Contents

Chapter 7

Software Systems for Linear Algebra Problems

This is Chapter 7 of "Sophomore Level Self-Teaching Webbook for Computational and Algorithmic Linear Algebra and n-Dimensional Geometry" by Katta G. Murty.

Nowadays there are several commercially available and highly popular software systems for solving the problems discussed in previous chapters. Among them the best known are: MATLAB, EXCEL, MATHEMATICA, MAPLE, and MACAULAY. We will provide illustarative numerical examples of how to solve these problems using MATLAB (this name is derived from *Matrix Laboratory*) in this chapter. For using the other software systems, see the references cited on them.

7.1 Difference Between Solving a Small Problem By Hand and Scientific Computing Using a Computer

When we solve small numerical problems by hand, we always use **exact arithmetic**, so all the numbers in the various stages of the work are exactly what they are supposed to be (for example, a fraction like "1/3" is maintained as it is). Hand computation is not feasible for large size problems encountered in applications. Such large problems are usually solved using a software package on a digital computer.

Digital computation uses **finite precision arithmetic**, not exact arithmetic; for example a fraction like "1/3" is **rounded** to 0.3333 or 0.3 . . . 3 depending on the number of **significant digits of precision** maintained, introducing a small **rounding error**. These rounding errors abound in scientific computation, and they accumulate during the course of computation. So results obtained in digital computation usually contain some rounding errors, and may only be approximately correct. These errors make it nontrivial to implement many of the algorithms discussed in earlier chapters to get answers of reasonable precision. That's why detailed descriptions of numerical implementations of algorithms are very different and much more complicated than their mathematical descriptions given in earlier chapters.

As an example consider the GJ pivotal method for solving a system of linear equations discussed in Section 1.16. Under exact arithmetic, we know that a redundant constraint in the system corresponds to a "$0 = 0$" equation in the final tableau and vice versa. However, under finite precision arithmetic some of the "0" entries in it may become nonzero numbers of hopefully small absolute value. So, what is actually a "$0 = 0$" equation in the final tableau, may in reality become an equation with nonzero entries of small absolute value, and it is very difficult to decide whether it is a redundant equation or not. In computer implementations, one difficult question faced at every stage is whether a nonzero entry of small absolute value is actually a nonzero entry, or a zero entry that has become nonzero due to rounding error accumulation. A practical rule that is often used selects a small positive tolerance, and replaces any entry in the tableau whose absolute value is less than this tolerance by zero. Under such a rule, we can only conclude that a "$0 = 0$" equation in the final tableau is *redundant to working precision.*

In the same way, when a square matrix has a determinant of very small absolute value, using finite precision arithmetic it is very difficult to decide whether it is actually nonsingular or singular, and software systems will conclude that this *matrix is singular to working precision.*

Similarly while the definitions of linear independence or dependence of a set of vectors and its rank defined in Chapter 4 are conceptually precise and mathematically unambiguous under exact arithmetic, using finite precision arithmetic they can only be determined *correct to the working precision.*

Numerical analysts have developed a variety of techniques to reduce the effects of rounding errors, some of these like partial pivoting or complete pivoting for pivot element selection are discussed very briefly in Chapter 1. As a detailed discussion of these techniques is beyond the scope of this book, the interested reader should refer to books in that area (for example, see Gill, Murray, Wright[4.1]).

We will see the effects of rounding errors in some of the illustrative numerical examples given below.

7.2 How to Enter Vectors and Matrices in MATLAB

First you need to begin a MATLAB session. It should be noted that there are many versions of MATLAB in the market, and input prompts, and error and warning messages may vary from version to version. Also, different computer systems may have different commands for initiating a MATLAB session on their system.

After getting the MATLAB prompt, if you want to enter a row vector, $a = (1, 6, 7, -9)$ say, type

$a = [1 \quad 6 \quad 7 \quad -9]$ or $a = [1, 6, 7, -9]$

with a blank space or a comma (,) between entries, and MATLAB responds with

$a = 1 \quad 6 \quad 7 \quad -9.$

To enter the same vector as a column vector, type

$a = [1, 6, 7, -9]'$ or $a = [1 \quad 6 \quad 7 \quad -9]'$ or $a = [1; 6; 7; -9]$

and MATLAB responds with

$$a = \begin{matrix} 1 \\ 6 \\ 7 \\ -9 \end{matrix}$$

To supress the system's response, a semicolon (;) is placed as the last character of the expression. For example, after typing *a = [1, 6, 7, −9];* MATLAB responds with a new line awaiting a new command.

MATLAB also lets one place several expressions on one line, a line being terminated by pressing the *Enter* button on the computer. In this case, each expression is separated by either a comma (,) or a semicolon (;). The comma results in the system echoing the output; the semicolon supresses it. For example, by typing

a = [1, 2, 6, 9], c = [2, −3, 1, −4]

the system responds with

```
a = 1   2   6   9
c = 2  −3   1  −4.
```

Matrices are entered into MATLAB row by row, with rows separated by either semicolons, or by pressing *Enter* button on the computer. For example, to enter the matrix

$$A = \begin{pmatrix} 10 & -2 & 4 \\ 0 & 8 & -7 \end{pmatrix}$$

type *A = [10, −2, 4; 0, 8, −7]*

or type

```
A = [ 10  −2   4
       0   8  −7 ]
```

using *Enter* to indicate the end of 1st row

or type

```
A = [ 10  −2   4;  ...
       0   8  −7    ]
```

the ... called ellipses is a continuation of a MATLAB expression to next line. Used to create more readable code.

MATLAB has very convenient ways to address the entries of a matrix. For example, to display the $(2, 3)$ entry in the above matrix A type *A(2, 3)*, to display 2nd column of A type *A(:, 2)*, and to display 2nd row of A type *A(2, :)*.

Some of the commands used for special matrices are:

eye(n)	identity matrix of order n
diag([a_1 a_2 $\dots$ a_n]*)*	diagonal matrix with diagonal entries $a_1, \dots, a_n$
zeros(m, n)	zero matrix of order $m \times n$
zero(n*)*	zero matrix of order $n \times n$
A'	transpose of matrix A

If B is a square matrix defined in MATLAB, *diag(B)* returns the diagonal elements of B. To see the contents of a vector or matrix A defined earlier, just type A and MATLAB responds by displaying it.

The arithmetic operators to perform addition, subtraction, multiplication, division, and exponentiation are: $+$, $-$, $*$, $/$, ^ respectively. As an example, suppose the vectors $x = (1, 0, 0)$, $y = (0, 1, 0)$, $z = (0, 0, 1)$ have been defined earlier. If you type $2 * x + 3 * y - 6 * z$ MATLAB responds with

ans $=$ *2* *3* -6.

If you type $w = 2 * x + 3 * y - 6 * z$ MATLAB responds with

$w =$ *2* *3* -6.

If you type an arithmetical expression that is not defined, for example $x + y'$ with x, y as defined above (i.e., the sum of a row vector and a column vector), MATLAB yields the warning

???Error using $\Longrightarrow +$
Matrix dimensions must agree.

Other arithmetical operations on matrices and vectors can be defined in MATLAB in the same way. Here are some MATLAB commands that we will use, and the outputs they produce. For a more complete list of MATLAB commands, see the MATLAB references given at the end of this chapter. Let A, B be two matrices, and x, y be two vectors.

length(x) or *size*(x)	the number of the elements in the vector x
size(A)	$m \quad n$ where $m \times n$ is the order of A
dot(x, y)	dot product of x and y (whether each is either a row or a column) provided they have same no. of elements. This can also be obtained by typing $x * y$ provided x is a row vector & y is a col. vector
norm(x)	the Eucledean norm $\|\|x\|\|$ of the vector x
det(A)	determinant of A, if it is a square matrix
rank(A)	rank of the matrix A
inv(A)	inverse of matrix A, if A is an invertible square matrix
$A * B$	matrix product AB if it is defined.
$A\backslash b$	solves the system of linear equations with A as the coefficient matrix, and b as the RHS constants vector
null(A)	outputs 0-vector if system $Ax = 0$ has no nonzero solution, or an orthonormal basis for the subspace which is the set of all solutions of $Ax = 0$ obtained by a method called singular value decomposition not discussed in the book if $Ax = 0$ has nonzero solutions.
rref(A)	outputs the reduced row echelon form of matrix A.
rrefmovie(A)	shows all the operations that MATLAB performs to obtain the RREF of the matrix A.
null$(A,'r')$	same as *null*(A) except that in the 2nd case it produces the basis for the set of solutions of $Ax = 0$ obtained by the GJ or the G elimination method as discussed in Section 1.23.
eig(A)	produces the eigen values of the square matrix A.

$[V, D] = \mathit{eig}(A)$	produces $D =$ diagonal matrix of eigen values of A, and $V =$ matrix whose column vectors are corresponding eigen vectors.
all(eig$(A + A') > 0)$	for a square matrix A, this checks whether all eigenvalues of $(A + A')$ are strictly positive. The output is 1 if the answer is yes, in this case A is a positive definite matrix; or 0 if the answer is no. In the latter case, using the command described above, one can generate all the eigenvalues and associated eigen vectors of $(A + A')$. The eigen vector x associated with a nonpositive eigen value of $(A + A')$ satisfies $x^T Ax \leq 0$.
all(eig$(A + A') >= 0)$	for a square matrix A, this checks whether all eigenvalues of $(A + A')$ are nonnegative. The output is 1 if the answer is yes, in this case A is a positive semidefinite matrix; or 0 if the answer is no and A is not PSD.
chol(A)	for a symmetric square matrix A, this command outputs the Cholesky factor of A (Cholesky factorization is not discussed in this book) if A is PD. If A is not PD, it outputs an error message that *Matrix must be positive definite* to use chol. So, this command can also be used to check if a given square symmetric matrix is PD.

When a system of linear equations $Ax = b$ has more than one solution, there are two ways of obtaining an expression of the general solution of the system using MATLAB. One is to get the RREF (a canonical tableau WRT a basic vector) of the system using the *rrefmovie* command (see Examples 2, 3 given below), and then construct an expression of the general solution of the system using it as explained in Section 1.8. The other is to get one solution, $\bar{x}$ say, using the $A\backslash b$ command; and then a basis for the null space of the coefficient matrix A using the *null*(A) command. If this basis is $\{x^1, \dots, x^s\}$, then the general solution of the system is $\bar{x} + \alpha_1 x^1 + \ldots + \alpha_s x^s$ where $\alpha_1, \dots, \alpha_s$ are parameters that take real values.

7.3 Illustrative Numerical Examples

1. Solving a Square System of Linear Equations:

Suppose we want to find a solution for the following system and check whether it is unique. We provide the MATLAB session for doing this. The coefficient matrix is called A, and the RHS constants vector is called b, and the solution vector is called x.

$$\begin{aligned} 2x_1 + x_2 - 3x_3 - x_4 &= 1 \\ -x_1 - 2x_2 + 4x_3 + 5x_4 &= 6 \\ 7x_1 - 6x_4 &= 7 \\ -3x_1 - 8x_2 - 9x_3 + 3x_4 &= 9 \end{aligned}$$

Type A = *[2, 1, −3, −1; −1, −2, 4, 5; 7, 0, 0, −6; −3, −8, −9, 3]*

Response $A = \begin{matrix} 2 & 1 & -3 & -1 \\ -1 & -2 & 4 & 5 \\ 7 & 0 & 0 & -6 \\ -3 & -8 & -9 & 3 \end{matrix}$

Type b = *[1, 6, 7, 9]'*

Response $b = \begin{matrix} 1 \\ 6 \\ 7 \\ 9 \end{matrix}$

Type $x = A\backslash b$

Response $x = \begin{matrix} 1.7843 \\ -1.5062 \\ 0.0491 \\ 0.9151 \end{matrix}$

The solution vector x is provided by MATLAB, this indicates that this is the unique solution, and that the coefficient matrix A is nonsingular.

In MATLAB the final canonical tableau is called RREF (reduced row echelon form). The command: *rref*(A, b), outputs the RREF for the augmented matrix $(A\vdots b)$ of the system. Here is the session to obtain that for this example.

Type $d =$*rref*$([Ab])$

Response $d =$

```
1.0000        0        0        0    1.7843
     0   1.0000        0        0   -1.5063
     0        0   1.0000        0    0.0491
     0        0        0   1.0000    0.9151
```

2. Another Square System of Linear Equations:

The ststem is given below. We call the coefficient matrix A, the RHS constants vector b, and the solution vector x.

$$\begin{aligned}
2x_1 + x_2 - 3x_3 + x_4 &= 4\\
-x_1 - 2x_2 + 4x_3 + 5x_4 &= -3\\
2x_1 - 2x_2 + 2x_3 + 12x_4 &= 6\\
x_1 - x_2 + x_3 + 6x_4 &= 3
\end{aligned}$$

The MATLAB session is carried out exactly as above, but this time MATLAB gives the following final response.

Warning: Matrix is singular to working precision

```
x = Inf
    Inf
    Inf
    Inf
```

This indicates that the coefficient matrix A is singular, and that the syatem has no solution.

In MATLAB the final canonical tableau is called RREF (reduced row echelon form). The command: *rref*(A, b), outputs the RREF for the augmented matrix $(A\vdots b)$ of the system. When rank$(A\vdots b) = 1 +$

rank(A), as in this case, MATLAB performs also a final pivot step in the column vector B, which we do not want. In this case the command:

rrefmovie$(A\vdots b)$

helps you to observe all the operations that MATLAB performs to obtain the RREF of the augmented matrix $(A\vdots b)$. The matrix before the final pivot step in the column of b is the RREF to our system. For this example, here is that output (here we did all the computations in fractional form).

```
1  0  −2/3    7/3   7/3
0  1  −5/3  −11/3  −2/3
0  0     0      0    −2
0  0     0      0     0
```

The rows in this final tableau do not directly correspond to rows (constraints) in the original problem, because of row interchanges performed during the algorithm. To find out which original constraint each row in the final tableau corresponds to, you can use the row interchange information that is displayed in the rrefmovie output.

3. A Rectangular System of Linear Equations:

The system is given below. The symbols A, b, x refer to the coefficient matrix, RHS constants vector, solution vector respectively.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
2	1	0	0	−1	−3	4	3
0	−2	1	2	0	4	3	−2
4	0	1	2	−2	−2	11	4
−1	2	3	1	−2	0	0	5
5	1	5	5	−5	−1	18	10

The session is carried out exactly as above, and MATLAB produced the following final response.

Warning: Rank deficient, rank = 3   tol = 3.3697e-014.

```
x =    0.7647
       1.4706
       0.9412
            0
            0
            0
            0
```

Here since the system has a solution, the final canonical tableau, RREF can be produced with the command: *rref*(A, b). Here it is:

```
1  0  0    0.2941  −0.3529  −0.7059   2.4118  0.7647
0  1  0   −0.5882   0.2941  −1.5882  −0.8235  1.4706
0  0  1    0.8235  −0.5882   0.8235   1.3529  0.9412
0  0  0         0        0        0        0       0
0  0  0         0        0        0        0       0
```

As in the above example, to trace the correspondence of rows in the RREF to the constraints in the original statement of the problem, one has to take into account the row interchanges performed by MATLAB in the process of getting this RREF by using the command: *rrefmovie*(A, b).

4. Another Rectangular System of Linear Equations:

The system is given below. The symbols A, b, x refer to the coefficient matrix, RHS constants vector, solution vector respectively.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
1	0	−1	1	−2	1	−3	−4
0	−1	2	3	1	−1	2	2
1	−2	3	7	0	−1	1	−2
−3	1	2	−1	0	0	2	3
−2	0	3	3	−1	0	1	3

The canonical tableau for this system, RREF, produced using the command: *rrefmovie*(A, b) is given below.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	b
1	0	0	6	−7	3	−8	−12
0	1	0	7	−11	5	−12	−17
0	0	1	5	−5	2	−5	−8
0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	3

The last two rows in the canonical tableau represent inconsistent equations, "$0 = \alpha$" for some $\alpha \neq 0$; so this system has no solution. To trace the correspondence of rows in the RREF to the constraints in the original statement of the problem, one has to take into account the row interchanges performed by MATLAB in the process of getting this RREF.

5. A Homogeneous System of Linear Equations:

Calling the coefficient matrix, RHS constants vector, solution vector as A, b, x respectively, here is the session.

Type *A = [1,1,0; 0,1,1; 1, 0, 1; 2, 2, 2]*

Response $A = \begin{matrix} 1 & 1 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 2 & 2 & 2 \end{matrix}$

Type *b = zeros(4, 1)* (Note: this command generates a 0-matrix of order 4×1)

Response $b = \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix}$

Type $x = A\backslash b$

Response $x = \begin{matrix} 0 \\ 0 \\ 0 \end{matrix}$

Type *rank(A)*

Response 3

Since rank of the coefficient mstrix is 3, from the results in Sections 4.5 and 1.22, we conclude that this system has no nonzero solution.

6. Another Homogeneous System of Linear Equations: Calling the coefficient matrix, RHS constants vector, solution vector as A, b, x respectively, here is the session.

Type *A = [1,−1,0, 2, 3, −2, 1; 0,1, −2, 1, −1, 3, −2; 1, 0, −2, 3, 2, 1, −1; −1, 2, 1, −2, 1, 3, 2]*

```
Response   A =     1  −1   0   2   3  −2   1
                   0   1  −2   1  -1   3  −2
                   1   0  −2   3   2   1  −1
                  −1   2   1  −2   1   3   2
```

Type *b = zeros(4, 1)* (Note: this command generates a 0-matrix of order 4 × 1)

```
Response  b =  0
               0
               0
               0
```

Type *d =rref*([*Ab*])

Response *d* =

```
1  0  0    7/3   16/3   −1/3   7/3  0
0  1  0    1/3    7/3    5/3   4/3  0
0  0  1   −1/3    5/3   −2/3   5/3  0
0  0  0     0      0      0     0   0
```

From the RREF we can see that this homogeneous system has nonzero solutions. From the RREF a basic set of nonzero solutions for the system can be constructed as discussed in Section 1.23.

7. Checking Linear Independence: The most convenient way to check linear independence of a given set of vectors (either all row vectors or all column vectors) using MATLAB is to write each of these vectors as a column vector of a matrix, A say. Then check whether the homogeneous system of equations $Ax = 0$ has a nonzero solution. If this system has no nonzero solution, the set is linearly independent. If $\bar{x}$ is a nonzero solution of the system, the set is linearly dependent; and $\bar{x}$ is the vector of coefficients in a linear dependence relation for it in the order in which the vectors are entered as column vectors in the matrix A.

As an example, consider the set of 4 column vectors, $\{A_{.1}.A_{.2}, A_{.3}, A_{.4}\}$ of the matrix A in Example 1. Here is the session for checking its linear independence; we suppress the display of the matrix in this session.

Type $A =$ *[2, 1, −3, −1; −1, −2, 4, 5; 7, 0,0, −6; −3, −8, −9, 3];* *null*(A)

Response ans = Empty matrix: 4-by-0.

This indicates that the system $Ax = 0$ has no nonzero solution, i.e., the set of column vectors of A is linearly independent.

8. Checking Linear Independence, Another Example:

Consider the set of 4 column vectors, $\{A_{.1}.A_{.2}, A_{.3}, A_{.4}\}$ of the matrix A in Example 2. Here is the session for checking its linear independence; we suppress the display of the matrix in this session.

Type $A =$ *[2, 1, −3, 1; −1, −2, 4, 5; 2, −2, 2, 12; 1, −1, 1, 6];* *null*(A)

$$\text{Response} \quad \text{ans} = \begin{array}{rr} -0.7818 & -0.3411 \\ -0.2469 & 0.9111 \\ -0.5436 & 0.1382 \\ 0.1797 & 0.1857 \end{array}$$

Each of the column vectors in the output above is a nonzero solution of $Ax = 0$ and these two vectors together form an orthonormal basis for the subspace which is the set of all solutions of this homogeneous system, obtained by using a method called singular value decomposition, which is not discussed in the book. If you do not require an orthonormal basis, but want the basis obtained by the GJ or G elimination methods as discussed in Section 1.22, change the command *null*(A) to *null*$(A,'r')$, then the output is (each column vector is a nonzero solution, together they form a basis for the subspace which is the set of all solutions of $Ax = 0$):

$$\text{Response} \quad \text{ans} = \begin{array}{rr} 0.6667 & -2.3333 \\ 1.6667 & 3.6667 \\ 1.0000 & 0 \\ 0 & 1.0000. \end{array}$$

Each of the column vectors in these outputs is the vector of coefficients in a linear dependence relation for the set of column vectors of the matrix A in this example. For instance the first column vector in the output under the command *null*(A) yields the linear dependence relation

$$-0.7818A_{.1} - 0.2469A_{.2} - 0.5436A_{.3} + 0.1797A_{.4} = 0.$$

9. Matrix Inversion: Calling the matrix A, here is the session.

Type $\quad A =$ *[1,2,1, −1; −1, 1, 1, 2; 0, −1, 2, 1; 2, 2, −1, 0]*

$$\textit{Response} \quad A = \begin{array}{rrrr} 1 & 2 & 1 & -1 \\ -1 & 1 & 1 & 2 \\ 0 & -1 & 2 & 1 \\ 2 & 2 & -1 & 0 \end{array}$$

Type $\quad$ *inv*(A)

$$\textit{Response} \quad \text{ans} = \begin{array}{rrrr} -0.1071 & -0.2500 & 0.3929 & 0.4286 \\ 0.2500 & 0.2500 & -0.2500 & 0.0000 \\ 0.2857 & 0.0000 & 0.2857 & -0.1429 \\ -0.3214 & 0.2500 & 0.1786 & 0.2857 \end{array}$$

10. Another Matrix Inversion: Calling the matrix A, here is the session.

Type $\quad A =$ *[0, 1, 1, 2; 1, 0, 2, 1; 2, 1, 0, 1; 3, 2, 3, 4]*

Response $A = \begin{matrix} 0 & 1 & 1 & 2 \\ 1 & 0 & 2 & 1 \\ 2 & 1 & 0 & 1 \\ 3 & 2 & 3 & 4 \end{matrix}$

Type $inv(A)$

Response Warning: Matrix is close to singular or badly scaled. Results may be inaccurate. RCOND $= 4.336809e-018$.

$$\text{ans} = 1.0e+016* \begin{matrix} 0.1801 & 0.1801 & 0.1801 & -0.1801 \\ -1.2610 & -1.2610 & -1.2610 & 1.2610 \\ -0.5404 & -0.5404 & -0.5404 & 0.5404 \\ 0.9007 & 0.9007 & 0.9007 & \text{-0.9007} \end{matrix}$$

According to the warning the input matrix A in this example appears to be singular, but reaching this exact conclusion is made difficult due to rounding errors. Of course a singular matrix does not have an inverse, but an inverse is obtained because rounding errors have altered the outcome of singularity. The warning message indicates that the outputted inverse is probably not accurate due to the singularity of the original matrix. Most likely, an inverse is still computed due to roundoff error introduced into the computation by computer arithmetic.

11. Nearest Point to $\bar{x}$ On a Given Straight Line:

Find the nearest point, x^*, (in terms of the Euclidean distance), to the given point $\bar{x}$, on the straight line L given in parametric form: $L = \{x : x = a + \lambda c$, where $a = (2,1,3,3)^T, c = (0,1,-1,2)^T$, and λ is the real valued parameter$\}$.

We denote the point $\bar{x}$ by $x1$ to avoid confusion, and we call the nearest point xn. Here is the session.

Type $a = [2,1,3,3]$;
$c = [0,1,-1,2]'$;
$x1 = [1,-1,2,0]'$;
$lambda = c' * (x1 - a)/[c(1,1)^2 + c(2,1)^2 + c(3,1)^2 + c(4,1)^2]$

Response $lambda = -1.1667$

Type $xn = 1 + lambda * c$

Response $xn =$

```
  2.0000
 −0.1667
  4.1667
  0.6667
```

Type $dist = [[(x1(1,1) - xn(1,1)]^2 + [(x1(2,1) - xn(2,1)]^2 + [(x1(3,1) - xn(3,1)]^2 + [(x1(4,1) - xn(4,1)]^2]^(1/2)$

Response *dist = 2.6141*

"dist" is the Euclidean distance between $x1$ and the nearest point to $x1$ on the given straight line. All these formulae are from Section 3.16.

12. Nearest Point to $\bar{x}$ On a Given Hyperplane:

Find the nearest point, x^*, (in terms of the Euclidean distance), to the given point $\bar{x}$, on the hyperplane $H = \{x : ax = a_0$ where $a = (1, -2, 3, -4), a_0 = -13\}$.

Again we denote $\bar{x}$ by $x1$, and the nearest point $x*$ by xn to avoid confusion. Here is the session.

Type $a = [1, -2, 3, -4]; a_0 = -13; x1 = [1, -1, 2, 0]';$
$xn = x1 + a' * [-(a * x1 - a_0)/(a(1,1)^2 + a(1,2)^2 + a(1,3)^2 + a(1,4)^2)]$

Response $xn =$

```
  0.2667
  0.4667
 −0.2000
  2.9333
```

Type $dist = [[(x1(1,1) - xn(1,1)]^2 + [(x1(2,1) - xn(2,1)]^2 + [(x1(3,1) - xn(3,1)]^2 + [(x1(4,1) - xn(4,1)]^2]^(1/2)$

Response *dist = 4.0166*

"dist" is the Euclidean distance between $x1$ and the nearest point to $x1$ on the given hyperplane. All these formulae are from Section 3.16.

13. Nearest Point to $\bar{x}$ In a Given Affine Space:

Find the nearest point, x^*, (in terms of the Euclidean distance), to the given point $\bar{x} = (1, 1, 0)^T$, in the affine space $F = \{x : Ax = b$ where A, b are given below$\}$.

$$A = \begin{pmatrix} 1 & 1 & -1 \\ 2 & 1 & 1 \end{pmatrix}, b = \begin{pmatrix} 2 \\ 9 \end{pmatrix}.$$

Here is the portion of the session after reading in $A, b, x1$ where we are calling the given point $\bar{x}$ as $x1$. The nearest point to $x1$ in the given affine space will be denoted by xn.

Type $\quad xn = x1 - A' * inv(A * A') * (A * x1 - b)$

Response $xn =$

```
2.7143
1.4286
2.1429
```

The Euclidean distance between $x1$ and xn using the same command as in the above examples is 2.7775.

14. Checking PD, PSD:

Check whether the following matrices A, B, C are PD, PSD, or neither.

$$A = \begin{pmatrix} 3 & 1 & 2 & 2 \\ -1 & 2 & 0 & 2 \\ 0 & 4 & 4 & 5/3 \\ 0 & 2 & -13/3 & 6 \end{pmatrix}, B = \begin{pmatrix} 8 & 2 & 1 & 0 \\ 4 & 7 & -1 & 1 \\ 2 & 1 & 1 & 2 \\ 2 & -2 & 1 & 1 \end{pmatrix}$$

$$C = \begin{pmatrix} 0 & -2 & -3 & -4 & 5 \\ 2 & 3 & 3 & 0 & 0 \\ 3 & 3 & 3 & 0 & 0 \\ 4 & 0 & 0 & 8 & 4 \\ -5 & 0 & 0 & 4 & 2 \end{pmatrix}$$

Here are the sessions after reading in the matrices.

Type $\mathit{all}(\mathit{eig}(A + A') > 0$

Response *ans = 0*. This means that A is not PD.

We show how the same thing is done using the *chol* cammand.

Type *chol(A + A')*

Response *??? Error using* $\Longrightarrow$ *chol. Matrix must be positive definite.*

We check that B is not PD using same procedures. Now to check whether B is PSD, we do the following.

Type $\mathit{all}(\mathit{eig}(B + B') >= 0)$

Response *1*. This means that B is PSD.

Since C has a 0 diagonal element, it is clearly not PD. To check whether it is PSD, we use the command $[V, D] = \mathit{eig}(Cs)$ command where $Cs = C + C'$. We get

$$D = \begin{array}{rrrrr} 0 & 0 & 0 & 0 & 0 \\ 0 & 12 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{array}$$

Since all the eigenvalues are nonnegative C is PSD.

15. Eigen Values and Eigen Vectors: Calling the matrix A, here is the session.

Type $A = [-3, 1, -3; 20, 3, 10; 2, -2, 4]$

Response $A = \begin{array}{rrr} -3 & 1 & -3 \\ 20 & 3 & 10 \\ 2 & -2 & 4 \end{array}$

Type $\mathit{eig}(A)$

```
                     −2.0000
Response   ans  =     3.0000
                      3.0000
```

$[V, D] = eig(A)$

```
                  0.4082  −0.4472   0.4472
Response   V =   −0.8165   0.0000   0.0000
                 −0.4082   0.8944  −0.8944

                 −2.0000        0        0
Response   D =         0   3.0000        0
                       0        0   3.0000
```

16. Matrix Diagonalization:

Diagonalize the following matrices A, B.

$$A = \begin{pmatrix} 1 & 3 & 3 \\ -3 & -5 & -3 \\ 3 & 3 & 1 \end{pmatrix}, B = \begin{pmatrix} 2 & 4 & 3 \\ -4 & -6 & -3 \\ 3 & 3 & 1 \end{pmatrix}.$$

The command $[V, D] = eig(A)$ gives the following output.

```
         0.5774         0   −0.7459
  V =   −0.5774   −0.7071    0.0853
         0.5774    0.7071    0.6606


          1    0    0
     D =  0   −2    0
          0    0   −2
```

-2 is an eigenvalue of A with algebraic multiplicity 2. Also, since two eigenvectors (which are not scalar multiples of each other) are associated with this eigenvalue, its geometric multiplicity is also 2. So, A has a complete set of eigenvectors which is linearly independent. So, A can be diagonalized. In fact by the results discussed in Chapter 6, the matrix V whose column vectors are the distinct eigenvectors of A diagonalizes A, i.e., $V^{-1}AV = D$.

The command $[V, D] = eig(B)$ gives the following output.

$$V = \begin{array}{rrr} -0.5774 & 0.7071 & 0.7071 \\ 0.5774 & -0.7071 & -0.7071 \\ -0.5774 & 0.0000 & 0.0000 \end{array}$$

$$D = \begin{array}{rrr} 1 & 0 & 0 \\ 0 & -2 & 0 \\ 0 & 0 & -2 \end{array}$$

Since the two eigenvectors associated with the eigenvalue -2 are the same, it indicates that the geometric multiplicity of the eigenvalue -2 is 1, while its algebraic multiplicity is 2. So, B does not have a complete set of eigenvectors, and hence it is not diagonalizable.

17. Orthogonal Diagonalization: Orthogonally diagonalize the following matrix A.

$$A = \begin{pmatrix} 5 & 2 & 9 & -6 \\ 2 & 5 & -6 & 9 \\ 9 & -6 & 5 & 2 \\ -6 & 9 & 2 & 5 \end{pmatrix}.$$

Since A is symmetric, it is possible to orthogonally diagonalize A by the results in Chapter 6. The command $[V, D] = eig(A)$ produced the following output:

$$V = \begin{array}{rrrr} 0.5 & 0.5 & -0.5 & 0.5 \\ 0.5 & 0.5 & 0.5 & -0.5 \\ 0.5 & -0.5 & -0.5 & -0.5 \\ 0.5 & -0.5 & 0.5 & 0.5 \end{array}$$

$$D = \begin{array}{rrrr} 10 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 \\ 0 & 0 & 18 & 0 \\ 0 & 0 & 0 & -12 \end{array}$$

By the results in Chapter 6, the matrix V whose column vectors form a complete set of eigenvectors for A orthogonally diagonalize A, i.e., $V^TAV = D$.

7.4 References

References on MATLAB

[7.1] M. Golubitsky and M. Dellnitz, *Linear Algebra and Differential Equations Using MATLAB*, Brooks/Cole Publishing Co., NY, 1999.
[7.2] E. B. Magrab and others, *An Engineer's Guide to MATLAB*, Prentice Hall, Upper Saddle River, NJ, 2000.
7.3] C. F. Van Loan, *Introduction to Scientific Computing*, MATLAB Curriculum Series, Prentice Hall, Upper Saddle River, NJ, 1997.
[7.4] *The Student Edition of MATLAB, Users Guide*, The MathWorks Inc., Natick, MA.

References on Other Software Systems

[7.5] B. W. Char, et. al. *First Leaves: A Tutorial Introduction to Maple V*, Springer Verlag, NY, 1992.
[7.6] J. H. Davenport, *Computer Algebra: Systems and Algorithms for Algebraic Computation*, Academic Press, San diago, 1993.
[7.7] D. Eugene, *Schaum's Outline of Theory and Problems of Mathematica*, McGraw-Hill, NY, 2001.
[7.8] M. B. Monagan, et. al., *Maple V Programming Guide*, Springer, NY, 1998.
[7.9] Microsoft Corp., *Microsoft Excel User's Guide, Version 5.0*, Redmond, WA, 1993.
[7.10] S. Wolfram, *The Mathematica Book*, 3rd ed., Wolfram Media, Cambridge University Press, 1996.