c*t/m))/c;
26 disp(v,"The final value of velocity=")
27 disp(n,"The final no. of load parcels=")
28 disp("m",r,"The chute radius=")
29 disp((c0+c1*l+c2*A*A)*n,"The minimum cost ($)=")

```

---

#### Scilab code Exa 15.5 One dimensional Optimization

```

1 clc;
2 clear;
3 function y=fx(x)
4     y=-(2*sin(x))+x^2/10
5 endfunction
6 x=fminsearch(fx,0)
7 disp("After maximization:")
8 disp(x,"x=")
9 disp(fx(x),"f(x)=")

```

---

#### Scilab code Exa 15.6 Multidimensional Optimization

```

1 clc;

```



```
2 clear;
3 function f=fx(x)
4     f=-(2*x(1)*x(2)+2*x(1)-x(1)^2-2*x(2)^2)
5 endfunction
6 x=fminsearch(fx,[-1 1])
7 disp("After maximization:")
8 disp(x,"x=")
9 disp(fx(x),"f(x)=")
```

---

#### Scilab code Exa 15.7 Locate Single Optimum

```
1 clc;
2 clear;
3 function y=fx(x)
4     y=-(2*sin(x)-x^2/10)
5 endfunction
6 x=fminsearch(fx,0)
7 disp("After maximization:")
8 disp(x,"x=")
9 disp(fx(x),"f(x)=")
```

---

# Chapter 17

## Least squares regression

Scilab code Exa 17.1 Linear regression

```
1 //clc()
2 x = [1,2,3,4,5,6,7];
3 y = [0.5,2.5,2,4,3.5,6,5.5];
4 n = 7;
5 s = 0;
6 xsq = 0;
7 xsum = 0;
8 ysum = 0;
9 for i = 1:7
10     s = s + (det(x(1,i)))*(det(y(1,i)));
11     xsq = xsq + (det(x(1,i))^2);
12     xsum = xsum + det(x(1,i));
13     ysum = ysum + det(y(1,i));
14 end
15 disp(s,"sum of product of x and y =")
16 disp(xsq,"sum of square of x = ")
17 disp(xsum,"sum of all the x = ")
18 disp(ysum,"sum of all the y = ")
19 a = xsum/n;
20 b = ysum/n;
21 a1 = (n*s - xsum*ysum)/(n*xsq - xsum^2);
```

```

22 a0 = b - a*a1;
23 disp(a1,"a1 = ")
24 disp(a0,"a0 = ")
25 disp("The equation of the line obtained is y = a0 +
      a1*x")

```

---

Scilab code Exa 17.2 Estimation of errors for the linear least square fit

```

1 //clc()
2 x = [1,2,3,4,5,6,7];
3 y = [0.5,2.5,2,4,3.5,6,5.5];
4 n = 7;
5 s = 0;
6 ssum = 0;
7 xsq = 0;
8 xsum = 0;
9 ysum = 0;
10 msum = 0;
11 for i = 1:7
12     s = s + (det(x(1,i)))*(det(y(1,i))));
13     xsq = xsq + (det(x(1,i))^2);
14     xsum = xsum + det(x(1,i));
15     ysum = ysum + det(y(1,i));
16 end
17 a = xsum/n;
18 b = ysum/n;
19 a1 = (n*s - xsum*ysum)/(n*xsq - xsum^2);
20 a0 = b - a*a1;
21 for i = 1:7
22     m(i) = (det(y(1,i)) - ysum/7)^2;
23     msum = msum +m(i);
24     si(i) = (det(y(1,i)) - a0 - a1*det(x(1,i)))^2;
25     ssum = ssum + si(i);
26 end
27 disp(ysum,"sum of all y =")

```

```

28 disp(m,"(yi - yavg)^2 = ")
29 disp(msum,"total (yi - yavg)^2 = ")
30 disp(si,"(yi - a0 - a1*x)^2 = ")
31 disp(ssum,"total (yi - a0 - a1*x)^2 = ")
32 sy = (msum / (n-1))^(0.5);
33 sxy = (ssum/(n-2))^(0.5);
34 disp(sy,"sy = ")
35 disp(sxy,"sxy = ")
36 r2 = (msum - ssum)/(msum);
37 r = r2^0.5;
38 disp(r,"r = ")
39 disp("The result indicate that 86.8 percent of the
      original uncertainty has been explained by linear
      model")

```

---

### Scilab code Exa 17.3.a linear regression using computer

```

1 //clc()
2 s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
3 v =
      [10,16.3,23,27.5,31,35.6,39,41.5,42.9,45,46,45.5,46,49,50];

4 g = 9.8//m/s^2
5 m = 68.1;//kg
6 c = 12.5//kg/s
7 for i = 1:15
8     v1(i) = g*m*(1 - exp(-c*s(i)/m))/c;
9     v2(i) = g*m*s(i)/(c*(3.75+s(i)));
10 end
11 disp(s,"time = ")
12 disp(v,"measured v =")
13 disp(v1,"using equation (1.10) v1 = ")
14 disp(v2,"using equation ((17.3)) v2 = ")
15 plot2d(v,v1);
16 xtitle('v vs v1 ','v ','v1 ');

```

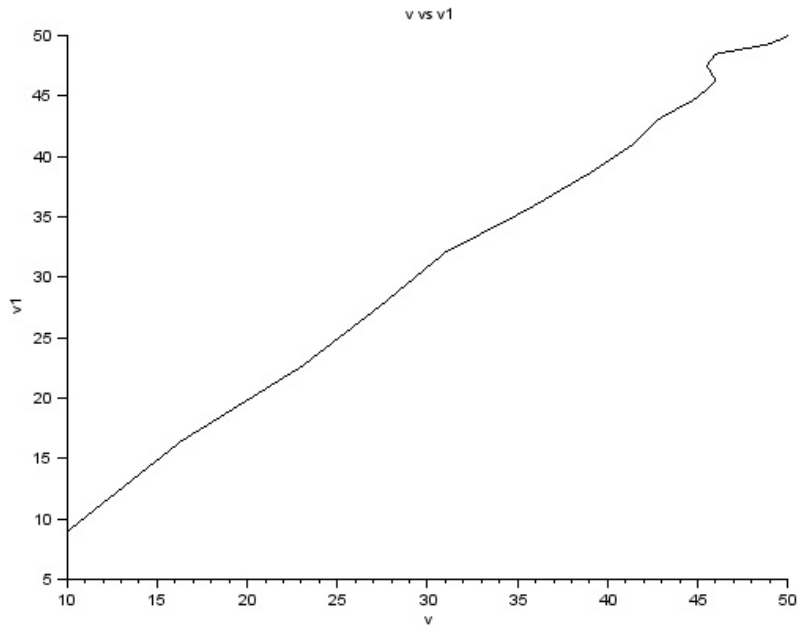


Figure 17.1: linear regression using computer

---

Scilab code Exa 17.3.b linear regression using computer

```

1 //clc()
2 s = [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15];
3 v =
    [10,16.3,23,27.5,31,35.6,39,41.5,42.9,45,46,45.5,46,49,50];
4 g = 9.8//m/s^2

```

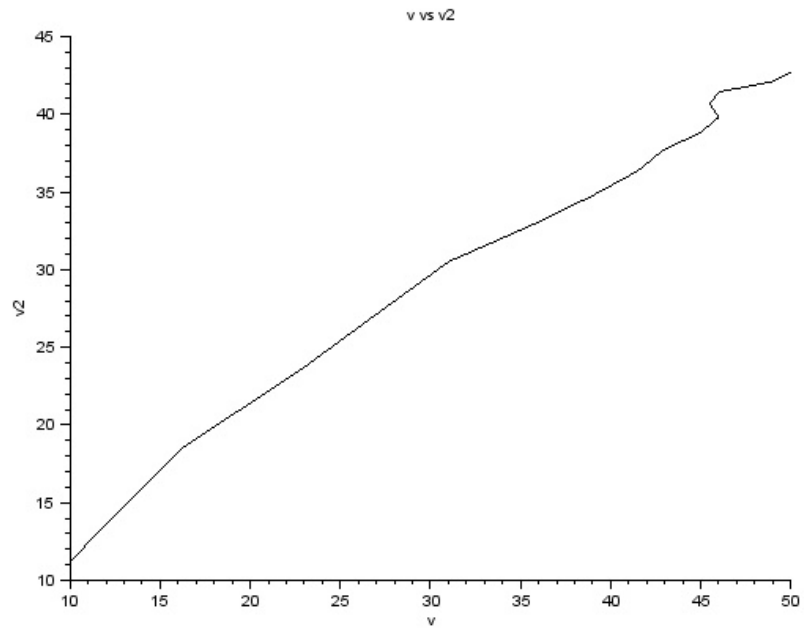


Figure 17.2: linear regression using computer

```

5 m = 68.1; //kg
6 c = 12.5 //kg/s
7 for i = 1:15
8     v1(i) = g*m*(1 - exp(-c*s(i)/m))/c;
9     v2(i) = g*m*s(i)/(c*(3.75+s(i)));
10 end
11 disp(s,"time = ")
12 disp(v,"measured v =")
13 disp(v1,"using equation (1.10) v1 = ")
14 disp(v2,"using equation ((17.3)) v2 = ")
15 plot2d(v,v2);
16 xtitle('v vs v2','v','v2');

```

---

#### Scilab code Exa 17.4 Linearization of a power function

```

1 //clc()
2 //y = a*x^b
3 a1 = -0.3000;
4 a = 10^(a1);
5 b = 1.75;
6 disp(a)
7 for i=1:5
8     x(i) = i;
9     y(i) = a*x(i)^b;
10    m(i) = log10(x(i));
11    n(i) = log10(y(i));
12 end
13 disp(x(1:5),"x = ")
14 disp(y(1:5),"y = ")
15 disp(m(1:5),"m = ")
16 disp(n(1:5),"n = ")
17 plot2d(x(1:5),y(1:5))
18 zoom_rect([0,0,7,7])
19 xtitle('y vs x','x','y')
20 plot2d(m(1:5),n(1:5))

```

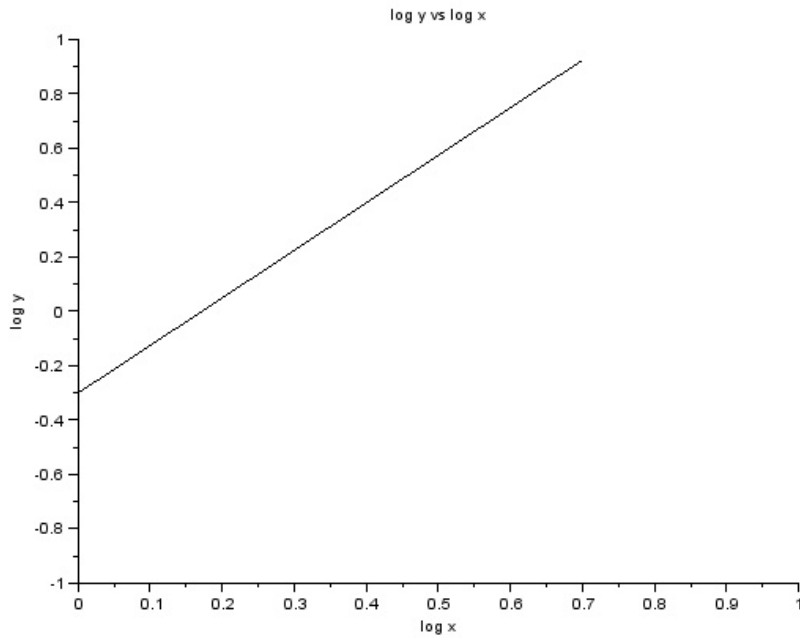


Figure 17.3: Linearization of a power function

```
21 zoom_rect([0,-1,1,1])
22 xtitle('log y vs log x', 'log x', 'log y')
```

---

#### Scilab code Exa 17.5 polynomial regression

```
1 //clc()
2 x = [0,1,2,3,4,5];
```



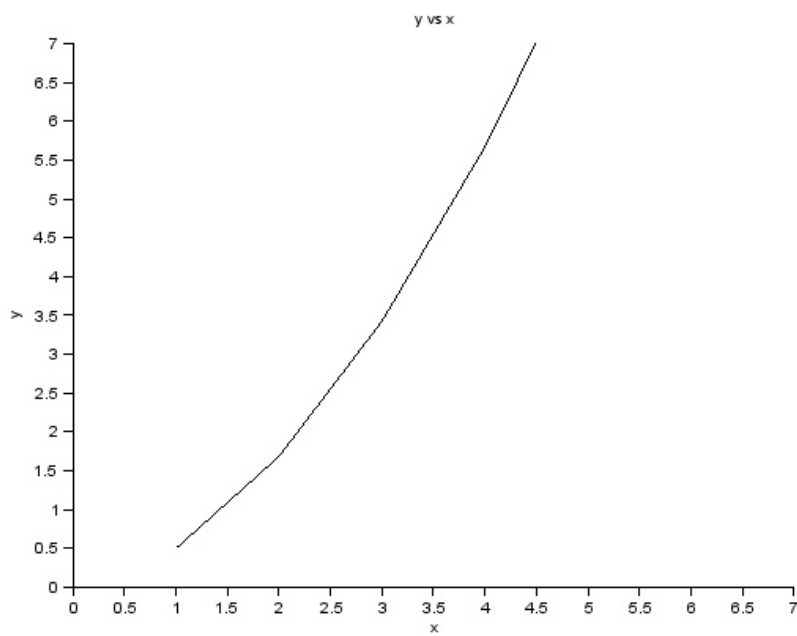


Figure 17.4: Linearization of a power function

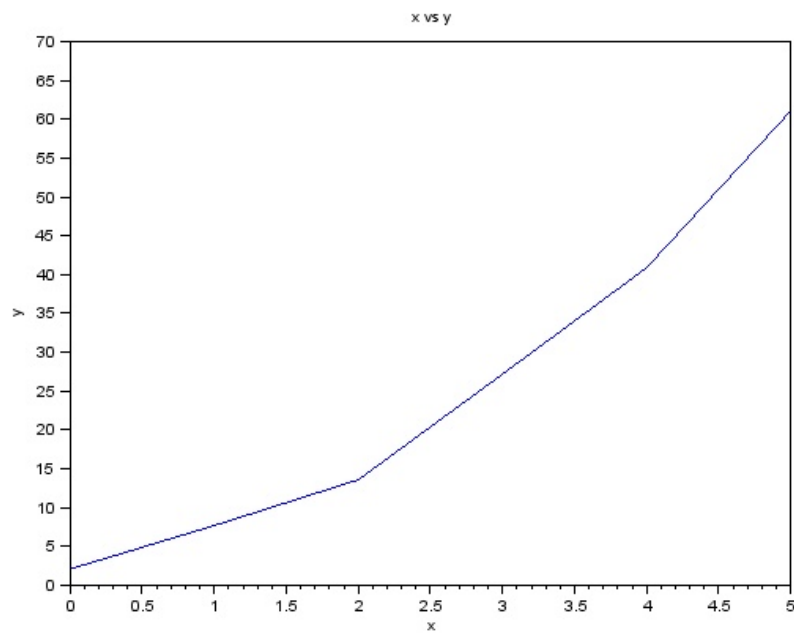


Figure 17.5: polynomial regression

```

3 y = [2.1,7.7,13.6,27.2,40.9,61.1];
4 sumy = 0;
5 sumx = 0;
6 m = 2;
7 n = 6;
8 s = 1.12;
9 xsqsum = 0;
10 xcsum = 0;
11 x4sum = 0;
12 xysum = 0;
13 x2ysum = 0;
14 rsum = 0;
15 usum = 0;
16 for i=1:6
17     sumy = sumy+y(i);
18     sumx = sumx+x(i);
19     r(i) = (y(i) - s/n)^2;
20     xsqsum = xsqsum + x(i)^2;
21     xcsum = xcsum +x(i)^3;
22     x4sum = x4sum + x(i)^4;
23     xysum = xysum + x(i)*y(i);
24     x2ysum = x2ysum + y(i)*x(i)^2;
25     rsum = r(i) + rsum;
26 end
27 disp(sumy,"sum y =")
28 disp(sumx,"sum x =")
29 xavg = sumx/n;
30 yavg = sumy/n;
31 disp(xavg,"xavg = ")
32 disp(yavg,"yavg = ")
33 disp(xsqsum,"sum x^2 =")
34 disp(xcsum,"sum x^3 =")
35 disp(x4sum,"sum x^4 =")
36 disp(xysum,"sum x*y =")
37 disp(x2ysum,"sum x^2 * y =")
38 J = [n,sumx,xsqsum;sumx,xsqsum,xcsum;xsqsum,xcsum,
      x4sum];
39 I = [sumy;xysum;x2ysum];

```

```

40 X = inv(J) * I;
41 a0 = det(X(1,1));
42 a1 = det(X(2,1));
43 a2 = det(X(3,1));
44 for i=1:6
45     u(i) = (y(i) - a0 - a1*x(i) - a2*x(i)^2)^2;
46     usum = usum + u(i);
47 end
48 disp(r,"(yi - yavg)^2 = ")
49 disp(u,"(yi - a0 - a1*x - a2*x^2)^2 = ")
50 plot(x,y);
51 xtitle('x vs y', 'x', 'y');
52 syx = (usum/(n-3))^0,5;
53 disp(syx,"The standard error of the estimate based
    on regression polynomial =")
54 R2 = (rsum - usum)/rsum;
55 disp("%",R2*100,"Percentage of original uncertainty
    that has been explained by the model = ")

```

---

### Scilab code Exa 17.6 Multiple linear regression

```

1 //clc()
2 x1 = [0,2,2.5,1,4,7];
3 x2 = [0,1,2,3,6,2];
4 x1sum = 0;
5 x2sum = 0;
6 ysum = 0;
7 x12sum = 0;
8 x22sum = 0;
9 x1ysum = 0;
10 x2ysum = 0;
11 x1x2sum = 0;
12 n = 6;
13 for i=1:6
14     y(i) = 5 + 4*x1(i) - 3*x2(i);

```

```

15     x12(i) = x1(i)^2;
16     x22(i) = x2(i)^2;
17     x1x2(i) = x1(i) * x2(i);
18     x1y(i) = x1(i) * y(i);
19     x2y(i) = x2(i) * y(i);
20     x1sum = x1sum + x1(i);
21     x2sum = x2sum + x2(i);
22     ysum = ysum + y(i);
23     x1ysum = x1ysum + x1y(i);
24     x2ysum = x2ysum + x2y(i);
25     x1x2sum = x1x2sum + x1x2(i);
26     x12sum = x12sum + x12(i);
27     x22sum = x22sum + x22(i);
28 end
29 X = [n, x1sum, x2sum; x1sum, x12sum, x1x2sum; x2sum,
      x1x2sum, x22sum];
30 Y = [ysum; x1ysum; x2ysum];
31 Z = inv(X)*Y;
32 a0 = det(Z(1,1));
33 a1 = det(Z(2,1));
34 a2 = det(Z(3,1));
35 disp(a0, "a0 = ")
36 disp(a1, "a1 = ")
37 disp(a2, "a2 = ")
38 disp("Thus, y = a0 + a1*x1 + a2*x2")

```

---

### Scilab code Exa 17.7 Confidence interval for linear regression

```

1 //clc()
2 //y = -0.859 + 1.032*x
3 Z =
      [1, 10; 1, 16.3; 1, 23; 1, 27.5; 1, 31; 1, 35.6; 1, 39; 1, 41.5; 1, 42.9; 1, 45; 1, 46
4 for i = 1:15
5     Y(i) = 9.8*68.1*(1-exp(-12.5*i/68.1))/12.5;

```

```

6 end
7 M = Z';
8 R = M*Z;
9 S = M*Y;
10 P = inv(R);
11 X = inv(R)*S;
12 a0 = det(X(1,1));
13 a1 = det(X(2,1));
14 disp(a0,"a0 = ")
15 disp(a1,"a1 = ")
16 sxy = 0.863403;
17 sa0 = (det(P(1,1)) * sxy^2)^0.5;
18 sa1 = (det(P(2,2)) * sxy^2)^0.5;
19 disp(sa0,"standard error of co efficient a0 = ")
20 disp(sa1,"standard error of co efficient a1 = ")
21 TINV = 2.160368;
22 a0 = [a0 - TINV*(sa0),a0 + TINV*(sa0)];
23 a1 = [a1 - TINV*(sa1),a1 + TINV*(sa1)];
24 disp(a0,"interval of a0 = ")
25 disp(a1,"interval of a1 = ")

```

---

#### Scilab code Exa 17.8 Gauss Newton method

```

1 //clc()
2 x = [0.25,0.75,1.25,1.75,2.25];
3 y = [0.28,0.57,0.68,0.74,0.79];
4 a0 = 1;
5 a1 = 1;
6 sr = 0.0248;
7 for i = 1:5
8     pda0(i) = 1 - exp(-a1 * x(i));
9     pda1(i) = a0 * x(i)*exp(-a1*x(i));
10 end
11 Z0 = [pda0(1),pda1(1);pda0(2),pda1(2);pda0(3),pda1
(3);pda0(4),pda1(4);pda0(5),pda1(5)]

```

```
12 disp(Z0,"Z0 = ")
13 R = Z0'*Z0;
14 S = inv(R);
15 for i = 1:5
16     y1(i) = a0 * (1-exp(-a1*x(i)));
17     D(i) = y(i) - y1(i);
18 end
19 disp(D,"D = ")
20 M = Z0'*D;
21 X = S *M;
22 disp(X,"X = ")
23 a0 = a0 + det(X(1,1));
24 a1 = a1 + det(X(2,1));
25 disp(a0,"The value of a0 after 1st iteration = ")
26 disp(a1,"The value of a1 after 1st iteration = ")
```

---

# Chapter 18

## Interpolation

Scilab code Exa 18.1 Linear interpolation

```
1 //clc ()
2 //f1(x) = f0(x) +(f(x1) - f(x0) *(x - x0)/ (x1 - x0)
3 x = 2;
4 x0 = 1;
5 x1 = 6;
6 m = 1.791759;
7 n = 0;
8 r = log(2);
9 f = 0 + (m - n) * (x - x0) / (x1 - x0);
10 disp(f,"value of ln2 for interpolation region 1 to 6
    =")
11 e = (r - f) * 100/r;
12 disp("%",e,"error by interpolation for interval [1,6]
    =")
13 x2 = 4;
14 p = 1.386294;
15 f = 0 + (p - n) * (x - x0) / (x2 - x0);
16 disp(f,"value of ln2 for interpolation region 1 to 6
    =")
17 e = (r - f) * 100/r;
18 disp("%",e,"error by interpolation for interval [1,6]
```



=")

---

### Scilab code Exa 18.2 Quadratic interpolation

```
1 //clc()
2 x = 2;
3 x0 = 1;
4 m = 0;
5 x1 = 4;
6 n = 1.386294;
7 x2 = 6;
8 p = 1.791759;
9 b0 = m;
10 b1 = (n - m)/(x1 - x0);
11 b2 = ((p - n)/(x2 - x1) - (n - m)/(x1 - x0))/(x2 -
      x0);
12 disp(b0,"b0 = ")
13 disp(b1,"b1 = ")
14 disp(b2,"b2 = ")
15 f = b0 + b1*(x - x0) + b2*(x - x0)*(x - x1);
16 disp(f,"f(2) = ")
17 r = log(2);
18 e = (r - f)*100/r;
19 disp("%",e,"error = ")
```

---

### Scilab code Exa 18.3 Newtons divided difference Interpolating polynomials

```
1 //clc()
2 x = 2;
3 x0 = 1;
4 m = 0;
5 x1 = 4;
6 n = 1.386294;
```

```

7 x3 = 5;
8 p = 1.609438;
9 x2 = 6;
10 o = 1.791759;
11 f01 = (m - n)/(x0 - x1);
12 f12 = (n - o)/(x1 - x2);
13 f23 = (p - o)/(x3 - x2);
14 f210 = (f12 - f01)/(x2 - x0);
15 f321 = (f23 - f12)/(x3 - x1);
16 f0123 = (f321 - f210) / (x3 - x0);
17 b0 = m;
18 b1 = f01;
19 b2 = f210;
20 b3 = f0123;
21 disp(b0,"b0 = ")
22 disp(b1,"b1 = ")
23 disp(b2,"b2 = ")
24 disp(b3,"b3 = ")
25 f = b0 + b1*(x - x0) + b2*(x - x0)*(x - x1) + b3*(x
    - x0)*(x - x1)*(x - x2);
26 disp(f,"f(2) = ")
27 r = log(2);
28 e = (r -f)*100/r;
29 disp("%",e,"error = ")

```

---

#### Scilab code Exa 18.4 Error estimation for Newtons polynomial

```

1 //clc()
2 x = 2;
3 x0 = 1;
4 m = 0;
5 x1 = 4;
6 n = 1.386294;
7 x3 = 5;
8 p = 1.609438;

```

```

9 x2 = 6;
10 o = 1.791759;
11 f01 = (m - n)/(x0 - x1);
12 f12 = (n - o)/(x1 - x2);
13 f23 = (p - o)/(x3 - x2);
14 f210 = (f12 - f01)/(x2 - x0);
15 f321 = (f23 - f12)/(x3 - x1);
16 f0123 = (f321 - f210) / (x3 - x0);
17 b0 = m;
18 b1 = f01;
19 b2 = f210;
20 b3 = f0123;
21 R2 = b3 * (x - x0) * (x - x1)*(x-x2);
22 disp(R2,"error R2 = ")

```

---

### Scilab code Exa 18.5 Error Estimates for Order of Interpolation

```

1 clc;
2 clear;
3 x=[1 4 6 5 3 1.5 2.5 3.5];
4 y=[0 1.3862944 1.7917595 1.6094379 1.0986123
    0.4054641 0.9162907 1.2527630];
5 n=8;
6 for i=1:n
7     fdd(i,1)=y(i);
8 end
9 for j=2:n
10    for i=1:n-j+1
11        fdd(i,j)=(fdd(i+1,j-1)-fdd(i,j-1))/(x(i+j-1)
                -x(i));
12    end
13 end
14 xterm=1;
15 yint(1)=fdd(1,1);
16

```

```

17 for order=2:n
18     xterm=xterm*(2-x(order-1));
19     yint2=yint(order-1)+fdd(1,order)*xterm;
20     Ea(order-1)=yint2-yint(order-1);
21     yint(order)=yint2;
22
23 end
24 disp(yint,"F(x)=")
25 disp(Ea,"Ea=")

```

---

#### Scilab code Exa 18.6 Lagrange interpolating polynomials

```

1 //clc()
2 x = 2;
3 x0 = 1;
4 m = 0;
5 x1 = 4;
6 n = 1.386294;
7 x2 = 6;
8 p = 1.791759;
9 f1 = (x - x1)*m/((x0 - x)) + (x - x0) * n/(x1 - x0);
10 disp(f1,"first order polynomial f1(2) = ")
11 f2 = (x - x1)*(x - x2)*m/((x0 - x1)*(x0 - x2)) + (x
    - x0)*(x - x2)*n/((x1-x0)*(x1-x2)) + (x - x0)*(x
    - x1)*p/((x2 - x0)*(x2 - x1));
12 disp(f2,"second order polynomial f2(2) = ")

```

---

#### Scilab code Exa 18.7 Lagrange interpolation using computer

```

1 //clc()
2 z = 10;
3 x =[1,3,5,7,13];
4 v = [800,2310,3090,3940,4755];

```

```

5 f1 = (z - x(5)) * v(4) / (x(4) - x(5)) + (z - x(4))
    * v(5) / (x(5) - x(4));
6 f2 = (z - x(4))*(z - x(5)) *v(3)/((x(3) - x(4))*(x
    (3) - x(5)))+(z - x(3))*(z - x(5)) *v(4)/((x(4) -
    x(3))*(x(4) - x(5)))+(z - x(4))*(z - x(3)) *v(5)
    /((x(5) - x(4))*(x(5) - x(3)));
7 f3 = (z - x(3))*(z - x(4))*(z - x(5)) *v(2)/((x(2) -
    x(4))*(x(2) - x(5))*(x(2) - x(3)))+(z - x(4))*(z
    - x(2))*(z - x(5)) *v(3)/((x(3) - x(2))*(x(3) -
    x(5))*(x(3) - x(4)))+(z - x(2))*(z - x(3))*(z - x
    (5)) *v(4)/((x(4) - x(3))*(x(4) - x(2))*(x(4) - x
    (5)))+(z - x(3))*(z - x(4))*(z - x(2)) *v(5)/((x
    (5) - x(4))*(x(5) - x(2))*(x(5) - x(3)));
8 f4 = (z - x(2))*(z - x(3))*(z - x(4))*(z - x(5)) *v
    (1)/((x(1) - x(2))*(x(1) - x(4))*(x(1) - x(5))*(x
    (1) - x(3)))+(z - x(1))*(z - x(3))*(z - x(4))*(z
    - x(5)) *v(2)/((x(2) - x(1))*(x(2) - x(4))*(x(2)
    - x(5))*(x(2) - x(3)))+(z - x(1))*(z - x(4))*(z
    - x(2))*(z - x(5)) *v(3)/((x(3) - x(1))*(x(3) - x
    (2))*(x(3) - x(5))*(x(3) - x(4)))+(z - x(1))*(z -
    x(2))*(z - x(3))*(z - x(5)) *v(4)/((x(4) - x(1))
    *(x(4) - x(3))*(x(4) - x(2))*(x(4) - x(5)))+(z -
    x(1))*(z - x(3))*(z - x(4))*(z - x(2)) *v(5)/((x
    (5) - x(1))*(x(5) - x(4))*(x(5) - x(2))*(x(5) - x
    (3)));
9 disp(f1," Velocity at 10 sec by first order
    interpolation = ")
10 disp(f2," Velocity at 10 sec by second order
    interpolation = ")
11 disp(f3," Velocity at 10 sec by third order
    interpolation = ")
12 disp(f4," Velocity at 10 sec by fourth order
    interpolation = ")

```

---

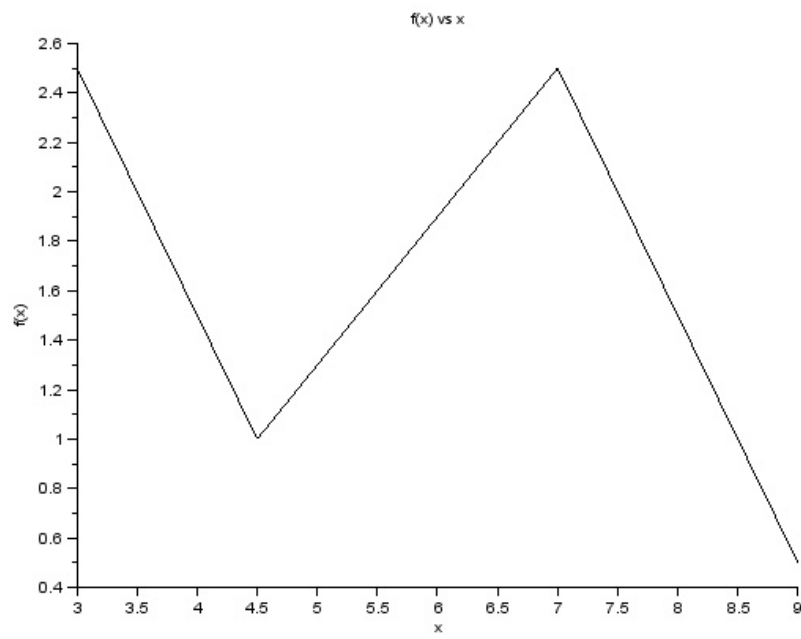


Figure 18.1: First order splines

### Scilab code Exa 18.8 First order splines

```
1 //clc()
2 x = [3,4.5,7,9];
3 fx = [2.5,1,2.5,0.5];
4 m1 = (fx(2) - fx(1))/(x(2) - x(1));
5 m2 = (fx(3) - fx(2))/(x(3) - x(2));
6 m3 = (fx(4) - fx(3))/(x(4) - x(3));
7 x1 = [3,4.5];
8 x2 = [4.5,7];
9 x3 = [7,9];
10 plot2d(x1,m1*x1+5.5);
11 plot2d(x2,m2*x2-1.7);
12 plot2d(x3,m3*x3+9.5);
13 xtitle("f(x) vs x","x","f(x)")
14 r = 5
15 z = m2*r -1.7;
16 disp(z,"The value at x = 5 is")
```

---

### Scilab code Exa 18.9 Quadratic splines

```
1 //clc()
2 x = [3,4.5,7,9];
3 fx = [2.5,1,2.5,0.5];
4 p = 4;
5 n = 3;
6 uk = n*(p-1);
7 c = 2*n - 2;
8 //20.25*a1 + 4.5*b1 + c1 = 1
9 //20.25*a2 + 4.5*b2 + c2 = 1
10 //49*a2 + 7*b2 + c2 = 2.5
11 //49*a3 + 7*b3 + c3 = 2.5
12 //9a1 + 3b1 + c1 = 2.5
```

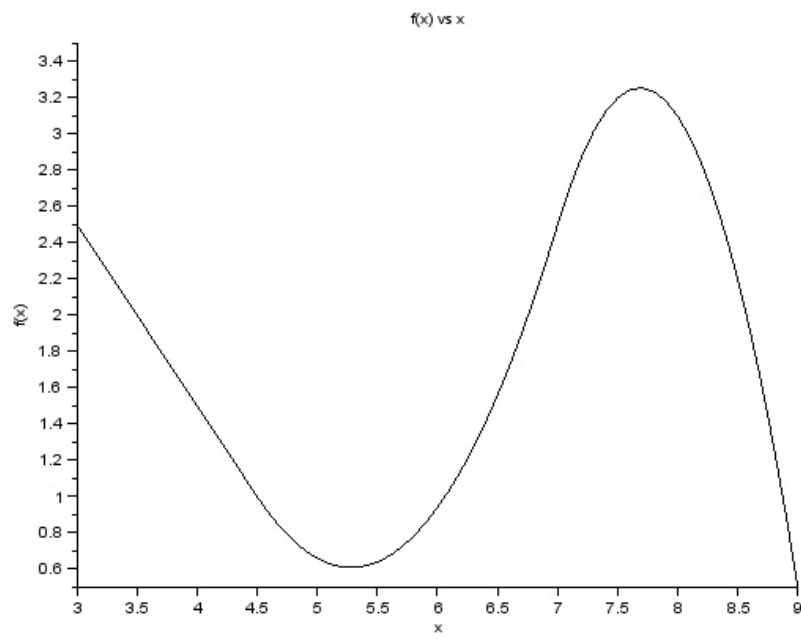


Figure 18.2: Quadratic splines



```

13 //81*a3 + 9*b3 + c3 = 0.5
14 //9*a1 + b1 = 9*a2 + b2
15 //14a2 + b2 = 14 *a3 + b3
16 a1 = 0;
17 //thus we have above 9 equations and 9 unknowns a1,
    a2,a3,b1,b2,b3,c1,c2,c3
18 //thus we get
19 A =
    [20.25,4.5,1,0,0,0,0,0,0;0,0,0,20.25,4.5,1,0,0,0;0,0,0,49,7,1,0,0,0,

20 disp(A,"A = ")
21 B = [1;1;2.5;2.5;2.5;0.5;0;0;0];
22 disp(B,"B =")
23 X = inv(A)*B;
24 a1 = det(X(1,1));
25 b1 = det(X(2,1));
26 c1 = det(X(3,1));
27 a2 = det(X(4,1));
28 b2 = det(X(5,1));
29 c2 = det(X(6,1));
30 a3 = det(X(7,1));
31 b3 = det(X(8,1));
32 c3 = det(X(9,1));
33 disp(a1,"a1 = ")
34 disp(b1,"b1 = ")
35 disp(c1,"c1 = ")
36 disp(a2,"a2 = ")
37 disp(b2,"b2 = ")
38 disp(c2,"c2 = ")
39 disp(a3,"a3 = ")
40 disp(b3,"b3 = ")
41 disp(c3,"c3 = ")
42 //thus ,f1(x) = -x + 5.5           3 <
    x < 4.5
43 //f2(x) = 0.64*x^2 -6.76*x + 18.46   4.5
    < x < 7
44 //f3(x) = -1.6*x^2 + 24.6*x - 91.3   7 <
    x < 9

```

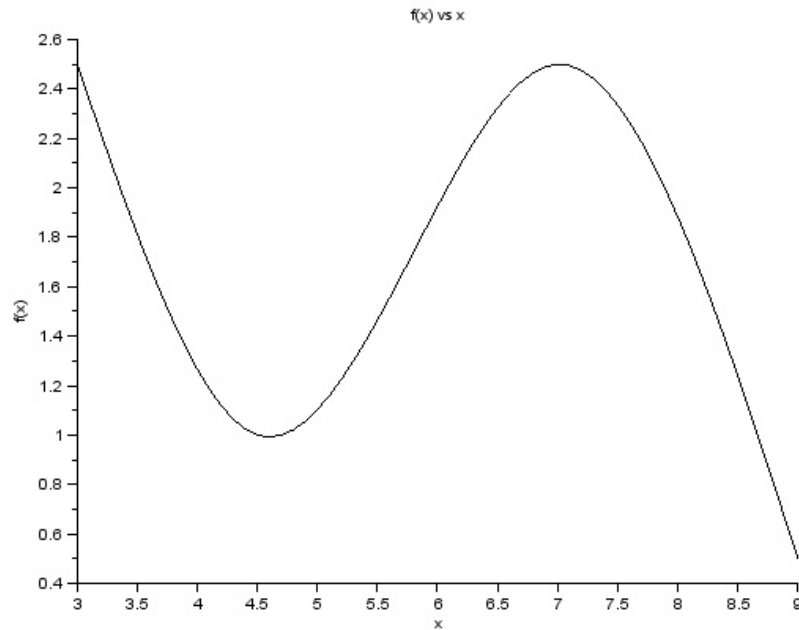


Figure 18.3: Cubic splines

```

45 x1 = 3:0.1:4.5;
46 x2 = 4.5:0.1:7;
47 x3 = 7:0.1:9;
48 plot2d(x1, -x1 + 5.5);
49 plot2d(x2, 0.64*x2^2 - 6.76*x2 + 18.46);
50 plot2d(x3, -1.6*x3^2 + 24.6*x3 - 91.3);
51 xtitle("f(x) vs x", "x", "f(x)")
52 x = 5;
53 fx = 0.64*x^2 - 6.76*x + 18.46;
54 disp(fx, "The value at x = 5 is")

```

---

### Scilab code Exa 18.10 Cubic splines

```
1 //clc()
2 x = [3,4.5,7,9];
3 fx = [2.5,1,2.5,0.5];
4 //we get the following equations for cubic splines
5 //8*f''(4.5) + 2.5*f''(7) = 9.6
6 //2.5*f''(4.5) + 9*f''(7) = -9.6
7 //above two equations give
8 m = 1.67909; //(m = f''(4.5))
9 n = -1.53308; //(n = f''(7))
10 //this values can be used to yield the final
    equation
11 //f1(x) = 0.186566 * (x - 3)^3 + 1.66667*(4.5 - x) +
    0.246894*(x - 3)
12 //in similar manner other equations can be found too
13 //f2(x) = 0.111939(7 - x)^3 - 0.102205*(x - 4.5)^3 -
    0.299621 * (7 - x) + 1.638783 * (x - 4.5)
14 //f3(x) = -0.127757*(9 - x)^3 + 1.761027 *(9 - x) +
    0.25*(x - 7)
15 x1 = 3:0.1:4.5;
16 x2 = 4.5:0.1:7;
17 x3 = 7:0.1:9;
18 plot2d(x1,0.186566 * (x1 - 3)^3 + 1.66667*(4.5 - x1)
    + 0.246894*(x1 - 3));
19 plot2d(x2,0.111939*(7 - x2)^3 - 0.102205*(x2 - 4.5)
    ^3 - 0.299621 * (7 - x2) + 1.638783 * (x2 - 4.5))
    ;
20 plot2d(x3,-0.127757*(9 - x3)^3 + 1.761027 *(9 - x3)
    + 0.25*(x3 - 7));
21 xtitle("f(x) vs x", "x", "f(x)")
22 x = 5;
23 fx = 0.111939*(7 - x)^3 - 0.102205*(x - 4.5)^3 -
    0.299621 * (7 - x) + 1.638783 * (x - 4.5);
24 disp(fx,"The value at x = 5 is")
```

---

# Chapter 19

## Fourier Approximation

Scilab code Exa 19.1 Least Square Fit

```
1  clc;
2  clear;
3  function y=f(t)
4      y=1.7+cos(4.189*t+1.0472)
5  endfunction
6  deltat=0.15;
7  t1=0;
8  t2=1.35;
9  omega=4.189;
10 del=(t2-t1)/9;
11 for i=1:10
12     t(i)=t1+del*(i-1);
13 end
14 sumy=0;
15 suma=0;
16 sumb=0;
17 for i=1:10
18     y(i)=f(t(i));
19     a(i)=y(i)*cos(omega*t(i));
20     b(i)=y(i)*sin(omega*t(i));
21     sumy=sumy+y(i);
```

```

22     suma=suma+a(i);
23     sumb=sumb+b(i);
24 end
25 A0=sumy/10;
26 A1=2*suma/10;
27 B1=2*sumb/10;
28 disp("The least square fit is  $y=A_0+A_1\cos(w_0*t)+A_2\sin(w_0*t)$ , where")
29 disp(A0,"A0=")
30 disp(A1,"A1=")
31 disp(B1,"B1=")
32 theta=atan(-B1/A1);
33 C1=(A1^2 + B1^2)^0.5;
34 disp("Alternatively, the least square fit can be expressed as")
35 disp("y=A0+C1*cos(w0*t + theta), where")
36 disp(A0,"A0=")
37 disp(theta,"Theta=")
38 disp(C1,"C1=")
39 disp("Or")
40 disp("y=A0+C1*sin(w0*t + theta + pi/2), where")
41 disp(A0,"A0=")
42 disp(theta,"Theta=")
43 disp(C1,"C1=")

```

---

### Scilab code Exa 19.2 Continuous Fourier Series Approximation

```

1  clc;
2  clear;
3  a0=0;
4  //f(t)=-1 for -T/2 to -T/4
5  //f(t)=1 for -T/4 to T/4
6  //f(t)=-1 for T/4 to T/2
7  //ak=2/T* (integration of f(t)*cos(w0*t) from -T/2
   to T/2)

```

```

8 //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))
9 //Therefore ,
10 //ak=4/(k*%pi) for k=1,5,9,.....
11 //ak=-4/(k*%pi) for k=3,7,11,.....
12 //ak=0 for k=even integers
13 //similarly we find the b's.
14 //all the b's=0
15 disp("The fourier approximtation is:")
16 disp("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) +
   .....")

```

---

### Scilab code Exa 19.3 Trendline

```

1 clc;
2 clear;
3 x=0.5:0.5:5.5;
4 for i=1:11
5     y(i)=0.9846*log(x(i))+1.0004;
6 end
7 plot(x,y)
8 xtitle("y vs x", "x", "y")

```

---

### Scilab code Exa 19.4 Data Analysis

```

1 clc;
2 clear;
3 s=[0.0002 0.0002 0.0005 0.0005 0.001 0.001];

```

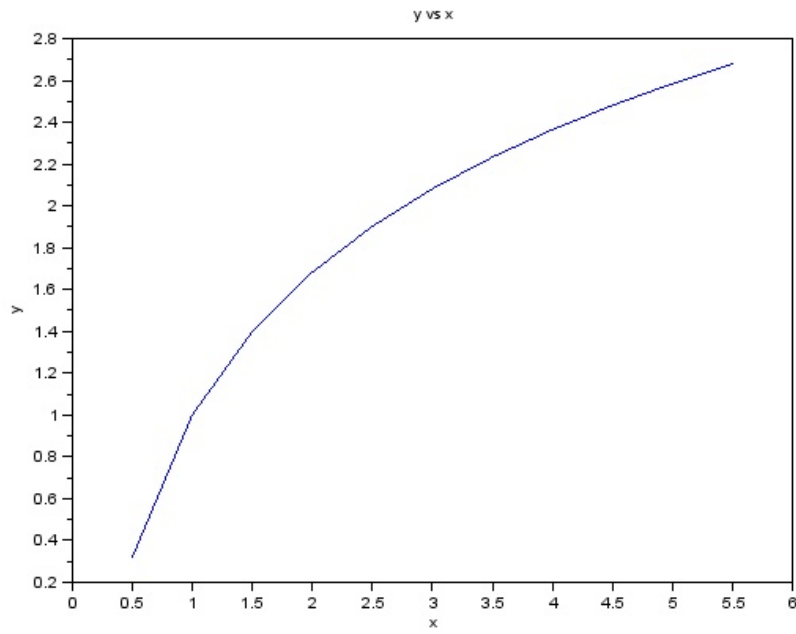


Figure 19.1: Trendline

```

4 r=[0.2 0.5 0.2 0.5 0.2 0.5];
5 u=[0.25; 0.5; 0.4; 0.75; 0.5; 1];
6 logs=log10(s);
7 logr=log10(r);
8 logu=log10(u);
9 for i=1:6
10     m(i,1)=1;
11     m(i,2)=logs(i);
12     m(i,3)=logr(i);
13 end
14
15 a=m\logu;
16 disp(10^a(1),"alpha=")
17 disp(a(2),"sigma=")
18 disp(a(3),"rho=")

```

---

### Scilab code Exa 19.5 Curve Fitting

```

1 clc;
2 clear;
3 x=0:10;
4 y=sin(x);
5 xi=0:.25:10;
6 //part a
7 yi=interp1(x,y,xi);
8 plot2d(xi,yi)
9 xtitle("y vs x (part a)","x","y")
10 //part b
11 //fitting x and y in a fifth order polynomial
12 clf();
13 p=[0.0008 -0.0290 0.3542 -1.6854 2.586 -0.0915];
14
15 for i=1:41
16     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);

```



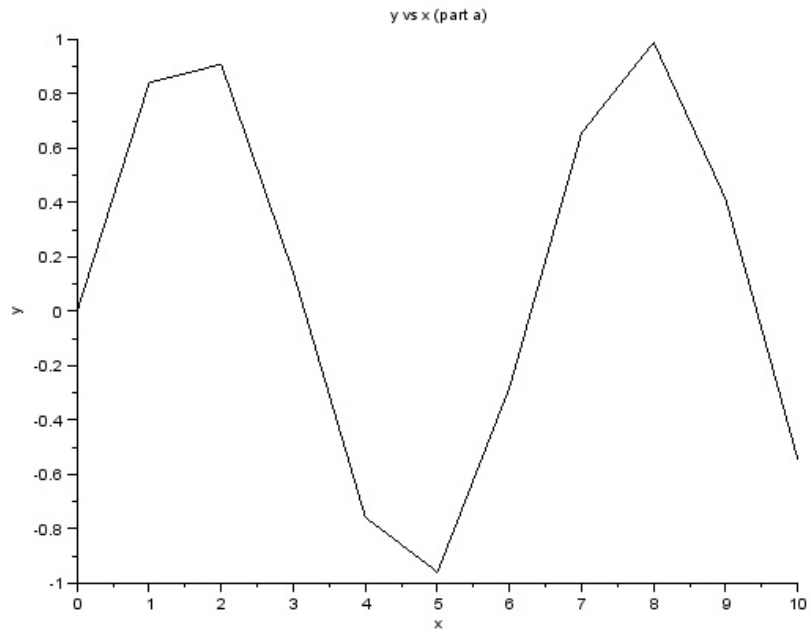


Figure 19.2: Curve Fitting

```

17 end
18 plot2d(xi,yi);
19 xtitle("y vs x (part b)","x","y")
20 //part c
21 clf();
22 d=splin(x,y)
23 yi=interp(xi,x,y,d)
24 plot2d(xi,yi)
25 xtitle("y vs x (part c)","x","y")

```

---

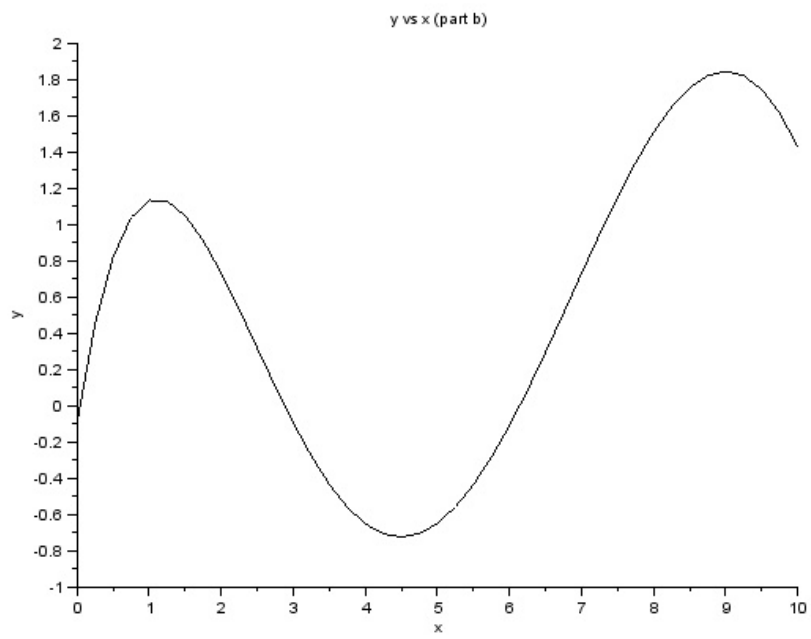


Figure 19.3: Curve Fitting

### Scilab code Exa 19.6 Polynomial Regression

```
1  clc;
2  clear;
3  x=[0.05 0.12 0.15 0.3 0.45 0.7 0.84 1.05];
4  y=[0.957 0.851 0.832 0.72 0.583 0.378 0.295 0.156];
5  sx=sum(x);
6  sxx=sum(x.*x);
7  sx3=sum(x.*x.*x);
8  sx4=sum(x.*x.*x.*x);
9  sx5=sum(x.*x.*x.*x.*x);
10 sx6=sum(x.*x.*x.*x.*x.*x);
11 n=8;
12 sy=sum(y);
13 sxy=sum(x.*y);
14 sx2y=sum(x.*x.*y);
15 sx3y=sum(x.*x.*x.*y);
16 m=[n sx sxx sx3;sx sxx sx3 sx4;sxx sx3 sx4 sx5;sx3
     sx4 sx5 sx6];
17 p=[sy;sxy;sx2y;sx3y];
18 a=m\p;
19 disp("The cubic polynomial is  $y=a_0 + a_1*x + a_2*x^2 +$ 
      $a_3*x^3$ , where  $a_0$ ,  $a_1$ ,  $a_2$  and  $a_3$  are")
20 disp(a)
```

---

# Chapter 21

## Newton Cotes Integration Formulas

Scilab code Exa 21.1 Single trapezoidal rule

```
1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  tval=1.640533;
7  a=0;
8  b=0.8;
9  fa=f(a);
10 fb=f(b);
11 l=(b-a)*((fa+fb)/2);
12 Et=tval-l;//error
13 et=Et*100/tval;//percent relative error
14
15 //by using approximate error estimate
16
17 //the second derivative of f
18 function y=g(x)
19     y=-400+4050*x-10800*x^2+8000*x^3
```

```

20 endfunction
21 f2x=intg(0,0.8,g)/(b-a); //average value of second
    derivative
22 Ea=-(1/12)*(f2x)*(b-a)^3;
23 disp(Et,"The Error Et=")
24 disp("%",et,"The percent relative error et=")
25 disp(Ea,"The approximate error estimate without
    using the true value=")

```

---

### Scilab code Exa 21.2 Multiple trapezoidal rule

```

1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  a=0;
7  b=0.8;
8  tval=1.640533;
9  n=2;
10 h=(b-a)/n;
11 fa=f(a);
12 fb=f(b);
13 fh=f(h);
14 l=(b-a)*(fa+2*fh+fb)/(2*n);
15 Et=tval-l; //error
16 et=Et*100/tval; //percent relative error
17
18 //by using approximate error estimate
19
20 //the second derivative of f
21 function y=g(x)
22     y=-400+4050*x-10800*x^2+8000*x^3
23 endfunction
24 f2x=intg(0,0.8,g)/(b-a); //average value of second

```

```

    derivative
25 Ea=-(1/12)*(f2x)*(b-a)^3/(n^2);
26 disp(Et,"The Error Et=")
27 disp("%",et,"The percent relative error et=")
28 disp(Ea,"The approximate error estimate without
    using the true value=")

```

---

### Scilab code Exa 21.3 Evaluating Integrals

```

1  clc;
2  clear;
3  g=9.8; //m/s^2; acceleration due to gravity
4  m=68.1; //kg
5  c=12.5; //kg/sec; drag coefficient
6  function v=f(t)
7      v=g*m*(1-exp(-c*t/m))/c
8  endfunction
9  tval=289.43515; //m
10 a=0;
11 b=10;
12 fa=f(a);
13 fb=f(b);
14 for i=10:10:20
15     n=i;
16     h=(b-a)/n;
17     disp(i,"No. of segments=")
18     disp(h,"Segment size=")
19     j=a+h;
20     s=0;
21     while j<b
22         s=s+f(j);
23         j=j+h;
24     end
25     l=(b-a)*(fa+2*s+fb)/(2*n);
26     Et=tval-l; //error

```

```

27     et=Et*100/tval; //percent relative error
28     disp("m",1," Estimated d=")
29     disp(et," et (%)")
30     disp("
        ")
31 end
32 for i=50:50:100
33     n=i;
34     h=(b-a)/n;
35     disp(i,"No. of segments=")
36     disp(h,"Segment size=")
37     j=a+h;
38     s=0;
39     while j<b
40         s=s+f(j);
41         j=j+h;
42     end
43     l=(b-a)*(fa+2*s+fb)/(2*n);
44     Et=tval-l; //error
45     et=Et*100/tval; //percent relative error
46     disp("m",1," Estimated d=")
47     disp(et," et (%)")
48     disp("
        ")
49 end
50 for i=100:100:200
51     n=i;
52     h=(b-a)/n;
53     disp(i,"No. of segments=")
54     disp(h,"Segment size=")
55     j=a+h;
56     s=0;
57     while j<b
58         s=s+f(j);
59         j=j+h;
60     end

```

```

61     l=(b-a)*(fa+2*s+fb)/(2*n);
62     Et=tval-1; //error
63     et=Et*100/tval; //percent relative error
64     disp("m",1," Estimated d=")
65     disp(et," et (%)")
66     disp("
        _____
        ")
67 end
68 for i=200:300:500
69     n=i;
70     h=(b-a)/n;
71     disp(i,"No. of segments=")
72     disp(h,"Segment size=")
73     j=a+h;
74     s=0;
75     while j<b
76         s=s+f(j);
77         j=j+h;
78     end
79     l=(b-a)*(fa+2*s+fb)/(2*n);
80     Et=tval-1; //error
81     et=Et*100/tval; //percent relative error
82     disp("m",1," Estimated d=")
83     disp(et," et (%)")
84     disp("
        _____
        ")
85 end
86 for i=1000:1000:2000
87     n=i;
88     h=(b-a)/n;
89     disp(i,"No. of segments=")
90     disp(h,"Segment size=")
91     j=a+h;
92     s=0;
93     while j<b
94         s=s+f(j);

```



```

95         j=j+h;
96     end
97     l=(b-a)*(fa+2*s+fb)/(2*n);
98     Et=tval-1; //error
99     et=Et*100/tval; //percent relative error
100    disp("m",l," Estimated d=")
101    disp(et," et (%)")
102    disp("
        _____
        ")
103 end
104 for i=2000:3000:5000
105     n=i;
106     h=(b-a)/n;
107     disp(i,"No. of segments=")
108     disp(h,"Segment size=")
109     j=a+h;
110     s=0;
111     while j<b
112         s=s+f(j);
113         j=j+h;
114     end
115     l=(b-a)*(fa+2*s+fb)/(2*n);
116     Et=tval-1; //error
117     et=Et*100/tval; //percent relative error
118     disp("m",l," Estimated d=")
119     disp(et," et (%)")
120     disp("
        _____
        ")
121 end
122 for i=5000:5000:10000
123     n=i;
124     h=(b-a)/n;
125     disp(i,"No. of segments=")
126     disp(h,"Segment size=")
127     j=a+h;
128     s=0;

```

```

129     while j<b
130         s=s+f(j);
131         j=j+h;
132     end
133     l=(b-a)*(fa+2*s+fb)/(2*n);
134     Et=tval-l; //error
135     et=Et*100/tval; //percent relative error
136     disp("m",l," Estimated d=")
137     disp(et," et (%)")
138     disp("
-----
")
139 end
-----

```

#### Scilab code Exa 21.4 Single Simpsons 1 by 3 rule

```

1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  a=0;
7  b=0.8;
8  tval=1.640533;
9  n=2;
10 h=(b-a)/n;
11 fa=f(a);
12 fb=f(b);
13 fh=f(h);
14 l=(b-a)*(fa+4*fh+fb)/(3*n);
15 disp(l," l=")
16 Et=tval-l; //error
17 et=Et*100/tval; //percent relative error
18
19 //by using approximate error estimate

```

```

20
21 //the fourth derivative of f
22 function y=g(x)
23     y=-21600+48000*x
24 endfunction
25 f4x=intg(0,0.8,g)/(b-a); //average value of fourth
    derivative
26 Ea=-(1/2880)*(f4x)*(b-a)^5;
27 disp(Et,"The Error Et=")
28 disp("%",et,"The percent relative error et=")
29 disp(Ea,"The approximate error estimate without
    using the true value=")

```

---

#### Scilab code Exa 21.5 Multiple Simpsons 1 by 3 rule

```

1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  a=0;
7  b=0.8;
8  tval=1.640533;
9  n=4;
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     if (-1)^count== -1 then
18         s=s+4*f(j);
19     else
20         s=s+2*f(j);

```

```

21     end
22     count=count+1;
23     j=j+h;
24 end
25 l=(b-a)*(fa+s+fb)/(3*n);
26 disp(l,"l=")
27 Et=tval-l;//error
28 et=Et*100/tval;//percent relative error
29
30 //by using approximate error estimate
31
32 //the fourth derivative of f
33 function y=g(x)
34     y=-21600+48000*x
35 endfunction
36 f4x=intg(0,0.8,g)/(b-a);//average value of fourth
    derivative
37 Ea=-(1/(180*4^4))*(f4x)*(b-a)^5;
38 disp(Et,"The Error Et=")
39 disp("%",et,"The percent relative error et=")
40 disp(Ea,"The approximate error estimate without
    using the true value=")

```

---

### Scilab code Exa 21.6 Simpsons 3 by 8 rule

```

1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  a=0;
7  b=0.8;
8  tval=1.640533;
9  //part a
10 n=3;

```

```

11 h=(b-a)/n;
12 fa=f(a);
13 fb=f(b);
14 j=a+h;
15 s=0;
16 count=1;
17 while j<b
18     s=s+3*f(j);
19     count=count+1;
20     j=j+h;
21 end
22 l=(b-a)*(fa+s+fb)/(8);
23 disp(" Part A:")
24 disp(1," l=")
25 Et=tval-1;//error
26 et=Et*100/tval;//percent relative error
27
28 //by using approximate error estimate
29
30 //the fourth derivative of f
31 function y=g(x)
32     y=-21600+48000*x
33 endfunction
34 f4x=intg(0,0.8,g)/(b-a);//average value of fourth
    derivative
35 Ea=-(1/6480)*(f4x)*(b-a)^5;
36 disp(Et," The Error Et=")
37 disp("%",et," The percent relative error et=")
38 disp(Ea," The approximate error estimate without
    using the true value=")
39
40 //part b
41 n=5;
42 h=(b-a)/n;
43 l1=(a+2*h-a)*(fa+4*f(a+h)+f(a+2*h))/6;
44 l2=(a+5*h-a-2*h)*(f(a+2*h)+3*(f(a+3*h)+f(a+4*h))+fb)
    /8;
45 l=l1+l2;

```

```

46 disp("
    ")
47 disp(" Part B:")
48 disp(1," l=")
49 Et=tval-1;//error
50 et=Et*100/tval;//percent relative error
51 disp(Et,"The Error Et=")
52 disp("%",et,"The percent relative error et=")

```

---

#### Scilab code Exa 21.7 Unequal Trapezoidal segments

```

1 clc;
2 clear;
3 function y=f(x)
4     y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5 endfunction
6 tval=1.640533;
7 x=[0 0.12 0.22 0.32 0.36 0.4 0.44 0.54 0.64 0.7
    0.8];
8 for i=1:11
9     func(i)=f(x(i));
10 end
11 l=0;
12 for i=1:10
13     l=l+(x(i+1)-x(i))*(func(i)+func(i+1))/2;
14 end
15 disp(1," l=")
16 Et=tval-1;//error
17 et=Et*100/tval;//percent relative error
18 disp(Et,"The Error Et=")
19 disp("%",et,"The percent relative error et=")

```

---

### Scilab code Exa 21.8 Simpsons Uneven data

```
1  clc;
2  clear;
3  function y=f(x)
4      y=(0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5)
5  endfunction
6  tval=1.640533;
7  x=[0 0.12 0.22 0.32 0.36 0.4 0.44 0.54 0.64 0.7
    0.8];
8  for i=1:11
9      func(i)=f(x(i));
10 end
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 disp(1," l=")
19 Et=tval-l; //error
20 et=Et*100/tval; //percent relative error
21 disp(Et," The Error Et=")
22 disp("%",et," The percent relative error et=")
```

---

### Scilab code Exa 21.9 Average Temperature Determination

```
1  clc;
2  clear;
3  function t=f(x,y)
4      t=2*x*y+2*x-x^2-2*y^2+72
5  endfunction
6  len=8; //m, length
```

```

7  wid=6; //m, width
8  a=0;
9  b=len;
10 n=2;
11 h=(b-a)/n;
12 a1=0;
13 b1=wid;
14 h1=(b1-a1)/n;
15
16 fa=f(a,0);
17 fb=f(b,0);
18 fh=f(h,0);
19 lx1=(b-a)*(fa+2*fh+fb)/(2*n);
20
21 fa=f(a,h1);
22 fb=f(b,h1);
23 fh=f(h,h1);
24 lx2=(b-a)*(fa+2*fh+fb)/(2*n);
25
26 fa=f(a,b1);
27 fb=f(b,b1);
28 fh=f(h,b1);
29 lx3=(b-a)*(fa+2*fh+fb)/(2*n);
30
31 l=(b1-a1)*(lx1+2*lx2+lx3)/(2*n);
32
33 avg_temp=l/(len*wid);
34 disp(avg_temp,"The average temperature is=")

```

---



# Chapter 22

## Integration of equations

Scilab code Exa 22.1 Error corrections of the trapezoidal rule

```
1 //clc()
2 h = [0.8,0.4,0.2];
3 I = [0.1728,1.0688,1.4848];
4 E = [89.5,34.9,9.5];
5 I1 = 4 * I(2) / 3 - I(1) / 3;
6 t = 1.640533;
7 et1 = t - I1 ;
8 Et1 = et1 * 100/t;
9 disp("%",Et1," Error of the improved integral for
      segment 1 and 2 = ")
10 I2 = 4 * I(3) / 3 - I(2) / 3;
11 et2 = t - I2 ;
12 Et2 = et2 * 100/t;
13 disp("%",Et2," Error of the improved integral for
      segment 4 and 2 = ")
```

---

Scilab code Exa 22.2 Higher order error correction of integral estimates

```

1 //clc ()
2 I1 = 1.367467;
3 I2 = 1.623467;
4 I = 16 * I2 /15 - I1 / 15;
5 disp(I,"Obtained integral which is the correct
   answer till the seventh decimal")

```

---

### Scilab code Exa 22.3 Two point gauss legendre formulae

```

1 //clc ()
2 //f(x) = 0.2 + 25*x - 200*x^2 + 675*x^3 - 900*x^4 +
   400*x^5
3 // for using two point gauss legendre formulae , the
   intervals have to be changed to -1 and 1
4 //therefore , x = 0.4 + 0.4 * xd
5 //thus the integral is transferred to
6 //(0.2 + 25*(0.4+0.4*x) - 200*(0.4 + 0.4*x)^2 +
   675*(0.4 + 0.4*x)^3 - 900*(0.4 + 0.4*x)^4 +
   400*(0.4 + 0.4*x)^5)*0.4
7 //for three point gauss legendre formulae
8 x1 = -(1/3) ^ 0.5;
9 x2 = (1/3) ^ 0.5;
10 I1 = (0.2 + 25*(0.4+0.4*x1) - 200*(0.4 + 0.4*x1)^2 +
   675*(0.4 + 0.4*x1)^3 - 900*(0.4 + 0.4*x1)^4 +
   400*(0.4 + 0.4*x1)^5)*0.4;
11 I2 = (0.2 + 25*(0.4+0.4*x2) - 200*(0.4 + 0.4*x2)^2 +
   675*(0.4 + 0.4*x2)^3 - 900*(0.4 + 0.4*x2)^4 +
   400*(0.4 + 0.4*x2)^5)*0.4;
12 I = I1 + I2;
13 disp(I,"Integral obtained using gauss legendre
   formulae =")
14 t = 1.640533;
15 e = (t - I)*100/t;
16 disp("%",e,"error = ")

```

---

### Scilab code Exa 22.4 Three point gauss legendre method

```
1 //clc ()
2 //f(x) = 0.2 + 25*x - 200*x^2 + 675*x^3 - 900*x^4 +
   400*x^5
3 // for using three point gauss legendre formulae ,
   the intervals have to be changed to -1 and 1
4 //therefore , x = 0.4 + 0.4 * xd
5 //thus the integral is transferred to
6 //(0.2 + 25*(0.4+0.4*x) - 200*(0.4 + 0.4*x)^2 +
   675*(0.4 + 0.4*x)^3 - 900*(0.4 + 0.4*x)^4 +
   400*(0.4 + 0.4*x)^5)*0.4
7 //for three point gauss legendre formulae
8 x1 = -0.7745967;
9 x2 = 0;
10 x3 = 0.7745967;
11 c0 = 0.5555556;
12 c1 = 0.8888889;
13 c2 = 0.5555556;
14 I1 = (0.2 + 25*(0.4+0.4*x1) - 200*(0.4 + 0.4*x1)^2 +
   675*(0.4 + 0.4*x1)^3 - 900*(0.4 + 0.4*x1)^4 +
   400*(0.4 + 0.4*x1)^5)*0.4;
15 I2 = (0.2 + 25*(0.4+0.4*x2) - 200*(0.4 + 0.4*x2)^2 +
   675*(0.4 + 0.4*x2)^3 - 900*(0.4 + 0.4*x2)^4 +
   400*(0.4 + 0.4*x2)^5)*0.4;
16 I3 = (0.2 + 25*(0.4+0.4*x3) - 200*(0.4 + 0.4*x3)^2 +
   675*(0.4 + 0.4*x3)^3 - 900*(0.4 + 0.4*x3)^4 +
   400*(0.4 + 0.4*x3)^5)*0.4;
17 I = c0 * I1 + c1 * I2 + c2 * I3;
18 disp(I,"integral obtained using three point gauss
   legendre formulae = ")
```

---

Scilab code Exa 22.5 Applying Gauss Quadrature to the falling Parachutist problem

```
1 //clc()
2 //f(t) = g*m*(int(0,10,(1-exp(-c*t/m))))/c
3 //for using gauss quadrature method, limits are
   changed to -1 to 1 by replcing x = 5 + 5*xd
4 //the new integral obtained is
5 //(1 - exp(-c*(5 + 5*x)/m ))*5
6 g = 9.8;
7 c = 12.5;
8 m = 68.1;
9 //for two point method
10 x1 = -(1/3)^0.5;
11 x2 = (1/3)^0.5;
12 I1 = g*m*(1 - exp(-c*(5 + 5*x1)/m ))*5 / c;
13 I2 = g*m*(1 - exp(-c*(5 + 5*x2)/m ))*5 / c;
14 I = I1 + I2;
15 disp(I,"integral by two point method = ")
16 x1 = -0.7745967;
17 x2 = 0;
18 x3 = 0.7745967;
19 c0 = 0.5555556;
20 c1 = 0.8888889;
21 c2 = 0.5555556;
22 I1 = g*m*(1 - exp(-c*(5 + 5*x1)/m ))*5 / c;
23 I2 = g*m*(1 - exp(-c*(5 + 5*x2)/m ))*5 / c;
24 I3 = g*m*(1 - exp(-c*(5 + 5*x3)/m ))*5 / c;
25 I = c0*I1 + c1 * I2 + c2 * I3;
26 disp(I,"integral by three point method =")
27 x1 = -0.861136312;
28 x2 = -0.339981044;
29 x3 = 0.339981044;
30 x4 = 0.861136312;
31 c1 = 0.3478548;
32 c2 = 0.6521452;
33 c3 = 0.6521452;
34 c4 = 0.3478548;
35 I1 = g*m*(1 - exp(-c*(5 + 5*x1)/m ))*5 / c;
```

```

36 I2 = g*m*(1 - exp(-c*(5 + 5*x2)/m ))*5 / c;
37 I3 = g*m*(1 - exp(-c*(5 + 5*x3)/m ))*5 / c;
38 I4 = g*m*(1 - exp(-c*(5 + 5*x4)/m ))*5 / c;
39 I = c1*I1 + c2 * I2 + c3 * I3 + c4 * I4;
40 disp(I,"integral by four point method =")
41 x1 = -0.906179846;
42 x2 = -0.538469310;
43 x3 = 0;
44 x4 = 0.538469310;
45 x5 = 0.906179846
46 c1 = 0.2369269;
47 c2 = 0.4786287;
48 c3 = 0.5688889;
49 c4 = 0.4786287;
50 c5 = 0.2369269;
51 I1 = g*m*(1 - exp(-c*(5 + 5*x1)/m ))*5 / c;
52 I2 = g*m*(1 - exp(-c*(5 + 5*x2)/m ))*5 / c;
53 I3 = g*m*(1 - exp(-c*(5 + 5*x3)/m ))*5 / c;
54 I4 = g*m*(1 - exp(-c*(5 + 5*x4)/m ))*5 / c;
55 I5 = g*m*(1 - exp(-c*(5 + 5*x5)/m ))*5 / c;
56 I = c1*I1 + c2 * I2 + c3 * I3 + c4 * I4 + c5 * I5;
57 disp(I,"integral by five point method =")

```

---

### Scilab code Exa 22.6 Evaluation of improper integral

```

1 //clc()
2 //N(x) = (int(-infinity,-2,exp(-(x^2)/2)) + int
   (-2,1,exp(-(x^2)/2)))/(2*pi)^0.5
3 //first integral can be solved as
4 //int(-infinity,-2,exp(-(x^2)/2)) = int(-0.5,0,exp
   (-1/(2*t^2))/t^2)
5 h = 1/8;
6 //int(-0.5,0,exp(-1/(2*t^2))/t^2) = h*(f(x-7/16) + f
   (x-5/16) + f(x-3/16) + f(x-1/16))
7 t1 = -7/16;

```

```

8 t2 = -5/16;
9 t3 = -3/16;
10 t4 = -1/16;
11 m1 = exp(-1/(2*t1^2))/t1^2;
12 m2 = exp(-1/(2*t2^2))/t2^2;
13 m3 = exp(-1/(2*t3^2))/t3^2;
14 m4 = exp(-1/(2*t4^2))/t4^2;
15 I1 = h*(m1 + m2 + m3 + m4);
16 disp(I1,"the value of first integral = ")
17 //simpsons 1/3rd rule is applied for the second
    integral
18 h1 = 0.5;
19 x1 = -2;
20 x2 = -1.5;
21 x3 = -1;
22 x4 = -0.5;
23 x5 = 0;
24 x6 = 0.5;
25 x7 = 1;
26 n1 = exp(-(x1^2)/2);
27 n2 = exp(-(x2^2)/2);
28 n3 = exp(-(x3^2)/2);
29 n4 = exp(-(x4^2)/2);
30 n5 = exp(-(x5^2)/2);
31 n6 = exp(-(x6^2)/2);
32 n7 = exp(-(x7^2)/2);
33 I2 =(1-(-2)) * (n1 + 4 *(n2 + n4 + n6) + 2*(n3 + n5)
    + n7)/(18);
34 disp(I2,"The value of second integral = ")
35 f = (I1 + I2)/(2 * %pi)^0.5;
36 disp(f,"Therefore the final result can be computed
    as ")
37 N = 0.8413;
38 e = (N - f) * 100 / N;
39 disp("%",e," error = ")

```

---

# Chapter 23

## Numerical differentiation

Scilab code Exa 23.1 High accuracy numerical differentiation formulas

```
1 //clc()
2 //f(x) = -0.1*x^4 - 0.15*x^3 - 0.5 * x^2 - 0.25 *x +
    1.2
3 h = 0.25;
4 t = -0.9125;
5 x = 0:h:1;
6 disp(x,"x = ")
7 fx = -0.1*x^4 - 0.15*x^3 - 0.5 * x^2 - 0.25 *x +
    1.2;
8 disp(fx,"f(x) = ")
9 fd = (- fx(5) + 4*fx(4) - 3 * fx(3))/(2 * h);
10 efd = (t - fd) * 100 / t;
11 disp(fd,"by forward difference")
12 disp("%",efd,"error in forward difference method = ")
    )
13 bd = (3 * fx(3) - 4 * fx(2) + fx(1))/ (2*h);
14 ebd = (t - bd) * 100 / t;
15 disp(bd,"by backward difference")
16 disp("%",ebd,"error in backward difference method = ")
    )
17 cdm = (-fx(5) + 8*(fx(4)) -8*fx(2) + fx(1) ) / (12*h
```

```

    );
18 ecdm = (t - cdm) * 100 / t;
19 disp(cdm,"by central difference")
20 disp("%",ecdm,"error in central difference method =
    ")

```

---

### Scilab code Exa 23.2 Richardson extrapolation

```

1 //clc()
2 //f(x) = -0.1*x^4 - 0.15*x^3 - 0.5 * x^2 - 0.25 * x +
    1.2
3 h = 0.5;
4 t = -0.9125;
5 x = 0:h:1;
6 disp(x,"x with h = 0.5 is")
7 fx = -0.1*x^4 - 0.15*x^3 - 0.5 * x^2 - 0.25 * x +
    1.2;
8 disp(fx,"f(x) with h = 0.5 is")
9 cdm = (fx(3) - fx(1))/ 1;
10 ecdm = (t - cdm) * 100 / t;
11 disp(cdm,"by central difference ( h = 0.5 ) ")
12 disp("%",ecdm,"error in central difference method (
    h = 0.5 ) = ")
13 h1 = 0.25;
14 x1 = 0:h1:1;
15 disp(x1,"x with h = 0.25 is")
16 fx1 = -0.1*x1^4 - 0.15*x1^3 - 0.5 * x1^2 - 0.25 * x1
    + 1.2;
17 disp(fx1,"fx with h = 0.25 is")
18 cdm1 = (fx1(4) - fx1(2))/ (2*h1);
19 ecdm1 = (t - cdm1) * 100 / t;
20 disp(cdm1,"by central difference ( h = 0.25 ) ")
21 disp("%",ecdm1,"error in central difference method (
    h = 0.25 ) = ")
22 D = 4 * cdm1 / 3 - cdm / 3;

```



```
23 disp(D,"improved estimate =")
```

---

### Scilab code Exa 23.3 Differentiating unequally spaced data

```
1 //clc()
2 //q(z = 0) = -k*p*C*(dT/dz)/(z = 0)
3 k = 3.5 * 10^-7; //m^2/s
4 p = 1800; //kg/m^3
5 C = 840; //(J/(kg.C))
6 x = 0;
7 fx0 = 13.5;
8 fx1 = 12;
9 fx2 = 10;
10 x0 = 0;
11 x1 = 1.25;
12 x2 = 3.75;
13 dfx = fx0 *(2*x - x1 - x2)/((x0 - x1)*(x0 - x2)) +
        fx1 *(2*x - x0 - x2)/((x1 - x0)*(x1 - x2)) + fx2
        *(2*x - x1 - x0)/((x2 - x1)*(x2 - x0));
14 disp("C/cm",dfx,"(dT/dz) = ")
15 q = - k * p * C * dfx*100;
16 disp("W/m^2",q,"q(z = 0) =")
```

---

### Scilab code Exa 23.4 Integration and Differentiation

```
1 clc;
2 clear;
3 function y=f(x)
4     y=0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5
5 endfunction
6 a=0;
7 b=0.8;
8 Q=intg(0,0.8,f);
```

```

9  disp(Q,"Q=")
10 x=[0 0.12 0.22 0.32 0.36 0.4 0.44 0.54 0.64 0.7
      0.8];
11 y=f(x);
12 integral=inttrap(x,y)
13 disp(integral,"intergral=")
14 disp(diff(x),"diff(x)=")
15 d=diff(y)./diff(x);
16 disp(d,"d=")

```

---

### Scilab code Exa 23.5 Integrate a function

```

1  clc;
2  clear;
3  function y=f(x)
4      y=0.2+25*x-200*x^2+675*x^3-900*x^4+400*x^5
5  endfunction
6  a=0;
7  b=0.8;
8  Qt=1.640533;
9  Q=intg(0,0.8,f);
10 disp(Q,"Computed=")
11 disp(abs(Q-Qt)*100/Qt,"Error estimate=")

```

---

# Chapter 25

## Runga Kutta methods

Scilab code Exa 25.1 Eulers method

```
1 //clc()
2 //dy/dx = -2*x^3 + 12*x^2 - 20*x + 8.5
3 //therefore , y = -0.5*x^4 + 4*x^3 - 10*x^2 + 8.5 + c
4 x1 = 0;
5 y1 = 1;
6 h = 0.5;
7 c = -(-0.5*x1^4 + 4*x1^3 - 10*x1^2 + 8.5*x1 - y1);
8 x = 0:0.5:4;
9 disp(x,"x = ")
10 y = -0.5*x^4 + 4*x^3 - 10*x^2 + 8.5*x + c;
11 disp(y,"true values of y = ")
12 fxy = -2*x^3 + 12*x^2 - 20*x + 8.5;
13 y2(1) = y(1);
14 e(1) = (y(1) - y2(1)) * 100 / y(1);
15 for i = 2:9
16     y2(i) = y2(i-1) + fxy(i-1)*h;
17     e(i) = (y(i) - y2(i))*100/y(i);
18 end
19 disp(y2,"y by euler method =")
20 disp(e,"error =")
```

---

**Scilab code Exa 25.2** Taylor series estimate for error by eulers method

```
1 // clc ()
2 //f(x,y) = dy/dx = -2*x^3 + 12*x^2 - 20*x + 8.5
3 //f'(x,y) = -6*x^2 + 24*x - 20
4 //f''(x,y) = -12*x + 24
5 //f'''(x,y) = -12
6 x = 0;
7 Et2 = (-6*x^2 + 24*x - 20) * 0.5^2 / 2;
8 Et3 = (-12*x + 24) * (0.5)^3 / 6;
9 Et4 = (-12) *(0.5 ^ 4) / 24;
10 Et = Et2 + Et3 + Et4;
11 disp(Et,"Total truncation error =")
```

---

**Scilab code Exa 25.3** Effect of reduced step size on Eulers method

```
1 // clc ()
2 //dy/dx = -2*x^3 + 12*x^2 - 20*x + 8.5
3 //therefore , y = -0.5*x^4 + 4*x^3 - 10*x^2 + 8.5*x + c
4 x1 = 0;
5 y1 = 1;
6 h = 0.25;
7 c = -(-0.5*x1^4 + 4*x1^3 - 10*x1^2 + 8.5*x1 - y1);
8 x = 0:h:4;
9 disp(x,"x = ")
10 y = -0.5*x^4 + 4*x^3 - 10*x^2 + 8.5*x + c;
11 disp(y,"true values of y = ")
12 fxy = -2*x^3 + 12*x^2 - 20*x + 8.5;
13 y2(1) = y(1);
14 e(1) = (y(1) - y2(1)) * 100 / y(1);
15 for i = 2:17
16     y2(i) = y2(i-1) + fxy(i-1)*h;
```

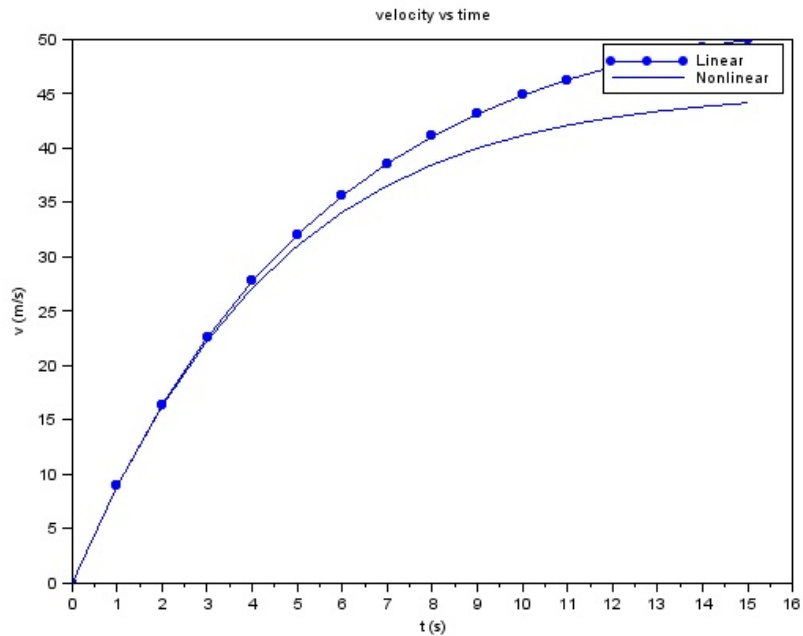


Figure 25.1: Solving ODEs

```

17     e(i) = (y(i) - y2(i))*100/y(i);
18 end
19 disp(y2,"y by euler method =")
20 disp(e,"error =")

```

---

#### Scilab code Exa 25.4 Solving ODEs

```

1  clc;
2  clear;
3  m=68.1;
4  g=9.8;

```

```

5 c=12.5;
6 a=8.3;
7 b=2.2;
8 vmax=46;
9 function yp=f(t,v)
10     yp=g-c*v/m;
11 endfunction
12 v0=0;
13 t=0:15;
14 x=ode(v0,0,t,f);
15 disp(x)
16 plot(t,x,'.-')
17
18 function yp=f1(t,v)
19     yp=g-(c/m)*(v+a*(v/vmax)^b)
20 endfunction
21 x1=ode(v0,0,t,f1);
22 plot(t,x1)
23 xtitle("velocity vs time","t (s)","v (m/s)")
24 h1=legend(["Linear";"Nonlinear"])

```

---

#### Scilab code Exa 25.5 Heuns method

```

1 //clc()
2 //y' = 4*exp(0.8*x) - 0.5*y
3 //y = 4*(exp(0.8*x) - exp(-0.5*x))/1.3 + 2*exp(-0.5*
4     x)
5 x = 0:1:4;
6 disp(x)
7 x1 = 0;
8 y1 = 2;
9 y2(1) = y1;
10 for i = 1:5
11     y(i) = 4*(exp(0.8*x(i)) - exp(-0.5*x(i)))/1.3 +
12         2*exp(-0.5*x(i));

```

```

11     dy(i) = 4*exp(0.8*x(i)) - 0.5*y2(i);
12     y2(i + 1) = y2(i) + dy(i);
13     if i>1 then
14         m(i) = (dy(i) + dy(i-1))/2;
15         y2(i) = y2(i-1) + m(i);
16         dy(i) = 4*exp(0.8*x(i)) - 0.5*y2(i);
17     end
18     e(i) = (y(i) - y2(i)) * 100 / y(i);
19 end
20 disp(y2(1:5),"By heuns method(1 iteration)")
21 disp("%",e(1:5),"error = ")

```

---

#### Scilab code Exa 25.6 Comparison of various second order RK 4 method

```

1 //clc()
2 //f'(x,y) = -2*x^3 + 12*x^2 -20*x + 8.5
3 //f(x,y) = -x^4 / 2 + 4*x^3 - 10*x^2 + 8.5*x + 1
4 h = 0.5;
5 x = 0:h:4;
6 y1 = -x^4 / 2 + 4*x^3 - 10*x^2 + 8.5*x + 1;
7 y(1) = 1;
8 disp(x,"x =")
9 disp(y1,"true value of y =")
10 for i = 1:8
11     k1(i) = -2*x(i)^3 + 12*x(i)^2 -20*x(i) + 8.5;
12     x1(i) = x(i) + h/2;
13     k2(i) = -2*x1(i)^3 + 12*x1(i)^2 -20*x1(i) + 8.5;
14     y(i+1) = y(i) + k2(i)*h;
15     e(i) = (y1(i) - y(i))*100/y1(i);
16 end
17 disp(y(1:9),"y by midpoint method")
18 disp(e,"error = ")
19 for i = 1:8
20     k1(i) = -2*x(i)^3 + 12*x(i)^2 -20*x(i) + 8.5;
21     x(i) = x(i) + 3*h/4;

```

```

22     k2(i) = -2*x(i)^3 + 12*x(i)^2 -20*x(i) + 8.5;
23     y(i+1) = y(i) + (k1(i)/3 + 2*k2(i)/3)*h;
24     e(i) = (y1(i) - y(i))*100/y1(i);
25 end
26 disp(y(1:9),"y by second order Ralston RK")
27 disp(e,"error = ")

```

---

### Scilab code Exa 25.7 Classical fourth order RK method

```

1 //clc()
2 //f'(x,y) = -2*x^3 + 12*x^2 -20*x + 8.5
3 //f(x,y) = -x^4 / 2 + 4*x^3 - 10*x^2 + 8.5*x + 1
4 h = 0.5;
5 x = 0:h:4;
6 y1 = -x^4 / 2 + 4*x^3 - 10*x^2 + 8.5*x + 1;
7 y(1) = 1;
8 for i=1:8
9     k1(i) = -2*x(i)^3 + 12*x(i)^2 -20*x(i) + 8.5;
10    x1(i) = x(i) + h/2;
11    k2(i) = -2*x1(i)^3 + 12*x1(i)^2 -20*x1(i) + 8.5;
12    k3(i) = -2*x1(i)^3 + 12*x1(i)^2 -20*x1(i) + 8.5;
13    x2(i) = x(i) + h;
14    k4(i) = -2*x2(i)^3 + 12*x2(i)^2 -20*x2(i) + 8.5;
15    y(i+1) = y(i) + (k1(i)+2*k2(i)+2*k3(i)+k4(i))*h
        /6;
16    e(i) = (y1(i) - y(i))*100/y1(i);
17 end
18 disp("f(x,y) = -2*x^3 + 12*x^2 -20*x + 8.5")
19 disp(y(1:9),"y by fourth order Ralston RK")
20 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
21 x = 0:h:0.5;
22 y(1) = 2;
23 k1 = 4*(exp(0.8*x(1))) -0.5*y(1);
24 x1 = x(1) + 0.5*h;
25 y1 = y(1) + 0.5*k1*h;

```



```

26 k2 = 4*exp(0.8*x1) - 0.5*y1;
27 y2 = y(1) + 0.5*k2*h;
28 k3 = 4*exp(0.8*x1) - 0.5*y2;
29 x1 = x(1) + h;
30 y1 = y(1) + k3*h;
31 k4 = 4*exp(0.8*x1) - 0.5*y1;
32 y(2) = y(1) + (k1+2*k2+2*k3+k4)*h/6;
33 disp(y(1:2), "y = ")

```

---

### Scilab code Exa 25.8 Comparison of Runge Kutta methods

```

1 //clc()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 h = 1;
4 x = 0:h:4;
5 y(1) = 2;
6 for i = 1:5
7     k1(i) = 4*(exp(0.8*x(i)))-0.5*y(i);
8     x1 = x(i) + h;
9     y1 = y(i) + k1(i)*h;
10    k2(i) = 4*(exp(0.8*x1))-0.5*y1;
11    y(i+1) = y(i) + (k1(i)/2 + k2(i)/2)*h;
12 end
13 disp(y(1:5), "y(second order RK method) = ")
14 for i = 1:5
15    k1(i) = 4*(exp(0.8*x(i)))-0.5*y(i);
16    x1 = x(i) + 0.5*h;
17    y1 = y(i) + 0.5*h*k1(i);
18    k2(i) = 4*(exp(0.8*x1))-0.5*y1;
19    x1 = x(i) + h;
20    y1 = y(i) -k1(i)*h + 2*k2(i)*h;
21    k3(i) = 4*(exp(0.8*x1))-0.5*y1;
22    y(i+1) = y(i) + (k1(i) + 4*k2(i) + k3(i))*h/6;
23 end
24 disp(y(1:5), "y(third order RK method) = ")

```

```

25 for i = 1:5
26 k1(i) = 4*(exp(0.8*x(i)))-0.5*y(i);
27 x1 = x(i) + 0.5*h;
28 y1 = y(i) + 0.5*k1(i)*h;
29 k2(i) = 4*exp(0.8*x1) - 0.5*y1;
30 y2 = y(i) + 0.5*k2(i)*h;
31 k3(i) = 4*exp(0.8*x1) - 0.5*y2;
32 x1 = x(i) + h;
33 y1 = y(i) + k3(i)*h;
34 k4(i) = 4*exp(0.8*x1) - 0.5*y1;
35 y(i+1) = y(i) + (k1(i)+2*k2(i)+2*k3(i)+k4(i))*h/6;
36 end
37 disp(y(1:5), "y(fourth order RK method) = ")
38 for i = 1:5
39
40 k1(i) = 4*(exp(0.8*x(i)))-0.5*y(i);
41 x1 = x(i) + 0.25*h;
42 y1 = y(i) + 0.25*k1(i)*h;
43 k2(i) = 4*exp(0.8*x1) - 0.5*y1;
44 y2 = y(i) + 0.125*k2(i)*h + 0.125*k1(i)*h;
45 k3(i) = 4*exp(0.8*x1) - 0.5*y2;
46 x1 = x(i) + 0.5*h;
47 y1 = y(i) -0.5*k2(i)*h + k3(i)*h;
48 k4(i) = 4*exp(0.8*x1) - 0.5*y1;
49 x1 = x(i) + 0.75*h;
50 y1 = y(i) + 3*k1(i)*h/16 + 9*k4(i)*h/16;
51 k5(i) = 4*exp(0.8*x1) - 0.5*y1;
52 x1 = x(i) + h;
53 y1 = y(i) - 3*k1(i)*h/7 + 2*k2(i)*h/7 + 12*k3(i)*h/7
      - 12*k4(i)*h/7 + 8*k5(i)*h/7;
54 k6(i) = 4*exp(0.8*x1) - 0.5*y1;
55 y(i+1) = y(i) + (7*k1(i)+32*k3(i)+12*k4(i) + 32*k5(i)
      ) + 7*k6(i))*h/90;
56 end
57 disp(y(1:5), "y(fifth order RK method)")

```

---

**Scilab code Exa 25.9** Solving systems of ODE using Eulers method

```
1 // clc ()
2 //dy1/dx = -0.5*y1
3 //dy2/dx = 4 - 0.3*y2 - 0.1*y1
4 x1 = 0;
5 h =0.5;
6 y1(1) = 4;
7 y2(1) = 6;
8 x = 0:h:2;
9 for i = 2:5
10     y1(i) = y1(i-1) - 0.5*y1(i-1)*h;
11     y2(i) = y2(i-1) + (4 - 0.3*y2(i-1) - 0.1*y1(i-1)
12         ) *h;
12 end
13 disp(x,"x = ")
14 disp(y1,"y1 = ")
15 disp(y2,"y2 = ")
```

---

**Scilab code Exa 25.10** Solving systems of ODE using RK 4 method

```
1 // clc ()
2 //dy1/dx = -0.5*y1
3 //dy2/dx = 4 - 0.3*y2 - 0.1*y1
4 x1 = 0;
5 h =0.5;
6 y1(1) = 4;
7 y2(1) = 6;
8 x = 0:h:2;
9 for i = 1:5
10     k11(i) = -0.5*y1(i);
11     k12(i) = 4 - 0.3*y2(i) - 0.1*y1(i);
```

```

12     y11 = y1(i) + k11(i) * h/2;
13     y21 = y2(i) + k12(i) * h/2;
14     k21(i) = -0.5*y11;
15     k22(i) = 4 - 0.3*y21 - 0.1*y11;
16     y11 = y1(i) + k21(i) * h/2;
17     y21 = y2(i) + k22(i) * h/2;
18     k31(i) = -0.5*y11;
19     k32(i) = 4 - 0.3*y21 - 0.1*y11;
20     y11 = y1(i) + k31(i) * h;
21     y21 = y2(i) + k32(i) * h;
22     k41(i) = -0.5*y11;
23     k42(i) = 4 - 0.3*y21 - 0.1*y11;
24     y1(i+1) = y1(i) + ( k11(i) + 2*k21(i) + 2*k31(i)
        + k41(i) ) * h / 6;
25     y2(i+1) = y2(i) + ( k12(i) + 2*k22(i) + 2*k32(i)
        + k42(i) ) * h / 6;
26 end
27 disp("using fourth order RK method")
28 disp(x,"x =")
29 disp(y1(1:5),"y1 = ")
30 disp(y2(1:5),"y2 = ")

```

---

### Scilab code Exa 25.11 Solving systems of ODEs

```

1  clc;
2  clear;
3  function yp=f(x,y)
4      yp=[y(2); -16.1*y(1)];
5  endfunction
6  x=0:0.1:4
7  y0=[0.1 0];
8  sol=ode(y0,0,x,f);
9  count=1;
10 disp(sol)
11 for i=1:2:81

```

```

12     a(count)=sol(i);
13     b(count)=sol(i+1);
14     count=count+1;
15 end
16 plot(x,a)
17 plot(x,b,".-")
18 h1=legend(["y1,y3","y2,y4"])
19 xtitle("y vs x","x","y")
20 function yp=g(x,y)
21     yp=[y(2);-16.1*sin(y(1))];
22 endfunction
23 sol=ode(y0,0,x,g);
24 count=1;
25 disp(sol)
26 for i=1:2:81
27     a(count)=sol(i);
28     b(count)=sol(i+1);
29     count=count+1;
30 end
31 plot(x,a)
32 plot(x,b,".-")
33 clf();
34 y0=[%pi/4 0];
35 sol=ode(y0,0,x,f);
36 count=1;
37 disp(sol)
38 for i=1:2:81
39     a(count)=sol(i);
40     b(count)=sol(i+1);
41     count=count+1;
42 end
43 plot(x,a)
44 plot(x,b,".-")
45
46 xtitle("y vs x","x","y")
47 sol=ode(y0,0,x,g);
48 count=1;
49 disp(sol)

```

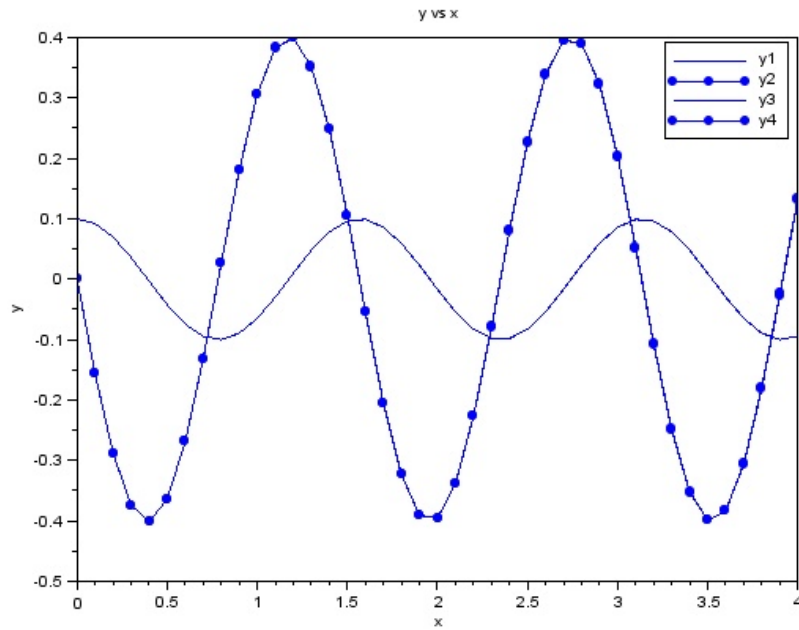


Figure 25.2: Solving systems of ODEs

```

50 for i=1:2:81
51     a(count)=sol(i);
52     b(count)=sol(i+1);
53     count=count+1;
54 end
55 plot(x,a,'o-')
56 plot(x,b,'x-')
57 h1=legend(["y1" ,"y2" ,"y3" ,"y4"])

```

---

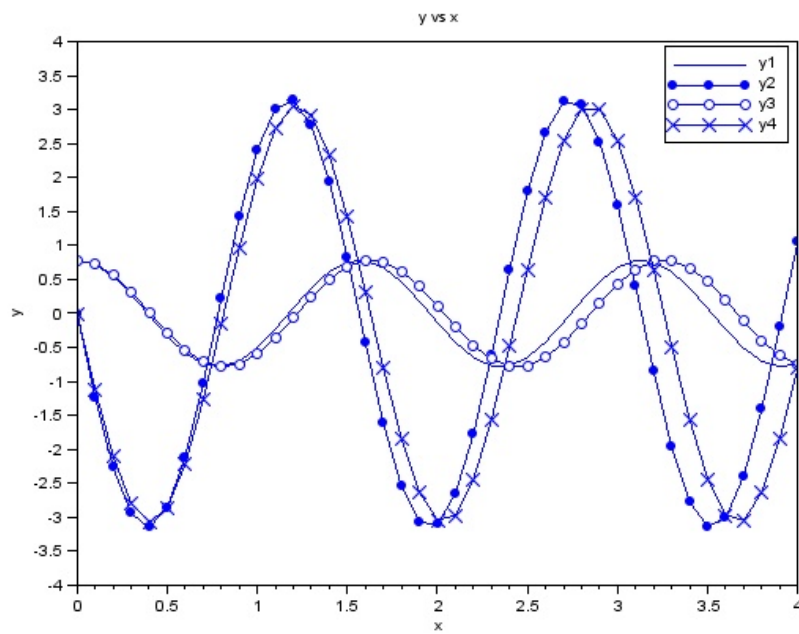


Figure 25.3: Solving systems of ODEs

### Scilab code Exa 25.12 Adaptive fourth order RK method

```
1 //clc()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 //f'(x,y) = 4*exp(0.8*x) - 0.5*y
4 x = 0:2:2;
5 y(1) = 2;
6 h =2;
7 t = 14.84392;
8 k1 = 4*exp(0.8*x(1)) - 0.5*y(1);
9 x1 = x(1) + h/2;
10 y1 = y(1) + k1*h/2;
11 k2 = 4*exp(0.8*x1) - 0.5*y1;
12 x1 = x(1) + h/2;
13 y1 = y(1) + k2*h/2;
14 k3 = 4*exp(0.8*x1) - 0.5*y1;
15 x1 = x(1) + h;
16 y1 = y(1) + k3*h;
17 k4 = 4*exp(0.8*x1) - 0.5*y1;
18 y(2) = y(1) + (k1 + 2*k2 + 2*k3 + k4)*h/6;
19 e = (t - y(2))/(t);
20 disp(y(1:2),"y by h = 2 is")
21 disp(e,"error = ")
22 h = 1;
23 x = 0:h:2;
24 for i=1:3
25     k1(i) = 4*exp(0.8*x(i)) - 0.5*y(i);
26     x1 = x(i) + h/2;
27     y1 = y(i) + k1(i)*h/2;
28     k2(i) = 4*exp(0.8*x1) - 0.5*y1;
29     x1 = x(i) + h/2;
30     y1 = y(i) + k2(i)*h/2;
31     k3(i) = 4*exp(0.8*x1) - 0.5*y1;
32     x1 = x(i) + h;
33     y1 = y(i) + k3(i)*h;
34     k4(i) = 4*exp(0.8*x1) - 0.5*y1;
35     y(i+1) = y(i) + (k1(i) + 2*k2(i) + 2*k3(i) + k4(
        i))*h/6;
```



```

36 end
37 e = (t - (y(3)))/t;
38 disp(y(1:3),"y by h = 1 is")
39 disp(e,"error = ")

```

---

### Scilab code Exa 25.13 Runge kutta Fehlberg method

```

1 //clc()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 //f'(x,y) = 4*exp(0.8*x) - 0.5*y
4 h = 2;
5 x = 0:h:2;
6 y(1) = 2;
7 t = 14.84392;
8 k1 = 4*exp(0.8*x(1)) - 0.5*y(1);
9 x1 = x(1) + h/5;
10 y1 = y(1) + k1*h/5;
11 k2 = 4*exp(0.8*x1) - 0.5*y1;
12 x1 = x(1) + 3*h/10;
13 y1 = y(1) + 3*k1*h/40 + 9*k2*h/40;
14 k3 = 4*exp(0.8*x1) - 0.5*y1;
15 x1 = x(1) + 3*h/5;
16 y1 = y(1) + 3*k1*h/10 - 9*k2*h/10 + 6*k3*h/5;
17 k4 = 4*exp(0.8*x1) - 0.5*y1;
18 x1 = x(1) + h;
19 y1 = y(1) - 11*k1*h/54 + 5*k2*h/2 - 70*k3*h/27 + 35*
    k4*h/27;
20 k5 = 4*exp(0.8*x1) - 0.5*y1;
21 x1 = x(1) + 7*h/8;
22 y1 = y(1) + 1631*k1*h/55296 + 175*k2*h/512 + 575*k3*
    h/13824 + 44275*k4*h/110592 + 253*k5*h/4096;
23 k6 = 4*exp(0.8*x1) - 0.5*y1;
24 y1 = y(1) + (37*k1/378 + 250*k3/621 + 125*k4/594 +
    512*k6/1771)*h;
25 y2 = y(1) + (2825*k1/27648 + 18575*k3/48384 + 13525*

```

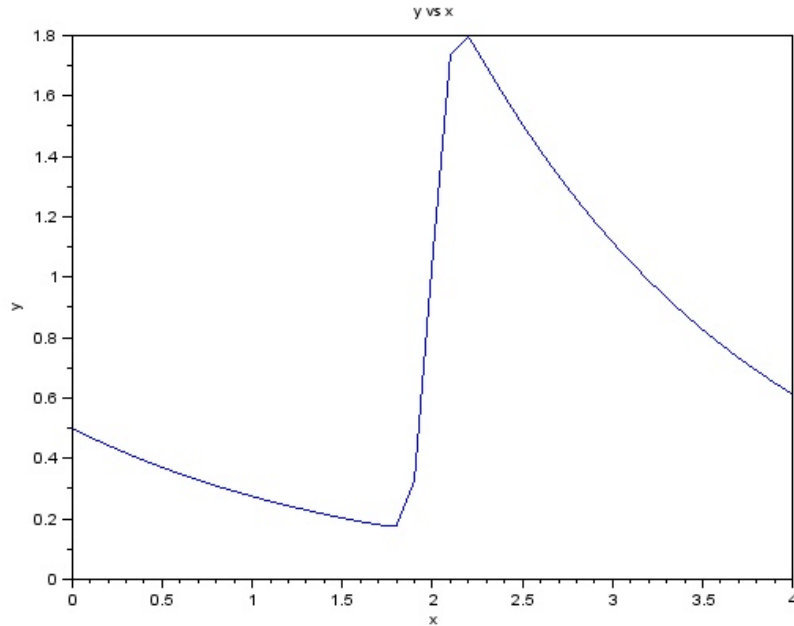


Figure 25.4: Adaptive Fourth order RK scheme

```

k4/55296 + 277*k5/14336 + k6/4)*h;
26 disp(y1,"y ( fourth order prediction ) = ")
27 disp(y2,"y ( fifth order prediction ) = ")

```

---

#### Scilab code Exa 25.14 Adaptive Fourth order RK scheme

```

1 clc;
2 clear;
3 function yp=f(x,y)
4     yp=10*exp(-(x-2)^2/(2*(0.075^2)))-0.6*y
5 endfunction

```

```
6 x=0:0.1:4
7 y0=0.5;
8 sol=ode(y0,0,x,f);
9 plot(x,sol)
10 xtitle("y vs x","x","y")
```

---

# Chapter 26

## Stiffness and multistep methods

Scilab code Exa 26.1 Explicit and Implicit Euler

```
1  clc;
2  clear;
3  function yp=f(t,y)
4      yp=-1000*y+3000-2000*exp(-t)
5  endfunction
6  y0=0;
7  //explicit euler
8  h1=0.0005;
9  y1(1)=y0;
10 count=2;
11 t=0:0.0001:0.006
12 for i=0:0.0001:0.0059
13     y1(count)=y1(count-1)+f(i,y1(count-1))*h1
14     count=count+1;
15 end
16 plot(t,y1)
17 h2=0.0015;
18 y2(1)=y0;
19 count=2;
20 t=0:0.0001:0.006
21 for i=0:0.0001:0.0059
```

```

22     y2(count)=y2(count-1)+f(i,y2(count-1))*h2
23     count=count+1;
24 end
25 plot(t,y2,'.-')
26 xtitle("y vs t","t","y")
27 h=legend(["h=0.0005","h=0.0015"])
28 clf();
29 //implicit order
30 h3=0.05;
31 y3(1)=y0;
32 count=2;
33 t=0:0.01:0.4
34 for j=0:0.01:0.39
35     y3(count)=(y3(count-1)+3000*h3-2000*h3*exp(-(j
        +0.01)))/(1+1000*h3)
36     count=count+1;
37 end
38
39 plot(t,y3)
40 xtitle("y vs t","t","y")

```

---

### Scilab code Exa 26.2 Non self starting Heun method

```

1 //clc()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 //f'(x,y) = 4*exp(0.8*x) - 0.5*y
4 h = 1;
5 x=0:h:4;
6 y(1) = 2;
7 x1 = -1;
8 y1 = -0.3929953;

```

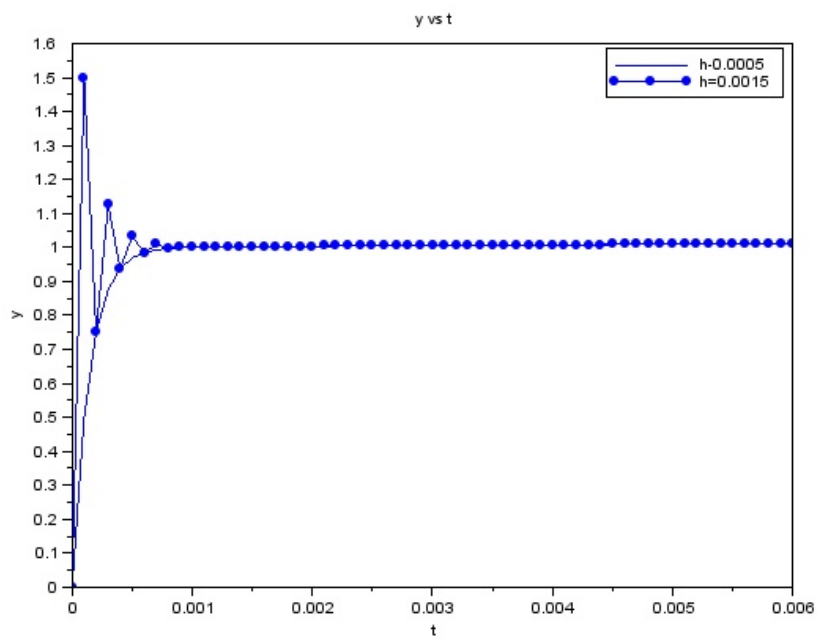


Figure 26.1: Explicit and Implicit Euler

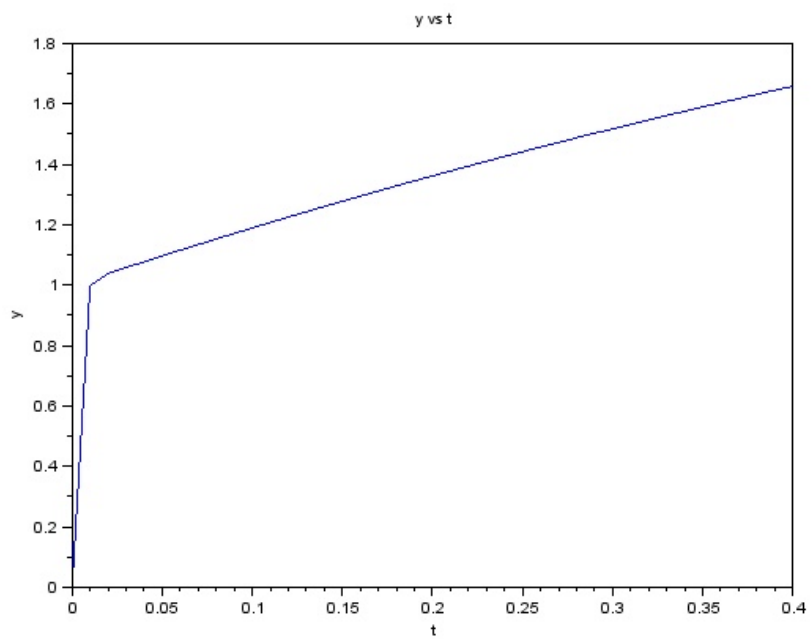


Figure 26.2: Explicit and Implicit Euler

```

9 y10 = y1 + (4*exp(0.8*x(1)) - 0.5*y(1))*2;
10 y11 = y(1) + (4*exp(0.8*x(1)) - 0.5*y(1) + 4*exp
    (0.8*x(2)) - 0.5*y10)*h/2;
11 y12 = y(1) + (3 + 4*exp(0.8*x(2)) - 0.5*y11)*h/2;
12 t = 6.360865;
13 y20 = y(1) + (4*exp(0.8*x(2)) - 0.5*t) *2;
14 y21 = t + (4*exp(0.8*x(2)) - 0.5*t + 4*exp(0.8*x(3))
    - 0.5*y20)*h/2;
15 disp(y21,"the first corrector yields y = ")
16 t = 14.84392
17 e = (t - y21)*100/t;
18 disp("%",e," error = ")

```

---

### Scilab code Exa 26.3 Estimate of per step truncation error

```

1 //clc()
2 x1 = 1;
3 x2 = 2;
4 y1 = 6.194631;
5 y2 = 14.84392;
6 y10 = 5.607005;
7 y11 = 6.360865;
8 y20 = 13.44346;
9 y21 = 15.30224;
10 Ec1 = -(y11 - y10)/5;
11 disp(Ec1,"Ec (x = 1) = ")
12 e1 = y1 - y11;
13 disp(e1,"true error (x = 1) = ")
14 Ec2 = -(y21 - y20)/5;
15 disp(Ec2,"Ec (x = 2) = ")
16 e2 = y2 - y21;
17 disp(e2,"true error (x = 2) = ")

```

---



### Scilab code Exa 26.4 Effect of modifier on Predictor Corrector results

```
1 //clc ()
2 //clc ()
3 x0 = 0;
4 x1 = 1;
5 x2 = 2;
6 y1 = 6.194631;
7 y2 = 14.84392;
8 y10 = 5.607005;
9 y11 = 6.360865;
10 y1m = y11 - (y11 - y10)/5;
11 e = (y1 - y1m)*100/y1;
12 disp(y1m,"ym")
13 disp("%",e,"error = ")
14 y20 =2+(4*exp(0.8*x1) - 0.5*y1m)*2;
15 e2 = (y2 - y20)*100/y2;
16 disp(y20,"y20 = ")
17 disp("%",e2,"error = ")
18 y2o = y20 + 4* (y11 - y10)/5;
19 e2 = (y2 - y2o)*100/y2;
20 disp(y2o,"y20 = ")
21 disp("%",e2,"error = ")
22 y21 = 15.21178;
23 y23 = y21 - (y21 - y20)/5;
24 disp(y23,"y2 = ")
25 e3 = (y2 - y23)*100/y2;
26 disp("%",e3,"error = ")
```

---

### Scilab code Exa 26.5 Milnes Method

```
1 //clc ()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 //f'(x,y) = 4*exp(0.8*x) - 0.5*y
4 h = 1;
```

```

5 x = -3:h:4;
6 y(1) = -4.547302;
7 y(2) = -2.306160;
8 y(3) = -0.3929953;
9 y(4) = 2;
10 y1(4) = 2;
11 for i = 4:7;
12     y(i+1) = y(i-3) + 4*h*(2*(4*exp(0.8*x(i)) - 0.5*
        y(i)) - 4*exp(0.8*x(i-1)) + 0.5*y(i-1) +
        2*(4*exp(0.8*x(i-2)) - 0.5*y(i-2)))/3;
13     y1(i+1) = y(i-1) + h*(4*exp(0.8*x(i-1)) - 0.5*y(i-1)
        +4 * (4*exp(0.8*x(i)) - 0.5*y(i)) + 4*
        exp(0.8*x(i+1)) - 0.5*y(i+1))/3;
14 end
15 disp(x(4:8),"x = ")
16 disp(y(4:8),"y0 = ")
17 disp(y1(4:8),"corrected y1 = ")

```

---

#### Scilab code Exa 26.6 Fourth order Adams method

```

1 //clc()
2 disp("f(x,y) = 4*exp(0.8*x) - 0.5*y")
3 //f'(x,y) = 4*exp(0.8*x) - 0.5*y
4 h = 1;
5 x = -3:h:4;
6 y(1) = -4.547302;
7 y(2) = -2.306160;
8 y(3) = -0.3929953;
9 y(4) = 2;
10 m(4) = y(4);
11 for i =4:7
12     y(i+1) = y(i) + h *(55* (4*exp(0.8*x(i)) - 0.5*y
        (i)) / 24 - 59 * (4*exp(0.8*x(i-1)) - 0.5*y(i-1)) / 24 + 37*(4*exp(0.8*x(i-2)) - 0.5*y(i-2))/24 - 9*(4*exp(0.8*x(i-3)) - 0.5*y(i-3))

```

```

        /24);
13     m(i+1) = y(i+1)
14     y(i+1) = y(i) + h*(9*(4*exp(0.8*x(i+1)) - 0.5*y(
        i+1))/24 +19*(4*exp(0.8*x(i)) - 0.5*y(i))/24
        - 5*(4*exp(0.8*x(i-1)) - 0.5*y(i-1))/24 + (4*
        exp(0.8*x(i-2)) - 0.5*y(i-2))/24);
15 end
16 disp(x(4:8),"x = ")
17 disp(m(4:8),"y0 = ")
18 disp(y(4:8),"y1 = ")

```

---

Scilab code Exa 26.7 stability of Milnes and Fourth order Adams method

```

1 // clc()
2 //dy/dx = -y
3 //y = exp(-x)
4 h = 0.5;
5 x = -1.5:h:10;
6 y(1) = exp(-x(1));
7 y(2) = exp(-x(2));
8 y(3) = exp(-x(3));
9 y(4) = 1;
10 m(4) = y(4);
11 for i = 4:23;
12     y(i+1) = y(i-3) + 4*h*(2*(-y(i)) + y(i-1) + 2*(-
        y(i-2)))/3;
13     m(i+1) = y(i+1);
14     y(i+1) = y(i-1) + h*(-y(i-1) +4 * (-y(i)) + (-y(
        i+1)))/3;
15 end
16 disp(x(4:24),"x = ")
17 disp(m(4:24),"y0(milnes method) = ")
18 disp(y(4:24),"corrected y1(milnes method) = ")
19 for i =4:23
20     y(i+1) = y(i) + h *(55* (-y(i)) / 24 - 59 * (-y(

```

```
        i-1)) / 24 + 37*(-y(i-2))/24 - 9*(-y(i-3))
        /24);
21     m(i+1) = y(i+1)
22     y(i+1) = y(i) + h*(9*(-y(i+1))/24 +19*(-y(i))/24
        - 5*(-y(i-1))/24 + (-y(i-2))/24);
23 end
24 disp(m(4:23),"y0(fourth order adams method) = ")
25 disp(y(4:23),"y1(fourth order adams method) = ")
```

---

# Chapter 27

## Boundary Value and Eigen Value problems

Scilab code Exa 27.1 The shooting method

```
1 //clc ()
2 l = 10; //m
3 h = 0.01; //m-2
4 Ta = 20;
5 T0 = 40;
6 T10 = 200;
7 //dT/dx = z
8 //dz/dx = h'(T-Ta)
9 // we use z = 10 as initial guess and integrating
   above equations simultaneously, solving using RK4
   method, we get T10 = 168.3797
10 // similarly, with z = 20, T10 = 285.898
11 z01 = 10;
12 z02 = 20;
13 T10a = 168.3797;
14 T10b = 285.898;
15 z0 = z01 + (z02 - z01)*(T10 - T10a)/(T10b - T10a);
16 disp(z0,"z0 = ")
17 disp("this value of z can be used to get the correct
```

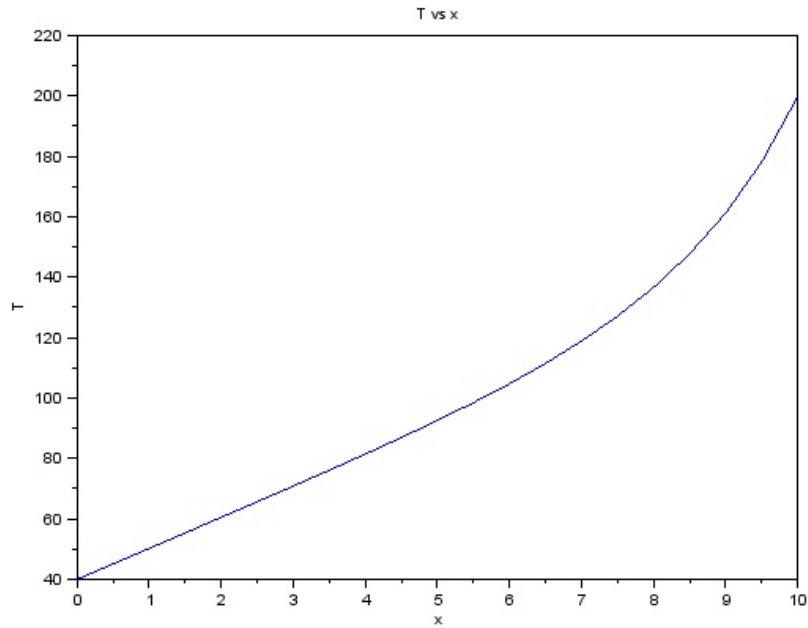


Figure 27.1: The shooting method for non linear problems

value”)

---

**Scilab code Exa 27.2** The shooting method for non linear problems

```

1 //clc()
2 z(1) = 10.0035;
3 T(1) = 40;
4 Ta = 20;
5 h = 0.5;
6 for i = 1:20
7     k11(i) = z(i);

```

```

8     k12(i) = 5*10^-8*(T(i) - Ta)^4;
9     z1 = z(i) + h/2;
10    T1 = T(i) + h/2;
11    k21(i) = z1;
12    k22(i) = 5*10^-8*(T1 - Ta)^4;
13    z1 = z(i) + h/2;
14    T1 = T(i) + h/2;
15    k31(i) = z1;
16    k32(i) = 5*10^-8*(T1 - Ta)^4;
17    z1 = z(i) + h;
18    T1 = T(i) + h;
19    k41(i) = z1;
20    k42(i) = 5*10^-8*(T1 - Ta)^4;
21    T(i+1) = T(i) + ( k11(i) + 2* k21(i) + 2*k31(i)
      + k41(i))*h/6;
22    z(i+1) = z(i) + ( k12(i) + 2* k22(i) + 2*k32(i)
      + k42(i))*h/6;
23 end
24 x=0:0.5:10;
25 plot(x,T(1:21))
26 xtitle("T vs x", "x", "T")

```

---

### Scilab code Exa 27.3 Finite Difference Approximation

```

1  clc;
2  clear;
3  h=0.01;
4  delx=2;
5  x=2+h*delx^2;
6  a=[x -1 0 0;-1 x -1 0;0 -1 x -1;0 0 -1 x];
7  b=[40.8; 0.8; 0.8; 200.8];
8  T=linsolve(a,b);
9  disp("The temperature at the interior nodes:")
10 disp(abs(T))

```

---

### Scilab code Exa 27.4 Mass Spring System

```
1  clc;
2  clear;
3  m1=40; //kg
4  m2=40; //kg
5  k=200; //N/m
6  sqw=poly(0,"s");
7  p=sqw^2-20*sqw+75;
8  r=roots(p);
9  f1=(r(1))^0.5;
10 f2=(r(2))^0.5;
11 Tp1=(2*%pi)/f1;
12 Tp2=(2*%pi)/f2;
13 //for first mode
14 disp("For first mode:")
15 disp(Tp1,"Period of oscillation:")
16 disp("A1=-A2")
17 disp("
    ")
18 //for first mode
19 disp("For second mode:")
20 disp(Tp2,"Period of oscillation:")
21 disp("A1=A2")
```

---

### Scilab code Exa 27.5 Axially Loaded column

```
1  clc;
2  clear;
3  E=10*10^9; //Pa
4  I=1.25*10^-5; //m^4
```



```

5 L=3; //m
6 for i=1:8
7     p=i*%pi/L;
8     P=i^2*(%pi)^2*E*I/(L^2*1000);
9     disp(i,"n=")
10    disp("m^-2",p,"p=")
11    disp("kN",P,"P=")
12    disp("
        _____
            ")
13 end
        _____

```

#### Scilab code Exa 27.6 Polynomial Method

```

1  clc;
2  clear;
3  E=10*10^9; //Pa
4  I=1.25*10^-5; //m^4
5  L=3; //m
6  true=[1.0472 2.0944 3.1416 4.1888];
7  //part a
8  h1=3/2;
9  p=poly(0,"s")
10 a=2-h1^2*p^2;
11 x=roots(a);
12 e=abs(abs(x(1))-true(1))*100/true(1);
13 disp(x,"p=")
14 disp(e,"error=")
15 disp("
        _____
            ")
16 //part b
17 h2=3/3;
18 p=poly(0,"s")
19 a=(2-h2^2*p^2)^2 - 1;

```

```

20 x=roots(a);
21 e(1)=abs(abs(x(1))-true(2))*100/true(2);
22 e(2)=abs(abs(x(2))-true(1))*100/true(1);
23 disp(x,"p=")
24 disp(e,"error=")
25 disp("

```

---

```

    ")

```

```

26 //part c
27 h3=3/4;
28 p=poly(0,"s")
29 a=(2-h3^2*p^2)^3 - 2*(2-h3^2*p^2);
30 x=roots(a);
31 e(1)=abs(abs(x(1))-true(3))*100/true(3);
32 e(2)=abs(abs(x(2))-true(2))*100/true(2);
33 e(3)=abs(abs(x(3))-true(1))*100/true(1);
34 disp(x,"p=")
35 disp(e,"error=")
36 disp("

```

---

```

    ")

```

```

37
38
39 //part d
40 h4=3/5;
41 p=poly(0,"s")
42 a=(2-h4^2*p^2)^4 - 3*(2-h4^2*p^2)^2 + 1;
43 x=roots(a);
44 e(1)=abs(abs(x(1))-true(4))*100/true(4);
45 e(2)=abs(abs(x(2))-true(3))*100/true(3);
46 e(3)=abs(abs(x(3))-true(2))*100/true(2);
47 e(4)=abs(abs(x(4))-true(1))*100/true(1);
48 disp(x,"p=")
49 disp(e,"error=")
50 disp("

```

---

```

    ")

```

---

### Scilab code Exa 27.7 Power Method Highest Eigenvalue

```
1  clc;
2  clear;
3  a=[3.556 -1.668 0;-1.778 3.556 -1.778;0 -1.778
      3.556];
4  b=[1.778;0;1.778];
5  ea=100;
6  count=1;
7  while ea>0.1
8      maxim=b(1);
9      for i=2:3
10         if abs(b(i))>abs(maxim) then
11             maxim=b(i);
12         end
13     end
14     eigen(count)=maxim;
15     b=a*(b./maxim);
16     if count==1 then
17         ea=20;
18         count=count+1;
19     else
20         ea=abs(eigen(count)-eigen(count-1))*100/abs(
21             eigen(count));
22         count=count+1;
23     end
24 end
25 disp(eigen(count-1),"The largest eigen value")
```

---

### Scilab code Exa 27.8 Power Method Lowest Eigenvalue

```

1  clc;
2  clear;
3  a=[3.556 -1.668 0;-1.778 3.556 -1.778;0 -1.778
      3.556];
4  b=[1.778;0;1.778];
5  ea=100;
6  count=1;
7  ai=inv(a);
8  while ea>4
9      maxim=b(1);
10     for i=2:3
11         if abs(b(i))>abs(maxim) then
12             maxim=b(i);
13         end
14     end
15     eigen(count)=maxim;
16     b=ai*(b./maxim);
17     if count==1 then
18         ea=20;
19         count=count+1;
20
21     else
22         ea=abs(eigen(count)-eigen(count-1))*100/abs(
                eigen(count));
23         count=count+1;
24     end
25 end
26 disp((1/eigen(count-1))^0.5,"The smallest eigen
      value")

```

---

### Scilab code Exa 27.9 Eigenvalues and ODEs

```

1  clc;
2  clear;
3  function yp=predprey(t,y)

```

```

4      yp=[1.2*y(1)-0.6*y(1)*y(2);-0.8*y(2)+0.3*y(1)*y
        (2)];
5  endfunction
6  t=0:0.1:20;
7  y0=[2 1];
8  sol=ode(y0,0,t,predprey);
9  count=1;
10 for i=1:2:401
11     x(count)=sol(i);
12     z(count)=sol(i+1);
13     count=count+1;
14 end
15
16 plot(t,x)
17 plot(t,z)
18 xtitle("y vs t", "t","y")
19 clf();
20 plot(x,z)
21 xtitle("space-space plot (y1 vs y2)", "y1","y2")

```

---

#### Scilab code Exa 27.10.a Stiff ODEs

```

1  clc;
2  clear;
3  function yp=vanderpol(t,y)
4      yp=[y(2);1*(1-y(1)^2)*y(2)-y(1)]
5  endfunction
6  y0=[1 1];
7  t=0:0.1:20;

```

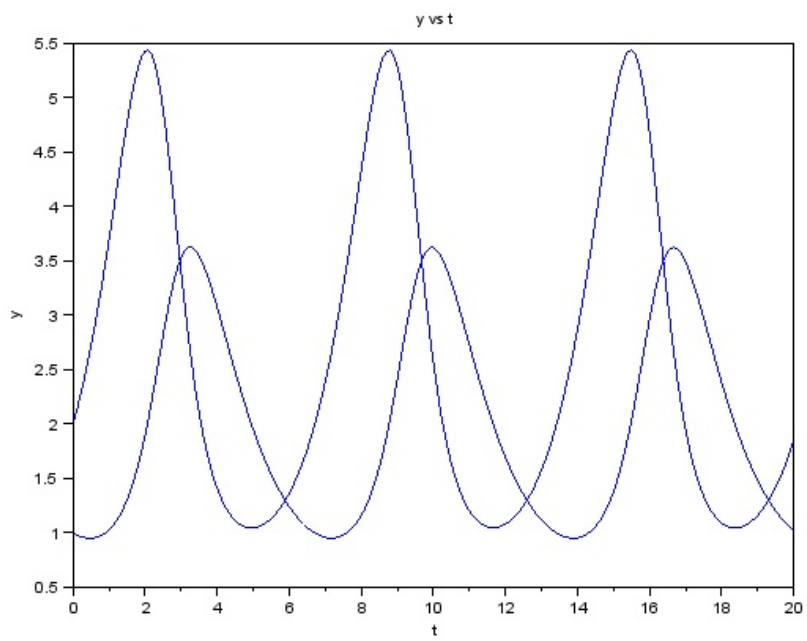


Figure 27.2: Eigenvalues and ODEs

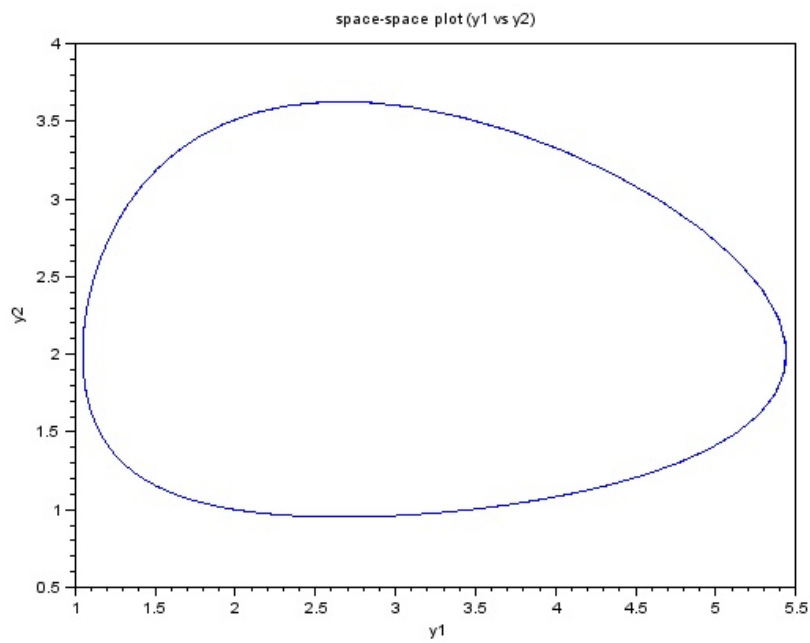


Figure 27.3: Eigenvalues and ODEs

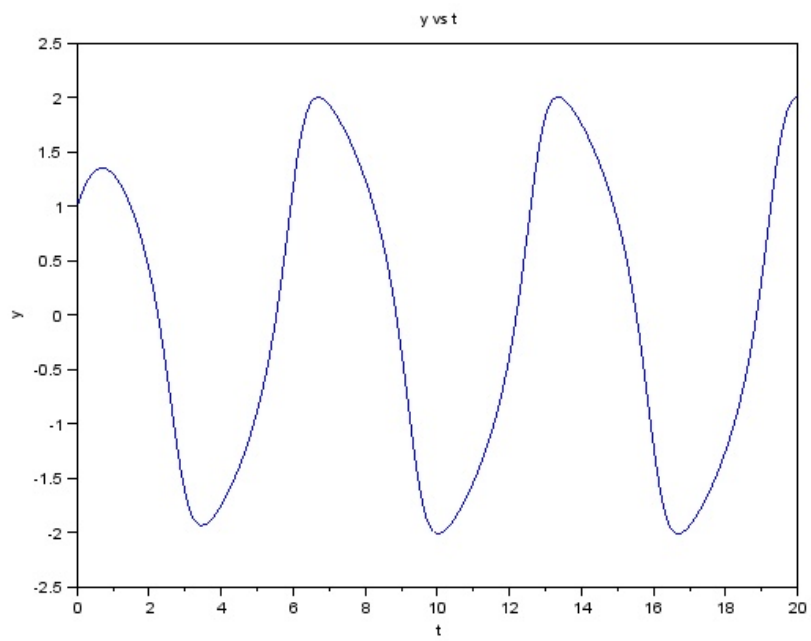


Figure 27.4: Stiff ODEs



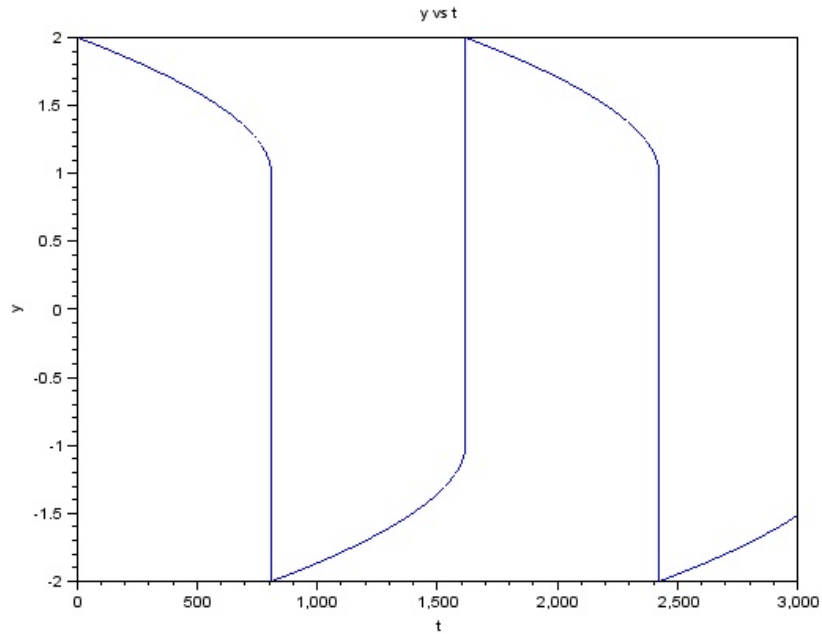


Figure 27.5: Stiff ODEs

```

8 sol=ode(y0,0,t,vanderpol);
9 count=1;
10 for i=1:2:401
11     x(count)=sol(i);
12     count=count+1;
13 end
14 plot(t,x)
15 xtitle("y vs t","t","y")

```

---

Scilab code Exa 27.10.b Stiff ODEs

```

1  clc;
2  clear;
3  clf();
4  function yp=vanderpol1(t,y)
5      yp=[y(2);1000*(1-y(1)^2)*y(2)-y(1)]
6  endfunction
7  t=0:3000;
8  yo=[1 1];
9  sol=ode(yo,0,t,vanderpol1);
10 count=1;
11
12 for i=1:2:6001
13     z(count)=sol(i)*-1;
14     count=count+1;
15 end
16
17 plot(t,z)
18 xtitle("y vs t","t","y")

```

---

### Scilab code Exa 27.11 Solving ODEs

```

1  clc;
2  clear;
3  function yp=predprey(t,y)
4      yp=[1.2*y(1)-0.6*y(1)*y(2);-0.8*y(2)+0.3*y(1)*y
5          (2)];
6  endfunction
7  t=0:10;
8  y0=[2 1];
9  sol=ode(y0,0,t,predprey);
10 count=0;
11 for i=1:2:22
12     disp(count,"istep=")
13     disp(count,"time=")
14     disp(sol(i),"y1=")

```

```
14     disp(sol(i+1), "y2=")
15     disp("
        _____
        ")
16     count=count+1;
17 end
_____
```

## Chapter 29

# Finite Difference Elliptic Equations

Scilab code Exa 29.1 Temperature of a heated plate with fixed boundary conditions

```
1 //clc()
2 wf = 1.5;
3 for i = 1:5
4     for j = 1:5
5         T(i,j) = 0;
6         if j == 1 then
7             T(i,j) = 0; //C
8         else
9             if j == 5 then
10                T(i,j) = 100; //C
11            end
12        end
13        if i == 1 then
14            T(i,j) = 75; //C
15        else
16            if i == 5 then
17                T(i,j) = 50; //C
18            end
19        end
20    end
21 end
```

```

20     end
21 end
22 e = 100;
23 while e>1
24 for i=1:5
25     for j = 1:5
26         if i>1 & j>1 & i<5 & j<5 then
27             Tn(i,j) = (T(i + 1,j) + T(i-1,j) + T(i,j+1)
28                 + T(i,j-1))/4;
29             Tn(i,j) = wf * Tn(i,j) + (1-wf)*T(i,j);
30             if i==2 & j==2 then
31                 e = abs((Tn(i,j) - T(i,j)) * 100/ (Tn(i,
32                     j)));
33             end
34             T(i,j) = Tn(i,j);
35         end
36     end
37 disp(T(2:4,2:4),"for error < 1, the temperatures are
    ")

```

---

### Scilab code Exa 29.2 Flux distribution for a heated plate

```

1 //clc()
2 k = 0.49; //cal/(s.cm.C)
3 T1 = 33.29755; //C
4 T2 = 75; //C
5 T3 = 63.21152; //C
6 T4 = 0; //C
7 qx = -k * (T1 - T2) / (2*10);
8 qy = -k * (T3 - T4) / (2*10);
9 disp(qx,"qx = ")
10 disp(qy,"qy = ")
11 q = (qx^2 + qy^2)^0.5;

```



```

24 disp(T20,"T20 = ")
25 disp(T30,"T30 = ")
26 disp(T11,"T11 = ")
27 disp(T21,"T21 = ")
28 disp(T31,"T31 = ")
29 disp(T12,"T12 = ")
30 disp(T22,"T22 = ")
31 disp(T32,"T32 = ")
32 disp(T13,"T13 = ")
33 disp(T23,"T23 = ")
34 disp(T33,"T33 = ")

```

---

#### Scilab code Exa 29.4 Heated plate with an irregular boundary

```

1 //clc()
2 //for irregular boundary,
3 //x = y
4 alpha1 = 0.732;
5 beta1 = 0.732;
6 alpha2 = 1;
7 beta2 = 1;
8 //substituting the above value to obtain
   simultaneous equation we get the following matrix
   ,
9 A =
   [4, -0.845, 0, -0.845, 0, 0, 0, 0, 0, 0; -1, 4, -1, 0, -1, 0, 0, 0, 0, 0; 0, -1, 4, 0, 0, -1, 0,
10 disp(A,"A = ")
11 B = [173.2; 75; 125; 75; 0; 50; 175; 100; 150];
12 disp(B,"B = ")
13 T = inv(A)*B;
14 T11 = det(T(1,1));
15 T21 = det(T(2,1));
16 T31 = det(T(3,1));
17 T12 = det(T(4,1));

```

```
18 T22 = det(T(5,1));
19 T32 = det(T(6,1));
20 T13 = det(T(7,1));
21 T23 = det(T(8,1));
22 T33 = det(T(9,1));
23 disp(T11,"T11 = ")
24 disp(T21,"T21 = ")
25 disp(T31,"T31 = ")
26 disp(T12,"T12 = ")
27 disp(T22,"T22 = ")
28 disp(T32,"T32 = ")
29 disp(T13,"T13 = ")
30 disp(T23,"T23 = ")
31 disp(T33,"T33 = ")
```

---



# Chapter 30

## Finite Difference Parabolic Equations

Scilab code Exa 30.1 Explicit solution of a one dimensional heat conduction equation

```
1 //clc ()
2 l = 10; //cm
3 k1 = 0.49; //cal/(s.cm.C)
4 x = 2; //cm
5 dt = 0.1; //sec
6 T0 = 100; //C
7 T10 = 50; //C
8 C = 0.2174; //cal/(g.C)
9 rho = 2.7; //g/cm^3
10 k = k1/(C*rho);
11 L = k * dt / x^2;
12 disp(L, "L =")
13 T(1,1) = 100;
14 T(1,2) = 0;
15 T(1,3) = 0;
16 T(1,4) = 0;
17 T(1,5) = 0;
18 T(1,6) = 50;
19 T(2,1) = 100;
```

```

20 T(2,6) = 50;
21 for i = 2:3
22     for j = 2:5
23         T(i,j) = T(i-1,j) + L * (T(i-1,j+1) - 2* T(i
                -1,j) + T(i-1,j-1));
24     end
25 end
26 disp(T(2,2), "T11 = ")
27 disp(T(2,3), "T12 = ")
28 disp(T(2,4), "T13 = ")
29 disp(T(2,5), "T14 = ")
30 disp(T(3,2), "T21 = ")
31 disp(T(3,3), "T22 = ")
32 disp(T(3,4), "T23 = ")
33 disp(T(3,5), "T24 = ")

```

---

Scilab code Exa 30.2 Simple implicit solution of a heat conduction equation

```

1 //clc()
2 l = 10; //cm
3 k1 = 0.49; //cal/(s.cm.C)
4 x = 2; //cm
5 dt = 0.1; //sec
6 C = 0.2174; //cal/(g.C)
7 rho = 2.7; //g/cm^3
8 k = k1/(C*rho);
9 L = k * dt / x^2;
10 disp(L, "L =")
11 //now, at t = 0, 1.04175 *T'1 + 0.020875 *T'2 = 0 +
    0.020875*100
12 //similarly getting other simultaneous equations, we
    get the following matrix
13 A =
    [1.04175, -0.020875, 0, 0; -0.020875, 1.04175, -0.020875, 0; 0, -0.020875,

```

```

14 B = [2.0875;0;0;1.04375];
15 X = inv(A) * B;
16 T11 = det(X(1,1));
17 T12 = det(X(2,1));
18 T13 = det(X(3,1));
19 T14 = det(X(4,1));
20 disp("for t = 0")
21 disp(T11,"T11 = ")
22 disp(T12,"T12 = ")
23 disp(T13,"T13 = ")
24 disp(T14,"T14 = ")
25 //to solve for t = 0.2, the right hand side vector
    is modified to D,
26 D = [4.09215;0.04059;0.02090;2.04069];
27 Y = inv(A)*D;
28 T21 = det(Y(1,1));
29 T22 = det(Y(2,1));
30 T23 = det(Y(3,1));
31 T24 = det(Y(4,1));
32 disp("for t = 0.2")
33 disp(T21,"T21 = ")
34 disp(T22,"T22 = ")
35 disp(T23,"T23 = ")
36 disp(T24,"T24 = ")

```

---

**Scilab code Exa 30.3** Crank Nicolson solution to the heat conduction equation

```

1 //clc()
2 //using crank nicolson method, we get simultaneous
    equations which can be simplified to following
    matrices
3 A =
    [2.04175, -0.020875, 0, 0; -0.020875, 2.04175, -0.020875, 0; 0, -0.020875,
4 B = [4.175;0;0;2.0875];

```

```

5 X = inv(A)*B;
6 disp(" at t = 0.1 s")
7 disp(det(X(1,1)), "T11 = ")
8 disp(det(X(2,1)), "T12 = ")
9 disp(det(X(3,1)), "T13 = ")
10 disp(det(X(4,1)), "T14 = ")
11 C = [8.1801;0.0841;0.0427;4.0901];
12 Y = inv(A)*C;
13 disp(" at t = 0.2 s")
14 disp(det(Y(1,1)), "T21 = ")
15 disp(det(Y(2,1)), "T22 = ")
16 disp(det(Y(3,1)), "T23 = ")
17 disp(det(Y(4,1)), "T24 = ")

```

---

#### Scilab code Exa 30.4 Comparison of Analytical and Numerical solution

```

1 //clc()
2 x = 2; //cm
3 L = 10; //cm
4 k = 0.835; //cm^2/s
5 t = 10; //s
6 n = 1:10000;
7 T = 324.* (x/L + sum(2.*((-1)^n).*sin(n.*x/L).*exp(-
      n^2.* %pi^2.* k.* t / L^2)/(n.*%pi)));
8 disp(T, "T[2,10] analytical = ")
9 Tex = 64.97;
10 disp(Tex, "T[2,10] explicit = ")
11 Tim = 64.49;
12 disp(Tim, "T[2,10] implicit = ")
13 Tcn = 64.73;
14 disp(Tcn, "T[2,10] crank-nicolson = ")

```

---

#### Scilab code Exa 30.5 ADI Method

```

1 //clc ()
2 x = 10; //cm
3 L = 0.0835;
4 t1 = 5;
5 //for first step t = 5 is applied to nodes (1,1) ,
   (1,2) and (1,3) to yield following matrices
6 A =
   [2.167, -0.0835, 0; -0.0835, 2.167, -0.0835; 0, -0.0835, 2.167];

7 B = [6.2625; 6.2625; 14.6125];
8 X = inv(A)*B;
9 disp("At t = 5 s")
10 disp(det(X(1,1)), "T11 = ")
11 disp(det(X(2,1)), "T12 = ")
12 disp(det(X(3,1)), "T13 = ")
13 //similarly we get ,
14 T21 = 0.1274;
15 T22 = 0.2900;
16 T23 = 4.1291;
17 T31 = 2.0181;
18 T32 = 2.2477;
19 T33 = 6.0256;
20 disp(T21, "T21 = ")
21 disp(T22, "T22 = ")
22 disp(T23, "T23 = ")
23 disp(T31, "T31 = ")
24 disp(T32, "T32 = ")
25 disp(T33, "T33 = ")
26 C = [13.0639; 0.2577; 8.0619];
27 Y = inv(A)*C;
28 disp("At t = 10 s")
29 disp(det(Y(1,1)), "T11 = ")
30 disp(det(Y(2,1)), "T12 = ")
31 disp(det(Y(3,1)), "T13 = ")
32 //similarly we get ,
33 T21 = 6.1683;
34 T22 = 0.8238;
35 T23 = 4.2359;

```

```
36 T31 = 13.1120;  
37 T32 = 8.3207;  
38 T33 = 11.3606;  
39 disp(T21,"T21 = ")  
40 disp(T22,"T22 = ")  
41 disp(T23,"T23 = ")  
42 disp(T31,"T31 = ")  
43 disp(T32,"T32 = ")  
44 disp(T33,"T33 = ")
```

---

# Chapter 31

## Finite Element Method

Scilab code Exa 31.1 Analytical Solution for Heated Rod

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

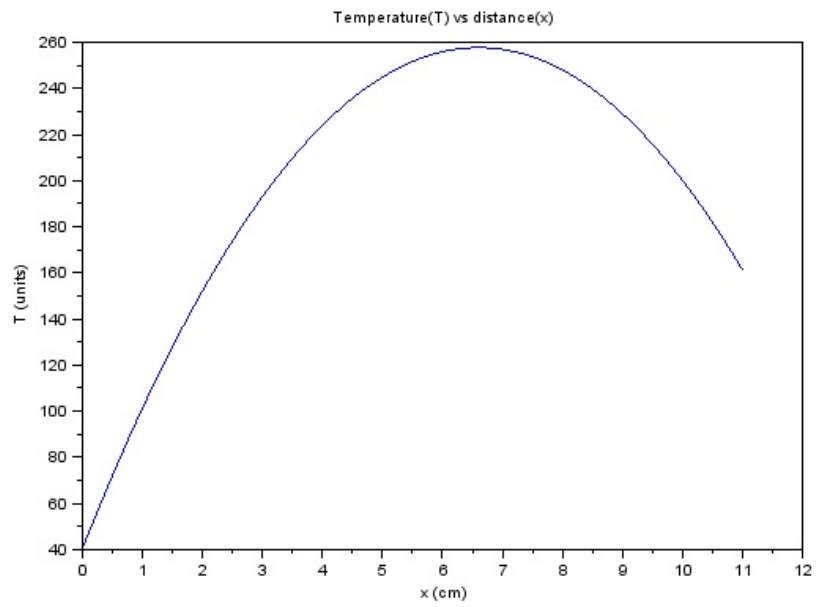


Figure 31.1: Analytical Solution for Heated Rod



```

20     T(count)=f(i);
21     count=count+1;
22 end
23 x=0:0.1:11
24 plot(x,T)
25 xtitle("Temperature(T) vs distance(x)","x (cm)","T (
    units)")

```

---

### Scilab code Exa 31.2 Element Equation for Heated Rod

```

1  clc;
2  clear;
3  xf=10; //cm
4  xe=2.5; //cm
5  //T(0,t)=40; boundary condition
6  //T(10,t)=200; boundary condition
7  //f(x)=10; uniform heat source
8  function y=f(x)
9      y=10*(xe-x)/xe;
10 endfunction
11 int1=intg(0,xe,f)
12 function y=g(x)
13     y=10*(x-0)/xe;
14 endfunction
15 int2=intg(0,xe,g)
16 disp("The results are:")
17 disp("0.4*T1-0.4*T2=-(dT/dx)*x1 + c1")
18 disp(int1," where c1=")
19 disp(" and")
20 disp("-0.4*T1+0.4*T2=-(dT/dx)*x2 + c2")
21 disp(int2," where c2=")

```

---

### Scilab code Exa 31.3 Temperature of a heated plate

```

1 //clc()
2 wf = 01.5;
3 for i = 1:41
4     for j = 1:41
5         T(i,j) = 0;
6         if j == 1 then
7             T(i,j) = 0; //C
8         else
9             if j == 41 then
10                T(i,j) = 100; //C
11            end
12        end
13        if i == 1 then
14            T(i,j) = 75; //C
15        else
16            if i == 41 then
17                T(i,j) = 50; //C
18            end
19        end
20    end
21 end
22 e = 100;
23 while e>1
24 for i=1:41
25     for j = 1:41
26         if i>1 & j>1 & i<41 & j<41 then
27             Tn(i,j) = (T(i + 1,j) + T(i-1,j) + T(i,j+1)
28                 + T(i,j-1))/4;
29             Tn(i,j) = wf * Tn(i,j) + (1-wf)*T(i,j);
30             if i==2 & j==2 then
31                 e = abs((Tn(i,j) - T(i,j)) * 100/ (Tn(i,
32                     j)));
33             end
34         end
35     end
36 end

```

37 `disp(T,"for error < 1, the temperatures are")`

---