Scilab Textbook Companion for Elementary Linear Algebra by Stephen Andrilli, David Hecker[1]

Created by
Parth Birthare
Bachelor of Technology
Computer Engineering
Vellore Institute Of Technology
Cross-Checked by
Scilab Tbc Team

March 1, 2022

[1]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: Elementary Linear Algebra

Author: Stephen Andrilli, David Hecker

Publisher: Academic Press - Elsevier

Edition: 5

Year: 2016

ISBN: 9780128008539

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 Vectors and Matrices 5

2 Systems of Linear Equations 30

3 Determinants and Eigenvalues 61

4 Finite Dimensional Vector Spaces 93

5 Linear Transformations 103

6 Orthogonality 105

7 Complex Vector Spaces and General Inner Products 110

8 Additional Applications 114

9 Numerical Techniques 116

List of Scilab Codes

Exa 1.1.1	Length or magnitude of a vector	5
Exa 1.1.2	Calculating unit vector	5
Exa 1.1.3	Calculating resultant velocity	6
Exa 1.1.4	Calculating acceleration to the object	7
Exa 1.2.1	Verify the Cauchy Schwarz Inequality	9
Exa 1.2.2	Angle between vector	10
Exa 1.2.3	Angle between vector	11
Exa 1.2.4	orthogonality of vectors	12
Exa 1.2.5	Parallel vectors	12
Exa 1.2.6	Projection of vectors	13
Exa 1.2.7	Component of vector orthogonal to projection vector	14
Exa 1.2.8	Calculating work done by force	16
Exa 1.4.1	Symmetric and skew symmetric matrix	17
Exa 1.4.2	Deriving symmetric and skew symmetric matrix	18
Exa 1.4.3	Decomposing a matrix	20
Exa 1.5.1	Multiplication of matrix	21
Exa 1.5.2	Multiplication of matrix	21
Exa 1.5.3	DVD warehouse problem	22
Exa 1.5.4	Linear Combinations Using Matrix Multiplication	23
Exa 1.5.5	Powers of square matrix	24
Exa 1.5.6	Identity matrix operations	25
Exa 1.5.7	Matrix multiplication operations	26
Exa 1.5.8	Matrix transpose operations	27
Exa 2.1.1	Solving linear equations	30
Exa 2.1.2	Solving linear equations	31

Exa 2.1.3 Solving linear equations 33
Exa 2.1.4 Solving linear equations 36
Exa 2.1.5 Solving linear equations 39
Exa 2.1.6 Gaussian elimination 41
Exa 2.1.7 Gaussian elimination 42
Exa 2.1.8 Gaussian elimination 42
Exa 2.1.9 Row operations effect on matrix multiplications 44
Exa 2.2.1 Gauss jordan elimination 46
Exa 2.2.2 Reduced Row Echelon Form 47
Exa 2.2.3 Reduced Row Echelon Form 48
Exa 2.2.4 Reduced Row Echelon Form 49
Exa 2.3.1 Rank of a matrix 49
Exa 2.3.2 Linear Combinations of Vectors 50
Exa 2.3.3 Linear Combinations of Vectors 51
Exa 2.3.4 Linear Combinations of Vectors 52
Exa 2.3.5 Linear Combinations of Vectors 53
Exa 2.3.6 Linear Combinations of Vectors 54
Exa 2.4.1 Inverse of matrix 54
Exa 2.4.2 Properties of inverse of matrix 56
Exa 2.4.3 Inverse of matrix 56
Exa 2.4.4 Inverse of matrix 58
Exa 2.4.5 Inverse of matrix 58
Exa 2.4.6 Solving equations using inverse of coefficient matrix 59
Exa 3.1.1 Finding determinant 61
Exa 3.1.2 Finding volume of parallelepiped 62
Exa 3.1.3 Submatrix 62
Exa 3.1.4 Cofactor 63
Exa 3.1.5 Finding determinant 65
Exa 3.1.6 Finding determinant 66
Exa 3.2.1 Determinant of upper triangular matrix . . 68
Exa 3.2.2 Effect of row operations on determinant . . 68
Exa 3.2.3 Multiplication of scalar in matrix 70
Exa 3.2.4 Determinant by row reduction 71
Exa 3.2.5 Determinant by row reduction 72
Exa 3.2.6 Determinant criterion for matrix singularity 74
Exa 3.3.1 Matrix product 75

Exa 3.3.2 Determinant of transpose 76
Exa 3.3.3 Row operation effect 77
Exa 3.3.4 Determinant of matrix 78
Exa 3.3.5 Cramers rule 79
Exa 3.4.1 Eigenvalue 80
Exa 3.4.2 Characteristic polynomial of matrix 80
Exa 3.4.3 Characteristic polynomial of matrix 82
Exa 3.4.4 Characteristic polynomial of matrix 83
Exa 3.4.5 Diagonalization 85
Exa 3.4.6 Diagonalization 86
Exa 3.4.7 Diagonalization 87
Exa 3.4.9 Algebraic multiplicity 87
Exa 3.4.10 Algebraic multiplicity 88
Exa 3.4.11 Diagonalization 89
Exa 3.4.12 Diagonalization 90
Exa 3.4.13 Rounding off eigen value 91
Exa 4.4.4 Linear Dependence and Independence With One and Two Element Sets 93
Exa 4.4.6 Using Row Reduction to Test for Linear Independence 94
Exa 4.4.7 Using Row Reduction to Test for Linear Independence 95
Exa 4.4.8 Using Row Reduction to Test for Linear Independence 95
Exa 4.4.9 Using Row Reduction to Test for Linear Independence 96
Exa 4.5.15 Diagonalization and Bases 97
Exa 4.6.1 Using the Simplified Span Method to Construct a Basis 98
Exa 4.6.2 Using the Simplified Span Method to Construct a Basis 99
Exa 4.6.3 Using the Simplified Span Method to Construct a Basis 100
Exa 4.6.4 Using the Simplified Span Method to Construct a Basis 101
Exa 4.6.5 CONSTRUCTING SPECIAL BASES Using the Simplified Span Method to Construct a Basis 102

Exa 5.3.7 Rank of the Transpose 103
Exa 6.1.1 Orthogonal and Orthonormal Vectors 105
Exa 6.1.2 Orthogonal and Orthonormal Vectors 106
Exa 6.1.6 Orthogonal Matrices 107
Exa 6.2.7 Orthogonal Projection Onto a Subspace . . 107
Exa 6.2.10 Matrix with respect to standard basis 108
Exa 6.2.11 Distance From a Point to a Subspace 109
Exa 7.1.1 Complex matrices operations 110
Exa 7.1.2 Complex matrices finding entry 111
Exa 7.1.3 Conjugate transpose of complex matrix . . . 111
Exa 7.1.4 Hermitian and skew hermitian matrix 112
Exa 7.1.5 Normal matrix 113
Exa 8.4.1 Markov chains 114
Exa 8.4.2 Markov chains 115
Exa 9.1.1 Gaussian elimination without partial pivoting 116
Exa 9.1.2 Gaussian elimination with partial pivoting . 118

Chapter 1

Vectors and Matrices

Scilab code Exa 1.1.1 Length or magnitude of a vector

```
1 // length/magnitude of a vector
2
3 clc
4 clear
5
6 // initializing and assigning vector
7 vector1 = [4,-3,0,2];
8 disp("vector =", vector1);
9
10 // finding length of vector
11 vector1_length = norm(vector1); // norm to calculate
      vector magnitude
12 disp('Length of vector =', vector1_length);
```

Scilab code Exa 1.1.2 Calculating unit vector

```
1 // ex 1.1 -> example 2 -> calculating unit vector
2
```

```
clc
clear

// initializing and assigning vector
vector1 = [2,3,-1,1];

// calculating magnitude of vector
vector1_magnitude = norm(vector1);

// calculating unit vector
vector1_unit = vector1/vector1_magnitude

disp('Unit vector in the direction of vector =',
   vector1_unit);
```

Scilab code Exa 1.1.3 Calculating resultant velocity

```
// ex 1.1 -> example 3 -> calculating resultant
   velocity

clc
clear

// velocity of man in calm water
velocity_man = [5,0]; // in east direction x
   component of velocity used. y component is 0
disp('Velocity of man in calm water =', velocity_man
   );

// velocity of current
velocity_current = [3*cos(%pi-%pi/4),3*sin(%pi-%pi
   /4)];
disp('Velocity of current', velocity_current);

// resultant velocity of man
```

```
15 velocity_resultant = velocity_man + velocity_current
   ;
16 disp('Resultant velocity of man =',
   velocity_resultant);
17 disp('That is', velocity_resultant(1), 'km/hr  in
   east direction and', velocity_resultant(2), 'km/
   hr in north direction');
18
19 // calculating the magnitude of resultant velocity
20 velocity_resultant_magnitude = norm(
   velocity_resultant);
21
22 // Answer varies due to round off error
23 disp('The resultant speed is',
   velocity_resultant_magnitude, 'km/hr');
```

Scilab code Exa 1.1.4 Calculating acceleration to the object

```
1 // ex 1.1 -> example 4 -> newton's second law,
   calculating acceleration to the object
2
3 clc
4 clear
5
6 // mass of object in kg
7 mass = 5;
8
9 disp('Mass of object is', mass, 'kg');
10
11 // force1 of 10 newtons applied to object in the
   direction of the vector [ 2 , 1, 2]
12 force1_magnitude = 10;
13 force1_direction = [-2, 1, 2];
14
15 // force2 of 20 newtons applied to object in the
```

```
    direction of the vector [6, 3, -2]
force2_magnitude = 20;
force2_direction = [6,3,-2];

// calculating unit vectors in direction of force
    vectors
force1_norm = norm(force1_direction);
force1_unit = force1_direction/force1_norm

force2_norm = norm(force2_direction);
force2_unit = force2_direction/force2_norm

// calculating net force
force_net = (force1_magnitude*force1_unit) + (
    force2_magnitude*force2_unit);

disp('Net force applied on object is', force_net);

// calculating net acceleration on object
// force = mass*acceleration => acceleration = force
    /mass
disp('Acceleration = force/mass');
acceleration_net = force_net/mass;

disp('Net acceleration of object is',
    acceleration_net);

// calculating length/magnitude of acceleration
    vector
acceleration_net_norm = norm(acceleration_net);

// Answer varies due to round off error
disp('Magnitude of acceleration is',
    acceleration_net_norm); // in book magnitude of
    acceleration is rounded off 3.1773     3.18

// calculating unit vector in direction of
    acceleration
```

```
45 acceleration_net_unit = acceleration_net/
      acceleration_net_norm;
46
47 disp('Acceleration applied on object is',
      acceleration_net_norm, 'm/sec^2 in the direction'
      ,acceleration_net_unit);
```

Scilab code Exa 1.2.1 Verify the Cauchy Schwarz Inequality

```
1  // ex 1.2 -> example 1 -> verify the Cauchy-Schwarz
      Inequality
2
3  clc
4  clear
5
6  // initializing and assigning vectors
7  x = [-1,4,2,0,-3];
8  y = [2,1,-4,-1,0];
9
10 // calculating dot product of x and y
11 x_y_dot = sum(x.*y);
12
13 disp('|x.y| =', abs(x_y_dot));
14
15 // calculating magnitude of vecors
16 x_norm = norm(x);
17 y_norm = norm(y);
18
19 // calculating x_norm*y_norm
20 norm_product = x_norm*y_norm;
21
22 // Answer varies due to round off error
23 disp('|x|*|y| =', norm_product);
24
25 if abs(x_y_dot) <= norm_product then
```

```
26      disp('Cauchy-Schwarz Inequality verified as |x.y
           | <= |x|*|y|');
27  else
28      disp('Cauchy-Schwarz Inequality not verified');
29  end
```

Scilab code Exa 1.2.2 Angle between vector

```
1  // ex 1.2 -> example 2 -> angle between vector
2
3  clc
4  clear
5
6  // initializing and assigning vectors
7  x = [6,-4];
8  y = [-2,3];
9
10 disp('vectors are', x, 'and', y);
11
12 // dot product of vectors
13 x_y_dot = sum(x.*y);
14
15 // magnitude of vectors
16 x_norm = norm(x);
17 y_norm = norm(y);
18
19 // product of magnitude of vectors
20 norm_product = x_norm*y_norm;
21
22 // calculating angle
23 // x.y = |x|*|y|*cos(theta)
24
25 theta = acos(x_y_dot/norm_product);
26 disp('Angle between vector in radian is', theta, '
      And in degrees is ', theta*180/%pi); // Answer
```

```
varies due to round off error
```

Scilab code Exa 1.2.3 Angle between vector

```
// ex 1.2 -> example 3 -> angle between vector

clc
clear

// initializing and assigning vectors
x = [-1,4,2,0,-3];
y = [2,1,-4,-1,0];

disp('vectors are', x, 'and', y);

// dot product of vectors
x_y_dot = sum(x.*y);

// magnitude of vectors
x_norm = norm(x);
y_norm = norm(y);

// product of magnitude of vectors
norm_product = x_norm*y_norm;

// calculating angle
// x.y = |x|*|y|*cos(theta)

theta = acos(x_y_dot/norm_product);
disp('Angle between vector in radian is', theta, '
   And in degrees is ', theta*180/%pi); // Answer
   varies due to round off error
```

Scilab code Exa 1.2.4 orthogonality of vectors

```
1 // ex 1.2 -> example 4 -> orthogonality of vectors
2
3 clc
4 clear
5
6 // initializing and assigning vectors
7 x = [2,-5];
8 y = [-10,-4];
9
10 disp('vectors are', x, 'and', y);
11
12 // vectors are orthogonal if x.y = 0
13
14 // calculating dot product of vectors
15 x_y_dot = sum(x.*y);
16
17 // checking orthogonality of vectors
18 if x_y_dot == 0 then
19     disp('Vectors are orthogonal, as dot product is
          0. So angle between them is 90  ');
20 else
21     disp('Vectors are not orthogonal');
22 end
```

Scilab code Exa 1.2.5 Parallel vectors

```
1 // ex 1.2 -> example 5 -> parallel vectors
2
3 clc
4 clear
5
6 // initializing and assigning vectors
7 x = [8,-20,4];
```

```
y = [6,-15,3];

disp('vectors are', x, 'and', y);

// If vectors are parallel, the dot product of
   vectors will be 0. As cos0 = 0. Since the angle
   between the vectors is 0
// dot product of vectors
x_y_dot = sum(x.*y);

// magnitude of vectors
x_norm = norm(x);
y_norm = norm(y);

// product of magnitude of vectors
norm_product = x_norm*y_norm;

// calculating angle
// x.y = |x|*|y|*cos(theta)

theta = round(acos(x_y_dot/norm_product)); //
   rounding off to nearest integer

if theta == 0 then
    disp('Vectors are parallel');
else
    disp('Vectors are not parallel');
end
```

Scilab code Exa 1.2.6 Projection of vectors

```
// ex 1.2 -> example 6 -> projection of vectors

clc
clear
```

```
5
6 // initializing and assigning vectors
7 a = [4,0,-3];
8 b = [3,1,-7];
9
10 disp('Vectors are', a, 'and', b);
11
12 // calculating dot product of vectors
13 a_b_dot = sum(a.*b);
14
15 // calculating magnitude of vector1
16 a_norm = norm(b);
17
18 // projection vector of b onto a
19 // projection of b onto a = (b.a/|a|^2)*a
20 proj_b_on_a = (a_b_dot/a_norm^2)*a;
21
22 disp('Projection vector of b onto a is', proj_b_on_a
   );
```

Scilab code Exa 1.2.7 Component of vector orthogonal to projection vector

```
1 // ex 1.2 -> example 7 -> component of vector
   orthogonal to projection vector
2
3 clc
4 clear
5
6 // initializing and assigning vectors
7 a = [4,0,-3];
8 b = [3,1,-7];
9
10 disp('Vectors are', a, 'and', b);
11
12 // calculating dot product of vectors
```

```
13 a_b_dot = sum(a.*b);
14
15 // calculating magnitude of a
16 a_norm = norm(a);
17
18 // projection vector of b onto a
19 // projection of b onto a = (b.a/|a|^2)*a
20 proj_b_on_a = (a_b_dot/a_norm^2)*a;
21
22 disp('Projection vector of b onto a is', proj_b_on_a
   );
23
24 // calculating component of b orthogonal to a and
   projection vector
25 b_orthogonal_component = b - proj_b_on_a;
26
27 disp('Component of b orthogonal to a and projection
   vector is', b_orthogonal_component);
28
29 // verifying if the b component is orthogonal to a
   or not
30 if sum(b_orthogonal_component.*a == 0) then
31     disp('Component of b is orthogonal to a');
32 else
33     disp('Component of b is not orthogonal to a');
34 end
35
36 // verifying if the b component is orthogonal to
   projection vector of b onto a or not
37 if sum(b_orthogonal_component.*proj_b_on_a == 0)
   then
38     disp('Component of b is orthogonal to projection
          vector of b onto a');
39 else
40     disp('Component of b is not orthogonal to
         projection vector of b onto a');
41 end
```

Scilab code Exa 1.2.8 Calculating work done by force

```
1 // ex 1.2 -> example 8 -> calculating work done by
     force
2
3 clc
4 clear
5
6 // initializing and assigning vectors
7
8 // magnitude of force in Newton
9 f_magnitude = 8;
10
11 // direction vector to which force is applied to
12 f_direction = [1,-2,1];
13
14 // magnitude of distance travelled by the object
15 d_magnitude = 5;
16
17 // direction vector in which the object moves
18 d_direction = [2,-1,0];
19
20
21 // calculating unit vectors
22
23 // calculating unit vector in the direction of force
      vector
24 f_unit = f_direction/norm(f_direction);
25
26 // calculating unit vector in the direction of
     distance vector
27 d_unit = d_direction/norm(d_direction);
28
29
```

```
30 // calculating work done by force
31 // work done = force.displacement
32 work_done = sum((f_magnitude*f_unit).*(d_magnitude*
      d_unit));
33
34 disp('total work done in Joules is', work_done); //
      Answer varies due to round off error
```

Scilab code Exa 1.4.1 Symmetric and skew symmetric matrix

```
1 // ex 1.4 -> example 1 -> symmetric and skew-
      symmetric matrix
2
3 clc
4 clear
5
6 // defining A and B
7 A = [2,6,4; 6,-1,0; 4,0,-3];
8 B = [0,-1,3,6; 1,0,2,-5; -3,-2,0,4; -6,5,-4,0];
9
10 // defining transpose of A and B
11 A_transpose = A';
12 B_transpose = B';
13
14 // displaying A and B
15 disp('A =', A, 'B =', B);
16
17 // checking if A is symmetric or non-symmetric
18 if A == A_transpose then
19     disp('A is symmetric');
20 else if A == -A_transpose then
21         disp('A is skew symmetric');
22     else
23         disp('A is neither symmetric nor skew-
              symmetric');
```

```
24      end
25  end
26
27  // checking if B is symmetric or non-symmetric
28  if B == B_transpose then
29      disp('B is symmetric');
30  else if B == -B_transpose then
31          disp('B is skew symmetric');
32      else
33          disp('B is neither symmetric nor skew-
                symmetric');
34      end
35  end
```

Scilab code Exa 1.4.2 Deriving symmetric and skew symmetric matrix

```
1  // ex 1.4 -> example 2 -> deriving symmetric and
       skew-symmetric matrix
2
3  clc
4  clear
5
6  // defining matrix1
7  matrix1 = [-4,3,-2; 5,-1,6; -3,8,1];
8
9  // displaying matrix1
10 disp('matrix1 =', matrix1);
11
12 // defining transpose of matrix1
13 matrix1_transpose = matrix1';
14
15 // checking if matrix1 is symmetric or non-symmetric
16 if matrix1 == matrix1_transpose then
17     disp('matrix1 is symmetric');
18 else if matrix1 == -matrix1_transpose then
```

```
19          disp('matrix1 is skew symmetric');
20      else
21          disp('matrix1 is neither symmetric nor skew-
               symmetric');
22      end
23  end
24
25  // creating symmetric matrix from matrix1
26  matrix1_symmetric = matrix1 + matrix1_transpose
27
28  disp('new matrix1 derived from matrix1 +
       matrix1_transpose =', matrix1_symmetric);
29
30  // checking if matrix1_symmetric is symmetric or not
31  if matrix1_symmetric == matrix1_symmetric' then
32      disp('new matrix1 is symmetric');
33  else if matrix1_symmetric == -matrix1_symmetric'
       then
34          disp('matrix1 is skew symmetric');
35      else
36          disp('matrix1 is neither symmetric nor skew-
               symmetric');
37      end
38  end
39
40  // creating skew-symmetric matrix from matrix1
41  matrix1_skew_symmetric = matrix1 - matrix1_transpose
42
43  disp('new matrix1 derived from matrix1 -
       matrix1_transpose =', matrix1_skew_symmetric);
44
45  // checking if matrix1_symmetric is symmetric or not
46  if matrix1_skew_symmetric == matrix1_skew_symmetric'
       then
47      disp('new matrix1 is symmetric');
48  else if matrix1_skew_symmetric == -
       matrix1_skew_symmetric' then
49          disp('new matrix1 is skew symmetric');
```

```
50      else
51          disp('matrix1 is neither symmetric nor skew-
               symmetric');
52      end
53  end
```

Scilab code Exa 1.4.3 Decomposing a matrix

```
1  // ex 1.4 -> example 3 -> decomposing a matrix
2
3  clc
4  clear
5
6  // defining matrix A
7  A = [-4,2,5; 6,3,7; -1,0,2];
8
9  // displaying matrix A
10 disp('Matrix A =', A);
11
12 // deriving symmetric matrix S from matrix A
13 S = 1/2*(A + A');
14 disp('Symmetric matrix S =', S);
15
16 // deriving skew-symmetric matrix V from matrix A
17 V = 1/2*(A - A');
18 disp('Skew-symmetric matrix V =', V);
19
20 // decomposing A in terms of of symmetric and skew
     symmetric matrix
21 if S + V == A then
22      disp('Matrix  A is decomposed into sum of a
          symmetric S and skew-symmmetric matrix V');
23      disp('Symmetric matrix S + skew symmetric matrix
           V =', S + V, "= A");
24 end
```

Scilab code Exa 1.5.1 Multiplication of matrix

```
1 // ex 1.5 -> example 1 -> multiplication of matrix
2
3 clc
4 clear
5
6 // defining matrix
7 A = [5,-1,4; -3,6,0];
8 B = [9,4,-8,2; 7,6,-1,0; -2,5,3,-4];
9
10 // displaying A and B
11 disp('A =', A, 'B =', B);
12
13 // multiplying A and B
14 C = A*B;
15
16 // displaying matrix multiplication
17 disp('C = A*B =', C);
```

Scilab code Exa 1.5.2 Multiplication of matrix

```
1 // ex 1.5 -> example 2 -> multiplication of matrix
2
3 clc
4 clear
5
6 // defining matrix
7
8 D = [-2,1; 0,5; 4,-3];
9 E = [1,-6; 0,2];
```

```
10 F = [-4,2,1];
11 G = [7; -1; 5];
12 H = [5,0; 1,-3];
13
14 // displaying matrix
15 disp('D =', D,'E =', E,'F =', F,'G =', G,'H =', H);
16
17 // multiplying matrix
18 DE = D*E;
19 DH = D*H;
20 GF = G*F;
21 EE = E*E;
22 EH = E*H;
23 HE = H*E;
24 HH = H*H;
25 FG = F*G;
26 FD = F*D;
27
28 disp('D*E =', DE);
29 disp('D*H =', DH);
30 disp('G*F =', GF);
31 disp('E*E =', EE);
32 disp('E*H =', EH);
33 disp('H*E =', HE);
34 disp('H*H =', HH);
35 disp('F*G =', FG);
36 disp('F*D =', FD);
```

Scilab code Exa 1.5.3 DVD warehouse problem

```
1 // ex 1.5 -> example 3 -> dvd-warehouse problem
2
3 clc
4 clear
5
```

```
6  // defining matrix
7  A = [130,160,240,190; 210,180,320,240;
       170,200,340,220];
8  B = [3,3; 4,2; 3,4; 2,2];
9
10 disp('The number of each type of DVD shipped from
       each warehouse during the past week is ', A);
11
12 disp('The shipping cost and profit collected for
       each DVD sold is', B);
13
14 // calculating combined shipping cost and profit
15 AB = A*B;
16 disp('Combined total shipping costs and profits last
        week is', AB);
17
18 // calculating total profit for warehouse1,
       warehouse2 ad warehouse3
19 Warehouse1 = A(1,:)*B(:,2);
20 Warehouse2 = A(2,:)*B(:,2);
21 Warehouse3 = A(3,:)*B(:,2);
22
23 // displaying total profit for different warehouse
24 disp('total profit for warehouse1 =', Warehouse1);
25 disp('total profit for warehouse2 =', Warehouse2);
26 disp('total profit for warehouse3 =', Warehouse3);
```

Scilab code Exa 1.5.4 Linear Combinations Using Matrix Multiplication

```
1  // ex 1.5 -> example 4 -> Linear Combinations Using
       Matrix Multiplication
2
3  clc
4  clear
5
```

```
6  // defining A
7  A = [3,-2,6,5; -1,4,-1,-3; 2,-5,3,-6];
8
9  disp('A =', A);
10
11 // for creating linear combination of rows -> 7(1st
     row of matrix)     8(2nd row of matrix) + 9(3rd
     row of matrix)
12 disp('7(1st row of matrix)     8(2nd row of matrix)
     + 9(3rd row of matrix) =');
13
14 vector_to_multiply = [7,-8,9];
15 linear_combination1 = vector_to_multiply*A;
16
17 disp(linear_combination1);
18
19
20 // for creating linear combination of the columns ->
     10(1st column of matrix)     11(2nd column of
     matrix) + 12(3rd column of matrix)     13(4th
     column of matrix)
21 disp('10(1st column of matrix)     11(2nd column of
     matrix) + 12(3rd column of matrix)     13(4th
     column of matrix) =');
22
23 vector_to_multiply = [10; -11; 12; -13];;
24 linear_combination2 = A*vector_to_multiply;
25
26 disp(linear_combination2);
```

Scilab code Exa 1.5.5 Powers of square matrix

```
1  // ex 1.5 -> example 5 -> powers of square matrix
2
3  clc
```

```
4  clear
5
6  // defining matrix
7  A = [2,1; -4,3];
8
9  // displaying matrix
10 disp('A =', A);
11
12 // finding matrix of power 2
13 A_2 = A*A; // or -> A_2 = A^2;
14
15 // displaying matrix of power 2
16 disp('matrix of power 2 = A^2 =', A_2);
17
18 // finding matrix of power 3
19 A_3 = A_2*A; // or -> A_3 = A^3;
20
21 // displaying matrix of power3
22 disp('matrix of power 3 = A^3 =', A_3);
```

Scilab code Exa 1.5.6 Identity matrix operations

```
1  // ex 1.5 -> example 6 -> identity matrix operations
2
3  clc
4  clear
5
6  // defining matrix
7  // considering 3x3 identity matrix
8  identity_matrix = [1,0,0; 0,1,0; 0,0,1];
9  matrix1 = [1,4,2; 3,5,3; 7,4,5];
10
11 // displaying matrix
12 disp('3x3 identity matrix =', identity_matrix);
13 disp('matrix1 =', matrix1);
```

```
14
15 // identity matrix operations
16
17 // multiplication with matrix1
18 disp('matrix1*identity_matrix =', matrix1*
      identity_matrix);
19
20 // identity_matrix*identity_matrix
21 disp('identity_matrix*identity_matrix =',
      identity_matrix*identity_matrix);
22
23 // power of identity matrix
24 // assume the power of identity matrix =
      power_of_identity_matrix
25 power_of_identity_matrix = 10;
26 disp('identity_matrix^power_of_identity_matrix = ',
      identity_matrix^power_of_identity_matrix);
```

Scilab code Exa 1.5.7 Matrix multiplication operations

```
1 // ex 1.5 -> example 7 -> matrix multiplication
      operations
2
3 clc
4 clear
5
6 // defining matrix
7 A = [2,-4; 1,3];
8 B = [3,2; -1,5];
9
10 // displaying matrix
11 disp('A =', A);
12 disp('B =', B);
13
14 // calculating (A*B)^2
```

```
15 A_B_multiplication = A*B;
16 A_B_of_power_2 = A_B_multiplication^2;
17
18 // displaying (A*B)^2
19 disp('(A*B)^2 =', A_B_of_power_2);
20
21 // calculating power of matrix
22 A_of_power_2 = A^2;
23 B_of_power_2 = B^2;
24
25 // calculating (A^2)*(B^2)
26 matrix_power_multiplication = A_of_power_2*
      B_of_power_2;
27
28 // displaying (A^2)*(B^2)
29 disp('(A^2)*(B^2)', matrix_power_multiplication);
30
31 // comparing (A*B)^2 and (A^2)*(B^2)
32 if A_B_of_power_2 == matrix_power_multiplication
      then
33     disp('(A*B)^2 and (A^2)*(B^2) are equal');
34 else
35     disp('(A*B)^2 and (A^2)*(B^2) are unequal');
36 end
```

Scilab code Exa 1.5.8 Matrix transpose operations

```
1 // ex 1.5 -> example 8 -> matrix transpose
      operations
2
3 clc
4 clear
5
6 // defining matrix
7 A = [2,-4; 1,3];
```

```
8  B = [3,2; -1,5];
9
10 // displaying matrix
11 disp('A =', A);
12 disp('B =', B);
13
14 // calculating transpose of matrix
15 A_transpose = A';
16 B_transpose = B';
17
18 // calculating matrix multiplication
19 AB = A*B;
20
21 // B'*A'
22 disp('B_transpose*A_transpose =', B_transpose*
      A_transpose);
23
24 // (A*B)'
25 disp('(A*B)_transpose =', AB');
26
27 // check if B'*A' = (A*B)'
28 if B_transpose*A_transpose == AB' then
29     disp('B_transpose*A_transpose = (A*B)_transpose'
         );
30 else
31     disp('B_transpose*A_transpose != (A*B)_transpose
         ');
32 end
33
34 // A'*B'
35 disp('A_transpose*B_transpose =', A_transpose*
      B_transpose);
36
37 // check if A'*B' = (A*B)'
38 if A_transpose*B_transpose == AB' then
39     disp('A_transpose*B_transpose = (A*B)_transpose'
         );
40 else
```

```
41      disp('A_transpose*B_transpose != (A*B)_transpose
            ');
42  end
```

Chapter 2

Systems of Linear Equations

Scilab code Exa 2.1.1 Solving linear equations

```
1 // ex 2.1 -> example 1 -> Solving linear equations
2
3 clc
4 clear
5
6 // defining matrix
7 A = [4,-2,1,-3; 3,1,0,5];
8 B = [5;12];
9
10 // matrix representation of equations
11 disp('Linear equation can be represented as', A, '='
      , B);
12
13 // augmented matrix representation of equations
14 augmented_matrix = [A B];
15 disp('augmented matrix =', augmented_matrix);
16
17 // solving for w, x, y, z
18 variable_w_x_y_z = -linsolve(A, B); // linsolve is
      for solving linear equations
19
```

```
20 // displaying w,x,y,z values
21 disp('w =', variable_w_x_y_z(1), 'x =',
      variable_w_x_y_z(2), 'y =', variable_w_x_y_z(3),
      'z =', variable_w_x_y_z(4));
```

Scilab code Exa 2.1.2 Solving linear equations

```
1  // ex 2.1 -> example 2 -> solving linear equations
2
3  clc
4  clear
5
6  // defining matrix
7  matrix1 = [5,-5,-15; 4,-2,-6; 3,-6,-17];
8  equation_solution = [40;19;41];
9
10 // matrix representation of equations
11 disp('Linear equation can be represented as matrix1,
      ', matrix1, '=', equation_solution);
12
13 // two ways to solve problem
14 // 1. using linsolve
15 // 2. using Gaussian Elimination, row operations (
      book method)
16
17 // 1.
18
19 ordered_triplet = -linsolve(matrix1,
      equation_solution);
20 disp('1. Direct solution', 'Solutions to the linear
      equation is', ordered_triplet);
21
22 // 2.
23
24 // concatenate matrices to create an augmented
```

```
    matrix
25 disp('2. Using Gaussian Elimination', 'The augmented
     matrix is');
26 // create an augmented matrix by appending solution
    to the equation to original matrix
27 matrix1_augmented = cat(2,matrix1,equation_solution)
    ;
28 disp(matrix1_augmented);
29
30 // position(1,1) is the pivot entry and that row of
    pivot position is pivot row
31 // make it equal to 1
32 // row operation row1     1/5 *row1
33 disp('After row operation row1     1/5 *row1');
34 matrix1_augmented(1,:) = matrix1_augmented(1,:)/5
35 disp(matrix1_augmented);
36
37 // targeting entries - make entries below pivot
    entry equal to 0
38 // row operation row2     -4*row1 + row2
39 disp('After row operation row2     -4*row1 + row2');
40 matrix1_augmented(2,:) = -4*matrix1_augmented(1,:) +
     matrix1_augmented(2,:);
41 disp(matrix1_augmented);
42
43 // targeting entries - make entries below pivot
    entry equal to 0
44 // row operation row3     -3*row1 + row3
45 disp('After row operation row3     -3*row1 + row3');
46 matrix1_augmented(3,:) = -3*matrix1_augmented(1,:) +
     matrix1_augmented(3,:);
47 disp(matrix1_augmented);
48
49 // position(2,2) is the pivot entry and that row of
    pivot position is pivot row
50 // make it equal to 1
51 // row operation row2     1/2 *row2
52 disp('After row operation row2     1/2 *row2');
```

```
53 matrix1_augmented(2,:) = 1/2 *matrix1_augmented(2,:)
   ;
54 disp(matrix1_augmented);
55
56 // targeting entries - make entries below pivot
   entry equal to 0
57 // row operation row3     3*row2 + row3
58 disp('After row operation row3     3*row2 + row3');
59 matrix1_augmented(3,:) = 3*matrix1_augmented(2,:) +
   matrix1_augmented(3,:);
60 disp(matrix1_augmented);
61
62 // now position(3,3) is pivot entry
63 // it is already 1 and there is no entry below to
   target
64
65 // looking at the third equation
66 z = matrix1_augmented(3,4);
67 y = matrix1_augmented(2,4) - matrix1_augmented(2,3)*
   z;
68 x = matrix1_augmented(1,4) - y*matrix1_augmented
   (1,2) - z*matrix1_augmented(1,3);
69 disp('Therefore x =', x, 'y =', y, 'z =', z);
70
71 // displaying the ordered triplet
72 ordered_triplet = [x,y,z];
73 disp('Therefore the unique solution -  ordered
   triplet is', ordered_triplet);
```

Scilab code Exa 2.1.3 Solving linear equations

```
1 // ex 2.1 -> example 3 -> solving linear equations
2
3 clc
4 clear
```

```
5
6  // defining matrix
7  matrix1 = [3,1; -6,-2; 4,5];
8  equation_solution = [-5;10;8];
9
10 // matrix representation of equations
11 disp('Linear equation can be represented as matrix1,
       ', matrix1, '=', equation_solution);
12
13 // two ways to solve problem
14 // 1. using linsolve
15 // 2. using Gaussian Elimination, row operations (
       book method)
16
17 // 1.
18 solution_to_equation = -linsolve(matrix1,
       equation_solution);
19 disp('Solutions to the linear equation is',
       solution_to_equation);
20
21 // or
22
23 // 2.
24
25 // concatenate matrices to create an augmented
       matrix
26 disp('2. Using Gaussian Elimination', 'The augmented
        matrix is');
27 // create an augmented matrix by appending solution
       to the equation to original matrix
28 matrix1_augmented = cat(2,matrix1,equation_solution)
       ;
29 disp(matrix1_augmented);
30
31 // position(1,1) is the pivot entry and that row of
       pivot position is pivot row
32 // make it equal to 1
33 // row operation row1     1/3 *row1
```

```
34 disp('After row operation row1     1/3 *row1');
35 matrix1_augmented(1,:) = matrix1_augmented(1,:)/3
36 disp(matrix1_augmented);
37
38 // targeting entries - make entries below pivot
      entry equal to 0
39 // row operation row2     6*row1 + row2
40 disp('After row operation row2     6*row1 + row2');
41 matrix1_augmented(2,:) = 6*matrix1_augmented(1,:) +
      matrix1_augmented(2,:);
42 disp(matrix1_augmented);
43
44 // targeting entries - make entries below pivot
      entry equal to 0
45 // row operation row3     -4*row1 + row3
46 disp('After row operation row3     -4*row1 + row3');
47 matrix1_augmented(3,:) = -4*matrix1_augmented(1,:) +
       matrix1_augmented(3,:);
48 disp(matrix1_augmented);
49
50 // position(2,2) is the pivot entry and that row of
      pivot position is pivot row
51 // make it equal to 1
52 // row operation row2     row3
53 disp('After row operation row2     row3');
54 temp = matrix1_augmented(3,:); // temp is used for
      intermediate for swapping
55 matrix1_augmented(3,:) = matrix1_augmented(2,:);
56 matrix1_augmented(2,:) = temp;
57 disp(matrix1_augmented);
58
59 // position(2,2) is the pivot entry and that row of
      pivot position is pivot row
60 // make it equal to 1
61 // row operation row2     3/11 *row2
62 disp('After row operation row2     3/11 *row2');
63 matrix1_augmented(2,:) = 3/11 *matrix1_augmented
      (2,:);
```

```
64 disp(matrix1_augmented);
65
66 // there is no entry below to target
67
68 // looking at the second equation
69 y = matrix1_augmented(2,3);
70 x = matrix1_augmented(1,3) - y*matrix1_augmented
      (1,2);
71 disp('Therefore x =', x, 'y =', y);
72
73 // displaying the solution
74 solution_to_equation = [x,y];
75 disp('Therefore the unique solution is',
      solution_to_equation);
```

Scilab code Exa 2.1.4 Solving linear equations

```
1 // ex 2.1 -> example 4 -> solving linear equations
2
3 clc
4 clear
5 warning('off')
6
7 // defining matrix
8 matrix1 = [3,-6,0,3; -2,4,2,-1; 4,-8,6,7];
9 equation_solution = [9;-11;-5];
10
11 // matrix representation of equations
12 disp('Linear equation can be represented as matrix1,
      ', matrix1, '=', equation_solution);
13
14 // two ways to solve problem
15 // 1. using linsolve
16 // 2. using Gaussian Elimination, row operations (
      book method)
```

```
17
18 // 1.
19 solution_to_equation = -linsolve(matrix1,
      equation_solution); // linsolve is to solve a
      system of equations
20 if size(solution_to_equation) == 0 then
21     disp('The given system of equations have no
          solutions');
22 else
23     disp('Solutions to the linear equation is',
         solution_to_equation);
24 end
25
26 // or
27
28 // 2.
29
30 // concatenate matrices to create an augmented
      matrix
31 disp('2. Using Gaussian Elimination', 'The augmented
       matrix is');
32 // create an augmented matrix by appending solution
      to the equation to original matrix
33 matrix1_augmented = cat(2,matrix1,equation_solution)
      ;
34 disp(matrix1_augmented);
35
36 // position(1,1) is the pivot entry and that row of
      pivot position is pivot row
37 // make it equal to 1
38 // row operation row1     1/3 *row1
39 disp('After row operation row1     1/3 *row1');
40 matrix1_augmented(1,:) = matrix1_augmented(1,:)/3
41 disp(matrix1_augmented);
42
43 // targeting entries - make entries below pivot
      entry equal to 0
44 // row operation row2     2*row1 + row2
```

```
45 disp('After row operation row2     2*row1 + row2');
46 matrix1_augmented(2,:) = 2*matrix1_augmented(1,:) +
      matrix1_augmented(2,:);
47 disp(matrix1_augmented, '=');
48 equation_solution(2,:) = 2*equation_solution(1,:) +
      equation_solution(2,:);
49 disp(equation_solution);
50
51 // targeting entries - make entries below pivot
      entry equal to 0
52 // row operation row3     -4*row1 + row3
53 disp('After row operation row3     -4*row1 + row3');
54 matrix1_augmented(3,:) = -4*matrix1_augmented(1,:) +
       matrix1_augmented(3,:);
55 disp(matrix1_augmented);
56
57 // position(2,2) is the pivot entry and that row of
      pivot position is pivot row
58 // but since it is zero and there is no non-zero
      entry below it
59 // move to position(2,3) as the pivot entry and the
      same row as the pivot row
60 // make it equal to 1
61 // row operation row2     1/2 *row2
62 disp('After row operation row2     1/2 *row2');
63 matrix1_augmented(2,:) = matrix1_augmented(2,:)/2
64 disp(matrix1_augmented);
65
66 // targeting entries - make entries below pivot
      entry equal to 0
67 // row operation row3     -6*row2 + row3
68 disp('After row operation row3     -6*row2 + row3');
69 matrix1_augmented(3,:) = -6*matrix1_augmented(2,:) +
       matrix1_augmented(3,:);
70 disp(matrix1_augmented);
71
72 // position(3,4) is the pivot entry and that row of
      pivot position is pivot row
```

```
73 // but it is zero and there is no entry below it to
   switch with
74
75 // looking at the equations
76 disp('Since there is no row below the pivot row3 to
   switch with, end of augmentation bar is reached.'
   );
77 disp('From the last row, 0 can not be equal to -2');
78 disp('Therefore, equation have no solution');
```

Scilab code Exa 2.1.5 Solving linear equations

```
1 // ex 2.1 -> example 5 -> solving linear equations
2
3 clc
4 clear
5
6 // defining matrix
7 matrix1 = [3,1,7,2; 2,-4,14,-1; 5,11,-7,8;
   2,5,-4,-3];
8 equation_solution = [13;-10;59;39];
9
10 // matrix representation of equations
11 disp('Linear equation can be represented as matrix1,
   ', matrix1, '=', equation_solution);
12
13 // two ways to solve problem
14 // 1. using linsolve
15 // 2. using Gaussian Elimination, row operations (
   book method)
16
17 // 1.
18 solution_to_equation = -linsolve(matrix1,
   equation_solution);
19 if size(solution_to_equation) == 0 then
```

```
20      disp('The given system of equations have no
           solutions');
21  else
22     disp('One particular solution to the linear
          equation is', solution_to_equation);
23  end
24
25  // or
26
27  // 2.
28
29
30  // concatenate matrices to create an augmented
       matrix
31  disp('2. Using Gaussian Elimination', 'The augmented
        matrix is');
32  // create an augmented matrix by appending solution
       to the equation to original matrix
33  matrix1_augmented = cat(2,matrix1,equation_solution)
       ;
34  disp(matrix1_augmented);
35
36  // simplifying
37  simplified_matrix = [1,1/3,7/3,2/3,13/3;
       0,1,-2,1/2,4; 0,0,0,0,0; 0,0,0,-13/2,13];
38  disp('After simplifying the first two columns,
       augmented matrix becomes', simplified_matrix);
39
40  // row3     row4
41  temp = simplified_matrix(3,:);
42  simplified_matrix(3,:) = simplified_matrix(4,:);
43  simplified_matrix(4,:) = temp;
44  disp('row operation row3     row4 gives',
       simplified_matrix);
45
46  // Converting the pivot entry in the fourth column
       to 1
47  // row3     -2/13 * row3
```

```
48 simplified_matrix(3,:) = (-2/13)*simplified_matrix
   (3,:);
49 disp('row operation row3      -2/13 * row3 gives',
   simplified_matrix);
50
51 disp('Equation 3 gives x4 =  2  , as the values for
   x1,x2,x3 are not uniquely determined, there are
   infinitely many solutions.');
```

Scilab code Exa 2.1.6 Gaussian elimination

```
1 // ex 2.1 -> example 6 -> gaussian elimination
2
3 clc
4 clear
5
6 // matrix after gaussian elimination
7
8 matrix1=[1,-2,0,3,5,-1,1; 0,0,1,4,23,0,-9;
   0,0,0,0,0,1,16; 0,0,0,0,0,0,0];
9 disp('augmented matrix =', matrix1);
10
11 equation_solution=matrix1(:,7);
12 matrix1=matrix1(1:4,1:6);
13
14 disp('that means', matrix1, '=', equation_solution);
15
16 // one particular solution
17 sol = -linsolve(matrix1,equation_solution);
18 disp('one particular solution is', sol);
19 // Answer varies as there are 6 variables and 3
   equations. x2, x4 and x5 can be considered as any
    real value. This answer is one solution of many
   possible.
```

Scilab code Exa 2.1.7 Gaussian elimination

```
1 // ex 2.1 -> example 7 -> gaussian elimination
2
3 clc
4 clear
5
6 // matrix after gaussian elimination
7
8 matrix1=[1,4,-1,2,1,8; 0,1,3,-2,6,-11; 0,0,0,1,-3,9;
      0,0,0,0,0,0];
9 disp('augmented matrix =', matrix1);
10
11 equation_solution=matrix1(:,6);
12 matrix1=matrix1(1:4,1:5);
13
14 disp('that means', matrix1, '=', equation_solution);
15
16 // one particular solution
17 sol = -linsolve(matrix1,equation_solution);
18 disp('one particular solution is', sol);
19 // Answer varies as there are 5 variables and 3
     equations. x3 and x5 can be considered as any
     real value. This answer is one solution of many
     possible.
```

Scilab code Exa 2.1.8 Gaussian elimination

```
1 // ex 2.1 -> example 8 -> gaussian elimination
2
3 clc
4 clear
```

```
5
6  matrix1 = [4,-2,1; 1,1,1; 9,3,1];
7  equation_solution = [20; 5; 25];
8
9  // augmented matrix
10 matrix2 = cat(2, matrix1, equation_solution);
11 disp('The augmented matrix is', matrix2);
12
13 // row1     row2
14 temp = matrix2(1,:);
15 matrix2(1,:) = matrix2(2,:);
16 matrix2(2,:) = temp;
17 disp('row1     row2', matrix2);
18
19 // row3     row3 - row2
20 matrix2(3,:) = matrix2(3,:) - matrix2(2,:);
21 disp('row3     row3 - row2', matrix2);
22
23 // row2     row2 - 4*row1 and row3     row3 - 5*row1
24 matrix2(2,:) = matrix2(2,:) - 4*matrix2(1,:);
25 matrix2(3,:) = matrix2(3,:) - 5*matrix2(1,:);
26 disp('row2     row2 - 4*row1 and row3     row3 - 5*
      row1', matrix2);
27
28 // row2     (-1/6)*row2
29 matrix2(2,:) = (-1/6)*matrix2(2,:);
30 disp('row2     (-1/6)*row2', matrix2);
31
32 // row3     (-1/5)*row3
33 matrix2(3,:) = (-1/5)*matrix2(3,:);
34 disp('row3     (-1/5)*row3', matrix2);
35
36 // Final augmented matrix is wrongly given in the
      book
37 // The answer provided in the textbook is wrong
38
39 c=matrix2(3,4);
40 b=matrix2(2,4)-matrix2(2,3)*c;
```

```
41 a=matrix2(1,4)-matrix2(1,3)*c-matrix2(1,2)*b;
42 disp('(a,b,c) =', a,b,c);
43
44 x=poly(0,'x');
45 y=a*x^2 + b*x +c;
46 disp('quadratic equation is', y);
```

Scilab code Exa 2.1.9 Row operations effect on matrix multiplications

```
1  // row operations effect on matrix multiplications
2
3  clc
4  clear
5
6  // defining matrix
7  A = [1,-2,1; 3,4,2];
8  B = [3,7; 0,-1; 5,2];
9
10 disp('A =', A, 'B =', B);
11
12 // finding AB -> (multiplication of both matrix)
13 AB = A*B;
14
15 // performing row operation on AB -> R(AB)
16 // row2       2 *row1 + row2
17 AB(2,:) = -2*AB(1,:) + AB(2,:);
18 disp('R(AB)', AB);
19
20 // performaing row operation on A and multiplying
      with B -> R(A)B
21 A(2,:) = -2*A(1,:) + A(2,:);
22 AB = A*B;
23 disp('R(A)B', AB)
24
25 // new row operations
```

```
26 // R1: row1      row2
27 // R2: row1       3  *row2 + row1
28 // R3: row1      4*row1
29
30 // redefining A and B matrix to original -> given in
       question
31 A = [1,-2,1; 3,4,2];
32 B = [3,7; 0,-1; 5,2];
33
34 // performing row operation R3(R2(R1(AB))) on AB
35 AB = A*B; // finding AB -> (multiplication of both
      matrix)
36
37 // performing R1 on AB
38 temp = AB(1,:);
39 AB(1,:) = AB(2,:);
40 AB(2,:) = temp;
41
42 // performing R2 on R1(AB)
43 AB(1,:) = -3*AB(2,:) + AB(1,:);
44
45 // performing R3 on R2(R1(AB))
46 AB(1,:) = 4*AB(1,:);
47
48 disp('R3(R2(R1(AB)))', AB);
49
50
51 // redefining A and B matrix to original -> given in
       question
52 A = [1,-2,1; 3,4,2];
53 B = [3,7; 0,-1; 5,2];
54
55 // performing row operation R3(R2(R1(A))) on A and
      multiplying with B -> R3(R2(R1(A)))B
56
57 // performing R1 on A
58 temp = A(1,:);
59 A(1,:) = A(2,:);
```

```
60 A(2,:) = temp;
61
62 // performing R2 on R1(A)
63 A(1,:) = -3*A(2,:) + A(1,:);
64
65 // performing R3 on R2(R1(A))
66 A(1,:) = 4*A(1,:);
67
68 // multiplying R3(R2(R1(A))) with B
69 AB = A*B;
70
71 disp('R3(R2(R1(A)))B', AB);
```

Scilab code Exa 2.2.1 Gauss jordan elimination

```
1 // gauss jordan elimination
2
3 clc
4 clear
5
6 matrix1 = [2,1,3,0; 3,2,0,1; 2,0,12,-5];
7 equation_solution = [16;16;5];
8
9 disp(matrix1, '=', equation_solution);
10
11 // augmented matrix
12 matrix2=[matrix1 equation_solution];
13
14 // rref for gauss jordan elimination
15 matrix2 = rref(matrix2);
16
17 // final matrix
18 disp(' The corresponding system for this final
     matrix is', matrix2);
19
```

```
20 equation_solution=matrix2(:,5);
21 matrix2=matrix2(1:3,1:4);
22
23 // one particular solution
24 sol = -linsolve(matrix2,equation_solution);
25 disp('one particular solution is', sol);
```

Scilab code Exa 2.2.2 Reduced Row Echelon Form

```
1 // Reduced Row Echelon Form
2
3 clc
4 clear
5
6 A=[1,0,0,6; 0,1,0,-2; 0,0,1,3];
7 B=[1,0,2,0,-1; 0,1,3,0,4; 0,0,0,1,2; 0,0,0,0,0];
8 C=[1,4,0,-3,0; 0,0,1,2,0; 0,0,0,0,1];
9 disp('A=',A,'B=',B,'C=',C);
10
11 equation_solution_A=A(:,4);
12 A=A(1:3,1:3);
13
14 equation_solution_B=B(:,5);
15 B=B(1:4,1:4);
16
17 equation_solution_C=C(:,5);
18 C=C(1:3,1:4);
19
20 solA= -linsolve(A,equation_solution_A);
21 solB= -linsolve(B,equation_solution_B);
22 solC= -linsolve(C,equation_solution_C);
23
24 disp('Solution for A=', solA);
25 disp('Particular solution for B=', solB);
26 disp('Solution for C=', solC);
```

```
27 if isempty(solC) then
28     disp('No solution');
29 end
```

Scilab code Exa 2.2.3 Reduced Row Echelon Form

```
1  // Reduced Row Echelon Form
2
3  clc
4  clear
5
6  matrix1=[4,-8,-2; 3,-5,-2; 2,-8,1];
7  matrix2=[2,1,4; 3,2,5; 0,-1,1];
8  equation_solution1=[0;0;0];
9  equation_solution2=[0;0;0];
10
11 disp(matrix1,'=',equation_solution1, "and", matrix2,
       "=", equation_solution2);
12
13 // Applying Gauss-Jordan row reduction to the
       coefficient matrices for these systems
14 matrix1=rref(matrix1);
15 matrix2=rref(matrix2);
16 disp('Applying Gauss-Jordan row reduction', matrix1,
       "and", matrix2);
17
18 // matrix1 has a nontrivial solution because only 2
       of the 3 columns of the coefficient matrix are
       pivot columns
19 // matrix2 has only the trivial solution because all
       3 columns of the coefficient matrix are pivot
       columns
20 sol1=[-matrix1(1,3), -matrix1(2,3), 1];
21
22 disp('Solution for matrix1 is multiple of', sol1);
```

```
23 disp('Matrix2 has only the trivial solution');
```

Scilab code Exa 2.2.4 Reduced Row Echelon Form

```
1 // Reduced Row Echelon Form
2
3 clc
4 clear
5
6 matrix1 = [1,-3,2,-4,8,17; 3,-9,6,-12,24,49;
     -2,6,-5,11,-18,-40];
7 equation_solution=[0;0;0];
8 disp(matrix1, '=', equation_solution);
9
10 // augmented matrix
11 matrix1 = [matrix1 equation_solution];
12
13 // Reduced Row Echelon Form
14 matrix1 = rref(matrix1);
15 disp('Reduced Row Echelon Form=', matrix1);
16
17 disp('one particular solution is', -linsolve(matrix1
     , equation_solution));
18 disp('The complete solution set is {(3b     2d     4
     e, b, 3d     2e, d, e, 0)| b, d, e     R}');
```

Scilab code Exa 2.3.1 Rank of a matrix

```
1 // rank of a matrix
2
3 clc
4 clear
5
```

```
6  A=[3,1,0,1,-9; 0,-2,12,-8,-6; 2,-3,22,-14,-17];
7  B=[2,1,4; 3,2,5; 0,-1,1];
8
9  // reduced row echelon form
10
11 reducedA=rref(A);
12 reducedB=rref(B);
13
14 disp('Reduced row echelon form for A=', reducedA);
15 disp('Reduced row echelon form for B=', reducedB);
16
17 // rank of matrix
18 disp('rank of matrix A =', rank(A)); // since the
     reduced row echelon form of A has 2 nonzero rows
     (and hence 2 nonzero pivot entries)
19 disp('rank of matrix B =', rank(B)); // since the
     reduced row echelon form of B has 3 nonzero rows
     (and hence 3 nonzero pivot entries)
```

Scilab code Exa 2.3.2 Linear Combinations of Vectors

```
1  // Linear Combinations of Vectors
2
3  clc
4  clear
5
6  a1=[-4;1;2];
7  a2=[2;1;0];
8  a3=[6;-3;-4];
9  vector1 = [-18;15;16];
10
11 if vector1 == 2*a1 + 4*a2 -3*a3 then
12     disp('vector', vector1, 'can be written as 2*a1
         + 4*a2 -3*a3');
13 end
```

```
14
15 // for vector2
16 vector2=[16;-3;8];
17
18 // augmented matrix
19 matrix1 = [a1 a2 a3 vector2];
20 disp('augmented matrix =', matrix1);
21
22 // reduced row echelon form
23 reduced_matrix1=rref(matrix1);
24 disp('reduced row echelon form of matrix=',
     reduced_matrix1);
25 // the reduced row echelon form of matrix is wrong
     in book
26
27 disp('since the third row of matrix is 0, therefore,
     [16, 3 , 8] is not a linear combination of the
     vectors [ 4 , 1, 2], [2, 1, 0], and [6, 3 ,
     4 ]')
```

Scilab code Exa 2.3.3 Linear Combinations of Vectors

```
1 // Linear Combinations of Vectors
2
3 clc
4 clear
5
6 a1=[2;-3;1];
7 a2=[-4;6;-2];
8 vector1 = [14;-21;7];
9
10 // augmented matrix
11 matrix1 = [a1 a2 vector1];
12 disp('augmented matrix =', matrix1);
13
```

```
14 // reduced row echelon form
15 reduced_matrix1=rref(matrix1);
16 disp('reduced row echelon form of matrix=',
      reduced_matrix1);
17
18 disp('Because c2 is an independent variable, c2 can
      be any real value. c1 = 2c2 + 7. Hence, the
      number of solutions to the system is infinite.');
19
20 // The answer varies as there are infinite values
      possible for this answer.
21
22 disp('one particular solution is', -linsolve([a1 a2
      ], vector1));
```

Scilab code Exa 2.3.4 Linear Combinations of Vectors

```
1 // Linear Combinations of Vectors
2
3 clc
4 clear
5
6 A=[3,1,-2; 4,0,1; -2,4,-3];
7 vector1 = [5;17;-20];
8
9 // augmented matrix
10 matrix1 = [A' vector1];
11 disp('augmented matrix =', matrix1);
12
13 // reduced row echelon form
14 reduced_matrix1=rref(matrix1);
15 disp('reduced row echelon form of matrix=',
      reduced_matrix1);
16
17 // row space
```

```
18 if vector1' == reduced_matrix1(1,4)*A(1,:) +
     reduced_matrix1(2,4)*A(2,:) + reduced_matrix1
     (3,4)*A(3,:) then
19     disp(vector1, 'is the rowspace of matrix');
20 end
```

Scilab code Exa 2.3.5 Linear Combinations of Vectors

```
1  // Linear Combinations of Vectors
2
3  clc
4  clear
5
6  B=[2,-4; -1,2];
7  X=[3;5];
8
9  disp('B =', B);
10 disp('X =', X);
11
12 // augmented matrix
13 matrix1 = [B' X];
14 disp('augmented matrix =', matrix1);
15
16 // reduced row echelon form
17 reduced_matrix1=rref(matrix1);
18 disp('reduced row echelon form of matrix=',
     reduced_matrix1);
19 // the reduced row echelon form of matrix is wrong
     in book
20
21 disp('the corresponding linear system is
     inconsistent');
```

Scilab code Exa 2.3.6 Linear Combinations of Vectors

```
1 // Linear Combinations of Vectors
2
3 clc
4 clear
5
6 A=[5,10,12,33,19; 3,6,-4,-25,-11; 1,2,-2,-11,-5;
     2,4,-1,-10,-4];
7
8 // reduced row echelon form
9 B=rref(A);
10 disp('reduced row echelon form of matrix=', B);
11
12 x=[4,8,-30,-132,-64];
13
14 if x == -1*A(1,:) + 3*A(2,:) + 4*A(3,:) + -2*A(4,:)
     then
15     disp('x =     1[5, 10, 12, 33, 19] + 3[3, 6,
           4  ,  25  ,  11  ] + 4[1, 2,  2  ,  11  ,
           5  ]     2[2, 4,  1  ,  10  ,  4  ]');
16     disp('therefore, x is in the row space of A');
17 end
18
19 if x == 4*B(1,:) + -30*B(2,:) then
20     disp('x = 4[1, 2, 0,  3  ,  1  ]     30[0, 0, 1,
          4, 2]');
21     disp('therefore, x is in the row space of B')
22 end
```

Scilab code Exa 2.4.1 Inverse of matrix

```
1 // inverse of matrix
2
3 clc
```

```
clear

// defining matrix
A = [1,-4,1; 1,1,-2; -1,1,1];
B = [3,5,7; 1,2,3; 2,3,5];
I = [1,0,0; 0,1,0; 0,0,1]; // defining an identity
   matrix

disp('A =', A, 'B =', B, 'I =', I);

// finding AB and BA
AB = A*B;
BA = B*A;

// checking whether AB = BA
if AB == BA then
    disp('AB = BA');
end

// checking whether matrix A and B are inverse of
   each other
disp('AB =', AB);
if AB == I then
    disp('Matrix A and B are inverse of each other')
       ;
end

C = [2,1; 6,3];
disp('C =', C);

// Calculate determinant of C
det_C = round(det(C));

// if det(C) == 0 then inverse not possible
if det_C == 0 then
    disp('Matrix C inverse can not be found');
else
    disp('Inverse of matrix C is ', inv(C));
```

```
39 end
```

Scilab code Exa 2.4.2 Properties of inverse of matrix

```
1 // properties of inverse of matrix
2
3 clc
4 clear
5
6 // defining matrix
7 A = [1,-4,1; 1,1,-2; -1,1,1];
8 disp('A =', A);
9
10 // finding inverse of A
11 A_inv = inv(A); // inv command to find inverse of a
      square matrix
12 disp('(A)^-1', A_inv);
13
14 // finding (A)^-3
15 disp('(A)^-3 = (A^-1)^3');
16 A_inv_power_3 = A_inv^3;
17 disp('(A)^-3', A_inv_power_3);
```

Scilab code Exa 2.4.3 Inverse of matrix

```
1 // inverse of 2x2 matrix
2
3 clc
4 clear
5
6 // defining matrix A
7 A = [12,-4; 9,-3];
8 disp('A =', A);
```

```
9
10 // finding delta/determinant
11 // delta = A(1,1)*A(2,2) - A(1,2)*A(2,1);
12 // or
13 delta = det(A);
14 disp('   =', delta);
15
16 if delta == 0 then
17     disp('Inverse of matrix not possible');
18 end
19
20 // defining matrix M
21 M = [-5,2; 9,-4];
22 disp('M =', M);
23
24 // finding delta
25 // delta = M(1,1)*M(2,2) - M(1,2)*M(2,1);
26 // or
27 delta = det(M);
28 disp('   =', delta);
29
30 if delta == 0 then
31     disp('Inverse of matrix not possible');
32 else
33     disp('Inverse of matrix possible');
34 end
35
36 M_inv = inv(M);
37 disp('Inverse of M =', M_inv);
38
39 // verifying
40 I = [1,0; 0,1]; // defining a square matrix
41
42 if M*M_inv == I then
43     disp('Hence verified, inverse of M is correct');
44 end
```

Scilab code Exa 2.4.4 Inverse of matrix

```
// inverse of matrix

clc
clear

A=[2,-6,5; -4,12,-9; 2,-9,8];
disp('A =', A);

// adding identity matrix
B=[A eye(3,3)];
disp(B);

// row reduction
reducedB=rref(B);

disp('Row reduction yields', reducedB);

A_inverse=reducedB(1:3,4:6);
disp('A inverse =', A_inverse);
```

Scilab code Exa 2.4.5 Inverse of matrix

```
// inverse of matrix

clc
clear

A=[4,2,8,1; -2,0,-4,1; 1,4,2,0; 3,-1,6,-2];
disp('A =', A);

```

```
9  // adding identity matrix
10 B=[A eye(4,4)];
11 disp(B);
12
13 // row reduction
14 reducedB=rref(B);
15
16 disp('Row reduction yields', reducedB);
17
18 disp('Since the first four columns of reduced row
      echelon form matrix are not converted into the
      identity matrix, the original matrix A has no
      inverse');
```

Scilab code Exa 2.4.6 Solving equations using inverse of coefficient matrix

```
1  // solving equations using inverse of coefficient
      matrix
2
3  clc
4  clear
5
6  // defining matrix
7  A = [-7,5,3; 3,-2,-2; 3,-2,-1]; // coefficient
      matrix
8  B = [6; -3; 2];
9
10 // AX = B
11 // A -> coefficient matrix
12 // X -> variables matrix
13 // B -> equation individual solution in matrix form
14
15 // finding inverse of coefficient matrix
16 A_inv = inv(A);
17
```

```
18 // finding X -> solution of equations
19 // X = inverse of A * (B)
20 X = A_inv*B;
21
22 disp('X =', X, 'is the solution to the equations');
```

Chapter 3

Determinants and Eigenvalues

Scilab code Exa 3.1.1 Finding determinant

```
1 // finding determinant
2
3 clc
4 clear
5
6 // defining matrix
7 A = [4,-2,3; -1,5,0; 6,-1,-2];
8
9 // 1. using direct command
10 A_det = det(A);
11 disp('determinant of A using direct command, |A| =',
       A_det);
12
13 // 2. using basketweaving method
14 A_det = A(1,1)*A(2,2)*A(3,3) + A(1,2)*A(2,3)*A(3,1)
      + A(1,3)*A(2,1)*A(3,2) - A(1,3)*A(2,2)*A(3,1) - A
      (1,1)*A(2,3)*A(3,2) - A(1,2)*A(2,1)*A(3,3);
15 disp('determinant of A using basketweaving method, |
      A| =', A_det);
```

Scilab code Exa 3.1.2 Finding volume of parallelepiped

```
1 // finding volume of parallelepiped
2
3 clc
4 clear
5
6 // defining the sides of parallelepiped
7 X = [-2,1,3];
8 Y = [3,0,-2];
9 Z = [-1,3,7];
10
11 disp('sides of parallelepiped are',X,Y,Z);
12
13 // converting in form of matrix
14 dimension = [X;Y;Z];
15
16 // finding absolute value of determinant of
     dimension gives the volume
17 volume = abs(det(dimension));
18 disp('Volume of parallelepiped is', volume)
```

Scilab code Exa 3.1.3 Submatrix

```
1 // submatrix
2
3 clc
4 clear
5
6 // defining matrix
7 A =[5,-2,1; 0,4,-3; 2,-7,6];
8 B = [9,-1,4,7; -3,2,6,-2; -8,0,1,3; 4,7,-5,-1];
```

```
9
10 disp('A =', A, 'B =', B);
11
12 // finding submatrix
13 A_submatrix_1_3 = A(2:3, 1:2); //  delete all
      entries in the 1st row and all entries in the 3rd
       column
14 disp('The (1, 3) submatrix of A is', A_submatrix_1_3
      );
15
16 B_submatrix_3_4 = B([1 2 4], 1:3); // delete all
      entries in the 3rd row and all entries in the 4th
       column
17 disp('The (3, 4) submatrix of B is', B_submatrix_3_4
      );
18
19 // finding minors
20 // determinant of the submatrix
21 A_minor_1_3 = det(A_submatrix_1_3);
22 B_minor_3_4 = det(B_submatrix_3_4);
23
24 disp('The corresponding minor with (1, 3) submatrix
      of A is', A_minor_1_3);
25 disp('The corresponding minor with (3, 4) submatrix
      of B is', B_minor_3_4);
```

Scilab code Exa 3.1.4 Cofactor

```
1 // cofactor
2
3 clc
4 clear
5
6 // defining matrix
7 A =[5,-2,1; 0,4,-3; 2,-7,6];
```

```
8  B = [9,-1,4,7; -3,2,6,-2; -8,0,1,3; 4,7,-5,-1];
9
10 disp('A =', A, 'B =', B);
11
12 // finding submatrix
13 A_submatrix_1_3 = A(2:3, 1:2); //  delete all
       entries in the 1st row and all entries in the 3rd
        column
14 disp('The (1, 3) submatrix of A is', A_submatrix_1_3
       );
15
16 B_submatrix_3_4 = B([1 2 4], 1:3); // delete all
       entries in the 3rd row and all entries in the 4th
        column
17 disp('The (3, 4) submatrix of B is', B_submatrix_3_4
       );
18
19 // finding minors
20 // determinant of the submatrix
21 A_minor_1_3 = det(A_submatrix_1_3);
22 B_minor_3_4 = det(B_submatrix_3_4);
23
24 disp('The corresponding minor with (1, 3) submatrix
       of A is', A_minor_1_3);
25 disp('The corresponding minor with (3, 4) submatrix
       of B is', B_minor_3_4);
26
27 // finding cofactors
28 // cofactor = (-1)^index of element * minor
29 A_cofactor_1_3 = (-1)^(1+3)*det(A_minor_1_3);
30 B_cofactor_3_4 = (-1)^(3+4)*det(B_minor_3_4);
31
32 disp('The corresponding cofactor with (1, 3)
       submatrix of A is', A_cofactor_1_3);
33 disp('The corresponding cofactor with (3, 4)
       submatrix of B is', B_cofactor_3_4);
```

Scilab code Exa 3.1.5 Finding determinant

```
1 // finding determinant
2
3 clc
4 clear
5
6 // defining matrix
7 A =[5,-2,1; 0,4,-3; 2,-7,6];
8
9 disp('A =', A);
10
11 // finding submatrix
12 A_submatrix_3_1 = A(1:2, 2:3); //  delete all
      entries in the 3rd row and all entries in the 1st
       column
13 A_submatrix_3_2 = A(1:2, [1 3]); // delete all
      entries in the 3rd row and all entries in the 2nd
       column
14 A_submatrix_3_3 = A(1:2, 1:2); // delete all entries
       in the 3rd row and all entries in the 3rd column
15
16 // finding minors
17 // determinant of the submatrix
18 A_minor_3_1 = det(A_submatrix_3_1);
19 A_minor_3_2 = det(A_submatrix_3_2);
20 A_minor_3_3 = det(A_submatrix_3_3);
21
22 // finding cofactors
23 // cofactor = (-1)^index of element * minor
24 A_cofactor_3_1 = (-1)^(3+1)*det(A_minor_3_1);
25 A_cofactor_3_2 = (-1)^(3+2)*det(A_minor_3_2);
26 A_cofactor_3_3 = (-1)^(3+3)*det(A_minor_3_3);
27
```

```
28 disp('The corresponding cofactor with (3, 1)
     submatrix of A is', A_cofactor_3_1);
29 disp('The corresponding cofactor with (3, 2)
     submatrix of A is', A_cofactor_3_2);
30 disp('The corresponding cofactor with (3, 3)
     submatrix of A is', A_cofactor_3_3);
31
32 // finding determinant
33 // determinant = A(3,1)*cofactor of A(3,1) + A(3,2)*
     cofactor of A(3,2) + A(3,3)*cofactor of A(3,3)
34 determinant = A(3,1)*A_cofactor_3_1 + A(3,2)*
     A_cofactor_3_2 + A(3,3)*A_cofactor_3_3;
35 disp('Therefore determinant of A is', determinant);
```

Scilab code Exa 3.1.6 Finding determinant

```
1 // finding determinant
2
3 clc
4 clear
5
6 // defining matrix
7 A =[3,2,0,5; 4,1,3,-1; 2,-1,3,6; 5,0,2,-1];
8
9 disp('A =', A);
10
11 // perform cofactor expansion along last row
12 // finding submatrix
13 A_submatrix_4_1 = A(1:3, 2:4); //  delete all
     entries in the 4th row and all entries in the 1st
      column
14 A_submatrix_4_2 = A(1:3, [1 3 4]); // delete all
     entries in the 4th row and all entries in the 2nd
      column
15 A_submatrix_4_3 = A(1:3, [1 2 4]); // delete all
```

```
    entries in the 4th row and all entries in the 3rd
     column
16 A_submatrix_4_4 = A(1:3, 1:3); // delete all entries
     in the 4th row and all entries in the 4th column
17
18 // finding minors
19 // determinant of the submatrix
20 A_minor_4_1 = det(A_submatrix_4_1);
21 A_minor_4_2 = det(A_submatrix_4_2);
22 A_minor_4_3 = det(A_submatrix_4_3);
23 A_minor_4_4 = det(A_submatrix_4_4);
24
25 // finding cofactors
26 // cofactor = (-1)^index of element * minor
27 A_cofactor_4_1 = (-1)^(4+1)*det(A_minor_4_1);
28 A_cofactor_4_2 = (-1)^(4+2)*det(A_minor_4_2);
29 A_cofactor_4_3 = (-1)^(4+3)*det(A_minor_4_3);
30 A_cofactor_4_4 = (-1)^(4+4)*det(A_minor_4_4);
31
32 // The value of the cofactor varies as in the book,
    the sign of the term is taken out and used during
     final calculation of determinant.
33 // The values provided here are the actual cofactor
    and minor values calculated unlike the book where
     intermediate values from matrix is calculated
    and used in the last.
34
35 disp('The corresponding cofactor with (4, 1)
    submatrix of A is', A_cofactor_4_1);
36 disp('The corresponding cofactor with (4, 2)
    submatrix of A is', A_cofactor_4_2);
37 disp('The corresponding cofactor with (4, 3)
    submatrix of A is', A_cofactor_4_3);
38 disp('The corresponding cofactor with (4, 4)
    submatrix of A is', A_cofactor_4_4);
39
40 // finding determinant
41 // determinant = A(4,1)*cofactor of A(4,1) + A(4,2)*
```

```
    cofactor of A(4,2) + A(4,3)*cofactor of A(4,3) +
    A(4,4)*cofactor of A(4,4)
42 determinant = A(4,1)*A_cofactor_4_1 + A(4,2)*
    A_cofactor_4_2 + A(4,3)*A_cofactor_4_3 + A(4,4)*
    A_cofactor_4_4;
43 disp('Therefore determinant of A is', determinant);
```

Scilab code Exa 3.2.1 Determinant of upper triangular matrix

```
1 // determinant of upper triangular matrix
2
3 clc
4 clear
5
6 // defining matrix
7 A = [4,2,0,1; 0,3,9,6; 0,0,-1,5; 0,0,0,7];
8
9 disp('A =', A);
10 // to check if it is an upper triangular matrix
11 if A == triu(A) then
12     disp('Matrix A is upper triangular matrix');
13 end
14
15 // calculating determinant
16 determinant = prod(diag(A));
17 // diag is for extracting diagonal elements of
    matrix
18 // prod is to return product of all components of A
19 disp('Determinant of A is', determinant);
```

Scilab code Exa 3.2.2 Effect of row operations on determinant

```
1 // effect of row operations on determinant
```

```
2
3  clc
4  clear
5
6  // defining matrix A
7  A = [5,-2,1; 4,3,-1; 2,1,0];
8
9  disp('A =', A);
10 // calculating determinant of A
11 // 1. using direct command
12 // A_det = det(A);
13
14 // 2. using basketweaving method
15 A_det = A(1,1)*A(2,2)*A(3,3) + A(1,2)*A(2,3)*A(3,1)
      + A(1,3)*A(2,1)*A(3,2) - A(1,3)*A(2,2)*A(3,1) - A
      (1,1)*A(2,3)*A(3,2) - A(1,2)*A(2,1)*A(3,3);
16 disp('Determinant of A using basketweaving method is
      ', A_det);
17
18 // defining matrix B1 -> formed by row operation
      row3 <- -3*row3 on A
19 B1=A;
20 B1(3,:) = -3*A(3,:);
21 disp('B1 =', B1);
22
23 // finding determinant of B1
24 B1_det = det(B1);
25 disp('Determinant of B1 is', B1_det);
26 if B1_det == -3*A_det then
27     disp('Hence proved. |B1| = -3*|A|');
28 end
29
30 // defining matrix B2 -> formed by row operation
      row3 <- 2*row1 + row3 on A
31 B2=A;
32 B2(3,:) = 2*B2(1,:) + B2(3,:);
33 disp('B2 =', B2);
34
```

```
35 // finding determinant of B2
36 B2_det = round(det(B2)); // round for rounding off
      to nearest integer
37 disp('Determinant of B2 is', B2_det);
38 if B2_det == A_det then
39     disp('Hence proved. |B2| = |A|');
40 end
41
42 // defining matrix B3 -> formed by row operation
      row1     row2
43 B3=A;
44 temp = B3(1,:);
45 B3(1,:) = B3(2,:);
46 B3(2,:) = temp;
47 disp('B3 =', B3);
48
49 // finding determinant of B3
50 B3_det = round(det(B3)); // round for rounding off
      to nearest integer
51 disp('Determinant of B3 is', B3_det);
52 if B3_det == -A_det then
53     disp('Hence proved. |B1| = -|A|');
54 end
```

Scilab code Exa 3.2.3 Multiplication of scalar in matrix

```
1 // multiplication of scalar in matrix
2
3 clc
4 clear
5
6 // defining matrix A
7 A = [0,2,1;  3,-3,-2; 16,7,1];
8 disp('A =', A);
9
```

```
10 // finding determinant of A
11 A_det = round(det(A));
12 disp('Determinant of A is', A_det);
13
14 // defining matrix B
15 B = [0,-4,-2; -6,6,4; -32,-14,-2];
16 disp('B =', B);
17
18 // taking common out of matrix
19 common = gcd(B); // gcd will give greatest common
      divisor of all elements of matrix B
20
21 // matrix can be written as:-
22 B = common * B/common
23
24 // finding determinant of B
25 B_det = round(det(B));
26 disp('Determinant of B is', B_det);
27
28 // proving |cA| = c^n|A|
29 if common^3 == B_det/A_det then
30     disp('|cA| = c^n|A|. Hence proved');
31 end
```

Scilab code Exa 3.2.4 Determinant by row reduction

```
1 // determinant by row reduction
2
3 clc
4 clear
5
6 A = [0,-14,-8; 1,3,2; -2,0,6];
7
8 disp('A =', A);
9
```

```
10 // reduce A to upper triangular form
11 // row1     row2
12 B1=A;
13 temp = B1(1,:);
14 B1(1,:) = B1(2,:);
15 B1(2,:) = temp;
16 disp('B1 =', B1);
17 disp('|B1| =', det(B1), '|A| =', det(A));
18
19 // row3 <- 2*row1 + row3
20 B2=B1;
21 B2(3,:) = 2*B2(1,:) + B2(3,:);
22 disp('B2 =', B2);
23 disp('|B2| =', det(B2), '|A| =', det(A));
24
25 // row2 <- -1/14 * row2
26 B3=B2;
27 B3(2,:) = -1/14 * B3(2,:);
28 disp('B3 =', B3);
29 disp('|B3| =', det(B3), '|A| =', det(A));
30
31 // row3 <- -6*row2 + row3
32 B=B3;
33 B(3,:) = -6*B(2,:) + B(3,:);
34 disp('B =', B);
35 disp('|B| =', det(B), '|A| =', det(A));
36
37 if B == triu(B) then
38     disp('Matrix B is in upper triangular form');
39 end
```

Scilab code Exa 3.2.5 Determinant by row reduction

```
1 // determinant by row reduction
2
```

```
clc
clear

A = [0,-14,-8; 1,3,2; -2,0,6];

disp('A =', A);

// reduce A to upper triangular form
// row1     row2
B1=A;
temp = B1(1,:);
B1(1,:) = B1(2,:);
B1(2,:) = temp;
disp('B1 =', B1);
disp('|B1| =', det(B1), '|A| =', det(A));
P = det(B1)/det(A);
disp('P =', P);

// row3 <- 2*row1 + row3
B2=B1;
B2(3,:) = 2*B2(1,:) + B2(3,:);
disp('B2 =', B2);
disp('|B2| =', det(B2), '|A| =', det(A));
P = det(B2)/det(A);
disp('P =', P);

// row2 <- -1/14 * row2
B3=B2;
B3(2,:) = -1/14 * B3(2,:);
disp('B3 =', B3);
disp('|B3| =', det(B3), '|A| =', det(A));
P = det(B3)/det(A);
disp('P =', P);

// row3 <- -6*row2 + row3
B=B3;
B(3,:) = -6*B(2,:) + B(3,:);
disp('B =', B);
```

```
41 disp('|B| =', det(B), '|A| =', det(A));
42 P = det(B)/det(A);
43 disp('P =', P);
44
45 if B == triu(B) then
46     disp('Matrix B is in upper triangular form');
47 end
```

Scilab code Exa 3.2.6 Determinant criterion for matrix singularity

```
1  // determinant criterion for matrix singularity
2
3  clc
4  clear
5
6  // defining matrix
7  A = [1,6; -3,5];
8
9  // corollary: Let A be an n    n matrix. Then rank(A
       ) = n if and only if |A| ~= 0
10 if issquare(A) & det(A) ~= 0 then
11     [row, col] = size(A);
12     disp('Matrix rank is', row);
13 end
14
15 // equ1 for equation1
16 equ1 = [20; 9]; // RHS
17 A_sol = -linsolve(A, equ1);
18 disp('Therefore, the solution is', A_sol);
19
20 // defining matrix
21 B = [1,5,1; 2,1,-7; -1,2,6];
22
23 // corollary: Let A be an n    n matrix. Then rank(A
       ) = n if and only if |A| ~= 0
```

```
24 if issquare(B) & round(det(B)) ~= 0 then
25     [row, col] = size(B);
26     disp('Matrix rank is', row);
27 else
28     [row, col] = size(B);
29     printf('\nSince |B| = 0, rank < %d\n', row);
30 end
31
32 // equ2 for equation2
33 equ2 = [0; 0; 0]; // RHS
34 B_sol = -linsolve(B, equ2);
35 disp('Therefore, the solution is', B_sol); //
     linsolve finds any solution to the equation. here
     it is a trivial solution
```

Scilab code Exa 3.3.1 Matrix product

```
1 // matrix product
2
3 clc
4 clear
5
6 // defining matrix
7 A = [3,2,1; 5,0,-2; -3,1,4];
8 B = [1,-1,0; 4,2,-1; -2,0,3];
9 disp('A =', A, 'B =', B);
10
11 // calculating multiplication of matrix
12 AB = A*B;
13 disp("AB =", AB);
14
15 // finding determinant
16 A_det = round(det(A)); // rounding determinant to
     nearest integer
17 B_det = round(det(B)); // rounding determinant to
```

```
    nearest integer
18 AB_det = round(det(AB)); // rounding determinant to
    nearest integer
19
20 // displaying determinant
21 disp('|A| =', A_det);
22 disp('|B| =', B_det);
23 disp('|AB| =', AB_det);
24
25 // checking if |AB| = |A|*|B|}
26 if AB_det == A_det*B_det then
27     disp('|AB| = |A|*|B|');
28 end
```

Scilab code Exa 3.3.2 Determinant of transpose

```
1 // determinant of transpose
2
3 clc
4 clear
5
6 // defining matrix
7 A = [-1,4,1; 2,0,3; -1,-1,2];
8 disp('A =', A);
9
10 // calculating transpose
11 A_transpose = A';
12 disp('A =', A_transpose);
13
14 // calculating determinant
15 A_det = det(A);
16 A_transpose_det = det(A_transpose);
17
18 disp('|A| =', A_det, '|transpose(A)| =',
    A_transpose_det);
```

```
19
20 // comparing |A| and |A'|
21 if A_det == A_transpose_det then
22     disp('|A| = |transpose(A)|');
23 end
```

Scilab code Exa 3.3.3 Row operation effect

```
1 // row operation effect
2
3 clc
4 clear
5
6 // defining matrix
7 A = [2,5,1; 1,2,3; -3,1,-1];
8 disp('A =', A);
9
10 // performing column operation col2 <- -3*col1 +
   col2
11 B=A;
12 B(:,2) = -3*B(:,1) + B(:,2);
13 disp('B =', B);
14
15 // calculating determinant
16 disp('|A| =', det(A), '|B| =', det(B));
17
18 // comparing |A| and |B|
19 // this type column does not effect determinant
20 if det(A) == det(B) then
21     disp('This type (II) column operation has no
       effect on the determinant');
22 end
```

Scilab code Exa 3.3.4 Determinant of matrix

```
1 // determinant of matrix
2
3 clc
4 clear
5
6 A=[5,0,1,-2; 2,2,3,1; -1,3,2,5; 6,0,1,1];
7
8 // finding matrix of cofactors
9 cofactor = -coff(A)';
10 cofactor = horner(cofactor, 0);
11
12 disp('matrix of cofactors =', cofactor);
13
14 // cofactor expansion across row2
15 cofactor_expansion_row2 = A(2,1)*cofactor(2,1) + A
   (2,2)*cofactor(2,2) + A(2,3)*cofactor(2,3) + A
   (2,4)*cofactor(2,4);
16 disp('|A| from cofactor expansion across row2= ',
   cofactor_expansion_row2);
17
18 // cofactor expansion across column2
19 cofactor_expansion_column2 = A(1,2)*cofactor(1,2) +
   A(2,2)*cofactor(2,2) + A(3,2)*cofactor(3,2) + A
   (4,2)*cofactor(4,2);
20 disp('|A| from cofactor expansion across column2= ',
    cofactor_expansion_column2);
21
22 // cofactor expansion across column4
23 cofactor_expansion_column4 = A(1,4)*cofactor(1,4) +
   A(2,4)*cofactor(2,4) + A(3,4)*cofactor(3,4) + A
   (4,4)*cofactor(4,4);
24 disp('|A| from cofactor expansion across column4= ',
    cofactor_expansion_column4);
```

Scilab code Exa 3.3.5 Cramers rule

```
1  // cramers rule
2
3  clc
4  clear
5
6  // defining matrix
7  A = [5,-3,-10; 2,2,-3; -3,-1,5];
8  B = [-9; 4; -1];
9
10 disp('A =', A, 'B =', B);
11
12 // finding determinant of A
13 A_det = det(A);
14
15 // calculating A1, A2 and AS3
16 A1 = A;
17 A1(:,1) = B;
18 disp('A1 =', A1);
19 A1_det = det(A1);
20
21 A2 = A;
22 A2(:,2) = B;
23 disp('A2 =', A2);
24 A2_det = det(A2);
25
26 A3 = A;
27 A3(:,3) = B;
28 disp('A3 =', A3);
29 A3_det = det(A3);
30
31 // finding solutions x1, x2 and x3
32 x1 = A1_det/A_det;
```

```
33 x2 = A2_det/A_det;
34 x3 = A3_det/A_det;
35
36 // displaying the solution
37 disp('(x1, x2, x3) =', x1, x2, x3);
```

Scilab code Exa 3.4.1 Eigenvalue

```
1 // eigenvalue
2
3 clc
4 clear
5
6 // defining matrix
7 A = [-4,8,-12; 6,-6,12; 6,-8,14];
8 disp('A =', A);
9
10 // finding eigenvalue
11 e_value = round(spec(A)); // spec for calculating
      eigen value of matrix
12
13 disp('The eigenvalue of the matrix is', e_value);
14 disp('Therefore 2 is an eigenvalue');
```

Scilab code Exa 3.4.2 Characteristic polynomial of matrix

```
1 // characteristic polynomial of matrix
2
3 clc
4 clear
5
6 // defining matrix
7 A = [12,-51; 2,-11];
```

```
8  disp('A =', A);
9
10 // finding characteristic polynomial
11 char_poly = poly(A, "x"); // poly for making a
      polynomial from matrix A
12 disp('The characteristic polynomial is', char_poly);
13
14 // finding eigenvalue
15 // finding roots of this polynomial will give
      eigenvalue
16 roots_poly = roots(char_poly); // roots for finding
      roots of a polynomial
17 disp('The eigen values are');
18 for i = 1:length(roots_poly)
19     disp(roots_poly(i));
20 end
21
22 // finding eigenspace for root 1
23 disp('For ',roots_poly(1));
24 equ_1 = roots_poly(1)*eye(2,2) - A; // 6*I - A //
      eye(2,2) to create (2,2) identity matrix
25
26 //  convert to row echelon form
27 equ_1_row = rref(equ_1); // rref to convert to row
      echelon form
28 disp('Row echelon form is ', equ_1_row);
29
30 disp('The eigenspace is');
31 sol1 = [-equ_1_row(1,2), equ_1_row(1,1)];
32 disp(sol1);
33
34 // finding eigenspace for root 2
35 disp('For ',roots_poly(2));
36 equ_2 = roots_poly(2)*eye(2,2) - A; // -5*I - A
37
38 //  convert to row echelon form
39 equ_2_row = rref(equ_2); // rref to convert to row
      echelon form
```

```
40 disp('Row echelon form', equ_2_row);
41
42 disp('The eigenspace is');
43 sol2 = [-equ_2_row(1,2), equ_2_row(1,1)];
44 disp(sol2);
```

Scilab code Exa 3.4.3 Characteristic polynomial of matrix

```
1  // characteristic polynomial of matrix
2
3  clc
4  clear
5
6  // defining matrix
7  B = [7,1,-1; -11,-3,2; 18,2,-4];
8  disp('B =', B);
9
10 // finding characteristic polynomial
11 char_poly = round(poly(B, "x")); // poly for making
      a polynomial from matrix B
12 disp('The characteristic polynomial is', char_poly);
13
14 // finding eigenvalue
15 // finding roots of this polynomial will give
      eigenvalue
16 roots_poly = round(roots(char_poly)); // roots for
      finding roots of a polynomial
17 roots_poly = unique(roots_poly); // ignoring
      repeated roots
18 disp('The eigen values are');
19 for i = 1:length(roots_poly)
20     disp(roots_poly(i));
21 end
22
23 // finding eigenspace for root 1
```

```
24 disp('for ',roots_poly(1));
25 equ_1 = roots_poly(1)*eye(3,3) - B; // -2*I - B //
     eye(3,3) to create (3,3) identity matrix
26
27 //  convert to row echelon form
28 equ_1_row = rref(equ_1); // rref to convert to row
     echelon form
29 disp('row echelon form is ', equ_1_row);
30
31 disp('The eigenspace is');
32 sol1 = [equ_1_row(1,3), equ_1_row(2,3), equ_1_row
     (2,2)];
33 disp(sol1);
34
35 // finding eigenspace for root 2
36 disp('for ',roots_poly(2));
37 equ_2 = roots_poly(2)*eye(3,3) - B; // -5*I - B
38
39 //  convert to row echelon form
40 equ_2_row = rref(equ_2); // rref to convert to row
     echelon form
41 disp('row echelon form', equ_2_row);
42
43 disp('The eigenspace is');
44 sol2 = [equ_2_row(2,3), equ_2_row(1,3), equ_2_row
     (2,2)];
45 disp(sol2);
```

Scilab code Exa 3.4.4 Characteristic polynomial of matrix

```
1 // characteristic polynomial of matrix
2
3 clc
4 clear
5
```

```
// defining matrix
B = [-4,8,-12; 6,-6,12; 6,-8,14];
disp('B =', B);

// finding characteristic polynomial
char_poly = round(poly(B, "x")); // poly for making a polynomial from matrix B
disp('The characteristic polynomial is', char_poly);

// finding eigenvalue
// finding roots of this polynomial will give eigenvalue
roots_poly = round(roots(char_poly)); // roots for finding roots of a polynomial
roots_poly = unique(roots_poly); // ignoring repeated roots
disp('The eigen values are');
for i = 1:length(roots_poly)
    disp(roots_poly(i));
end

// finding eigenspace for root 1
disp('for ',roots_poly(1));
equ_1 = roots_poly(1)*eye(3,3) - B; // -2*I - B // eye(3,3) to create (3,3) identity matrix

//  convert to row echelon form
equ_1_row = rref(equ_1); // rref to convert to row echelon form
disp('row echelon form is ', equ_1_row);

disp('The eigenspace is');
sol1 = [equ_1_row(1,3), equ_1_row(2,3), equ_1_row(2,2)];
disp(sol1);

// finding eigenspace for root 2
disp('for ',roots_poly(2));
```

```
37 equ_2 = roots_poly(2)*eye(3,3) - B; // -5*I - B
38
39 //  convert to row echelon form
40 equ_2_row = rref(equ_2); // rref to convert to row
     echelon form
41 disp('row echelon form', equ_2_row);
42
43 disp('The eigenspace is');
44 sol2 = [equ_2_row(2,3), equ_2_row(1,3), equ_2_row
     (2,2)];
45 disp(sol2);
```

Scilab code Exa 3.4.5 Diagonalization

```
1 // Diagonalization
2
3 clc
4 clear
5
6 A=[-4,8,-12; 6,-6,12; 6,-8,14];
7
8 // eigen values
9 eigen_value = unique(round(spec(A)));
10 disp('Eigen values are ', eigen_value);
11
12 // finding eigen vector
13 [eigen_vector eigen_value] = spec(A);
14
15 disp('corresponding eigen vectors are', eigen_vector
     ); // book values differe since it has multiplied
      by a common number
16
17 P=[eigen_vector(:,2) eigen_vector(:,3) eigen_vector
     (:,1)]
18 P_inv = inv(P);
```

```
19
20 // calculating diagonal matrix D
21 D=round(P_inv*A*P);
22 disp('Diagonal matrix is', D);
```

Scilab code Exa 3.4.6 Diagonalization

```
1 // Diagonalization
2
3 clc
4 clear
5
6 A=[-4,7,1,4; 6,-16,-3,-9; 12,-27,-4,-15;
     -18,43,7,24];
7 disp('A=', A);
8
9 // eigen values
10 eigen_value = unique(round(spec(A)));
11 disp('Eigen values are ', eigen_value);
12
13 // finding eigen vector
14 [eigen_vector eigen_value] = spec(A);
15
16 disp('corresponding eigen vectors are', eigen_vector
     ); // book values differ since it has multiplied
     by a common number
17
18 // P matrix is arranged in this manner so as to
     match with the book to not confuse the reader.
19 // Even if the same order is taken without re-
     arranging, the answer will be the same.
20
21 P=[eigen_vector(:,4) eigen_vector(:,2) eigen_vector
     (:,1) eigen_vector(:,3)]
22 disp('P =', P);
```

```
23 P_inv = inv(P);
24
25 // calculating diagonal matrix D
26 D=round(P_inv*A*P);
27 disp('Diagonal matrix is', D);
```

Scilab code Exa 3.4.7 Diagonalization

```
1 // Diagonalization
2
3 clc
4 clear
5
6 B = [7,1,-1; -11,-3,2; 18,2,-4];
7 disp('B=', B);
8
9 // eigen values
10 eigen_value = unique(round(spec(B)));
11 disp('Eigen values are ', eigen_value);
12
13 // finding eigen vector
14 [eigen_vector eigen_value] = spec(B);
15
16 disp('corresponding eigen vectors are', round(real(
     eigen_vector))); // book values differe since it
     has multiplied by a common number
17
18 disp('Since the method yields only two fundamental
     eigenvectors for this 3    3 matrix, B cannot be
     diagonalized.');
```

Scilab code Exa 3.4.9 Algebraic multiplicity

```
1  // algebraic multiplicity
2
3  clc
4  clear
5
6  A=[-4,7,1,4; 6,-16,-3,-9; 12,-27,-4,-15;
       -18,43,7,24];
7  disp('A=', A);
8
9  characteristic_equation=poly(A, 'x');
10
11 characteristic_equation = round(
       characteristic_equation);
12 disp('characteristic equation is',
       characteristic_equation);
13
14 // eigen values
15 eigen_value = unique(round(spec(A)));
16 disp('Eigen values are ', eigen_value);
17
18 disp(characteristic_equation, '= x*(x-2)*(x+1)^2');
19
20 disp('The algebraic multiplicity of 1 = 1 is 2'
       );
21 disp('The algebraic multiplicity of 2 = 0 is 1');
22 disp('The algebraic multiplicity of 3 = 2 is 1');
```

Scilab code Exa 3.4.10 Algebraic multiplicity

```
1  // algebraic multiplicity
2
3  clc
4  clear
5
6  // characteristic polynomial of matrix
```

```
7
8  // defining matrix
9  B = [7,1,-1; -11,-3,2; 18,2,-4];
10 disp('B =', B);
11
12 characteristic_equation=poly(B, 'x');
13
14 characteristic_equation = round(
       characteristic_equation);
15 disp('characteristic equation is',
       characteristic_equation);
16
17 // eigen values
18 eigen_value = unique(round(spec(B)));
19 disp('Eigen values are ', eigen_value);
20
21 disp(characteristic_equation, '= (x - 4)(x + 2)^2');
22
23 disp('The algebraic multiplicity of 1 = 2 is 2'
       );
```

Scilab code Exa 3.4.11 Diagonalization

```
1  // Diagonalization
2
3  clc
4  clear
5
6  // characteristic polynomial of matrix
7
8  // defining matrix
9  A = [-3,-1,-2; -2,16,-18; 2,9,-7];
10 disp('A =', A);
11
12 characteristic_equation=poly(A, 'x');
```

```
13
14 characteristic_equation = round(
      characteristic_equation);
15 disp('characteristic equation is',
      characteristic_equation);
16
17 // eigen values
18 eigen_value = unique(round(spec(A)));
19 disp('Eigen values are ', eigen_value);
20
21 // considering only real eigen value
22 // eigen_value = eigen_value(eigen_value == real(
      eigen_value))
23 disp(eigen_value(1), 'is the only real eigen value')
      ;
24 disp('The diagonalization method can produce only
      one fundamental eigenvector for    . Therefore, A
      cannot be diagonalized');
```

Scilab code Exa 3.4.12 Diagonalization

```
1 // Diagonalization
2
3 clc
4 clear
5
6 A=[-4,7,1,4; 6,-16,-3,-9; 12,-27,-4,-15;
      -18,43,7,24];
7 disp('A=', A);
8
9 // eigen values
10 eigen_value = unique(round(spec(A)));
11 disp('Eigen values are ', eigen_value);
12
13 // finding eigen vector
```

```
14 [eigen_vector eigen_value] = spec(A);
15
16 disp('corresponding eigen vectors are', eigen_vector
   ); // book values differ since it has multiplied
   by a common number
17
18 // P is a matrix of eigenvectors
19 // In example 3.4.6 same matrix is there
20 // The matrix varies as there can be many eigen
   vectors possible and applying the formula gives
   this precise value
21 // P matrix is arranged in this manner so as to
   match with the book to not confuse the reader.
22 // Even if the same order is taken without re-
   arranging, the answer will be the same.
23
24 P=[eigen_vector(:,4) eigen_vector(:,2) eigen_vector
   (:,1) eigen_vector(:,3)]
25 disp('P =', P);
26 P_inv = inv(P);
27
28 // calculating diagonal matrix D
29 D=round(P_inv*A*P);
30 disp('Diagonal matrix is', D);
31
32 A_pow_11 = P*(D^11)*P_inv;
33 disp('A^11 =', A_pow_11);
```

Scilab code Exa 3.4.13 Rounding off eigen value

```
1 // rounding off eigen value
2
3 clc
4 clear
5
```

```
A=[0,2; 1,0];
disp('A=', A);

// characteristic equation
characteristic_equation = round(poly(A, 'x'));
disp('characteristic_equation =',
   characteristic_equation);

// eigen values
eigen_value = (spec(A));
disp('Eigen values are ', eigen_value);

// finding eigen vector
[eigen_vector eigen_value] = spec(A);

B = [1.414*eye(2,2)-A [0;0]];

// row reducing
B = rref(B);

disp('X = [ 2 , 1] is an eigenvector for A
   corresponding to 1 = 2 ');
```

Chapter 4

Finite Dimensional Vector Spaces

Scilab code Exa 4.4.4 Linear Dependence and Independence With One and Two Element S

```
1 // Linear Dependence and Independence With One- and
     Two-Element Sets
2
3 clc
4 clear
5
6 s1=[3,-1,4];
7 s2=[0,0,0,0];
8
9 disp('S1 =', s1);
10 if s1 == 0 then
11     disp('S1 is a linearly dependent subset of R3');
12 else
13     disp('S1 contains a single nonzero vector, S1 is
           a linearly independent subset of R3');
14 end
15
16 disp('S2 =', s2);
17 if s2 == 0 then
```

```
18      disp('S2 is a linearly dependent subset of R4');
19  else
20      disp('S2 contains a single nonzero vector, S2 is
            a linearly independent subset of R4');
21  end
```

Scilab code Exa 4.4.6 Using Row Reduction to Test for Linear Independence

```
1  // Using Row Reduction to Test for Linear
      Independence
2
3  clc
4  clear
5
6  disp('subset S = {[1,  1  , 0, 2], [0,  2  , 1, 0],
      [2, 0,  1  , 1]} of R4.');
7  a=[1,-1,0,2];
8  b=[0,-2,1,0];
9  c=[2,0,-1,1];
10 equation_solution = [0;0;0;0];
11
12 // rewriting in matrix form
13 matrix1 = [a;b;c]';
14
15 // augmented matrix
16 matrix1 = [matrix1 equation_solution];
17 disp('augmented matrix =', matrix1);
18
19 // row reduction
20 matrix1 = rref(matrix1);
21 disp('after row reduction', matrix1);
22
23 disp('this system has only the trivial solution a =
      b = c = 0. therefore, system is linearly
      independent');
```

Scilab code Exa 4.4.7 Using Row Reduction to Test for Linear Independence

```
1 // Using Row Reduction to Test for Linear
     Independence
2
3 clc
4 clear
5
6 disp('subset S = {[3,1,-1], [-5,-2,2], [2,2,-1]} of
     R3.');
7 a=[3,1,-1];
8 b=[-5,-2,2];
9 c=[2,2,-1];
10 equation_solution = [0;0;0];
11
12 // rewriting in matrix form
13 matrix1 = [a;b;c]';
14
15 // augmented matrix
16 matrix1 = [matrix1 equation_solution];
17 disp('augmented matrix =', matrix1);
18
19 // row reduction
20 matrix1 = rref(matrix1);
21 disp('after row reduction', matrix1);
22
23 disp('since there is a pivot for every column,
     therefore, system is linearly independent');
```

Scilab code Exa 4.4.8 Using Row Reduction to Test for Linear Independence

```
// Using Row Reduction to Test for Linear
   Independence

clc
clear

disp('subset S = {[1,0,2], [-1,-5,-12], [5,10,30]},
   [-3,0,-11]}, [6,-25,-18]} of R3.');
a=[1,0,2];
b=[-1,-5,-12];
c=[5,10,30];
d=[-3,0,-11];
e=[6,-25,-18];
equation_solution = [0;0;0];

// rewriting in matrix form
matrix1 = [a;b;c;d;e]';

// augmented matrix
matrix1 = [matrix1 equation_solution];
disp('augmented matrix =', matrix1);

// row reduction
matrix1 = rref(matrix1);
disp('after row reduction', matrix1);

disp('since there is no pivot in column 3 and 5, the
    set is linearly dependent');
```

Scilab code Exa 4.4.9 Using Row Reduction to Test for Linear Independence

```
// Using Row Reduction to Test for Linear
   Independence

clc
```

```
4  clear
5
6  a=[2,3; -1,4];
7  b=[-1,0; 1,1];
8  c=[6,-1; 3,2];
9  d=[-11,3; -2,2];
10 equation_solution = [0;0;0;0];
11
12 // reshaping matrix to vectors
13 a=matrix(a',1,4);
14 b=matrix(b',1,4);
15 c=matrix(c',1,4);
16 d=matrix(d',1,4);
17
18 // rewriting in matrix form
19 matrix1 = [a;b;c;d]';
20
21 // augmented matrix
22 matrix1 = [matrix1 equation_solution];
23 disp('augmented matrix =', matrix1);
24
25 // row reduction
26 matrix1 = rref(matrix1);
27 disp('after row reduction', matrix1);
28
29 disp('since there is no pivot in column 4, the set
     is linearly dependent');
```

Scilab code Exa 4.5.15 Diagonalization and Bases

```
1 // Diagonalization and Bases
2
3 clc
4 clear
5
```

```
6  A=[-2,12,-4; -2,8,-2; -3,9,-1];
7
8  // eigen values
9  eigen_value = (spec(A));
10 disp('Eigen values are ', eigen_value);
11
12 // finding eigen vector
13 [eigen_vector eigen_value] = spec(A);
14
15 disp('corresponding eigen vectors are', eigen_vector
      ); // book values differe since it has multiplied
       by a common number
16
17 P=[eigen_vector(:,1) eigen_vector(:,2) eigen_vector
      (:,3)]
18 disp('P =', P);
19
20 eigen_vector1=eigen_vector(:,1);
21 eigen_vector2=eigen_vector(:,2);
22 eigen_vector3=eigen_vector(:,3);
23
24 disp('eigen_vector1 =',eigen_vector1);
25 disp('eigen_vector2 =',eigen_vector2);
26 disp('eigen_vector3 =',eigen_vector3);
27 disp('the set of fundamental eigenvectors is a basis
       for R3.');
```

Scilab code Exa 4.6.1 Using the Simplified Span Method to Construct a Basis

```
1  // CONSTRUCTING SPECIAL BASES - Using the Simplified
       Span Method to Construct a Basis
2
3  clc
4  clear
5
```

```
6 A = [2,-2,3,5,5; -1,1,4,14,-8; 4,-4,-2,-14,18;
     3,-3,-1,-9,13];
7
8 // row reduce
9 reducedA=rref(A);
10
11 disp('row reduced A =', reducedA);
12
13 disp('Therefore, the desired basis for V is the set
     B =', [reducedA(1,:)], [reducedA(2,:)]);
14
15 dim = rank(reducedA);
16
17 disp('dim(V) =', dim);
```

Scilab code Exa 4.6.2 Using the Simplified Span Method to Construct a Basis

```
1 // CONSTRUCTING SPECIAL BASES - Using the Simplified
     Span Method to Construct a Basis
2
3 clc
4 clear
5
6 A = [0,0,0,1,3,-1; 0,0,3,0,6,15; 0,0,6,-1,9,31;
     2,0,-7,0,-22,-47];
7
8 // row reduce
9 reducedA=rref(A);
10
11 disp('row reduced A =', reducedA);
12
13 disp('Therefore, the desired basis for V is the set
     B =', [reducedA(1,:)], [reducedA(2,:)], [reducedA
     (3,:)]);
14
```

```
15 dim = rank(reducedA);
16
17 disp('dim(V) =', dim);
```

Scilab code Exa 4.6.3 Using the Simplified Span Method to Construct a Basis

```
1  // CONSTRUCTING SPECIAL BASES - Using the Simplified
       Span Method to Construct a Basis
2
3  clc
4  clear
5
6  A = [1,2,0,1; 3,1,5,-7; -2,4,-8,14];
7
8  // row reduce
9  reducedA=rref(A);
10
11 disp('row reduced A =', reducedA);
12
13 vector1=A(:,1);
14 vector2=A(:,2);
15 vector3=A(:,3);
16 vector4=A(:,4);
17
18 if (vector3 == 2*vector1-vector2) then
19     if (vector4== -3*vector1 + 2*vector2) then
20         disp('third and fourth vectors of S are
               linear combinations of the first two
               vectors');
21         disp('[0, 5, 8 ] = 2[1, 3, 2 ]    [2, 1,
              4]');
22         disp('[1, 7 , 14] = 3 [1, 3, 2 ] +
               2[2, 1, 4]');
23         disp('therefore, redundant vectors can be
               eliminated from S without affecting span(
```

```
            S)')
24      end
25  end
26
27  disp('Therefore, B is a subset of S that forms a
       basis for V = span(S), B=', [A(:,1)], [A(:,2)]);
28
29  dim = rank(reducedA);
30
31  disp('dim(V) =', dim);
```

Scilab code Exa 4.6.4 Using the Simplified Span Method to Construct a Basis

```
1  // CONSTRUCTING SPECIAL BASES - Using the Simplified
       Span Method to Construct a Basis
2
3  clc
4  clear
5
6  A = [1,3,4,0,-1; 2,6,1,0,5; -1,-3,2,0,-5];
7
8  // row reduce
9  C=rref(A);
10
11 disp('row reduced C =', C);
12
13 disp('There are nonzero pivots in the first and
       third columns of C');
14
15 disp('Therefore, B forms a basis for V = span(S). B=
       ', [A(:,1)], [A(:,3)]);
16
17 dim = rank(C);
18
19 disp('dim(V) =', dim);
```

Scilab code Exa 4.6.5 CONSTRUCTING SPECIAL BASES Using the Simplified Span Method

```
// CONSTRUCTING SPECIAL BASES - Using the Simplified
    Span Method to Construct a Basis

clc
clear

A = [1,0,2,0; -3,2,0,0; 0,1,3,4; 1,0,2,-5];

// row reduce
C=rref(A);

disp('row reduced C =', C);

disp('There are nonzero pivots in the  first, second
   , and fourth columns of C');

disp('Therefore, B forms a basis for V = span(S). B=
   ', [A(:,1)], [A(:,2)], [A(:,4)]);

dim = rank(C);

disp('dim(V) =', dim);
```

Chapter 5

Linear Transformations

Scilab code Exa 5.3.7 Rank of the Transpose

```
1  // Rank of the Transpose
2
3  clc
4  clear
5
6  A = [8,4,16,32,0; 4,2,10,22,-4; -2,-1,-5,-11,7;
       6,3,15,33,-7];
7  disp('A =', A);
8
9  reducedA=rref(A);
10 disp('A after row reduction =', reducedA);
11 rankA = rank(A);
12 disp('rank of A =', rankA);
13
14
15 A_transpose = A';
16 disp('A transpose =', A_transpose);
17 reducedA_transpose = rref(A_transpose);
18 disp('A transpose after row reduction =',
       reducedA_transpose);
19 rankA_transpose = rank(A_transpose);
```

```
20 disp('rank of A transpose =', rankA_transpose);
```

Chapter 6

Orthogonality

Scilab code Exa 6.1.1 Orthogonal and Orthonormal Vectors

```
1  // Orthogonal and Orthonormal Vectors
2
3  clc
4  clear
5
6  disp('{[1,0,-1,0], [3,0,3,0]}');
7
8  vector1=[1, 0, -1, 0];
9  vector2=[3, 0, 3, 0];
10
11 vector_dot_product = sum(vector1.*vector2);
12 disp('[1, 0, 1 , 0].[3, 0, 3, 0] =',
       vector_dot_product);
13 if vector_dot_product == 0 then
14     disp('dot product of vector is 0', 'therefore
          {[1, 0, 1 , 0], [3, 0, 3, 0]} is an
          orthogonal set');
15 end
16
17 orthonormal_vector1 = vector1/norm(vector1);
18 orthonormal_vector2 = vector2/norm(vector2);
```

```
19
20 disp('orthonormal set of vector is',
      orthonormal_vector1, orthonormal_vector2);
```

Scilab code Exa 6.1.2 Orthogonal and Orthonormal Vectors

```
1  // Orthogonal and Orthonormal Vectors
2
3  clc
4  clear
5
6  disp('{[1,0,-1,0], [3,0,3,0]}');
7
8  vector1=[1, 0, -1];
9  vector2=[-1, 4, -1];
10 vector3=[2, 1, 2];
11
12 vector_dot_product12 = sum(vector1.*vector2);
13 vector_dot_product23 = sum(vector2.*vector3);
14 vector_dot_product13 = sum(vector1.*vector3);
15
16 disp('[1, 0, 1].[-1, 4, -1] =', vector_dot_product12
      );
17 disp('[-1, 4, -1].[2, 1, 2] =', vector_dot_product23
      );
18 disp('[1, 0, 1].[2, 1, 2] =', vector_dot_product13);
19
20 if (vector_dot_product12 == 0) & (
      vector_dot_product23 == 0) & (
      vector_dot_product13 == 0) then
21     disp('dot product of vectors are 0', 'therefore
         {[1, 0, 1], [1, 4, 1], [2, 1, 2]} is
          an orthogonal set');
22 end
23
```

```
24 orthonormal_vector1 = vector1/norm(vector1);
25 orthonormal_vector2 = vector2/norm(vector2);
26 orthonormal_vector3 = vector3/norm(vector3);
27
28 disp('orthonormal basis for R3 is',
      orthonormal_vector1, orthonormal_vector2,
      orthonormal_vector3);
```

Scilab code Exa 6.1.6 Orthogonal Matrices

```
1  // Orthogonal Matrices
2
3  clear
4  clc
5
6  v1=[1/sqrt(2), 0, -1/sqrt(2)];
7  v2=[-1/(3*sqrt(2)), 4/(3*sqrt(2)), -1/(3*sqrt(2))];
8  v3=[2/3, 1/3, 2/3];
9
10 A = [v1; v2; v3];
11 disp('A =', A);
12 A_transpose = A';
13 disp('A transpose =', A_transpose);
14
15 A_mul_A_transpose = round(A*A_transpose);
16 disp('A * A transpose =', A_mul_A_transpose);
17
18 if A_mul_A_transpose == eye(3,3) then
19     disp('A and A transpose are orthogonal matrices'
         );
20 end
```

Scilab code Exa 6.2.7 Orthogonal Projection Onto a Subspace

```
// Orthogonal Projection Onto a Subspace

clc
clear

u1 = [1/sqrt(5), 0, -2/sqrt(5)];
u2 = [-2/sqrt(30), 5/sqrt(30), -1/sqrt(30)];
v = [-6,10,5];
// projWv = (v    u1) u1 + (v    u2) u2

disp('u1 =', u1, 'u2 =', u2, 'v =', v, 'projWv = (v
      u1) u1 + (v    u2) u2');

projWv = sum(v.*u1)*u1 + sum(v.*u2)*u2;
disp('projWv =', projWv);

// v     projWv
dist_vector = v - projWv;
disp('v - projWv =', dist_vector);
```

Scilab code Exa 6.2.10 Matrix with respect to standard basis

```
// matrix with respect to standard basis

clc
clear

// defining transition matrix P
P = [5,1,0; -1,0,1; 3,-5/3,1/3];

// calculating inverse of P
P_inverse = inv(P);

// defining matrix D
D = [-1,0,0; 0,1,0; 0,0,1];
```

```
14
15 // calculating A
16 A = P*D*P_inverse;
17
18 disp('A =', A);
```

Scilab code Exa 6.2.11 Distance From a Point to a Subspace

```
1  // Distance From a Point to a Subspace
2
3  clc
4  clear
5
6  u1 = [1/sqrt(5), 0, -2/sqrt(5)];
7  u2 = [-2/sqrt(30), 5/sqrt(30), -1/sqrt(30)];
8  v = [-6,10,5];
9  // projWv = (v    u1) u1 + (v    u2) u2
10
11 disp('u1 =', u1, 'u2 =', u2, 'v =', v, 'projWv = (v
       u1) u1 + (v    u2) u2');
12
13 projWv = sum(v.*u1)*u1 + sum(v.*u2)*u2;
14 disp('projWv =', projWv);
15
16 // v     projWv
17 dist_vector = v - projWv;
18 disp('v - projWv =', dist_vector);
19
20 min_dist = norm(dist_vector);
21
22 disp('minimum distance from P =', min_dist);
```

Chapter 7

Complex Vector Spaces and General Inner Products

Scilab code Exa 7.1.1 Complex matrices operations

```
1 // complex matrices
2
3 clc
4 clear
5
6 // defining matrices
7 z = [3-2*%i, -2+%i, -4-3*%i];
8 w = [-2+4*%i, 5-%i, -2*%i];
9
10 disp('z =', z);
11 disp('w =', w);
12
13 // finding conjugate of matrix
14 w_conj = conj(w);
15 z_conj = conj(z);
16
17 // calculating dot product with conjugate of other
     matrix
18 disp(sum(z.*w_conj));
```

```
19 disp(sum(w.*z_conj));
```

Scilab code Exa 7.1.2 Complex matrices finding entry

```
1  // complex matrices
2
3  clc
4  clear
5
6  // defining matrices
7  Z = [1-%i, 2*%i, -2+%i; -3*%i, 3-2*%i, -1-%i];
8  W = [-2*%i, 1-4*%i; -1+3*%i, 2-3*%i; -2+%i, -4+%i];
9
10 // calculating (1,1) entry of ZW
11 entry_ZW = (Z(1,:)*W(:,1));
12
13 // finding product of ZW
14 ZW = Z*W;
15
16 disp('(1, 1) entry of ZW is', entry_ZW);
17 disp('Product of ZW is', ZW);
```

Scilab code Exa 7.1.3 Conjugate transpose of complex matrix

```
1  // complex matrices
2
3  clc
4  clear
5
6  // defining matrix
7  Z = [2-3*%i, -%i, 5; 4*%i, 1+2*%i, -2-4*%i];
8  disp('Z =', Z);
9
```

```
10 // calculating conjugate transpose of matrix
11 Z_conj_transpose = Z';
12
13 disp('conjugate transpose of matrix =',
      Z_conj_transpose);
```

Scilab code Exa 7.1.4 Hermitian and skew hermitian matrix

```
1  // complex matrices
2
3  clc
4  clear
5
6  // defining matrix
7  H = [3, 2-%i, 1-2*%i; 2+%i, -1, -3*%i; 1+2*%i, 3*%i,
       4];
8  disp('H =', H);
9
10 // finding matrix transpose
11 H_transpose = H';
12 disp('H transpose =', H_transpose);
13
14 if H'==H then
15     disp('Matrix is hermitian as  H   = H');
16 end
17
18 disp('conjugate of H is', conj(H));
19 disp(' H   is', H_transpose);
20
21 // defining matrix
22 K = [-2*%i, 5+%i, -1-3*%i; -5+%i, %i, 6; 1-3*%i, -6,
       3*%i];
23 disp('K =', K);
24
25 // finding matrix transpose
```

```
26 K_transpose = K';
27 disp('H transpose =', K_transpose);
28
29 if K'==-K then
30     disp('Matrix is skew-hermitian as  K   = -K');
31 end
32
33 disp('conjugate of K is', conj(K));
34 disp(' H   is', K_transpose);
```

Scilab code Exa 7.1.5 Normal matrix

```
1 // complex matrices
2
3 clc
4 clear
5
6 Z = [1-2*%i, -%i; 1, 2-3*%i];
7 disp("Z =", Z);
8
9 Z_conj_transpose = Z';
10 Z_mul_Z_conj_transpose = Z*Z_conj_transpose;
11 Z_conj_transpose_mul_Z = Z_conj_transpose*Z;
12
13 if Z_mul_Z_conj_transpose == Z_conj_transpose_mul_Z
     then
14     disp("Matrix Z =",Z, "is normal as  Z Z   =  Z Z
         =", Z_mul_Z_conj_transpose);
15 end
```

Chapter 8

Additional Applications

Scilab code Exa 8.4.1 Markov chains

```
1  // markov chains
2
3  clc
4  clear
5
6  // defining matrix
7  p = [40/100; 10/100; 50/100];
8  disp('p =', p);
9
10 M = [.500, .167, .250; .250, .667, .250; .250, .167,
       .500];
11 disp('M =', M);
12
13 // calculating distribution
14 p1 = M*p;
15 disp('p1 = distribution after one years =', p1);
16
17 p2 = M*p1;
18 disp('p2 = distribution after two years =', p2);
19
20 p6 = (M^6)*p;
```

```
21 disp('p6 = distribution after six years =', p6);
```

Scilab code Exa 8.4.2 Markov chains

```
1 // markov chains
2
3 clc
4 clear
5
6 // defining matrix
7 p = [40/100; 10/100; 50/100];
8 M = [.500, .167, .250; .250, .667, .250; .250, .167,
      .500];
9 disp('M =', M);
10
11 pf = [.286, .429,.286];
12
13 Mf = [.286, .286, .286; .429, .429, .429; .286,
     .286, .286];
14
15 disp('pf =', pf);
16
17 // calculating probability vector
18 probability_vector = Mf*p;
19 disp('Probability vector = ', probability_vector);
```

Chapter 9

Numerical Techniques

Scilab code Exa 9.1.1 Gaussian elimination without partial pivoting

```
1 // gaussian elimination - without partial pivoting
2
3 clc
4 clear
5
6 matrix1 = [0.0006, -1, 1; 0.03, 30, -5; 0.04, 40,
     -7];
7 equation_solution = [10; 15; 19];
8
9 // matrix representation of equations
10 disp('Linear equation can be represented as matrix1,
     ', matrix1, '=', equation_solution);
11
12 // two ways to solve problem
13 // 1. using linsolve
14 // 2. using Gaussian Elimination, row operations (
     book method)
15
16 // 1.
17 solution_to_equation = -linsolve(matrix1,
     equation_solution);
```

```
18 disp('Unique solutions to the linear equation is',
      solution_to_equation);
19
20 // or Gaussian Elimination gives inaccurate answer
21
22 // 2.
23 // create augmented matrix
24 matrix2 = cat(2, matrix1, equation_solution); // 1
      for row, 2 for column
25 disp('augmented matrix', matrix2);
26
27 // row1     (1/0.0006)*row1
28 matrix2(1,:) = (1/0.0006)*matrix2(1,:);
29 disp('row1     (1/0.0006)*row1', matrix2);
30
31 // row2       0  .03*row1 + row2 and row3       0
      .04*row1 + row3
32 matrix2(2,:) = (-0.03)*matrix2(1,:) + matrix2(2,:);
33 matrix2(3,:) = (-0.04)*matrix2(1,:) + matrix2(3,:);
34 disp('row2       0  .03*row1 + row2 and row3       0
      .04*row1 + row3', matrix2);
35
36 // row2     (1/80)*row2
37 matrix2(2,:) = (1/80)*matrix2(2,:); // in the book
      it is divided by 80.01 due to round off error
38 disp('row2     (1/80)*row2', matrix2);
39
40 // row3       106  .7*row2 + row3
41 matrix2(3,:) = (-106.7)*matrix2(2,:) + matrix2(3,:);
42 disp('row3       106  .7*row2 + row3', matrix2);
43
44 // It is wrong printed in book. row3     ( 1
      /0.3131)+ row3 should be row3     ( 1 /0.3131)*
      row3
45 // row3     ( 1 /0.3131)*row3
46 matrix2(3,:) = (-1/0.3131)*matrix2(3,:);
47 disp('row3     ( 1 /0.3131)*row3', matrix2);
48
```

```
49 // the deviations are because the book rounded off
   some numbers.
50 x3 = matrix2(3,4);
51 x2 = matrix2(2,4) - matrix2(2,3)*x3;
52 x1 = matrix2(1,4) - matrix2(1,3)*x3 - matrix2(1,2)*
   x2;
53 disp('(x1,x2,x3) =', x1, x2, x3);
54
55 // Answer varies due to round off error
56
57 disp('This answer from back substitution is
   inaccurate answer and is largely the result of
   dividing row 1 through by 0.0006 which is much
   smaller than the other entries of the matrix (in
   row reduction for row(I)).');
```

Scilab code Exa 9.1.2 Gaussian elimination with partial pivoting

```
1 // gaussian elimination - with partial pivoting
2
3 clc
4 clear
5
6 matrix1 = [0.0006, -1, 1; 0.03, 30, -5; 0.04, 40,
   -7];
7 equation_solution = [10; 15; 19];
8
9 // matrix representation of equations
10 disp('Linear equation can be represented as matrix1,
   ', matrix1, '=', equation_solution);
11
12 // two ways to solve problem
13 // 1. using linsolve
14 // 2. using Gaussian Elimination, row operations (
   book method)
```

```
15
16 // 1.
17 solution_to_equation = -linsolve(matrix1,
      equation_solution);
18 disp('Unique solutions to the linear equation is',
      solution_to_equation);
19
20 // or Gaussian Elimination gives inaccurate answer
21
22 // 2.
23 // create augmented matrix
24 matrix2 = cat(2, matrix1, equation_solution); // 1
      for row, 2 for column
25 disp('augmented matrix', matrix2);
26
27 // row1     row3
28 temp = matrix2(1,:);
29 matrix2(1,:) = matrix2(3,:);
30 matrix2(3,:) = temp;
31 disp('row1     row3', matrix2);
32
33 // row1     (1/0.04)*row1
34 matrix2(1,:) = (1/0.04)*matrix2(1,:);
35 disp('row1     (1/0.04)*row1', matrix2);
36
37 // row2       0  .03*row1 + row2 and row3
      -0.0006*row1 + row3
38 matrix2(2,:) = (-0.03)*matrix2(1,:) + matrix2(2,:);
39 matrix2(3,:) = (-0.0006)*matrix2(1,:) + matrix2(3,:)
      ;
40 disp('row2       0  .03*row1 + row2 and row3
      -0.0006*row1 + row3', matrix2);
41
42 // row2     row3
43 temp = matrix2(2,:);
44 matrix2(2,:) = matrix2(3,:);
45 matrix2(3,:) = temp;
46 disp('row2     row3', matrix2);
```

```
47
48 // row2      (-1/1.6)*row2
49 matrix2(2,:) = (-1/1.6)*matrix2(2,:);
50 disp('row2      (-1/1.6)*row2', matrix2);
51
52 // row3      (1/0.25)*row3
53 matrix2(3,:) = (1/0.25)*matrix2(3,:);
54 disp('row3      (1/0.25)*row3', matrix2);
55
56 x3 = matrix2(3,4);
57 x2 = matrix2(2,4) - matrix2(2,3)*x3;
58 x1 = matrix2(1,4) - matrix2(1,3)*x3 - matrix2(1,2)*
     x2;
59 disp('(x1,x2,x3) =', x1, x2, x3);
60
61 disp('The correct solution is obtained by partial
     pivoting');
```