1 Matrix Algebra and Solution of Matrix Equations

1.1 INTRODUCTION

Computers are best suited for repetitive calculations and for organizing data into specialized forms. In this chapter, we review the *matrix* and *vector* notation and their manipulations and applications. Vector is a one-dimensional array of numbers and/or characters arranged as a single column. The number of rows is called the *order* of that vector. Matrix is an extension of vector when a set of numbers and/or characters are arranged in rectangular form. If it has M rows and N column, this matrix then is said to be of order M by N. When M = N, then we say this *square* matrix is of order N (or M). It is obvious that vector is a special case of matrix when there is only one column. Consequently, a vector is referred to as a column matrix as opposed to the row matrix which has only one row. Braces are conventionally used to indicate a vector such as {V} and brackets are for a matrix such as [M].

In writing a computer program, DIMENSION or DIM statements are necessary to declare that a certain variable is a vector or a matrix. Such statements instruct the computer to assign multiple memory spaces for keeping the values of that vector or matrix. When we deal with a large number of different entities in a group, it is better to arrange these entities in vector or matrix form and refer to a particular entity by specifying where it is located in that group by pointing to the row (and column) number(s). Such as in the case of having 100 numbers represented by the variable names A, B, …, or by A(1) through A(100), the former requires 100 different characters or combinations of characters and the latter certainly has the advantage of having only one name. The A(1) through A(100) arrangement is to adopt a vector; these numbers can also be arranged in a matrix of 10 rows and 10 columns, or 20 rows and five columns depending on the characteristics of these numbers. In the cases of collecting the engineering data from tests of 20 samples during five different days, then arranging these 100 data into a matrix of 20 rows and five columns will be better than of 10 rows and 10 columns because each column contains the data collected during a particular day.

In the ensuing sections, we shall introduce more definitions related to vector and matrix such as transpose, inverse, and determinant, and discuss their manipulations such as addition, subtraction, and multiplication, leading to the organizing of systems of linear algebraic equations into matrix equations and to the methods of finding their solutions, specifically the Gaussian Elimination method. An apparent application of the matrix equation is the transformation of the coordinate axes by a rotation about any one of the three axes. It leads to the derivation of the three basic transformation matrices and will be elaborated in detail.

Since the interactive operations of modern personal computers are emphasized in this textbook, how a simple three-dimensional brick can be displayed will be discussed. As an extended application of the display monitor, the transformation of coordinate axes will be applied to demonstrate how animation can be designed to simulate the continuous rotation of the three-dimensional brick. In fact, any three-dimensional object could be selected and its motion animated on a display screen.

Programming languages, **FORTRAN**, **QuickBASIC**, **MATLAB**, and **Mathematica** are to be initiated in this chapter and continuously expanded into higher levels of sophistication in the later chapters to guide the readers into building a collection of their own programs while learning the computational methods for solving engineering problems.

1.2 MANIPULATION OF MATRICES

Two matrices [A] and [B] can be added or subtracted if they are of same order, say M by N which means both having M rows and N columns. If the sum and difference matrices are denoted as [S] and [D], respectively, and they are related to [A] and [B] by the formulas [S] = [A] + [B] and [D] = [A]-[B], and if we denote the elements in [A], [B], [D], and [S] as a_{ij}, b_{ij}, d_{ij}, and s_{ij} for i = 1 to M and j = 1 to N, respectively, then the elements in [S] and [D] are to be calculated with the equations:

$$s_{ij} = a_{ij} + b_{ij} \tag{1}$$

and

$$d_{ij} = a_{ij} - b_{ij} \tag{2}$$

Equations 1 and 2 indicate that the element in the ith row and jth column of [S] is the sum of the elements at the same location in [A] and [B], and the one in [D] is to be calculated by subtracting the one in [B] from that in [A] at the same location. To obtain all elements in the sum matrix [S] and the difference matrix [D], the index i runs from 1 to M and the index j runs from 1 to N.

In the case of *vector* addition and subtraction, only one column is involved (N = 1). As an example of addition and subtraction of two vectors, consider the two vectors in a two-dimensional space as shown in Figure 1, one vector $\{V_1\}$ is directed from the origin of the x-y coordinate axes, point O, to the point 1 on the x-axis which has coordinates $(x_1,y_1) = (4,0)$ and the other vector $\{V_2\}$ is directed from the origin O to the point 2 on the y-axis which has coordinates $(x_2,y_2) = (0,3)$. One may want to find the resultant of $\{R\} = \{V_1\} + \{V_2\}$ which is the vector directed from the origin to the point 3 whose coordinates are $(x_3,y_3) = (4,3)$, or, one may want to find the difference vector $\{D\} = \{V_1\} - \{V_2\}$ which is the vector directed from the origin O to the point 4 whose coordinates are $(x_4,y_4) = (4,-3)$. In fact, the vector $\{D\}$ can be obtained by adding $\{V_1\}$ to the negative image of $\{V_2\}$, namely $\{V_{2-}\}$ which is a vector directed from the origin O to the point 5 whose coordinates are (x_5,y_5). Mathematically, based on Equations 1 and 2, we can have:

$$\{R\} = \{V_1\} + \{V_2\} = \begin{bmatrix} 4 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

and

$$\{D\} = \{V_1\} - \{V_2\} = \begin{bmatrix} 4 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ -3 \end{bmatrix}$$

When Equation 1 is applied to two arbitrary two-dimensional vectors which unlike $\{V_1\}$, $\{V_2\}$, and $\{V_{2-}\}$ but are not on either one of the coordinate axes, such as {D} and {E} in Figure 1, we then have the sum vector {F} = {D} + {E} which has components of 1 and –2 units along the x- and y-directions, respectively. Notice that O467 forms a parallelogram in Figure 1 and the two vectors {D} and {E} are the two adjacent sides of the parallelogram at O. To find the sum vector {F} of {D} and {E} graphically, we simply draw a diagonal line from O to the opposite vertex of the parallelogram — this is the well-known *Law of Parallelogram.*

It should be evident that to write out a vector which has a large number of rows will take up a lot of space. If this vector can be rotated to become from one column to one row, space saving would then be possible. This process is called transposition as we will be leading to it by first introducing the length of a vector.

For the calculation of the *length* of a two-dimensional or three-dimensional vector, such as $\{V_1\}$ and $\{V_2\}$ in Figure 1, it would be a simple matter because they are oriented along the directions of the coordinate axes. But for the vectors such as {R}

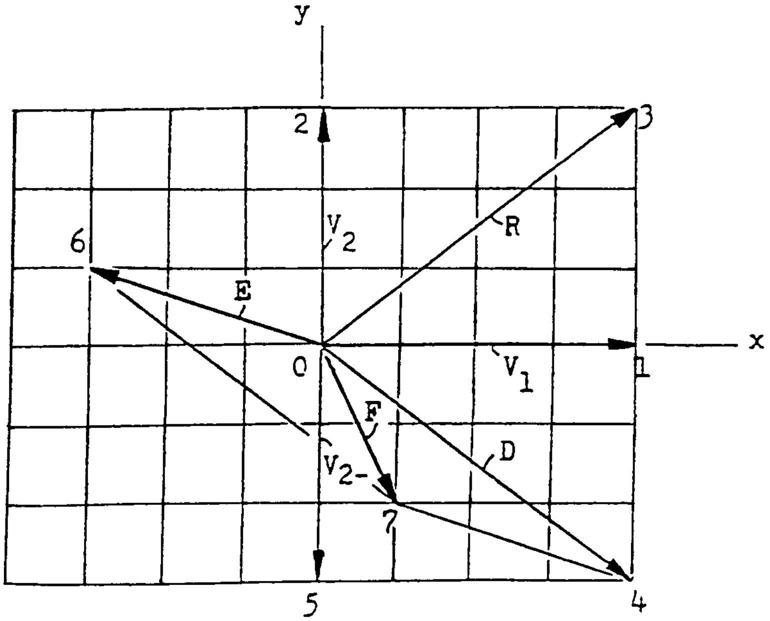


FIGURE 1. Two vectors in a two-dimensional space.

and {D} shown in Figure 1, the calculation of their lengths would need to know the *components* of these vectors in the coordinate axes and then apply the *Pythagorean theorem*. Since the vector {R} has components equal to $r_x = 4$ and $r_y = 3$ units along the x- and y-axis, respectively, its length, here denoted with the symbol | |, is:

$$|\{R\}| = \left[r_x^2 + r_y^2\right]^{0.5} = \left[4^2 + 3^2\right]^{0.5} = 5 \tag{3}$$

To facilitate the calculation of the length of a generalized vector {V} which has N components, denoted as v_1 through v_N, its length is to be calculated with the following formula obtained from extending Equation 3 from two-dimensions to N-dimensions:

$$|\{V\}| = \left[v_1^2 + v_2^2 + \ldots + v_N^2\right]^{0.5} \tag{4}$$

For example, a three-dimensional vector has components $v_1 = v_x = 4$, $v_2 = v_y = 3$, and $v_3 = v_z = 12$, then the length of this vector is $|\{V\}| = [4^2 + 3^2 + 12^2]^{0.5} = 13$. We shall next show that Equation 4 can also be derived through the introduction of the multiplication rule and transposition of matrices.

1.2 MULTIPLICATION OF MATRICES

A matrix [A] of order L (rows) by M (columns) and a matrix [B] of order M by N can be multiplied in the order of [A][B] to produce a new matrix [P] of order L by N. [A][B] is said as [A] *post-multiplied* by [B], or, [B] *pre-multiplied* by [A]. The elements in [P] denoted as p_{ij} for i = 1 to N and j = 1 to M are to be calculated by the formula:

$$p_{ij} = \sum_{k=1}^{M} a_{ik} b_{kj} \tag{5}$$

Equation 5 indicates that the value of the element p_{ij} in the ith row and jth column of the product matrix [P] is to be calculated by multiplying the elements in the ith row of the matrix [A] by the corresponding elements in the jth column of the matrix [B]. It is therefore evident that the number of elements in the ith row of [A] should be equal to the number of elements in the jth column of [B]. In other words, to apply Equation 5 for producing a product matrix [P] by multiplying a matrix [A] on the right by a matrix [B] (or, to say multiplying a matrix [B] on the left by a matrix [A]), the number of columns of [A] should be equal to the number of row of [B]. A matrix [A] of order L by M can therefore be post-multiplied by a matrix [B] of order M by N; but [A] cannot be pre-multiplied by [B] unless L is equal to N!

As a numerical example, consider the case of a square, 3×3 matrix post-multiplied by a rectangular matrix of order 3 by 2. Since L = 3, M = 3, and N = 2, the product matrix is thus of order 3 by 2.

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\begin{bmatrix} 6 & -3 \\ 5 & -2 \\ 4 & -1 \end{bmatrix} = \begin{bmatrix} 1(6)+2(5)+3(4) & 1(-3)+2(-2)+3(-1) \\ 4(6)+5(5)+6(4) & 4(-3)+5(-2)+6(-1) \\ 7(6)+8(5)+9(4) & 7(-3)+8(-2)+9(-1) \end{bmatrix}$$

$$= \begin{bmatrix} 6+10+12 & -3-4-3 \\ 24+25+24 & -12-10-5 \\ 42+40+32 & -21-16-9 \end{bmatrix} = \begin{bmatrix} 28 & -10 \\ 73 & -27 \\ 114 & -46 \end{bmatrix}$$

More exercises are given in the Problems listed at the end of this chapter for the readers to practice on the matrix multiplications based on Equation 5.

It is of interest to note that the square of the length of a vector $\{V\}$ which has N components as defined in Equation 4, $|\{V\}|^2$, can be obtained by application of Equation 5 to $\{V\}$ and its transpose denoted as $\{V\}^T$ which is a row matrix of order 1 by N (one row and N columns). That is:

$$|\{V\}|^2 = \{V\}^T\{V\} = v_1^1 + v_2^2 + \ldots + v_3^2 \tag{6}$$

For a L-by-M matrix having elements e_{ij} where the row index i ranges from 1 to L and the column index j ranges from 1 to M, the transpose of this matrix when its elements are designated as t_{rc} will have a value equal to e_{cr} where the row index r ranges from 1 to M and the column index c ranges from 1 to M because this transpose matrix is of order M by L. As a numerical example, here is a pair of a 3×2 matrix [G] and its 2×3 transpose [H]:

$$\underset{3\times2}{[G]} = \begin{bmatrix} 6 & -3 \\ 5 & -2 \\ 4 & -1 \end{bmatrix} \quad \text{and} \quad \underset{2\times3}{[H]} = [G]^T = \begin{bmatrix} 6 & 5 & 4 \\ -3 & -2 & -1 \end{bmatrix}$$

If the elements of [G] and [H] are designated respectively as g_{ij} and h_{ij}, then $h_{ij} = g_{ji}$. For example, from above, we observe that $h_{12} = g_{21} = 5$, $h_{23} = g_{32} = -1$, and so on. There will be more examples of applications of Equations 5 and 6 in the ensuing sections and chapters.

Having introduced the transpose of a matrix, we can now conveniently revisit the addition of $\{D\}$ and $\{E\}$ in Figure 1 in algebraic form as $\{F\} = \{D\} + \{E\} = [4\ -3]^T + [-3\ 1]^T = [4+(-3)\ -3+1]^T = [1\ -2]^T$. The resulting sum vector is indeed correct as it is graphically verified in Figure 1. The saving of space by use of transposes of vectors (row matrices) is not evident in this case because all vectors are two-dimensional; imagine if the vectors are of much higher order.

Another noteworthy application of matrix multiplication and transposition is to reduce a system of linear algebraic equations into a simple, (or, should we say a single) *matrix equation*. For example, if we have three unknowns x, y, and z which are to be solved from the following three linear algebraic equations:

$$\begin{aligned} x+2y+3z&=4 \\ 5x+6y+7z&=8 \\ -2x-37&=9 \end{aligned} \tag{7}$$

Let us introduce two vectors, {V} and {R}, which contain the unknown x, y, and z, and the right-hand-side constants in the above three equations, respectively. That is:

$$\{V\}=[x \;\; y \;\; z]^T=\begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad \text{and} \quad \{R\}=[4 \;\; 8 \;\; 9]^T=\begin{bmatrix} 4 \\ 8 \\ 9 \end{bmatrix} \tag{8}$$

Then, making use of the multiplication rule of matrices, Equation 5, the system of linear algebraic equations, 7, now can be written simply as:

$$[C]\{V\}=\{R\} \tag{9}$$

where the *coefficient* matrix [C] formed by listing the coefficients of x, y, and z in first equation in the first row and second equation in the second row and so on. That is,

$$[C]=\begin{bmatrix} 1 & 2 & 3 \\ 5 & 6 & 7 \\ -2 & -3 & 0 \end{bmatrix}$$

There will be more applications of matrix multiplication and transposition in the ensuing chapters when we discuss how matrix equations, such as [C]{V} = {R}, can be solved by employing the Gaussian Elimination method, and how ordinary differential equations are approximated by finite differences will lead to the matrix equations. In the abbreviated matrix form, derivation and explanation of computational methods becomes much simpler.

Also, it can be observed from the expressions in Equation 8 how the transposition can be conveniently used to define the two vectors not using the column matrices which take more lines.

FORTRAN Version

Since Equations 1 and 2 require repetitive computation of the elements in the sum matrix [S] and difference matrix [D], machine could certainly help to carry out this laborous task particularly when matrices of very high order are involved. For covering all rows and columns of [S] and [D], looping or application of **DO** statement of the **FORTRAN** programming immediately come to mind. The following program is provided to serve as a first example for generating [S] and [D] of two given matrices [A] and[B]:

```
C      Program MatxAlgb.1 - Matrix addition and subtraction.
       DIMENSION A(3,3),B(3,3),D(3,3),S(3,3)
       Data A/1.,2.,3.,4.,5.,6.,7.,8.,9./,B/1.,2*2.,3*3.,3*4./
       DO 5 I=1,3
       DO 5 J=1,3
       S(I,J)=A(I,J)+B(I,J)
     5 D(I,J)=A(I,J)-B(I,J)
       WRITE (*,*) ' Matrix A'
       WRITE (*,10) ((A(I,J),J=1,3),I=1,3)
    10 FORMAT(3F5.1)
       WRITE (*,*) ' Matrix B'
       WRITE (*,10) ((B(I,J),J=1,3),I=1,3)
       WRITE (*,15) ((S(I,J),J=1,3),(D(I,J),J=1,3),I=1,3)
    15 FORMAT('  Matrix S',7X,'Matrix D'/(6F5.1))
       STOP
       END
```

The resulting display on the screen is:

```
 Matrix A
 1.0  4.0  7.0
 2.0  5.0  8.0
 3.0  6.0  9.0
 Matrix B
 1.0  3.0  4.0
 2.0  3.0  4.0
 2.0  3.0  4.0
 Matrix S          Matrix D
 2.0  7.0 11.0      .0  1.0  3.0
 4.0  8.0 12.0      .0  2.0  4.0
 5.0  9.0 13.0     1.0  3.0  5.0
```

To review **FORTRAN** briefly, we notice that matrices should be declared as variables with two subscripts in a DIMENSION statement. The displayed results of matrices A and B show that the values listed between // in a DATA statment will be filling into the first column and then second column and so on of a matrix. To instruct the computer to take the values provided but to fill them into a matrix row-by-row, a more explicit DATA needs to be given as:

DATA ((A(I,J),J = 1,3),I = 1,3)/1.,4.,7.,2.,5.,8.,3.,6.,9./

When a number needs to be repeated, the * symbol can be conveniently applied in the DATA statement exemplified by those for the matrix [B].

Some sample WRITE and FORMAT statements are also given in the program. The first * inside the parentheses of the WRITE statement when replaced by a number allows a device unit to be specified for saving the message or the values of the variables listed in the statement. * without being replaced means the monitor will be the output unit and consequently the message or the value of the variable(s) will be displayed on screen. The second * inside the parentheses of the WRITE statement if not replaced by a statement number, in which formats for printing the listed variables are specified, means "unformatted" and takes whatever the computer provides. For example, statement number 15 is a FORMAT statement used by the WRITE statement preceding it. There are 18 variables listed in that WRITE statement but only six F5.1 codes are specified. F5.1 requests five column spaces and one digit after the decimal point to be used to print the value of a listed variable. / in a FORMAT statement causes the print/display to begin at the first column of the next line. 6F5.1 is, however, enclosed by the inner pair of parentheses that allows it to be reused and every time it is reused the next six values will be printed or displayed on next line. The use (*,*) in a WRITE statement has the convenience of viewing the results and then making a hardcopy on a connected printer by pressing the *PrtSc* (Print Screen) key.

Interactive Operation

Program **MatxAlgb.1** only allows the two particular matrices having their elements specified in the DATA statement to be added and subtracted. For finding the sum matrix [S] and difference matrix [D] for any two matrices of same order N, we ought to upgrade this program to allow the user to enter from keyboard the order N and then the elements of the two matrices involved. This is *interactive* operation of the program and proper messages should be given to instruct the user what to do which means the program should be *user-friendly*. The program **MatxAlgb.2** listed below is an attempt to achieve that goal:

```
C   Program MatxAlgb.2 - Interactive matrix addition and subtraction.
      DIMENSION A(25,25),B(25,25),D(25,25),S(25,25)
      WRITE (*,*) 'Enter the order of the two matrices, N (<25) :'
      READ (*,*) N
      DO 3 I=1,N
      WRITE (*,2) I
    2 FORMAT(' Enter all elements of [A] in row',I3/
     *       '   then press RETURN/ENTER key!')
    3 READ (*,*) (A(I,J),J=1,N)
      DO 6 I=1,N
      WRITE (*,5) I
    5 FORMAT(' Enter all elements of [B] in row',I3/
     *       '   then press RETURN/ENTER key!')
    6 READ (*,*) (B(I,J),J=1,N)
      DO 7 I=1,N
      DO 7 J=1,N
      S(I,J)=A(I,J)+B(I,J)
    7 D(I,J)=A(I,J)-B(I,J)
      WRITE (*,*)
      WRITE (*,*) 'Matrix A'
      DO 8 I=1,N
    8 WRITE (*,10) I,(A(I,J),J=1,N)
   10 FORMAT(' Row ',I5/(5E15.5))
      WRITE (*,*)
      WRITE (*,*) 'Matrix B'
      DO 12 I=1,N
   12 WRITE (*,10) I,(B(I,J),J=1,N)
      WRITE (*,*)
      WRITE (*,*) 'Matrix S'
      DO 15 I=1,N
   15 WRITE (*,10) I,(S(I,J),J=1,N)
      WRITE (*,*)
      WRITE (*,*) 'Matrix D'
      DO 22 I=1,N
   22 WRITE (*,10) I,(D(I,J),J=1,N)
      STOP
      END
```

The interactive execution of the problem solved by the previous version **Matxalgb.1** now can proceed as follows:

```
Enter the order of the two matrices, N (<25) :
3
Enter all elements of [A] in row  1
  then press RETURN/ENTER key!
1,4,7
Enter all elements of [A] in row  2
  then press RETURN/ENTER key!
2,5,8
1nter all elements of [A] in row  3
  then press RETURN/ENTER key!
3,6,9
Enter all elements of [B] in row  1
  then press RETURN/ENTER key!
1,3,4
Enter all elements of [B] in row  2
  then press RETURN/ENTER key!
2,3,4
Enter all elements of [B] in row  3
  then press RETURN/ENTER key!
2,3,4

Matrix A
Row     1
    .10000E+01     .40000E+01     .70000E+01
Row     2
    .20000E+01     .50000E+01     .80000E+01

Row     3
    .30000E+01     .60000E+01     .90000E+01
```

```
Matrix B
Row      1
     .10000E+01     .30000E+01     .40000E+01
Row      2
     .20000E+01     .30000E+01     .40000E+01
Row      3
     .20000E+01     .30000E+01     .40000E+01

Matrix S
Row      1
     .20000E+01     .70000E+01     .11000E+02
Row      2
     .40000E+01     .80000E+01     .12000E+02
Row      3
     .50000E+01     .90000E+01     .13000E+02

Matrix D
Row      1
     .00000E+01     .10000E+01     .30000E+01
Row      2
     .00000E+01     .20000E+01     .40000E+01
Row      3
     .10000E+01     .30000E+01     .50000E+01
```

The results are identical to those obtained previously. The READ statement allows the values for the variable(s) to be entered via keyboard. A WRITE statement has no variable listed serves for need of skipping a line to provide better readability of the display. Also the I and E format codes are introduced in the statement 10. Iw where w is an integer in a FORMAT statement requests w columns to be provided for displaying the value of the integer variable listed in the WRITE statement, in which the FORMAT statement is utilized. Ew.d where w and d should both be integer constants requests w columns to be provided for display a real value in the scientific form and carrying d digits after the decimal point. Ew.d format gives more feasibility than Fw.d format because the latter may cause an *error message* of insufficient width if the value to be displayed becomes too large and/or has a negative sign.

More Programming Review

Besides the operation of matrix addition and subtraction, we have also discussed about the transposition and multiplication of matrices. For further review of computer programming, it is opportune to incorporate all these matrix algebraic operations into a single interactive program. In the listing below, three subroutines for matrix addition and subtraction, transposition, and multiplication named as **MatrixSD**, **Transpos**, and **MatxMtpy**, respectively, are created to support a program called **MatxAlgb** (Matrix Algebra).

```
C   Program MatxAlgb - Interactive matrix addition and subtraction,
C                        transposition, and multiplication.
      DIMENSION A(25,25),AT(25,25),B(25,25),P(25,25),SorD(25,25)
      WRITE (*,50)
   50 FORMAT(' Enter -1/1/2/3 for matrix subtraction/addition/',
     *       'transposition/multiplication :')
      READ (*,*) K
      KP2=K+2
      GO TO (100,100,100,200,300) KP2
C     Matrix addition or subtraction.
  100 WRITE (*,103)
  103 FORMAT(' Enter the order of the two matrices [A] & [B],'
     *       ' M & N (both<26) :')
      READ (*,*) M,N
      DO 107 I=1,M
      WRITE (*,105) I
  105 FORMAT(' Enter all elements of [A] in row',I3/
     *       '   then press RETURN/ENTER key!')
  107 READ (*,*) (A(I,J),J=1,N)
      DO 126 I=1,M
      WRITE (*,121) I
  121 FORMAT(' Enter all elements of [B] in row',I3/
     *       '   then press RETURN/ENTER key!')
  126 READ (*,*) (B(I,J),J=1,N)
      CALL MatrixSD(A,B,25,25,M,N,K,SorD)
      IF (K) 131,131,145
  131 WRITE (*,*) 'Matrix [A]-[B]'
  135 DO 138 I=1,M
  138 WRITE (*,141) I,(SorD(I,J),J=1,N)
  141 FORMAT(' Row ',I5/(5E15.5))
      GO TO 500
  145 WRITE (*,*) 'Matrix [A]+[B]'
      GOTO 135
C     Matrix transposition.
  200 WRITE (*,203)
      GO TO 500
C     Matrix multiplication.
  300 WRITE (*,303)
  303 FORMAT(' To find P(L,N)=A(L,M)B(M,N), first enter L, M,'
     *       ' and N (all<26) : ')
      READ (*,*) L,M,N
      DO 307 I=1,L
      WRITE (*,105) I
  307 READ (*,*) (A(I,J),J=1,M)
      DO 326 I=1,M
      WRITE (*,121) I
  326 READ (*,*) (B(I,J),J=1,N)
      CALL MatxMtpy(A,B,25,25,25,L,M,N,P)
  331 WRITE (*,*) 'Matrix [P]'
      DO 338 I=1,L
  338 WRITE (*,141) I,(P(I,J),J=1,N)
  500 STOP
      END

      SUBROUTINE MATXMTPY(A,B,Lmax,Mmax,Nmax,L,M,N,P)
C     Matrix multiplication of P(L,N)=A(L,M)B(M,N).
C     Input arguments: A, B, L, M, and N.
C     Output argument: P
      DIMENSION A(Lmax,Mmax),B(Mmax,Nmax),P(Lmax,Nmax)
      DO 5 I=1,L
      DO 5 J=1,N
      P(I,J)=0.
      DO 5 K=1,M
      5 P(I,J)=P(I,J)+A(I,K)*B(K,J)
        RETURN
        END

        SUBROUTINE TRANSPOS(A,Mmax,Nmax,M,N,AT)
C       Finds the transpose AT(N,M) for a given matrix A(M,N).
        DIMENSION A(Mmax,Nmax),AT(Nmax,Mmax)
        DO 5 I=1,M
        DO 5 J=1,N
      5 AT(J,I)=A(I,J)
        RETURN
        END

        SUBROUTINE MATRIXSD(A,B,Mmax,Nmax,M,N,K,SORD)
C       Finds the sum matrix SORD(M,N)=A(M,N)+B(M,N) when K=1;
C         or the difference matrix SORD(M,N)=A(M,N)-B(M,N) when K=-1.
        DIMENSION A(Mmax,Nmax),B(Mmax,Nmax),SORD(Mmax,Nmax)
        DO 5 I=1,M
        DO 5 J=1,N
      5 SORD(I,J)=A(I,J)+K*B(I,J)
        RETURN
        END
```

The above program shows that Subroutines are independent units all started with a SUBROUTINE statement which includes a name followed by a pair of parentheses enclosing a number of *arguments*. The Subroutines are called in the main program by specifying which variables or constants should serve as arguments to connect to the subroutines. Some arguments provide input to the subroutine while other arguments transmit out the results determined by the subroutine. These are referred to as *input arguments* and *output arguments*, respectively. In many instances, an argument may serve a dual role for both input and output purposes. To construct as an independent unit, a subprogram which can be in the form of a SUBROUTINE, or **FUNCTION** (to be elaborated later) must have RETURN and END statements.

It should also be remarked that program **MatxAlgb** is arranged to handle any matrix having an order of no higher than 25 by 25. For this restriction and for having the flexibility of handling any matrices of lesser order, the Lmax, Mmax, and Nmax arguments are added in all three subroutines in order not to cause any mismatch of matrix sizes between the main program and the called subroutine when dealing with any L, M, and N values which are interactively entered via keyboard.

Computed GOTO and arithmetic IF statements are also introduced in the program **MatxAlgb**. GOTO (i,j,k,…) C will result in going to (execute) the statement numbered i, j, k, and so on when C has a value equal to 1, 2, 3, and so on, respectively. IF (Expression) a,b,c will result in going to the statement numbered a, b, or c if the value calculated by the expression or a single variable is less than, equal to, or, greater than zero, respectively.

It is important to point out that in describing any derived procedure of numerical computation, *indicial notation* such as Equation 5 should always be preferred to facilitate programming. In that notation, the indices are directly used, or, literally translated into the index variables for the DO loops as can be seen in Subroutine MatxMtpy which is developed according to Equation 5. Subroutine MatrixSD is another example of literally translating Equations 1 and 2. For defining the values of the element in the following *tri-diagonal band matrix:*

$$[C] = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 \\ -3 & 1 & 2 & 0 & 0 \\ 0 & -3 & 1 & 2 & 0 \\ 0 & 0 & -3 & 1 & 2 \\ 0 & 0 & 0 & -3 & 1 \end{bmatrix}$$

we ought not to write 25 separate statements for the 25 elements in this matrix but derive the indicial formulas for i,j = 1 to 5:

$$c_{ij} = 0, \quad \text{if } j > i + 2, \text{or}, j < i - 2$$

$$c_{i,i+1} = 2,$$

and

$$c_{i,i-1} = -3$$

Then, the matrix [C] can be generated with the DO loops as follows:

```
      DO 5 I=1,5
         DO 5 J=1,5
            C(I,J)=0.
            IF (J.EQ.I)       C(I,J)=1.
            IF (J.EQ.(I+1))   C(I,J)=2.
            IF (J.EQ.(I-1))   C(I,J)=-3.
    5 CONTINUE
```

The above short program also demonstrates the use of the **CONTINUE** statement for ending the DO loop(s), and the **logical IF** statements. The *true*, or, *false* condition of the expression inside the outer pair of parentheses directs the computer to execute the statement following the parentheses or the next statement immediately below the current IF statement. Reader may want to practice on deriving indicial formulas and then write a short program for calculating the elements of the matrix:

$$[M] = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 2 & 1 & 0 & 0 & 0 & 0 & 0 \\ 4 & 3 & 2 & 1 & 0 & 0 & 0 & 0 \\ 5 & 4 & 3 & 2 & 1 & 0 & 0 & 0 \\ 6 & 5 & 4 & 3 & 2 & 1 & 0 & 0 \\ 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ 8 & 7 & 6 & 5 & 4 & 3 & 2 & 1 \end{bmatrix} \qquad (10)$$

As another example of writing a computer program based on indicial notation, consider the case of calculating e^x based on the infinite series:

$$\begin{aligned} e^x &= 1 + \frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^2}{3!} + \ldots + \frac{x^i}{i!} + \ldots \\ &= \sum_{i=0}^{\infty} \frac{x^i}{i!} \end{aligned} \qquad (11)$$

With the understanding that 0! = 1, we have expressed the series as a summation involving the index i which ranges from zero to infinity. A FUNCTION **ExpoFunc** can be developed for calculating e^x based on Equation 11 and taking only a finite number of terms for a partial sum of the series when the contribution of additional term is less than certain percentage of the sum in magnitude, say 0.001%. This FUNCTION may be arranged as:

```
      FUNCTION ExpoFunc(X)
C     Calculates EXP(X)
      ExpoFunc=1.
      NT=1
      UP=X
      FACTO=1.
    5 TERM=UP/FACTO
      IF (ABS(TERM).LT.0.00001*ABS(ExpoFunc)) GOTO 999
      ExpoFunc=ExpoFunc+TERM
      NT=NT+1
      FACTO=FACTO*NT
      UP=UP*X
      GOTO 5
  999 RETURN
      END
```

To further show the advantage of adopting vector and matrix notation, here let us apply FUNCTION **ExpoFunc** to examine the surface $z(x,y) = e^{x+y}$ above the rectangular area $0 \le x \le 2.0$ and $0 \le y \le 1.5$. The following program, ExpTest, will enable us to compare the values of e^{x+y} generated by the FUNCTION **ExpoFunc** and by the function **EXP** available in the **FORTRAN** library (hence called *library function*).

```
C     Program ExpTest - Application of FUNCTION ExpoFunc for z=e**(x+y).
      DIMENSION X(5),Y(4),Z(4,5),ZF(4,5)
      WRITE (*,1)
    1 FORMAT(20X,'Z using ExpoFunc',25X,'Z using EXP'/)
      DO 3 J=1,4
    3 Y(J)=(J-1)*0.5
      WRITE (*,5) ((Y(J),J=1,4),I=1,2)
    5 FORMAT(7X,4(' Y=',F3.1,3X),3X,4(' Y=',F3.1,3X))
      DO 9 I=1,5
      X(I)=(I-1)*0.5
      DO 7 J=1,4
```

```
   Z(I,J)=ExpoFunc(X(I))*ExpoFunc(Y(J))
  7 ZF(I,J)=EXP(X(I))*EXP(Y(J))
  9 WRITE (*,10) X(I),(Z(I,J),J=1,4),(ZF(I,J),J=1,4)
 10 FORMAT(' X=',F3.1,4F9.5,3X,4F9.5)
   STOP
   END
```

The resulting printout is:

```
                  Z using ExpoFunc                                Z using EXP

          Y= .0     Y=0.5     Y=1.0     Y=1.5         Y= .0     Y=0.5     Y=1.0     Y=1.5

 X= .0   1.00000   1.64872   2.71825   4.48167       1.00000   1.64872   2.71828   4.48169

 X=0.5   1.64872   2.71828   4.48164   7.38902       1.64872   2.71828   4.48169   7.38906

 X=1.0   2.71825   4.48164   7.38891  12.18232       2.71828   4.48169   7.38906  12.18249

 X=1.5   4.48167   7.38902  12.18232  20.08537       4.48169   7.38906  12.18249  20.08554

 X=2.0   7.38900  12.18238  20.08517  33.11504       7.38906  12.18249  20.08554  33.11545
```

It is apparent that two approaches produce almost identical results, so the 0.001% accuracy appears quite adequate for the x and y ranges studied. Also, arranging the results in vector and matrix forms make the presentation much easy to comprehend.

We have experienced how the summation process for an indicial formula involving a Σ should be programmed. Another operation symbol of importance is Π which is for multiplication of many factors. That is:

$$\prod_{i=1}^{N} a_i = a_1 a_2 \dots a_N \qquad (12)$$

An obvious application of Equation 12 is for the calculation of factorials. For example, 5! = Πi for i ranges from 1 to 5. As an exercise, we display the values of 1! through 50! with the following program involving a subroutine IFACTO which calculates I! for a specified I value:

```
C  Program FT - Factorial Table
        WRITE (*,*) ' I    I!'
        WRITE (*,*)
        DO 5 I=1,30
        CALL IFACTO(I,RIF)
        WRITE (*,3) I,RIF
      3 FORMAT(I3,E15.7)
        STOP
        END
      SUBROUTINE IFACTO(I,RIF)
C   Calculates the I factorial and gives answer in real form as
C     RIF.
      RIF=1.
      DO 7 K=1,I
    7 RIF=RIF*K
      RETURN
      END
```

The resulting print out is (listed in three columns for saving space)

```
I   I!
1   .1000000E+01    7   .5040000E+04   13   .6227021E+10   19   .1216451E+18   25   .1551121E+26
2   .2000000E+01    8   .4032000E+05   14   .8717829E+11   20   .2432902E+19   26   .4032915E+27
3   .6000000E+01    9   .3628800E+06   15   .1307674E+13   21   .5109094E+20   27   .1088887E+29
4   .2400000E+02   10   .3628800E+07   16   .2092279E+14   22   .1124001E+22   28   .3048884E+30
5   .1200000E+03   11   .3991680E+08   17   .3556874E+15   23   .2585202E+23   29   .8841763E+31
6   .7200000E+03   12   .4790016E+09   18   .6402374E+16   24   .6204485E+24   30   .2653539E+33
```

Another application of Equation 12 is for calculation of the *binomial coefficients* for a real number r and an integer k defined as:

$$\binom{r}{k} = \frac{r(r-1)(r-2)\ldots(r-k+1)}{k!} = \prod_{i=1}^{k} \frac{r-i+1}{i} \tag{13}$$

We shall have the occasion of applying Equations 12 and 13 when the finite differences and Lagrangian interpolation are discussed.

Sample Applications

Program **MatxAlgb** has been tested interactively, the following are the resulting displays of four test cases:

```
Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
-1
Enter the order of the two matrices [A] & [B], M & N
(both<26) :
2,2
Enter all elements of [A] in row  1
  then press RETURN/ENTER key!
1,2
Enter all elements of [A] in row  2
  then press RETURN/ENTER key!
3,4
Enter all elements of [B] in row  1
  then press RETURN/ENTER key!
5,6
```

```
Enter all elements of [B] in row  2
  then press RETURN/ENTER key!
7,8
Matrix [A]-[B]
Row     1
   -.40000E+01    -.40000E+01
Row     2
   -.40000E+01    -.40000E+01
Stop - Program terminated.
Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
1
Enter the order of the two matrices [A] & [B], M & N
(both<26) :
2,2
Enter all elements of [A] in row  1
  then press RETURN/ENTER key!
1,2
Enter all elements of [A] in row  2
  then press RETURN/ENTER key!
3,4
Enter all elements of [B] in row  1
  then press RETURN/ENTER key!
5,6
Enter all elements of [b] in row  2
  then press RETURN/ENTER key!
7,8
Matrix [A]+[B]
Row     1
    .60000E+01     .80000E+01
Row     2
    .10000E+02     .12000E+02
Stop - Program terminated.

Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
2
Enter the order of the two matrices [A] & [B], M & N
(both<26) :
2,3
Enter all elements of [A] in row  1
  then press RETURN/ENTER key!
1,2,3
Enter all elements of [A] in row  2
  then press RETURN/ENTER key!
4,5,6
Transpose of [A]
Row     1
    .10000E+01     .40000E+01
```

```
Row      2
    .20000E+01     .50000E+01
Row      3
    .30000E+01     .60000E+01
Stop - Program terminated.

Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
3
To find P(L.N)=A(L,M)B(M,N), first enter L, M, and N
(all<26) :
2,3,2
Enter all elements of [A] in row  1
  then press RETURN/ENTER key!
1,2,3
Enter all elements of [A] in row  2
  then press RETURN/ENTER key!
4,5,6
Enter all elements of [B] in row  1
  then press RETURN/ENTER key!
1,2
Enter all elements of [B] in row  2
  then press RETURN/ENTER key!
2,3
Enter all elements of [B] in row  3
  then press RETURN/ENTER key!
3,4
Matrix [P]
Row      1
    .14000E+02     .20000E+02
Row      2
    .32000E+02     .47000E+02
Stop - Program terminated.
```

QuickBASIC Version

The **QuickBASIC** language has the advantage over the **FORTRAN** language for making quick changes and then running the revised program without compilation. Furthermore, it offers simple plotting statements. Let us have a **QuickBASIC** version of the program **MatxAlgb** and then discuss its basic features.

```
'   Program MatxAlgb - Interactive matrix addition and subtraction,
'                      transposition, and multiplication.
     DECLARE SUB MatrixSD (A(), B(), M, N, K, SorD())
     DECLARE SUB Transpos (C(), M, N, CT())
     DECLARE SUB MatxMtpy (D(), E(), L, M, N, P())
     PRINT "Enter -1/1/2/3 for matrix subtraction/addition/";
     PRINT "transposition/multiplication :"
     INPUT K
     IF (K = -1) THEN 100
     IF (K = 1) THEN 100
     IF (K = 2) THEN 200
     GOTO 300
```

```
'       Matrix addition or subtraction.
100     PRINT "Enter the order of the two matrices [A] & [B], M & N :"
        INPUT M, N: DIM A(M, N), B(M, N), SorD(M, N)
        FOR I = 1 TO M
            PRINT "Enter all elements of [A] in row"; I
            PRINT "  then press RETURN/ENTER key after entering each number!"
            FOR J = 1 TO N: INPUT ; A(I, J): NEXT J: PRINT : NEXT I
        FOR I = 1 TO M
            PRINT "Enter all elements of [B] in row"; I
            PRINT "  then press RETURN/ENTER key after entering each number!"
            FOR J = 1 TO N: INPUT ; B(I, J): NEXT J: PRINT : NEXT I
        CALL MatrixSD(A(), B(), M, N, K, SorD())
        IF (K < 0) THEN 131 ELSE 145
131     PRINT "Matrix [A]-[B]"
135     FOR I = 1 TO M
            PRINT "Row "; I
            FOR J = 1 TO N: PRINT USING "  #.####^^^^"; SorD(I, J); : NEXT J
            PRINT : NEXT I: GOTO 500
145     PRINT "Matrix [A]+[B]": GOTO 135
'       Matrix transposition.
200     PRINT "Enter the order of the matrix [C], M & N :"
        INPUT M, N: DIM C(M, N), CT(N, M)
        FOR I = 1 TO M
            PRINT "Enter all elements of [C] in row"; I
            PRINT "  then press RETURN/ENTER key after entering each number!"
            FOR J = 1 TO N: INPUT ; C(I, J): NEXT J: PRINT : NEXT I
        CALL Transpos(C(), M, N, CT()): PRINT "Transpose of [C]"
        FOR I = 1 TO N
            PRINT "Row "; I
            FOR J = 1 TO M: PRINT USING "  #.####^^^^"; CT(I, J); : NEXT J
            PRINT : NEXT I: GOTO 500
'       Matrix multiplication.
300     PRINT "To find P(L,N)=D(L,M)E(M,N), first enter L, M, and N :"
        INPUT L, M, N: DIM D(L, M), E(M, N), P(L, N)
        FOR I = 1 TO L
            PRINT "Enter all elements of [D] in row"; I
            PRINT "  then press RETURN/ENTER key after entering each number!"
            FOR J = 1 TO M: INPUT ; D(I, J): NEXT J: PRINT : NEXT I
        FOR I = 1 TO M
            PRINT "Enter all elements of [E] in row"; I
            PRINT "  then press RETURN/ENTER key after entering each number!"
            FOR J = 1 TO N: INPUT ; E(I, J): NEXT J: PRINT : NEXT I
        CALL MatxMtpy(D(), E(), L, M, N, P())
331     PRINT "Matrix [P]"
        FOR I = 1 TO L
            PRINT "Row "; I
            FOR J = 1 TO N: PRINT USING "  #.####^^^^"; P(I, J); : NEXT J
            PRINT : NEXT I
500     END

        SUB MatrixSD (A(), B(), M, N, K, SorD())
'       Finds the sum matrix SORD(M,N)=A(M,N)+B(M,N) when K=1;
'         or the difference matrix SORD(M,N)=A(M,N)-B(M,N) when K=-1.
        FOR I = 1 TO M
            FOR J = 1 TO N
                SorD(I, J) = A(I, J) + K * B(I, J): NEXT J: NEXT I
        END SUB

        SUB MatxMtpy (A(), B(), L, M, N, P())
'       Matrix multiplication of P(L,N)=A(L,M)B(M,N).
'       Input arguments: A, B, L, M, and N.
'       Output argument: P
        FOR I = 1 TO L
            FOR J = 1 TO N
                P(I, J) = 0!
                FOR K = 1 TO M
                    P(I, J) = P(I, J) + A(I, K) * B(K, J)
                    NEXT K: NEXT J: NEXT I
        END SUB
```

```
SUB Transpos (A(), M, N, AT())
'     Finds the transpose AT(N,M) for a given matrix A(M,N).
      FOR I = 1 TO M
          FOR J = 1 TO N
              AT(J, I) = A(I, J): NEXT J: NEXT I
END SUB
```

Notice that the order limit of 25 needed in the **FORTRAN** version is removed in the **QuickBASIC** version which allows the dim statement to be adjustable. ' is replacing C in **FORTRAN** to indicate a comment statement in **QuickBASIC**. READ and WRITE in **FORTRAN** are replaced by INPUT and PRINT in **QuickBASIC**, respectively. The DO loop in **FORTRAN** is replaced by the FOR and NEXT pair in **QuickBASIC**.

Sample Applications

When the four cases previously run by the **FORTRAN** version are executed by the **QuickBASIC** version, the screen prompting messages, the interactively entered data, and the computed results are:

```
Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
? -1
Enter the order of the two matrices [A] & [B], M & N :
? 2,2
Enter all elements of [A] row 1
  then press RETURN/ENTER key after entering each
number!
? 1? 2
Enter all elements of [A] row 2
  then press RETURN/ENTER key after entering each
number!
? 3? 4

Enter all elements of [B] row 1
  then press RETURN/ENTER key after entering each
number!
? 5? 6
Enter all elements of [B] row 2
  then press RETURN/ENTER key after entering each
number!
? 7? 8
Matrix [A]-[B]
Row  1
  -.4000E+01  -.4000E+01
Row  2
  -.4000E+01  -.4000E+01

Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
```

```
? 1
Enter the order of the two matrices [A] & [B], M & N :
? 2,2
Enter all elements of [A] row 1
  then press RETURN/ENTER key after entering each
number!
? 1? 2
Enter all elements of [A] row 2
  then press RETURN/ENTER key after entering each
number!
? 3? 4
Enter all elements of [B] row 1
  then press RETURN/ENTER key after entering each
number!
? 5? 6
Enter all elements of [B] row 2
  then press RETURN/ENTER key after entering each
number!
? 7? 8
Matrix [A]+[B]
Row  1
   .6000E+01   .8000E+01
Row  2
   .1200E+02   .1200E+02

Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
? 2
Enter the order of the matrix [C], M & N :
? 2,3
Enter all elements of [C] row 1
  then press RETURN/ENTER key after entering each
number!
? 1? 2? 3
Enter all elements of [C] row 2
  then press RETURN/ENTER key after entering each
number!
? 4? 5? 6
Transpose of [C]
Row  1
   .1000E+01   .4000E+01
Row  2
   .2000E+01   .5000E+01
Row  1
   .3000E+01   .6000E+01

Enter -1/1/2/3 for matrix
subtraction/addition/transposition/multiplication :
? 3
To find P(L,N)=D(L.M)E(M,N), first enter L, M, and N :
? 2,3,2
```

```
Enter all elements of [D] row 1
  then press RETURN/ENTER key after entering each
number!
? 1? 2? 3
Enter all elements of [D] row 2
  then press RETURN/ENTER key after entering each
number!
? 4? 5? 6
Enter all elements of [E] row 1
  then press RETURN/ENTER key after entering each
number!
? 1? 2
Enter all elements of [E] row 2
  then press RETURN/ENTER key after entering each
number!
? 2? 3
Enter all elements of [E] row 3
  then press RETURN/ENTER key after entering each
number!
? 3? 4
Matrix [P]
Row  1
  0.1400E+02  0.2000E+02
Row  2
  0.3200E+02  0.4700E+02
```

MATLAB Applications

MATLAB developed by the Mathworks, Inc. offers a quick tool for matrix manipulations. To *load* **MATLAB** after it has been set-up and stored in a subdirectory of a hard drive, say C, we first switch to this subdirectory by entering (followed by pressing ENTER)

```
C:\cd MATLAB
```

and then switch to its own subdirectory BIN by entering (followed by pressing ENTER)

```
C:\MATLAB>cd BIN
```

Next, we type MATLAB to obtain a display of:

```
C:\MATLAB>BIB>MATLAB
```

Pressing the ENTER key results in a display of:

```
>>
```

which indicates **MATLAB** is ready to begin. Let us rerun the cases of matrix subtraction, addition, transposition, and multiplication previously considered in the **FORTRAN** and **QuickBASIC** versions. First, we enter the matrix [A] in the form of:

```
>> A = [1,2;3,4]
```

When the ENTER key is pressed, the displayed result is:

```
A =
   1   2
   3   4
```

Notice that the elements of [A] should be entered row by row. While the rows are separated by ;, in each row elements are separated by comma. After the print out of the above results, >> sign will again appear. To eliminate the unnecessary line space (between A = and the first row 1 2), the statement **format compact** can be entered as follows (the phrase "pressing ENTER key" will be omitted from now on):

```
>> format compact, B = [5,6;7,8]
B =
   5   6
   7   8
```

Notice that *comma* is used to separate the statements. To demonstrate matrix subtraction and addition, we can have:

```
>> A-B
ans =
   –4   –4
   –4   –4
>> A + B
ans =
   6     8
   10   12
```

To apply **MATLAB** for transposition and multiplication of matrices, we can have:

```
>> C = [1,2,3;4,5,6]
C =
   1   2   3
   4   5   6
>> C'
ans =
   1   4
   2   5
   3   6
```

```
>> D = [1,2,3;4,5,6]; E = [1,2;2,3;3,4]; P = D*E

P =

   14   20

   32   47
```

Notice that **MATLAB** uses ' (*single quote*) in place of the superscripted symbol T for transposition. When ; (*semi-colon*) follows a statement such as the D statement, the results will *not* be displayed. As in **FORTRAN** and **QuickBASIC**, * is the multiplication operator as is used in P = D*E, here involving three matrices not three single variables. More examples of **MATLAB** applications including plotting will ensue. To *terminate* the **MATLAB** operation, simply enter *quit* and then the RETURN key.

Mathematica Applications

To *commence* the service of **Mathematica** from **Windows** setup, simply point the mouse to it and double click the left button. The **Input** display bar will appear on screen, applications of **Mathematica** can start by entering commands from keyboard and then press the **Shift** and **Enter** keys. To *terminate* the **Mathematica** application, enter **Quit[]** from keyboard and then press the **Shift** and **Enter** keys.

Mathematica commands, statements, and functions are gradually introduced and applied in increasing degree of difficulty. Its graphic capabilities are also utilized in presentation of the computed results.

For matrix operations, **Mathematica** can compute the sum and difference of two matrices of same order in symbolic forms, such as in the following cases of involving two matrices, A and B, both of order 2 by 2:

```
In[1]: = A = {{1,2},{3,4}}; MatrixForm[A]

Out[1]//MatrixForm =

   1   2

   3   4
```

In[1]: = is shown on screen by **Mathematica** while user types in A = {{1,2},{3,4}}; MatrixForm[A]. Notice that braces are used to enclose the elements in each row of a matrix, the elments in a same row are separated by commas, and the rows are also separated by commas. MatrixForm demands that the matrix be printed in a matrix form. *Out[1]//MatrixForm* = and the rest are response of **Mathematica**.

```
In[2]: = B = {{5,6},{7,8}}; MatrixForm[B]

Out[2]//MatrixForm =

    5   6
    7   8
```

In[3]: = MatrixForm[A + B]

Out[3]//MatrixForm =

```
6    8
10   12
```

In[4]: = Dif = A-B; MatrixForm[Dif]

Out[4]//MatrixForm =

```
–4  –4
–4  –4
```

In[3] and In[4] illustrate how matrices are to be added and subtracted, respectively. Notice that one can either use A + B directly, or, create a variable Dif to handle the *sum* and *difference* matrices.

Also, **Mathematica** has a function called **Transpose** for transposition of a matrix. Let us reuse the matrix A to demonstrate its application:

In[5]: = AT = Transpose[A]; MatrixForm[AT]

Out[5]//MatrixForm =

```
1  3

2  4
```

1.3 SOLUTION OF MATRIX EQUATION

Matrix notation offers the convenience of organizing mathematical expression in an orderly manner and in a form which can be directly followed in coding it into programming languages, particularly in the case of repetitive computation involving the *looping* process. The most notable situation is in the case of solving a system of linear algebraic equation. For example, if we need to determine a linear equation $y = a_1 + a_2x$ which geometrically represents a straight line and it is required to pass through two specified points (x_1,y_1) and (x_2,y_2). To find the values of the coefficients a_1 and a_2 in the y equation, two equations can be obtained by substituting the two given points as:

$$(1)a_1 + (x_1)a_2 = y_1 \tag{1}$$

and

$$(1)a_1 + (x_2)a_2 = y_2 \tag{2}$$

To facilitate programming, it is advantageous to write the above equations in matrix form as:

$$[C]\{A\}=\{Y\} \tag{3}$$

where:

$$[C]=\begin{bmatrix}1 & x_1\\ 1 & x_2\end{bmatrix},\{A\}=\begin{bmatrix}a_1\\ a_2\end{bmatrix}, \text{ and } \{Y\}=\begin{bmatrix}y_1\\ y_2\end{bmatrix} \tag{4}$$

The matrix equation 3 in this case is of small order, that is an order of 2. For small systems, Cramer's Rule can be conveniently applied which allows the unknown vector $\{A\}$ to be obtained by the formula:

$$\{A\}=\left[\left|[c_1]\right| \; \left|[c_2]\right|\right]^T/\left|[C]\right| \tag{5}$$

Equation 5 involves the calculation of three determinants, i.e., , $|[C_1]|$, $|[C_2]|$, and $|[C]|$ where $[C_1]$ and $[C_2]$ are matrices derived from the matrix [C] when the first and second columns of [C] are replaced by $\{Y\}$, respectively. If we denote the elements of a general matrix [C] of order 2 by c_{ij} for i,j = 1,2, the determinant of [C] by definition is:

$$|[C]|=c_{11}c_{22}-c_{12}c_{21} \tag{6}$$

The general definition of the determinant of a matrix [M] of order N and whose elements are denoted as m_{ij} for i,j = 1,2,…,N is to add all possible product of N elements selected one from each row but from different column. There are N! such products and each product carries a positive or negative sign depending on whether even or odd number of exchanges are necessary for rearranging the N subscripts in increasing order. For example, in Equation 6, c_{11} is selected from the first row and first column of [C] and only c_{22} can be selected and multiplied by it while the other possible product is to select c_{12} from the second row and first column of [C] and that leaves only c_{21} from the second row and first column of [C] available as a factor of the second product. In order to arrange the two subscripts in non-decreasing order, one exchange is needed and hence the product $c_{12}c_{21}$ carries a minus sign. We shall explain this sign convention further when a matrix of order 3 is discussed. However, it should be evident here that a matrix whose order is large the task of calculating its determinant would certainly need help from computer. This will be the a topic discussed in Section 1.5.

Let us demonstrate the application of Cramer's Rule by having a numerical case. If the two given points to be passed by the straight line $y = a_1 + a_2x$ are $(x_1,y_1) = (1,2)$ and $(x_2,y_2) = (3,4)$. Then we can have:

$$\|[C]\| = \begin{vmatrix} 1 & x_1 \\ 1 & x_2 \end{vmatrix} = \begin{vmatrix} 1 & 1 \\ 1 & 3 \end{vmatrix} = 1\times 3 - 1\times 1 = 3 - 1 = 2$$

$$\|[c_1]\| = \begin{vmatrix} y_1 & x_1 \\ y_2 & x_2 \end{vmatrix} = \begin{vmatrix} 2 & 1 \\ 4 & 3 \end{vmatrix} = 2\times 3 - 1\times 4 = 6 - 4 = 2$$

and

$$\|[C_2]\| = \begin{vmatrix} 1 & y_1 \\ 1 & y_2 \end{vmatrix} = \begin{vmatrix} 1 & 2 \\ 1 & 4 \end{vmatrix} = 1\times 4 - 2\times 1 = 4 - 2 = 2$$

Consequently, according to Equation 5 we can find the coefficients in the straight-line equation to be:

$$a_1 = \|[C_1]\| / \|[C]\| = 2/2 = 1 \text{ and } a_2 = \|[C_2]\| / \|[C]\| = 2/2 = 1$$

Hence, the line passing through the points (1,2) and (3,4) is $y = a_1 + a_2x = 1 + x$.

Application of Cramer's Rule can be extended for solving three unknowns from three linear algebraic equations. Consider the case of finding a plane which passes three points (x_i,y_i) for i = 1 to 3. The equation of that plane can first be written as $z = a_1 + a_2x + a_3y$. Similar to the derivation of Equation 3, here we substitute the three given points into the z equation and obtain:

$$(1)a_1 + (x_1)a_2 + (y_1)a_3 = z_1 \tag{7}$$

$$(1)a_1 + (x_2)a_2 + (y_2)a_3 = z_2 \tag{8}$$

and

$$(1)a_1 + (x_3)a_2 + (y_3)a_3 = z_3 \tag{9}$$

Again, the above three equations can be written in matrix form as:

$$[C]\{A\} = \{Z\} \tag{10}$$

where the matrix [C] and the vector {A} previously defined in Equation 4 need to be reexpanded and redefined as:

$$[C] = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix}, \{A\} = \begin{vmatrix} a_1 \\ a_2 \\ a_3 \end{vmatrix}, \text{ and } \{Z\} = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \end{bmatrix} \tag{11}$$

And, the Cramer's Rule for solving Equation 10 can be expressed as:

$$\{A\} = \left[\left|[C_1]\right| \left|[C_2]\right| \left|[C_3]\right| \right]^T / \left|[C]\right| \tag{12}$$

where $[C_i]$ for i = 1 to 3 for matrices formed by replacing the ith column of the matrix [C] by the vector {Z}, respectively. Now, we need the calculation of the determinant of matrices of order 3. If we denote the element in a matrix [M] as m_{ij} for i,j = 1 to 3, the determinant of [M] can be calculated as:

$$\begin{aligned} \left|[M]\right| = m_{11}m_{22}m_{33} - m_{11}m_{23}m_{32} + m_{12}m_{23}m_{31} \\ - m_{12}m_{21}m_{33} + m_{13}m_{21}m_{32} - m_{13}m_{22}m_{31} \end{aligned} \tag{13}$$

To give a numerical example, let us consider a plane passing the three points, $(x_1,y_1,z_1) = (1,2,3)$, $(x_2,y_2,z_2) = (-1,0,1)$, and $(x_3,y_3,z_3) = (-4,-2,0)$. We can then have:

$$\left|[C]\right| = \begin{vmatrix} 1 & x_1 & y_1 \\ 1 & x_2 & y_2 \\ 1 & x_3 & y_3 \end{vmatrix} = \begin{vmatrix} 1 & 1 & 2 \\ 1 & -1 & 0 \\ 1 & -4 & -2 \end{vmatrix} = -2$$

$$\left|[C_1]\right| = \begin{vmatrix} z_1 & x_1 & y_1 \\ z_2 & x_2 & y_2 \\ z_3 & x_3 & y_3 \end{vmatrix} = \begin{vmatrix} 3 & 1 & 2 \\ 1 & -1 & 0 \\ 0 & -4 & -2 \end{vmatrix} = 0$$

$$\left|[C_2]\right| = \begin{vmatrix} 1 & z_1 & y_1 \\ 1 & z_2 & y_2 \\ 1 & z_3 & y_3 \end{vmatrix} = \begin{vmatrix} 1 & 3 & 2 \\ 1 & 1 & 0 \\ 1 & 0 & -2 \end{vmatrix} = 2$$

and

$$\left|[C_3]\right| = \begin{vmatrix} 1 & x_1 & z_1 \\ 1 & x_2 & z_2 \\ 1 & x & z \end{vmatrix} = \begin{vmatrix} 1 & 1 & 3 \\ 1 & -1 & 1 \\ 1 & -4 & -2 \end{vmatrix} = -4$$

According to Equation 13, we find $a_1 = |[C_1]|/|[C]| = 0/(-2) = 0$, $a_2 = |[C_2]|/|[C]| = 2/(-2) = -1$, and $a_3 = |[C_3]|/|[C]| = -4/(-2) = 2$. Thus, the required plane equation is $z = a_1 + a_2x + a_3y = -x + 2y$.

QuickBASIC Version of the program CramerR

A computer program called **CramerR** has been developed as a reviewing exercise in programming to solve a matrix equation of order 3 by application of Cramer

Rule and the definition of determinant of a 3 by 3 square matrix according to Equations 12 and 13, respectively. First, a subroutine called **Determ3** is created explicitly following Equation 13 as listed below:

```
SUB Determ3(A(),D)
D=A(1,1)*(A(2,2)*A(3,3)-A(2,3)*A(3,2))
  +A(2,1)*(A(3,2)*A(1,3)-A(1,2)*A(3,3))
  +A(3,1)*(A(1,2)*A(2,3)-A(1,3)*A(2,2))
END SUB
```

To interactively enter the elements of the coefficient matrix [C] and also the elements of the right-hand-side vector {Z} in Equation 12 and to solve for {A}, the program **CramerR** can be arranged as:

```
' PROGRAM CramerR - solves a matrix equation C(3,3)X(3)=V(3) by Cramer Rule.
'
                        DECLARE SUB DETERM3(C(),D)
                     SCREEN 2: CLEAR : CLS : KEY OFF
  PRINT "Program CramerR - solves the matrix equation C(3,3)X(3)=V(3)"
  PRINT "                by Cramer Rule."
  DIM C(3,3),CT(3,3),DC(3),V(3)
' Input
  PRINT : PRINT "Input the elements of the coefficient matrix [C], row by row"
  PRINT "  and press <Enter> key after entering a number :"
  FOR I=1 TO 3
     FOR J=1 TO 3
        INPUT; C(I,J)
        NEXT J
     PRINT
     NEXT I
  PRINT
  PRINT "Input the elements of the constant vector {V} :";
  FOR I = 1 TO 3
     INPUT ; V(I)
     NEXT I
' Solve for {X}
  CALL DETERM3(C(),DC)
  FOR I=1 TO 3
     FOR IR=1 TO 3
        FOR JC=1 TO 3
           CT(IR.JC)=C(IR,JC)
           IF (JC.EQ.I) CT(IR,JC)=Z(IR)
           NEXT JC:
        NEXT IR
     CALL DETERM3(CT(),DC(I))
     X(I)=DC(I)/DC
     NEXT I
  PRINT : PRINT "The solution vector has elements : ";
  FOR I = 1 TO 3: PRINT X(I);: NEXT I: END
```

1.4 PROGRAM GAUSS

Program **Gauss** is designed for solving N unknowns from N simultaneous, linear algebraic equations by the Gaussian Elimination method. In matrix notation, the problem can be described as to solve a vector {X} from the matrix equation:

$$[C]\{X\}=\{V\} \tag{1}$$

where [C] is an NxN coefficient matrix and {V} is a Nx1 constant vector, and both are prescribed. For example, let us consider the following system:

$$9x_1 + x_2 + x_3 = 10 \tag{2}$$

$$3x_1 + 6x_2 + x_3 = 14 \tag{3}$$

$$2x_1 + 2x_2 + 3x_3 = 3 \tag{4}$$

If the above three equations are expressed in matrix form as Equation 1, then:

$$[C]=\begin{bmatrix} 9 & 1 & 1 \\ 3 & 6 & 1 \\ 2 & 2 & 3 \end{bmatrix}, \qquad \{V\}=\begin{bmatrix} 10 \\ 14 \\ 3 \end{bmatrix}, \tag{5,6}$$

and

$$\{X\}=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}=\begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T \tag{7}$$

where T designates the transpose of a matrix.

Gaussian Elimination Method

A systematic procedure named after Gauss can be employed for solving x_1, x_2, and x_3 from the above equations. It consists of first dividing Equation 28 by the leading coefficient, 9, to obtain:

$$x_1 + \frac{1}{9}x_2 + \frac{1}{9}x_3 = \frac{10}{9} \tag{8}$$

This step is called *normalization* of the first equation of the system (1). The next step is to eliminate x_1 term from the other (in this case, two) equations. To do that, we multiply Equation 8 by the coefficients associated with x_1 in Equations 3 and 4, respectively, to obtain:

$$3x_1 + \frac{1}{3}x_2 + \frac{1}{3}x_3 = \frac{10}{3} \tag{9}$$

and

$$2x_1 + \frac{2}{9}x_2 + \frac{2}{9}x_3 = \frac{20}{9} \tag{10}$$

If we subtract Equation 9 from Equation 3, and subtract Equation 10 from Equation 4, the x_1 terms are eliminated. The resulting equations are, respectively:

$$\frac{17}{3}x_2 + \frac{2}{3}x_3 = \frac{32}{3} \tag{11}$$

and

$$\frac{16}{9}x_2 + \frac{25}{9}x_3 = \frac{7}{9} \tag{12}$$

This completes the first *elimination* step. The next normalization is applied to Equation 11, and then the x_2 term is to be eliminated from Equation 12. The resulting equations are:

$$x_2 + \frac{2}{17}x_3 = \frac{32}{17} \tag{13}$$

and

$$\frac{393}{153}x_3 = -\frac{393}{153} \tag{14}$$

The last normalization of Equation 14 then gives:

$$x_3 = -1 \tag{15}$$

Equations 8, 13, and 15 can be organized in matrix form as:

$$\{V\} = \begin{bmatrix} 1 & 1/9 & 1/9 \\ 0 & 1 & 2/17 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 10/9 \\ 32/17 \\ -1 \end{bmatrix} \tag{16}$$

The coefficient matrix is now a so-called *upper triangular matrix* since all elements below the main diagonal are equal to zero.

As x_3 is already obtained in Equation 15, the other two unknowns, x_2 and x_3, can be obtained by a sequential *backward-substitution* process. First, Equation 13 can be used to obtain:

$$x_2 = \frac{32}{17} - \frac{2}{17}x_3 = \frac{32}{17} - \frac{2}{17}(-1) = \frac{32+2}{17} = 2$$

Once, both x_2 and x_3 have been calculated, x_1 can be obtained from Equation 8 as:

$$x_1 = \frac{10}{9} - \frac{1}{9}x_2 - \frac{1}{9}x_3 = \frac{10}{9} - \frac{1}{9}(2) - \frac{1}{9}(-1) = \frac{10-2+1}{9} = 1$$

To derive a general algorithm for the Gaussian elimination method, let us denote the elements in [C], {X}, and {V} as $c_{i,j}$, x_i, and v_i, respectively. Then the normalization of the first equation can be expressed as:

$$\left(c_{1,j}\right)_{new} = \left(c_{1,j}\right)_{old} \Big/ \left(c_{1,1}\right)_{old} \tag{17}$$

and

$$\left(v_1\right)_{new} = \left(v_1\right)_{old} \Big/ \left(c_{1,1}\right)_{old} \tag{18}$$

Equation 17 is to be used for calculating the new coefficient associated with x_j in the first, normalized equation. So, j should be ranged from 2 to N which is the number of unknowns (equal to 3 in the sample case). The subscripts old and new are added to indicate the values before and after normalization, respectively. Such designation is particularly helpful if no separate storage in computer are assigned for [C] for the values of its elements. Notice that $(c_{1,1})_{new} = 1$ is not calculated. Preserving this diagonal element enables the determinant of [C] to be computed. (See the topic on matrix inversion and determinant.)

The formulas for the elimination of x_1 terms from the second equation are:

$$\left(c_{2,j}\right)_{new} = \left(c_{2,j}\right)_{old} - \left(c_{2,1}\right)_{old}\left(c_{1,j}\right)_{old} \tag{19}$$

for j = 2,3,…,N (there is no need to include j = 1) and

$$\left(v_2\right)_{new} = \left(v_2\right)_{old} - \left(c_{2,1}\right)_{old}\left(v_1\right)_{old} \tag{20}$$

By changing the subscript 2 in Equations 19 and 20, x_1 term in the third equation can be eliminated. In other words, the general formulas for elimination of x_1 terms from all equation other than the first equation are, for k = 2,3,…,N

$$\left(c_{k,j}\right)_{new} = \left(c_{k,j}\right)_{old} - \left(c_{k,1}\right)_{old}\left(c_{1,j}\right)_{old} \tag{21}$$

for j = 2,3,…,N

$$\left(v_k\right)_{new} = \left(v_k\right)_{old} - \left(c_{k,1}\right)_{old}\left(v_1\right)_{old} \tag{22}$$

Instead of normalizing the first equation, we can generalize Equations 17 and 18 for normalization of the ith equation, for i = 1,2,…,N to the expressions:

$$\left(c_{i,j}\right)_{new} = \left(c_{i.j}\right)_{old} \Big/ \left(c_{i,i}\right)_{old} \tag{23}$$

for j = i + 1,i + 2,…,N and

$$\left(v_i\right)_{new} = \left(v_i\right)_{old} \Big/ \left(c_{i,i}\right)_{old} \tag{24}$$

Note that $(c_{i,i})_{new}$ should be equal to 1 but no need to calculate since it is not involved in later calculation for finding {X}.

Similarly, elimination of x_i term from kth equation for k = i + 1,i + 2,…,N consists of using the general formula:

$$\left(c_{k,j}\right)_{new} = \left(c_{k,j}\right)_{old} - \left(c_{k,i}\right)_{old}\left(c_{i,j}\right)_{old} \tag{25}$$

for j = i + 1,i + 2,…,N and

$$\left(v_k\right)_{new} = \left(v_k\right)_{old} - \left(c_{k,i}\right)_{old}\left(v_i\right)_{old} \tag{26}$$

Backward substitution for finding x_i involves the calculation of:

$$x_i = v_i - \sum_{j=i+1}^{N} c_{i,j} x_j \tag{27}$$

for i = N–1,N–2,…,2,1. Note that x_N is already found equal to v_N after the Nth normalization.

Program **Gauss** listed below in both **QuickBASIC** and **FORTRAN** languages is developed for interactive specification of the number of unknowns, N, and the values of the elements of [C] and {V}. It proceeds to solve for {X} and prints out the computed values. Sample applications of both languages are provided immediately following the listed programs.

A subroutine **Gauss.Sub** is also made available for the frequent need in the other computer programs which require the solution of matrix equations.

QuickBASIC Version

```
' PROGRAM Gauss - solves a matrix equation C(N,N)X(N)=V(N)
'                 by Gaussian Elimination method.
'                 X and V share same storage space.
'
                  SCREEN 2: CLEAR : CLS : KEY OFF
  PRINT "Program Gauss - solves matrix equation C(N,N)X(N)=V(N)"
  PRINT "                  by Gaussian-Elimination method."
  PRINT : PRINT "Input the order of matrix equation, N : ";
          INPUT N : DIM C(N, N), V(N)
  PRINT : PRINT "Input the elements of the coefficient matrix, row by row"
  PRINT "  and press <Enter> key after entering a number :"
  FOR I=1 TO N
      FOR J=1 TO N
          INPUT;C(I,J)
          NEXT J
      PRINT
      NEXT I
  PRINT
  PRINT "Input the elements of the constant vector :";
  FOR I = 1 TO N
      INPUT ; V(I)
      NEXT I
  FOR I = 1 TO N                                          ' *** Normalization
      FOR J=I+1 TO N: C(I,J)=C(I,J)/C(I,I): NEXT J
      V(I)=V(I)/C(I,I)  : IF  I = N THEN GOTO 271
      FOR K = I+1 TO N
          IF C(K,I)=0 THEN 265 ELSE V(K)=V(K)-C(K,I)*V(I)     ' *** Elimination
          FOR J=I+1 TO N: C(K,J)=C(K,J)-C(K,I)*C(I,J): NEXT J
265       NEXT K
      NEXT I
271 FOR I=N-1 TO 1 STEP -1
        FOR J=I+1 TO N: V(I)=V(I)-C(I,J)*V(J): NEXT J
        NEXT I
    PRINT : PRINT : PRINT "The solution vector has elements : ";
    FOR I = 1 TO N: PRINT V(I);: NEXT I: PRINT: PRINT: END
```

Sample Application

```
Program Gauss - solves matrix equation C(N,N)X(N)=V(N)
                by Gaussian elimination method.

Input the order of matrix equation, N : 3

Input the elements of the coefficient matrix, row by row
  and press <Enter> key after entering a number :
? 9? 1? 1
? 3? 6? 1
? 2? 2? 3

Input the elements of the constant vector :? 10? 14? 3
The solution vector has elements : 1  2  -1
```

FORTRAN Version

```
C     PROGRAM Gauss - solves matrix equation C(N,N)X(N)=V(N)
C                     by Gaussian elimination method.
C                     X and V share same storage space.
      DIMENSION C(50,50),V(50)
      WRITE (*,*) 'Program Gauss - solves matrix equation C(N,N)X(N)=V(N)'
      WRITE (*,*) '                by Gaussian elimination method.'
      WRITE (*,*) 'Input the order of matrix equation, N : '
      READ  (*,*) N
      WRITE (*,*) 'Input the elements of the coefficient matrix,'
      WRITE (*,*) '  row by row and press <Enter> key after entering'
      WRITE (*,*) '  each row :'
      DO 5 I=1,N
    5 READ (*,*) (C(I,J),J=1,N)
      WRITE (*,*) 'Input the elements of the constant vector :'
      READ (*,*) (V(I),I=1,N)
      CALL GAUSS(C,N,50,V)
      WRITE (*,15) (V(I),I=1,N)
   15 FORMAT(' The vector {V} is'/5E16.5)
      STOP
      END

      SUBROUTINE GAUSS(C,N,M,V)
C
C     SOLVES MATRIX EQUATION C(N,N)*X(N)=V(N) BY GAUSSIAN ELIMINATION.
C       X and V share same storage.  C is dimensioned (M,M) in the calling program.
C
      DIMENSION C(M,M),V(N)
C
      N1=N-1
      DO 25 K=1,N1
         KP1=K+1
C
C     NORMALIZATION
C
         DO 10 J=KP1,N
   10       C(K,J)=C(K,J)/C(K,K)
         V(K)=V(K)/C(K,K)
C
C     ELIMINATION
C
         DO 25 I=KP1,N
            DO 20 J=KP1,N
   20          C(I,J)=C(I,J)-C(I,K)*C(K,J)
   25       V(I)=V(I)-C(I,K)*V(K)
C
C     BACKWARD SUBSTITUTION
C
      V(N)=V(N)/C(N,N)
      DO 35 I=1,N1
      IR=N-I
      IR1=IR+1
      DO 35 J=IR1,N
   35    V(IR)=V(IR)-C(IR,J)*V(J)
      RETURN
      END
```

Sample Application

```
Program Gauss - solves matrix equation C(N,N)X(N)=V(N)
                by Gauss-Jordan elimination method.
Input the order of matrix equation, N :
3
Input the elements of the coefficient matrix,
  row by row and press <Enter> key after entering
  each row :
9,1,1
3,6,1
2,2,3
Input the elements of the constant vector :
10,14,3
The vector {V} is
     .10000E+01      .20000E+01     -.10000E+01
Stop - Program terminated.
```

Gauss-Jordan Method

One slight modification of the elimination step will make the backward substitution steps completely unnecessary. That is, during the elimination of the x_i terms from the linear algebraic equations except the ith one, Equations 25 and 26 should be applied for k equal to 1 through N and excluding k = i. For example, the x_3 terms should be eliminated from the first, second, fourth through Nth equations. In this manner, after the Nth normalization, [C] becomes an identity matrix and {V} will have the elements of the required solution {X}. This modified method is called Gauss-Jordan method.

A subroutine called **GauJor** is made available based on the above argument. In this subroutine, a block of statements are also added to include the consideration of the *pivoting* technique which is required if $c_{i,i} = 0$. The normalization steps, Equations 49 and 50, cannot be implemented if $c_{i,i}$ is equal to zero. For such a situation, a search for a nonzero $c_{i,k}$ is necessary for i = k + 1,k + 2,…,N. That is, to find in the kth column of [C] and below the kth row a nonzero element. Once this nonzero $c_{i,k}$ is found, then we can then interchange the ith and kth rows of [C] and {V} to allow the normalization steps to be implemented; if no nonzero $c_{i,k}$ can be found then [C] is *singular* because the determinant of [C] is equal to zero! This can be explained by the fact that when $c_{k,k} = 0$ and no pivoting is possible and the determinant D of [C] can be calculated by the formula:

$$D = c_{1,1}c_{2,2}\cdots c_{k,k}\cdots c_{N,N} = \prod_{k=1}^{N} c_{k,k} \tag{28}$$

where Π indicates a product of all listed factors.

A subroutine has been written based on the Gauss-Jordan method and called **GauJor.Sub**. Both **QuickBASIC** and **FORTRAN** versions are made available and they are listed below.

QuickBASIC Version

```
   SUB GauJor (A(), N, C(), D)
 ' Gauss-Jordan Elimination method for solving [A]{X}={C}.
 ' {C} exits as {X}.
 ' Also calculate the determinant D of [A].
       FOR I = 1 TO N
           IF A(I, I) = 0 THEN 220
 '
 '                         *** Normalization ***
 200       FOR J = I + 1 TO N
               A(I, J) = A(I, J) / A(I, I)
               NEXT J
           C(I) = C(I) / A(I, I)
           IF I=N THEN 280 ELSE 250
 '
 '                            *** Pivoting ***
 220       FOR J = I + 1 TO N
               IF A(J, I) = 0 THEN 230
               FOR K = I TO N
                   T = A(I, K)
                   A(I, K) = A(J, K)
                   A(J, K) = T
                   NEXT K
               T = C(I)
               C(I) = C(J)
               C(J) = T
               GOTO 200
 230           NEXT J
            PRINT "The coefficient matrix is singular.":
 GOTO 300
 '
 '                          *** Elimination ***
 250     FOR K = 1 TO N
             IF K = I THEN 265
             IF A(K, I) = 0 THEN 265
             C(K) = C(K) - A(K, I) * C(I)
             FOR J = I + 1 TO N
                 A(K, J) = A(K, J) - A(K, I) * A(I, J)
                 NEXT J
 265         NEXT K
     NEXT I
 '
 '                      Calculates determinant.
 280 D = 1!
     FOR I = 1 TO N
         D = D * A(I, I)
         NEXT I
 300 END SUB
```

FORTRAN Version

```
      SUBROUTINE GAUJOR(C,N,M,V,D)
C
C   SOLVES MATRIX EQUATION C(N,N)*X(N)=V(N) BY GAUSSIAN ELIMINATION.
C     CALCULATES DETERMINANT, D, of C.  Pivoting is provided.
C       X and V share same storage.
C       C is dimensioned (M,M) in the calling program.
C
      DIMENSION C(M,M),V(N)
      DO 35 K=1,N
      KP1=K+1
      IF (C(K,K).EQ.0.) GOTO 12
C
C     NORMALIZATION
C
    8    DO 10 J=KP1,N
   10    C(K,J)=C(K,J)/C(K,K)
         V(K)=V(K)/C(K,K)
         IF (I.EQ.N) RETURN
         GOTO 20
C
C     PIVOTING
C
   12    DO 14 I=KP1,N
         IF (C(I,K).NE.0.) GOTO 16
   14    CONTINUE
   15    WRITE (*,*) 'The coefficient matrix is singular!'
         STOP
   16    DO 18 J=KP1,N
         T=C(K,J)
         C(K,J)=C(I,J)
   18    C(I,J)=T
         T=V(K)
         V(K)=V(I)
         V(I)=T
         GOTO 8
C
C     ELIMINATION
C
   20    DO 30 I=1,N
         IF (I.EQ.K) GOTO 30
            DO 25 J=KP1,N
   25       C(I,J)=C(I,J)-C(I,K)*C(K,J)
         V(I)=V(I)-C(I,K)*V(K)
   30    CONTINUE
   35 CONTINUE
C     CALCULATES Determinant
      D=1.
      DO 40 K=1,N
   40 D=D*C(K,K)
      RETURN
      END
```

Sample Applications

The same problem previously solved by the program **Gauss** has been used again but solved by application of subroutine GauJor. The results obtained with the **QuickBASIC** and **FORTRAN** versions are listed, in that order, below:

```
Program GauJor - solves matrix equation A(N,N)X(N)=C(N)
                 by Gauss-Jordan elimination method.

Input the order of matrix equation, N : 3

Input the elements of the coefficient matrix, row by row
  and press <Enter> key after entering a number :
? 9? 1? 1
? 3? 6? 1
? 2? 2? 3

Input the elements of the constant vector :? 10? 14? 3
The solution vector is :
 1  2  -1
Determinant of [A] =  131

Program GauJor - solves matrix equation C(N,N)X(N)=V(N)
                 by Gauss-Jordan elimination method.
Input the order of matrix equation, N :
3
Input the elements of the coefficient matrix,
  row by row and press <Enter> key after entering
  each row :
9,1,1
3,6,1
2,2,3
Input the elements of the constant vector :
10,14,3
The vector {V} is :
     .10000E+01      .20000E+01     -.10000E+01
The determinant of the coefficient matrix =   .13100E+03
Stop - Program terminated.
```

MATLAB Applications

For solving the vector {X} from the matrix equation [C]{X} = {R} when both the coefficient matrix [C] and the right-hand side vector {R} are specified, **MATLAB** simply requires [C] and {R} to be interactively inputted and then uses a statement X = C\R to obtain the solution vector {X} by multiplying the vector {R} on the left of the inverse of [C] or dividing {R} on the left by [C]. More details are discussed in the program **MatxAlgb**. Here, for providing more examples in **MATLAB** applications, a m file called **GauJor.m** is presented below as a companion of the **FORTRAN** and **QuickBASIC** versions:

```
           function [X,D]=GauJor(C,N,R)
% Solves matrix equation [C]{X}={R} of order N by Gauss-Jordan Elimination.
% Also finds the determinant D of [C].
%
ExitFlag=0; D=0;
for I=1:N, ip1=I+1;
    if C(i,i)==0
%                                Pivoting
       for k=ip1:N
           if C(k,i)~=0
              for j=ip1:N, T=C(k,j); C(k,j)=C(i,j); C(i,j) =T; end
              T=R(k); R(k)=R(I); R(I)=T; break
           end
       end
       ExitFlag=1;
    end
%                                Normalization
    if ExitFlag==0;
       for j=ip1:N, C(i,j)=C(i,j)/C(i,i); end
       R(I)=R(I)/C(i,i);
%                                Elimination
       for k=1:N
           if I~=k
              for j=ip1:N, C(k,j)=C(k,j)-C(k,i)*C(i,j); end
              R(k)=R(k)-C(k,i)*R(I);
           end
       end
    end
    if I==N, break
    end
end
if ExitFlag==1;
   error('The coefficient matrix is singular.')
   else D=1; X=R; for I=1:N, D=D*C(i,i); end
end
```

This file **GauJor.m** should then be added into **MATLAB**. As an example of interactive application of this m file, the sample problem used in the **FORTRAN** and **QuickBASIC** versions is again solved by specifying the coefficient matrix [C] and the right hand side vector {R} to obtain the resulting display as follows:

```
>> C=[9,1,1;3,6,1;2,2,3]; R=[10,14,3]'; [X,D]=GauJor(C,3,R)
  X =
      1.0000
      2.0000
      1.0000
  D =
      131
```

The results of the vector {X} and determinant D for the coefficient matrix [C] are same as obtained before.

Mathematica Applications

For solving a system of linear algebraic equations which has been arranged in matrix form as [A]{X} = {R}, **Mathematica**'s function **LinearSolve** can be applied to solve for {X} when the coefficient matrix [A] and the right-hand side vector {R} are both provided. The following is an example of interactive application:

```
In[1]: = A = {{3,6,14},{6,14,36},{14,36,98}}
Out[1]: =
    {{3, 6, 14}, {6, 14, 36}, {14, 36, 98}}
In[2]: = MatrixForm[A]
Out[2]//MatrixForm: =
    3    6    14
    6    14   36
    14   36   98
In[3]: = R = {9,20,48}
Out[3]: =
    {9, 20, 48}
In[4]: = LinearSolve[A,R]
Out[4]: =
    {–9,13,–3}
```

Output[2] and *Output[1]* demonstrate the difference in display of matrix [A] when MatrixeForm is requested, or, not requested, respectively. It shows that without requesting of MatrixForm, some screen space saving can be gained. *Output[4]* gives the solution $\{X\} = [-9\ 13\ -3]^T$ for the matrix equation $[A]\{X\} = \{R\}$ where the coefficient matix [A] and vector {R} are provided by *Input[1]* and *Input[3]*, respectively.

1.5 MATRIX INVERSION, DETERMINANT, AND PROGRAM MatxInvD

Given a square matrix [C] of order N, its inverse as $[C]^{-1}$ of the same order is defined by the equation:

$$[C][C]^{-1} = [C]^{-1}[C] = [I] \tag{1}$$

where [I] is an identity matrix having elements equal to one along its main diagonal and equal to zero elsewhere. That is:

$$[I] = \begin{bmatrix} 1 & 0 & . & . & . & . & 0 \\ 0 & 1 & 0 & . & . & . & 0 \\ & . & . & . & . & . & . \\ 0 & 0 & . & . & . & . & 1 \end{bmatrix} \tag{2}$$

To find $[C]^{-1}$, let c_{ij} and d_{ij} be the elements at the ith row and jth column of the matrices [C] and $[C]^{-1}$, respectively. Both i and j range from 1 to N. Furthermore, let $\{D_j\}$ and $\{I_j\}$ be the jth column of the matrices $[C]^{-1}$ and [I], respectively. It is easy to observe that $\{I_j\}$ has elements all equal to zero except the one in the jth row which is equal to unity. Also,

$$\{D_j\} = [d_{1j} d_{2j} \ldots d_{Nj}]^T \tag{3}$$

and

$$[C]^{-1} = [D_1 D_2 \ldots D_N]^T \tag{4}$$

Based on the rules of matrix multiplication, Equation 1 can be interpreted as $[C]\{D_1\} = \{I_1\}$, $[C]\{D_2\} = \{I_2\}$, …, and $[C]\{D_N\} = \{I_N\}$. This indicates that program **Gauss** can be successively employed N times by using the same coefficient matrix [C] and the vectors $\{I_i\}$ to find the vectors $\{D_i\}$ for i = 1,2,…,N. Program **MatxInvD** is developed with this concept by modifying the program **Gauss**. It is listed below along with a sample interactive run.

QuickBASIC Version

```
 ' * Program MatxInvD - Calculates inverse and determinant of a square matrix
            SCREEN 2: CLS : CLEAR : KEY OFF
PRINT " * Program MatxInvD - Calculates inverse and determinant of a square matrix
*"
PRINT : INPUT "Enter the order of the matrix : ", N : DIM C(N, N), C1(N, N)
PRINT : PRINT "Enter the elements of the matrix row-by-row and press <Enter> key"
PRINT "  after entering each element : ": PRINT
FOR J = 1 TO N
    FOR J1=1 TO N: INPUT ; C(J, J1): NEXT J1: PRINT: NEXT J: GOSUB 195
PRINT : PRINT "Determinant = "; D  : IF D <> 0 THEN 170
    PRINT : PRINT "The matrix is singular!": PRINT : END
170 PRINT : PRINT "The inverse matrix is :": PRINT
    FOR J = 1 TO N
        FOR J1=1 TO N: PRINT USING " ##.###^^^^";C1(J,J1);: NEXT J1
        PRINT : NEXT J: PRINT : END
 '
195 ' * Find inverse C1 of C(N,N) by Gaussian elimination *
 '
FOR T1 = 1 TO N
    FOR T2=1 TO N: C1(T1,T2)=0: NEXT T2: C1(T1,T1)=1: NEXT T1: N1=N-1: D=1
FOR T1 = 1 TO N1
 '
 ' * PIVOTING *
 '
    Q1 = T1 + 1: IF C(T1, T1) <> 0 THEN 285
    FOR T2 = Q1 TO N: IF C(T2,T1)=0 THEN 245 ELSE D=D*(-1)^(T1+T2): GOTO 260
245     NEXT T2      : D = 0: RETURN
 '
260 ' * Interchanging rows *
 '
    FOR T4=1 TO N
        T = C (T2, T4):  C(T2, T4) =  C(T1, T4):  C(T1,T4)=T
        T = C1(T2, T4): C1(T2, T4) = C1(T1, T4): C1(T1,T4)=T: NEXT T4
 '
285 ' * Normalization *
 '
    FOR T4= Q1 TO N:  C(T1,T4) =  C(T1,T4) / C(T1,T1): NEXT T4
    FOR T4=  1 TO N: C1(T1,T4) = C1(T1,T4) / C(T1,T1): NEXT T4
```

```
     ' * Elimination *
     '
     FOR T4 = T1 + 1 TO N
         FOR T5 = Q1 TO N: C(T4,T5) = C(T4,T5) - C(T4,T1)* C(T1,T5): NEXT T5
         FOR T7 =  1 TO N: C1(T4,T7)=C1(T4,T7) - C(T4,T1)*C1(T1,T7): NEXT T7
         NEXT T4: NEXT T1
     '
     ' * Backward Substitution *
     '
 FOR T6 = 1 TO N: C1(N, T6) = C1(N, T6) / C(N, N)
     FOR T4 = 1 TO N1: T1 = N - T4
         FOR T5 = T1 + 1 TO N: C1(T1,T6) = C1(T1,T6)-C(T1,T5)*C1(T5,T6): NEXT T5
         NEXT T4: D = D * C(T6, T6): NEXT T6: RETURN
```

Sample Application

```
  * Program MatxInvD - Calculates inverse and determinant of a square matrix *

 Enter the order of the matrix : 3

 Enter the elements of the matrix row-by-row and press <Enter> key
   after entering each element :

 ? 3? 0? 0
 ? 0? 4? 0
 ? 0? 0? 5

 Determinant = 60

 The inverse matrix is :

   3.333E-01  0.000E+00  0.000E+00
   0.000E+00  2.500E-01  0.000E+00
   0.000E+00  0.000E+00  2.000E-01
```

FORTRAN Version

```
 C * Program MatxInvD - Calculates inverse and determinant of a square matrix of order N
       DIMENSION C(50,50),C1(50,50)
       WRITE (*,2)
     2 FORMAT(' * Program MatxInvD - Calculates inverse and '
      *       'determinant of a square matrix *')
       WRITE (*,4)
     4 FORMAT(' Enter the order of the matrix : ')
       READ (*,*) N
       WRITE (*,6)
     6 FORMAT(' Enter the elements of the matrix row-by-row and ',
      *       'press <Enter> key after entering an entire row : ')
       DO 10 I=1,N
    10 READ (*,*) (C(I,J),J=1,N)
       CALL MATXINVD(C,50,N,C1,D)
       WRITE (*,*) 'Determinant = ',D
       IF (D.NE.0.) GO TO 20
       WRITE (*,*) 'The matrix is singular!'
       STOP
    20 WRITE (*,*) 'The inverse matrix is :'
       DO 25 I=1,N
    25 WRITE (*,30) (C1(I,J),J=1,N)
    30 FORMAT(4E20.5)
       END

       SUBROUTINE MATXINVD(C,M,N,C1,D)
 C
 C * Find inverse C1 of C(N,N) by Gaussian elimination *
 C   Both C & C1 are dimensioned M by M in the calling program.
 C
```

```
      DIMENSION C(M,M),C1(M,M)
      DO 8 I=1,N
         DO 5 J=1,N
   5           C1(I,J)=0
   8     C1(I,I)=1
      N1=N-1
      D=1
      DO 800 IE=1,N1
C
C * PIVOTING *
C
         I1=IE+1
         IF (C(IE,IE).NE.0) GO TO 285
         DO 245 K=I1,N
            IF (C(K,IE).EQ.0.) GO TO 245
            D=D*(-1)**(IE+K)
            GO TO 260
 245        CONTINUE
         D=0
         RETURN


C
C * Interchanging rows *
C
 260  DO 270 J=1,N
         T=C(K,J)
         C(K,J)=C(IE,J)
         C(IE,J)=T
         T=C1(K,J)
         C1(K,J)=C1(IE,J)
 270     C1(IE,J)=T
C
C * Normalization *
C
 285  DO 300 J=I1,N
 300     C(IE,J)=C(IE,J)/C(IE,IE)
      DO 400 J=1,N
 400     C1(IE,J)=C1(IE,J)/C(IE,IE)
C
C * Elimination *
C
      DO 422 K=IE+1,N
         DO 411 J=I1,N
 411        C(K,J)=C(K,J)-C(K,IE)*C(IE,J)
         DO 416 J=1,N
 416        C1(K,J)=C1(K,J)-C(K,IE)*C1(IE,J)
 422     CONTINUE
 800  CONTINUE
C
C * Backward Substitution *
C
      DO 900 J=1,N
         C1(N,J)=C1(N,J)/C(N,N)
         DO 810 I=1,N1
            IR=N-I
            DO 805 K=IR+1,N
 805           C1(IR,J)=C1(IR,J)-C(IR,K)*C1(K,J)
 810        CONTINUE
 900  D=D*C(J,J)
      RETURN
      END
```

Sample Applications

```
* Program MatxInvD - Calculates inverse and determinant of a square matrix *
Enter the order of the matrix :
3
Enter the elements of the matrix row-by-row and press <Enter> key after entering
 an entire row :
3,0,0
0,4,0
0,0,5
Determinant =         60.0000000
The inverse matrix is :
         .33333E+00          .00000E+00          .00000E+00
         .00000E+00          .25000E+00          .00000E+00
         .00000E+00          .00000E+00          .20000E+00
```

MATLAB Application

MATLAB offers very simple matrix operations. For example, matrix inversion can be implemented as:

```
>> A=[1,2;3,4]
   A =
        1      2
        3      4
>> Ainv=inv(A)
   Ainv =
       -2.0000    1.0000
        1.5000   -0.5000
```

To check if the obtained inversion indeed satisfies the equation [A}$[A]^{-1}$ = [I] where [I] is the identity matrix, we enter:

```
>> I=A*Ainv
   I =
        1.0000          0
        0.0000     1.0000
```

Once $[A]^{-1}$ becomes available, we can solve the vector {X} in the matrix equation [A]{X} = {R} if {R} is prescribed, namely {X} = $[A]^{-1}${R}. For example, may enter a {R} vector and find {X} such as:

```
>> R=[13;31]
   R =
        13
        31
>> X=A\R
   X =
        5.0000
        4.0000
```

MATHEMATICA APPLICATIONS

Mathematica has a function called **Inverse** for inversion of a matrix. Let us reuse the matrix A that we have entered in earlier examples and continue to demonstrate the application of **Inverse**:

In[1]: = A = {{1,2},{3,4}}; MatrixForm[A]

Out[1]//MatrixForm =

```
1   2
3   4
```

In[2]: = B = {{5,6},{7,8}}; MatrixForm[B]

Out[2]//MatrixForm =

```
5   6
7   8
```

In[3]: = MatrixForm[A + B]

Out[3]//MatrixForm =

```
6    8
10   12
```

In[4]: = Dif = A-B; MatrixForm[Dif]

Out[4]//MatrixForm =

```
–4   –4
–4   –4
```

In[5]: = AT = Transpose[A]; MatrixForm[AT]

Out[5]//MatrixForm =

```
1   3
2   4
```

In[6]: = Ainv = Inverse[A]; MatrixForm[Ainv]

Out[6]//MatrixForm =

```
–2   1
```

$$\frac{3}{2} - \left(\frac{1}{2}\right)$$

To verify whether or not the inverse matrix Ainv obtained in *Output[6]* indeed satisfies the equations $[A][A]^{-1} = [I]$ which is the identity matrix, we apply **Mathematica** for matrix multiplication:

In[7]: = Iden = A.Ainv; MatrixForm[Iden]

Out[7]//MatrixForm =

```
1  0
0  1
```

A dot is to separate the two matrices A and Ainv which is to be multiplied in that order. Output[7] proves that the computed matrix, Ainv, is the inverse of A! It should be noted that D and I are two *reserved variables* in **Mathematica** for the determinant of a matrix and the identity matrix. In their places, here Dif and Iden are adopted, respectively. For further testing, we show that $[A][A]^T$ is a symmetric matrix:

In[8]: = S = A.AT; MatrixForm[S]

Out[8]//MatrixForm =

```
5    11
11   25
```

And, the unknown vector {X} in the matrix equation [A]{X} = {R} can be solved easily if {R} is given and $[A]^{-1}$ are available:

In[9]: = R = {13,31}; X = Ainv.R

Out[9] = {5, 4}

The solution of $x_1 = 5$ and $x_2 = 4$ do satisfy the equations $x_1 + 2x_2 = 13$ and $3x_1 + 4x_2 = 31$.

Transformation of Coordinate Systems, Rotation, and Animation

Matrix algebra can be effectively applied for transformation of coordinate systems. When the cartesian coordinate system, x-y-z, is rotated by an angle Θ_z about the z-axis to arrive at the system x′-y′-z′ as shown in Figure 2, where z and z′ axes coincide and directed outward normal to the plane of paper, the new coordinates of a typical point P whose coordinates are (x_P,y_P,z_P) can be easily obtained as follows:

$$x'_P = OP\cos(\theta_P - \theta_z) = (OP\cos\theta_P)\cos\theta_z + (OP\sin\theta_P)\sin\theta_z$$

$$= x_P\cos\theta_z + y_p\sin\theta_z$$

$$y'_P = OP\sin(\theta_P - \theta_z) = (OP\sin\theta_P)\cos\theta_z - (OP\cos\theta_P)\sin\theta_z$$

$$= x_p\sin\theta_z + y_p\sin\theta_z$$

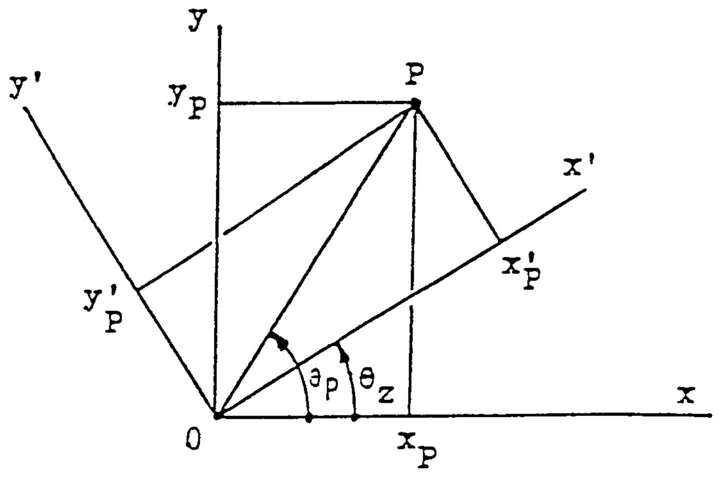


FIGURE 2. The cartesian coordinate system, x-y-z, is rotated by an angle Θ_z about the z-axis to arrive at the system x′-y′-z′.

and

$$z'_P = z_P$$

In matrix notation, we may define $\{P\} = [x_P\ y_P\ z_P]^T$ and $\{P'\} = [x_P'\ y_P'\ z_P']^T$ and write the above equations as $\{P'\} = [T_z]\{P\}$ where the transformation matrix for a rotation of z-axis by Θ_z is:

$$[T_z] = \begin{bmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

In a similar manner, it can be shown that the transformation matrices for rotating about the x- and y-axes by angles Θ_x and Θ_y, respectively, are:

$$[T_x] = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{bmatrix} \tag{6}$$

and

$$[T_y] = \begin{bmatrix} \cos\theta_y & 0 & -\sin\theta_y \\ 0 & 1 & 0 \\ \sin\theta_y & 0 & \cos\theta_y \end{bmatrix} \tag{7}$$

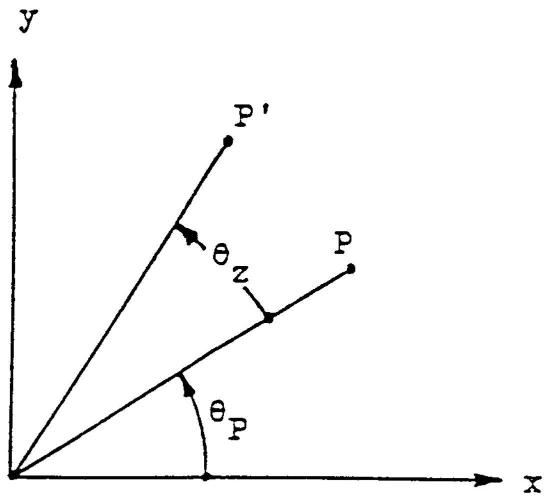


FIGURE 3. Point P whose coordinates are (x_P,y_P,z_P) is rotated to the point P' by a rotation of Θ_z.

It is interesting to note that if a point P whose coordinates are (x_P,y_P,z_P) is rotated to the point P' by a rotation of Θ_z as shown in Figure 3, the coordinates of P' can be easily obtained by the formula $\{P'\} = [R_z]\{P\}$ where $[R_z] = [T_z]^T$. If the rotation is by an angle Θ_x or Θ_y, then $\{P'\} = [R_x]\{P\}$ or $\{P'\} = [R_y]\{P\}$ where $[R_x] = [T_x]^T$ and $[R_y] = [T_y]^T$.

Having discussed about transformations and rotations of coordinate systems, we are ready to utilize the derived formulas to demonstrate the concept of **animation**. Motion can be simulated by first generating a series of rotated views of a three-dimensional object, and showing them one at a time. By erasing each displayed view and then showing the next one at an adequate speed, a smooth motion of the object is achievable to produce the desired animation. Program **Animate1.m** is developed to demonstrate this concept of animation by using a $4 \times 2 \times 3$ brick and rotating it about the x-axis by an angle of 25° and then rotating about the y-axis as many revolutions as desired. The front side of the block (x-y plane) is marked with a character F, and the right side (y-z plane) is marked with a character R, and the top side (x-z plane) is marked with a character T for helping the viewer to have a better three-dimensional perspective of the rotated brick (Figure 4). The x-rotation prior to y-rotation is needed to tilt the top side of the brick toward the front. The speed of animation is controlled by a parameter Damping. This parameter and the desired number of y-revolutions, Ncycle, are both to be interactively specified by the viewer (Figure 5).

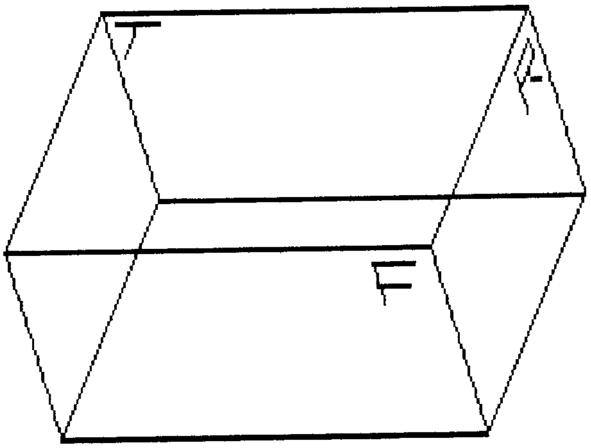
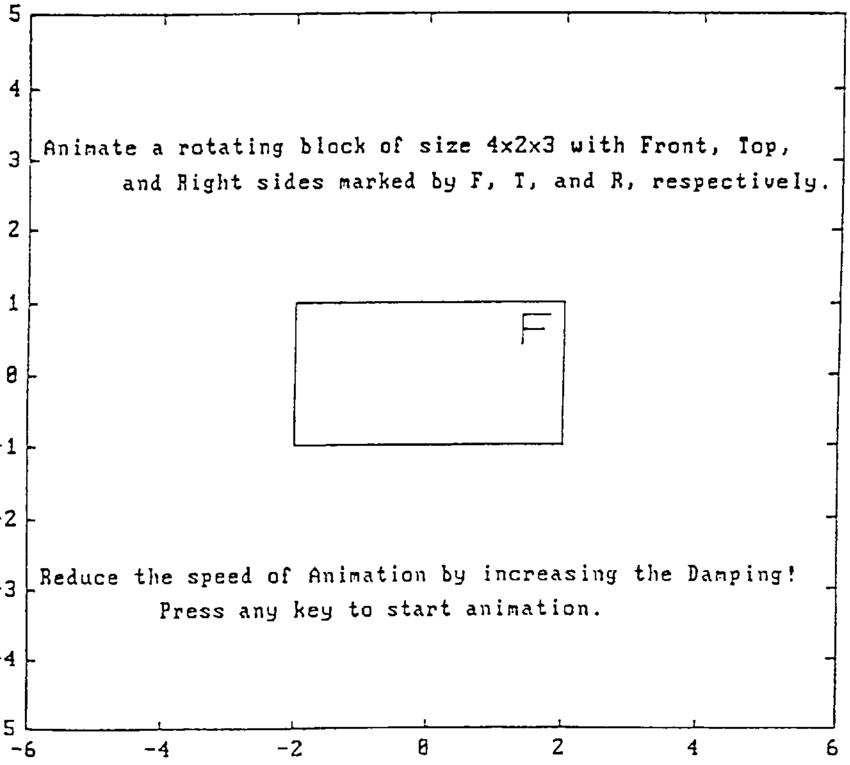


FIGURE 4. The characters F, R, and T help the viewer to have a better three-dimensional perspective of the rotated brick.



```
Animate a rotating block of size 4x2x3 with Front, Top,
        and Right sides marked by F, T, and R, respectively.

Reduce the speed of Animation by increasing the Damping!
           Press any key to start animation.
```

FIGURE 5. The speed of animation is controlled by a parameter Damping. This parameter and the desired number of y-revolutions, Ncycle, are both to be interactively specified by the viewer.

FUNCTION ANIMATE1(NCYCLE,DAMPING)

```
%
% Define the points for drawing block, F, T, and R
  xb=[ -2,  2,   2,  -2, -2, -2,  2,   2,  -2, -2,  2,  2,   2,   2,  -2,  -2];
  yb=[  1,  1,   1,   1,  1, -1, -1,  -1,  -1, -1, -1,  1,   1,  -1,  -1,   1];
  zb=[1.5,1.5,-1.5,-1.5,1.5,1.5,1.5,-1.5,-1.5,1.5,1.5,1.5,-1.5,-1.5,-1.5,-1.5];
  xs=[-6,6,6,-6,-6]; ys=[5,5,-5,-5,5];
  xf=[1.4,1.4,1.8,1.4,1.4,1.7]; yf=[.4,.8,.8,.8,.6,.6];
                                zf=[1.5,1.5,1.5,1.5,1.5,1.5];
  xt=[-1.8,-1.4,-1.6,-1.6]; yt=[1,1,1,1]; zt=[-1.3,-1.3,-1.3,-.9];
  xr=[2,2,2,2,2,2,2]; yr=[.4,.8,.8,.6,.6,.6,.4];
                      zr=[-.9,-.9,-1.3,-1.3,-.9,-1.1,-1.3];
%
% Describe the animation
  plot(xs,ys), hold, plot(xb,yb), plot(xf,yf), plot(xt,yt), plot(xr,yr)
  text(-5.8,3,'Animate a rotating block of size 4x2x3 with Front, Top,')
  text(-4.6,2.5,'and Right sides marked by F, T, and R, respectively.')
  text(-5.8,-3,'Reduce the speed of Animation by increasing the Damping!')
  text(-4,-3.5,'Press any key to start animation.')
  pause
%
% X-rotation
  for kxr=1:5; ax=5*kxr*pi/180; c=cos(ax); s=sin(ax);
      ybn=yb*c-zb*s; zbn=yb*s+zb*c; yfn=yf*c-zf*s; zfn=yf*s+zf*c;
      ytn=yt*c-zt*s; ztn=yt*s+zt*c; yrn=yr*c-zr*s; zrn=yr*s+zr*c;
      clg, plot(xb,ybn), plot(xf,yfn), plot(xt,ytn), plot(xr,yrn)
      for hold=1:Damping; end
      end
%
% Y-rotation
  kend=36*Ncycle;
  for kyr=1:kend; ay=-kyr*pi/18; c=cos(ay); s=sin(ay);
      xbn=xb*c+zbn*s; xfn=xf*c+zfn*s; xtn=xt*c+ztn*s; xrn=xr*c+zrn*s;
      clg, plot(xbn,ybn), plot(xfn,yfn), plot(xtn,ytn), plot(xrn,yrn)
      for hold=1:Damping; end
      end
  end
```

Notice that the coordinates for the corners of the brick are defined in arrays xb, yb, and zb. The coordinates of the points to be connected by linear segments for drawing the characters F, R, ant T are defined in arrays xf, yf, and zf, and xr, yr, and zr, and xt, yt, and zt, respectively.

The equations in deriving $[R_x]$ (= $[T_x]^T$) and $[R_y]$ (= $[T_y]^T$) are applied for x- and y- rotations in the above program. Angle increments of 5 and 10° are arranged for the x- and y-rotations, respectively. The rotated views are plotted using the new coordinates of the points, (xbn,ybn,zbn), (xfn,yfn,zfn), etc. Not all of these new arrays but only those needed in subsequent plot are calculated in this m file.

MATLAB command **clg** is used to erase the graphic window before a new rotated view the brick is displayed. The speed of animation is retarded by the "hold" loops in both x- and y-rotations involving the interactively entered value of the parameter Damping. The **MATLAB** command **pause** enables Figure 4 to be read and requires the viewer to press any key on the keyboard to commence the animation. Notice that a statement begins with a **%** character making that a comment statement, and that **%** can also be utilized for spacing purpose.

The xs and ys arrays allow the graphic window to be scaled by plotting them and then held (by command **hold**) so that all subsequent plots are using the same

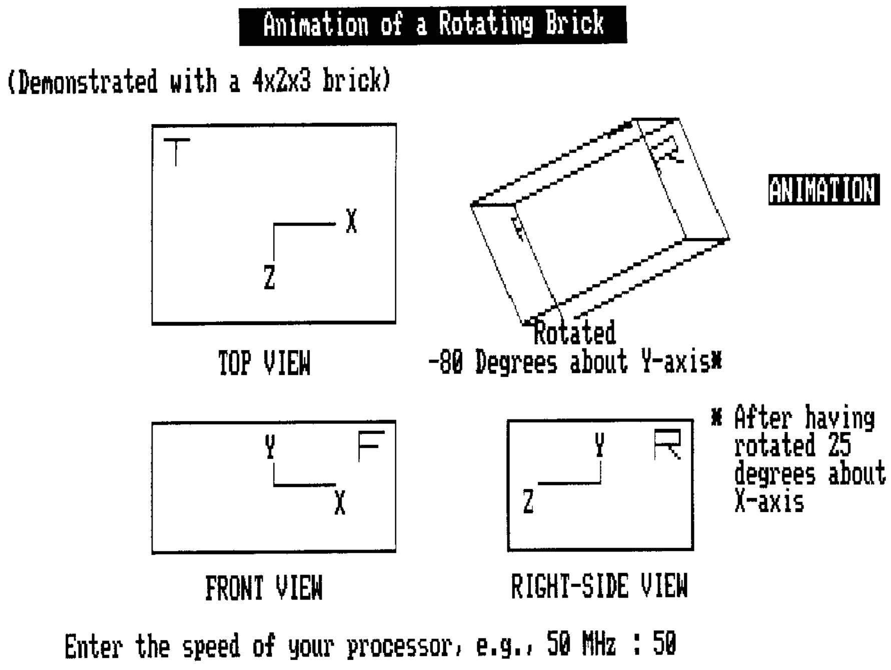

Animation of a Rotating Brick
(Demonstrated with a 4x2x3 brick)
T
X
Z
TOP VIEW
Rotated
-80 Degrees about Y-axis*
ANIMATION
Y
F
X
FRONT VIEW
Y
R
Z
RIGHT-SIDE VIEW
* After having rotated 25 degrees about X-axis
Enter the speed of your processor, e.g., 50 MHz : 50

FIGURE 6. Animation of a rotating brick.

scales in both x- and y-directions. The values in xs and ys arrays also control where to properly place the texts in Figure 4 as indicated in the **text** statements.

QuickBASIC Version

A **QuickBASIC** version of the program **Animate1.m** called **Animate1.QB** also is provided. It uses commands GET and PUT to animate the rotation of the 4 × 3 × 2 brick. More features have been added to show the three principal views of the brick and also the rotated view at the northeast corner of screen, as illustrated in Figure 6.

The window-viewport transformation of the rotated brick for displaying on the screen is implemented through the functions FNTX and FNTY. The actual ranges of the x and y measurements of the points used for drawing the brick are described by the values of V1 and V2, and V3 and V4, respectively. These ranges are mapped onto the screen matching the ranges of W1 and W2, and W4 and W3, respectively.

The rotated views of the brick are stored in arrays S1 through S10 using the **GET command**. Animation retrieves these views by application of the **PUT** command. Presently, animation is set for 10 y-swings (Ncycle = 10 in the program **Animate1.m**, arranged in Line 600). The parameter **Damping** described in the program **Animate1.m** here is set equal to 1500 (in Line 695).

```
'        * Program Animat1.QB - animates a Rotating Brick *
  DECLARE SUB XRotate (N,AX,XS(),YS(),ZS(),XN(),YN(),ZN())
  DECLARE SUB YRotate (N,AY,XS(),YS(),ZS(),XN(),YN(),ZN())
  CLEAR: CLS: KEY OFF: CV = 3.1416/180: SCREEN 2: W=8: H=8: ASPR=11/25: N=24: NS=1250
  DIM S1(NS),S2(NS),S3(NS),S4(NS),S5(NS),S6(NS),S7(NS),S8(NS),S9(NS),S10(NS),SV(200),
  DIM X(N),XO(N),XN(N),XS(N),Y(N),YO(N),YN(N),YS(N),ZO(N),ZN(N),ZS(N)
  FOR I = 1 TO N: READ XO(I), YO(I), ZO(I): NEXT I
  DATA   -2,-1,-1.5,    2,-1,-1.5,    2, 1,-1.5,  -2, 1,-1.5,  -2,-1,1.5,   2,-1,1.5,
2, 1, 1.5
  DATA   -2, 1, 1.5,  1.4,.4, 1.5,  1.4,.6, 1.5, 1.4,.8, 1.5, 1.8,.8,1.5, 1.7,.6,1.5,
-1.8, 1,-1.3
  DATA -1.6, 1,-1.3, -1.4, 1,-1.3, -1.6, 1, -.9,   2,.4, -.9,   2,.6,-.9,   2,.8,-.9,
2,.8,-1.3      DATA    2,.6,-1.3,     2,.6,-1.1,     2,.4,-1.3
  LOCATE 2, 26: PRINT " Animation of a Rotating Brick"
    GET (24.5 * W, H - 1)-(56.5 * W, 2 * H + 1), S1: PUT (24.5 * W, H - 1), S1, PRESET
  LOCATE 4, 5: PRINT "(Demonstrated with a 4x2x3 brick)"
    DEF FNTX (X) = V1 + (X - W1) * TS1 : DEF FNTY (Y) = V4 - (Y - W3) * TS2
    V1 =11*W: V2 = V1 + 31*W: V3 = 10*H: V4 = V3 + 31*W*ASPR:  W1=-3 : W2=3 : W3=-3 : W4=3
    TS1=(V2-V1)/(W2-W1): TS2=(V4-V3)/(W4-W3) :  LOCATE 22, 22: PRINT "FRONT VIEW"
    FOR I=1 TO N: X(I)=XO(I): Y(I)=YO(I): NEXT I             'DRAW FRONTAL VIEW, Transform
and plot
    GOSUB 480: GOSUB 490
    PSET (FNTX(1),FNTY(0)): LINE -(FNTX(0),FNTY(0)): LINE -(FNTX(0),FNTY(.5))
  LOCATE 18, 33: PRINT "X": LOCATE 16, 26: PRINT "Y"
    CALL YRotate(N,-90*CV,XO(),YO(),ZO(),XN(),YN(),ZN())
         V1 = 39 * W: V2 = V1 + 31 * W                                              'DRAW
RIGHT VIEW
  LOCATE 22, 48: PRINT "RIGHT-SIDE VIEW"
    FOR I=1 TO N: X(I)=XN(I): Y(I)=YN(I): NEXT I: GOSUB 480: GOSUB 490          'Transform
and plot
    PSET (FNTX(-1),FNTY(0))  : LINE -(FNTX(0),FNTY(0)): LINE -(FNTX(0),FNTY(.5))
  LOCATE 18, 49 :  PRINT "Z": LOCATE 16, 56: PRINT "Y"
    CALL XRotate(N,90*CV,XO(),YO(),ZO(),XN(),YN(),ZN())
         V1=11*W: V2=V1+31*W: V3=1.25*H: V4=V3+31*W*ASPR
  LOCATE 14, 23 : PRINT "TOP VIEW"                                                 'DRAW
TOP VIEW
    FOR I=1 TO N: X(I)=XN(I): Y(I)=YN(I): NEXT I: GOSUB 480: GOSUB 490          'Transform
and plot
    PSET (FNTX(1),FNTY(0))   : LINE -(FNTX(0),FNTY(0)): LINE -(FNTX(0),FNTY(-.5))
  LOCATE 9, 34  : PRINT "X" : LOCATE 11, 27: PRINT "Z"
    V1=43*W      : V2=V1+20*W: V3=3*H        : V4=V3+20*W*ASPR                 'ROTATE
ABOUT X-AXIS
    FOR XL=1 TO 5          : TAX=TAX+5    : GET (43*W,3*H)-(68*W,14*H),S1
        TS1 = (V2 - V1) / (W2 - W1): TS2 = (V4-V3)/(W4-W3)
        IF XL = 1 THEN 300 ELSE GOSUB 580: GOTO 305                            'Update
before rotate
300     FOR I = 1 TO N: XS(I) = XO(I): YS(I) = YO(I): ZS(I) = ZO(I): NEXT I
305     CALL XRotate(N,5*CV,XS(),YS(),ZS(),XN(),YN(),ZN()): PUT (43*W,3*H),S1,XOR
        FOR I =1 TO N: X(I)=XN(I): Y(I)=YN(I): NEXT I
        GOSUB 480: GOSUB 490: GOSUB 695                                         'Transform
and plot
        LOCATE 12,53: PRINT "Rotated" : LOCATE 13,45: PRINT USING "## Degrees about
X-axis";TAX
        NEXT XL      : AY = -10        : TAY=0
    FOR YL=1 TO 9: TAY=TAY+AY: X1=43*W: Y1 =3*H: X2=68*W: Y2=14*H: GOSUB 580   'ROTATE
ABOUT Y-AXIS
        ON YL GOTO 340, 345, 350, 355, 360, 365, 370, 375, 380
340     GET (X1, Y1)-(X2, Y2), S1: GOTO 385
345     GET (X1, Y1)-(X2, Y2), S2: GOTO 385
350     GET (X1, Y1)-(X2, Y2), S3: GOTO 385
355     GET (X1, Y1)-(X2, Y2), S4: GOTO 385
360     GET (X1, Y1)-(X2, Y2), S5: GOTO 385
365     GET (X1, Y1)-(X2, Y2), S6: GOTO 385
370     GET (X1, Y1)-(X2, Y2), S7: GOTO 385
375     GET (X1, Y1)-(X2, Y2), S8: GOTO 385
380     GET (X1, Y1)-(X2, Y2), S9
385     CALL YRotate(N,AY*CV,XS(),YS(),ZS(),XN(),YN(),ZN())
        FOR I = 1 TO N: X(I) = XN(I): Y(I) = YN(I): NEXT I
        ON YL GOTO 395, 400, 405, 410, 415, 420, 425, 430, 435
395     PUT (X1, Y1), S1, XOR: GOTO 440
400     PUT (X1, Y1), S2, XOR: GOTO 440
405     PUT (X1, Y1), S3, XOR: GOTO 440
410     PUT (X1, Y1), S4, XOR: GOTO 440
415     PUT (X1, Y1), S5, XOR: GOTO 440
420     PUT (X1, Y1), S6, XOR: GOTO 440
425     PUT (X1, Y1), S7, XOR: GOTO 440
430     PUT (X1, Y1), S8, XOR: GOTO 440
435     PUT (X1, Y1), S9, XOR
440     GOSUB 480: GOSUB 490                                                    'Transform
and plot
```

```
  LOCATE 12,53: PRINT "Rotated": LOCATE 13,44: PRINT USING "### Degrees about Y-axis*";TAY
  IF YL <> 1 GOTO 470
     LOCATE 16,65: PRINT "* After having": LOCATE 17,67: PRINT "rotated 25"
     LOCATE 18,67: PRINT "degrees about" : LOCATE 19,67: PRINT "X-axis"
470       NEXT YL: GET (X1, Y1)-(X2, Y2), S10
  FOR I=1 TO 500 : NEXT: GOSUB 590: LOCATE 22,1    :          END
480 FOR I=1 TO N : X(I)=FNTX(X(I)): Y(I)=FNTY(Y(I)): NEXT I: RETURN   'Converts window to
viewport
490 PSET  (X(5),Y(5)): LINE -(X(6),Y(6)): LINE -(X(2),Y(2)): LINE -(X(1),Y(1))
'DRAW BRICK
    LINE -(X(5),Y(5)): LINE -(X(8),Y(8)): LINE -(X(7),Y(7)): LINE -(X(3),Y(3)): LINE
-(X(4),Y(4))
    LINE -(X(8),Y(8)): PSET  (X(6),Y(6)): LINE -(X(7),Y(7)): PSET  (X(3),Y(3)): LINE
-(X(2),Y(2))
    PSET  (X(1),Y(1)): LINE -(X(4),Y(4))
    PSET  (X(9),Y(9)): LINE -(X(11),Y(11)): LINE -(X(12),Y(12)): PSET (X(10),Y(10))
'DRAW "F"
                       LINE -(X(13),Y(13))
    PSET (X(14),Y(14)): LINE -(X(16),Y(16)): PSET  (X(17),Y(17)): LINE -(X(15),Y(15))
'DRAW "T"
    PSET (X(18),Y(18)): LINE -(X(20),Y(20)): LINE -(X(21),Y(21)): LINE -(X(22), Y(22))
'DRAW "R"
    LINE -(X(19),Y(19)):PSET  (X(23),Y(23)): LINE -(X(24),Y(24)): RETURN
580 FOR I=1 TO N: XS(I)=XN(I): YS(I)=YN(I) : ZS(I)=ZN(I): NEXT I: RETURN
'Updating
590 LOCATE 8, 70: PRINT "ANIMATION": GET (68.5*W,7*H-1)-(78.5*W,8*H),SV
'Animation
                                    PUT (68.5*W,7*H-1),SV,PRESET
600 FOR I=1 TO 10: PUT (X1,Y1),S10,PSET: GOSUB 695: PUT (X1,Y1),S9,PSET: GOSUB 695
                   PUT (X1,Y1), S8,PSET: GOSUB 695: PUT (X1,Y1),S7,PSET: GOSUB 695
                   PUT (X1,Y1), S6,PSET: GOSUB 695: PUT (X1,Y1),S5,PSET: GOSUB 695
                   PUT (X1,Y1), S4,PSET: GOSUB 695: PUT (X1,Y1),S3,PSET: GOSUB 695
                   PUT (X1,Y1), S2,PSET: GOSUB 695: PUT (X1,Y1),S1,PSET: GOSUB 695
                   PUT (X1,Y1), S2,PSET: GOSUB 695: PUT (X1,Y1),S3,PSET: GOSUB 695
                   PUT (X1,Y1), S4,PSET: GOSUB 695: PUT (X1,Y1),S5,PSET: GOSUB 695
                   PUT (X1,Y1), S6,PSET: GOSUB 695: PUT (X1,Y1),S7,PSET: GOSUB 695
                   PUT (X1,Y1), S8,PSET: GOSUB 695: PUT (X1,Y1),S9,PSET: GOSUB 695: NEXT
I: RETURN
695 FOR PS = 1 TO 1500: NEXT PS: RETURN
 'pause

    SUB XRotate (N,AX,XS(),YS(),ZS(),XN(),YN(),ZN())
'   X-ROTATION
      CS = COS(AX): SN = SIN(AX)
    FOR I=1 TO N: XN(I)=XS(I): YN(I)=YS(I)*CS-ZS(I)*SN: ZN(I)=YS(I)*SN+ZS(I)*CS: NEXT I
    END SUB

    SUB YRotate (N,AY,XS(),YS(),ZS(),XN(),YN(),ZN())
'   Y-ROTATION
      CS = COS(AY): SN = SIN(AY)
    FOR I=1 TO N: YN(I)=YS(I): XN(I)=XS(I)*CS+ZS(I)*SN: ZN(I)=-XS(I)*SN+ZS(I)*CS: NEXT I
    END SUB
```

1.6 PROBLEMS

MATRIX ALGEBRA

1. Calculate the product [A][B][C] by (1) finding [T] = [A][B] and then [T][C], and (2) finding [T] = [B][C] and then [A][T] where:

$$[A]=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}\quad [B]=\begin{bmatrix}6 & 5\\ 4 & 3\\ 2 & 1\end{bmatrix}\quad [C]=\begin{bmatrix}-1 & -2\\ -3 & -4\end{bmatrix}$$

2. Calculate [A][B] of the two matrices given above and then take the transpose of product matrix. Is it equal to the product of $[B]^T[A]^T$?
3. Are $([A][B][C])^T$ and the product $[C]^T[B]^T[A]^T$ identical to each other?

4. Apply the **QuickBASIC** and **FORTRAN** versions of the program **Matx-Algb** to verify the results of Problems 1, 2, and 3.
5. Repeat Problem 4 but use **MATLAB**.
6. Apply the program **MatxInvD** to find $[C]^{-1}$ of the matrix [C] given in Problem 1 and also to $([C]^T)^{-1}$. Is $([C]^{-1})^T$ equal to $([C]^T)^{-1}$?
7. Repeat Problem 6 but use **MATLAB**.
8. For statistical analysis of a set of N given data $X_1, X_2, \ldots, X_N$, it is often necessary to calculate the *mean*, m, and *standard deviation*, 5, by use of the formulas:

$$m = \frac{1}{N}\left(X_1 + X_2 + \ldots + X_N\right)$$

and

$$\sigma = \left\{\frac{1}{N}\left[\left(X_1 - m\right)^2 + \left(X_2 - m\right)^2 + \ldots + \left(X_N - m\right)^2\right]\right\}^{0.5}$$

Use indicial notation to express the above two equations and then develop a subroutine meanSD(X,N,RM,SD) for taking the N values of X to compute the real value of mean, RM, and standard deviation, SD.
9. Express the ith term in the following series in indicial notation and then write an interactive program **SinePgrm** allowing input of the x value to calculate sin(x) by terminating the series when additional term contributes less than 0.001% of the partial sum of series in magnitude:

$$\text{Sin } x = \frac{x^1}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \ldots$$

Notice that Sin(x) is an *odd function* so the series contains only terms of odd powers of x and the series carries alternating signs. Compare the result of the program SinePgrm with those obtained by application of the library function Sin available in **FORTRAN** and **QuickBASIC**.
10. Same as Problem 9, but for the cosine series:

$$\text{Cos } x = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

Notice that Cos(x) is an *even function* so the series contains only terms of even powers of x and the series also carries alternating signs.
11. Repeat Problem 4 but use **Mathematica**.
12. Repeat Problem 6 but use **Mathematica**.

Gauss

1. Run the program **GAUSS** to solve the problem:

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 10 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 2 \\ 8 \\ 14 \end{bmatrix}$$

2. Run the program **GAUSS** to solve the problem:

$$\begin{bmatrix} 0 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -1 \\ 8 \\ 14 \end{bmatrix}$$

   What kind of problem do you encounter? "Divided by zero" is the message! This happens because the coefficient associated with x_1 in the first equation is equal to zero and the normalization in the program **GAUSS** cannot be implemented. In this case, the order of the given equations needs to be interchanged. That is to put the second equation on top or find *below* the first equation an equation which has a coefficient associated with x_1 not equal to zero and is to be interchanged with the first equation. This procedure is called "pivoting." Subroutine **GauJor** has such a feature incorporated, apply it for solving the given matrix equation.
3. Modify the program **GAUSS** by following the Gauss-Jordan elimination procedure and excluding the back-substitution steps. Name this new program **GauJor** and test it by solving the matrix equations given in Problems 1 and 2.
4. Show all details of the normalization, elimination, and backward substitution steps involved in solving the following equations by application of Gaussian Elimination method:

$$4x_1 + 2x_2 - 3x_3 = 8$$
$$5x_1 - 3x_2 + 7x_3 = 26$$
$$-x_1 + 9x_2 - 8x_3 = -10$$

5. Present every normalization and elimination steps involved in solving the following system of linear algebraic equations by the Gaussian Elimination Method:

$$5x_1 - 2x_2 + 2x_3 = 9,\ -2x_1 + 7x_2 - 2x_3 = 9,\ \text{and}\ 2x_1 - 2x_2 + 9x_3 = 41$$

6. Apply the Gauss-Jordan elimination method to solve for x_1, x_2, and x_3 from the following equations:

$$\begin{bmatrix} 0 & 1 & -1 \\ 2 & 9 & 3 \\ 4 & 24 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

   Show every normalization, elimination, and pivoting (if necessary) steps of your calculation.
7. Solve the matrix equation [A]{X} = {C} by **Gauss-Jordan** method where:

$$\begin{bmatrix} 3 & 2 & 1 \\ 2 & 5 & -1 \\ 4 & 1 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -2 \\ -3 \\ 3 \end{bmatrix}$$

   Show every interchange of rows (if you are required to do pivoting before normalization), normalization, and elimination steps by indicating the changes in [A] and {C}.
8. Apply the program **GauJor** to solve Problem 7.
9. Present every normalization and elimination steps involved in solving the following system of linear algebraic equations by the *Gauss-Jordan* Elimination Method:

$$5x_1 - 2x_2 + x_3 = 4$$

$$-2x_1 + 7x_2 - 2x_3 = 9$$

$$x_1 - 2x_2 + 9x_3 = 40$$

10. Apply the program **Gauss** to solve Problem 9 described above.
11. Use **MATLAB** to solve the matrix equation given in Problem 7.
12. Use **MATLAB** to solve the matrix equation given in Problem 9.
13. Use **Mathematica** to solve the matrix equation given in Problem 7.
14. Use **Mathematica** to solve the matrix equation given in Problem 9.

Matrix Inversion

1. Run the program **MatxInvD** for finding the inverse of the matrix:

$$[A] = \begin{bmatrix} 3 & 0 & 2 \\ 0 & 5 & 0 \\ 2 & 0 & 3 \end{bmatrix}$$

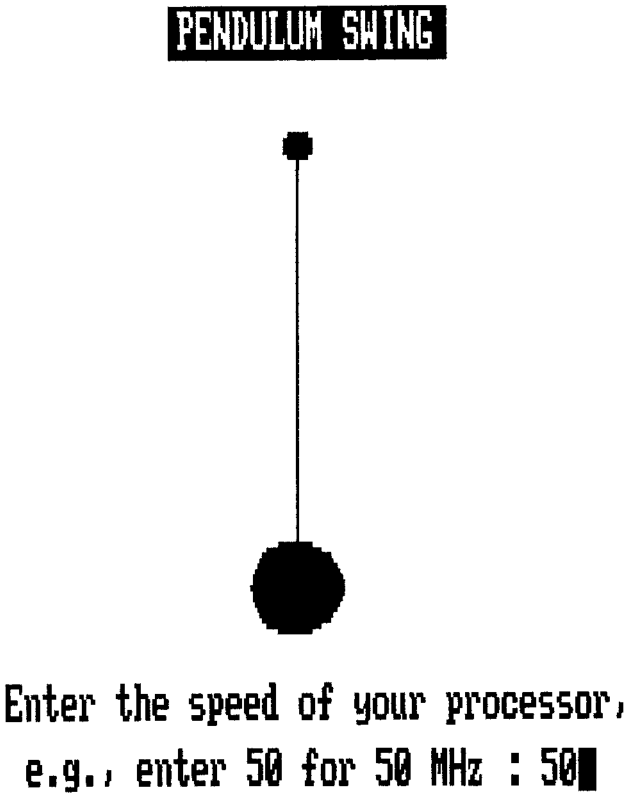
2. Write a program Invert3 which inverts a given 3×3 matrix [A] by using the cofactor method. A subroutine COFAC should be developed for calculating the cofactor of the element at Ith row and Jth column of [A] in term of the elements of [A] and the user-specified values of I and J. Let the inverse of [A] be designated as [AI] and the determinant of [A] be designated as D. Apply the developed program **Invert3** to generate all elements of [AI] by calling the subroutine **COFAC** and by using D.
3. Write a **QuickBASIC** or **FORTRAN** program **MatxSorD** which will perform the addition and subtraction of two matrices of same order.
4. Write a **QuickBASIC** or **FORTRAN** program **MxTransp** which will perform the transposition of a given matrix.
5. Translate the **FORTRAN** subroutine **MatxMtpy** into a **MATLAB** m file so that by entering the matrices [A] and [B] of order L by M and M by N, respectively, it will produce a product matrix [P] of order L by N.
6. Enter **MATLAB** commands interactively first a square matrix [A] and then calculate its trace.
7. Use **MATLAB** commands to first define the elements in its upper right corner including the diagonal, and then use the symmetric properties to define those in the lower left corner.
8. Convert either **QuickBasic** or FORTRAN version of the program MatxInvD into a MATLAB function file MatxInvD.m with a leading statement function [Cinv,D] = MatxInvD(C,N)
9. Apply the program MatxInvD to invert the matrix:

$$[A]=\begin{bmatrix}1 & 3 & 4\\ 5 & 6 & 7\\ 8 & 9 & 10\end{bmatrix}$$

   Verify the answer by using Equation 1.
10. Repeat Problem 9 but by **MATLAB** operation.
11. Apply the program **MatxInvD** to invert the matrix:

$$[A]=\begin{bmatrix}-9 & -1 & -2\\ -3 & -4 & -5\\ -6 & -7 & -8\end{bmatrix}$$

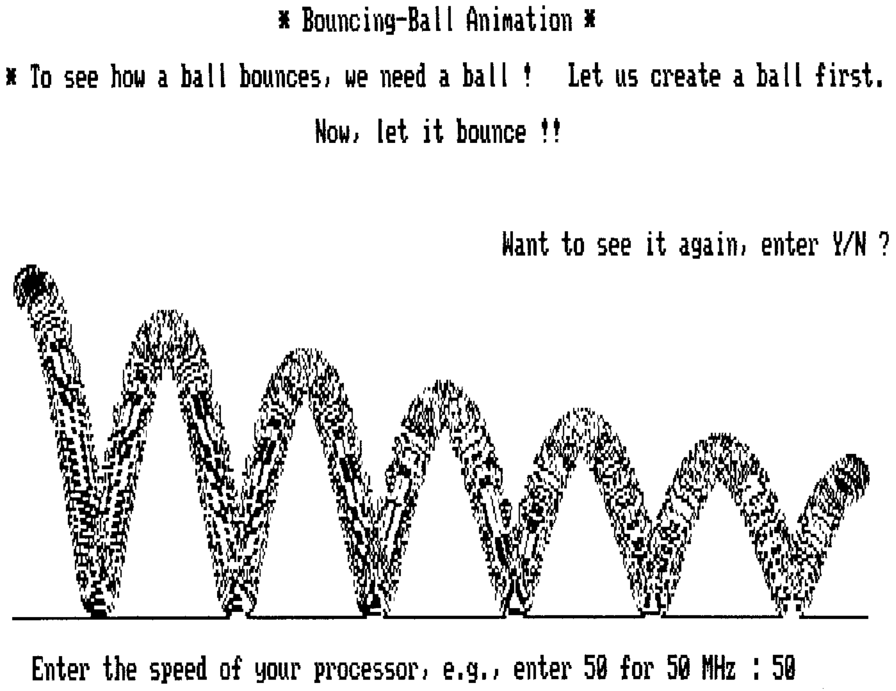
   Verify the answer by using Equation 1.
12. Repeat Problem 11 but by **MATLAB** operations.
13. Derive $[R_x]$ and verify that it is indeed equal to $[T_x]^T$. Repeat for $[R_y]$ and $[R_z]$.
14. Apply **MATLAB** to generate a matrix $[R_z]$ for $\theta_z = 45°$ and then to use $[R_z]$ to find the rotated coordinates of a point P whose coordinates before rotation are (1,–2,5).



Enter the speed of your processor,
e.g., enter 50 for 50 MHz : 50

FIGURE 7. Problem 18.

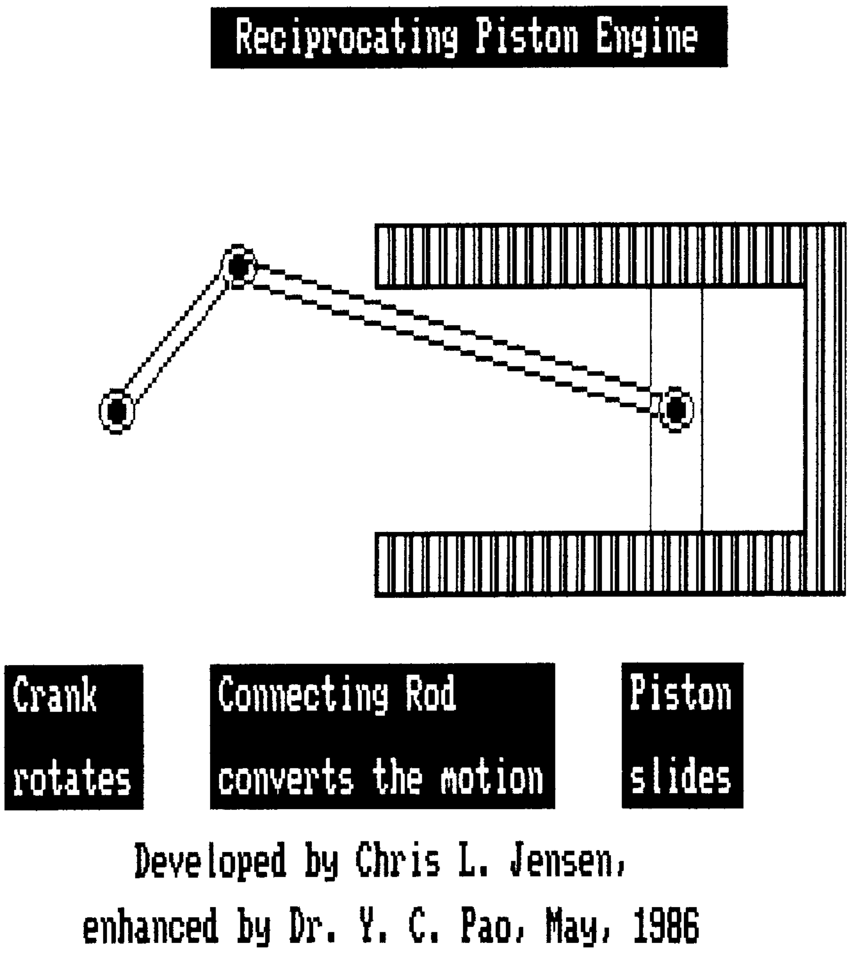
15. What will be the coordinates for the point P mentioned in Problem 14 if the coordinate axes are rotated counterclockwise about the z-axis by 45°? Use **MATLAB** to find your answer.
16. Apply **MATLAB** to find the location of a point whose coordinates are (1,2,3) after three rotations in succession: (1) about y-axis by 30°, (2) about z-axis by 45° and then (3) about x-axis by –60°.
17. Change m file Animate1.m to animate just the rotation of the front (F) side of the 4 × 2 × 3 brick in the graphic window.
18. Write a **MATLAB** m file for animation of pendulum swing[1] as shown in Figure 7.
19. Write a **MATLAB** m file for animation of a bouncing ball[1] using an equation of $y = 3e^{-0.1x}\sin(2x + 1.5708)$ as shown in Figure 8.
20. Write a **MATLAB** m file for animation of the motion of crank-piston system as shown in Figure 9.
21. Write a **MATLAB** m file to animate the vibrating system of a mass attached to a spring as shown in Figure 10.



```
* Bouncing-Ball Animation *

* To see how a ball bounces, we need a ball !   Let us create a ball first.

Now, let it bounce !!

                                    Want to see it again, enter Y/N ?

Enter the speed of your processor, e.g., enter 50 for 50 MHz : 50
```

FIGURE 8. Problem 19.

22. Write a **MATLAB** m file to animate the motion of a cam-follower system as shown in Figure 11.
23. Write a **MATLAB** m file to animate the rotary motion of a wankel cam as shown in Figure 12.
24. Repeat Problem 9 but by **Mathematica** operation.
25. Repeat Problem 11 but by **Mathematica** operation.
26. Repeat Problem 14 but by **Mathematica** operation.
27. Repeat Problem 15 but by **Mathematica** operation.
28. Repeat Problem 16 but by **Mathematica** operation.



Reciprocating Piston Engine

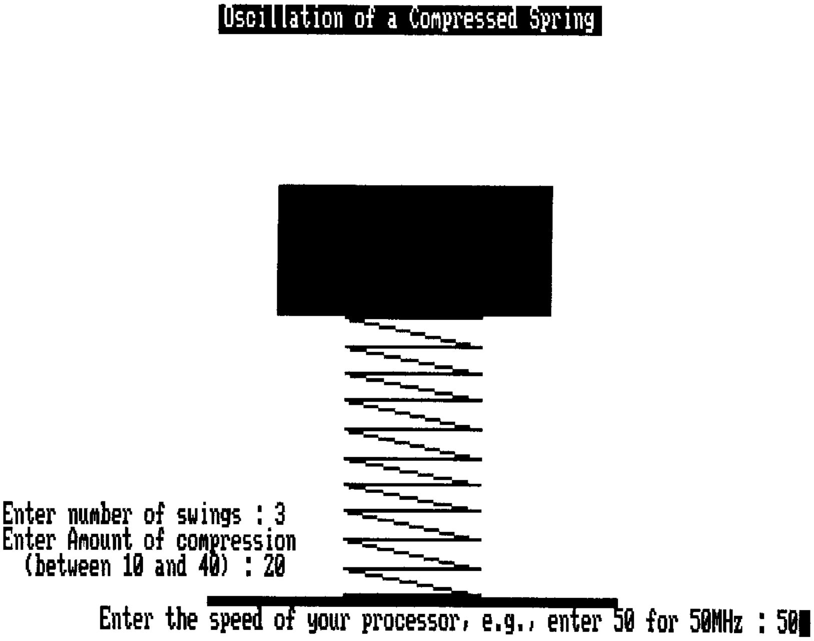
Crank rotates

Connecting Rod converts the motion

Piston slides

Developed by Chris L. Jensen,
enhanced by Dr. Y. C. Pao, May, 1986

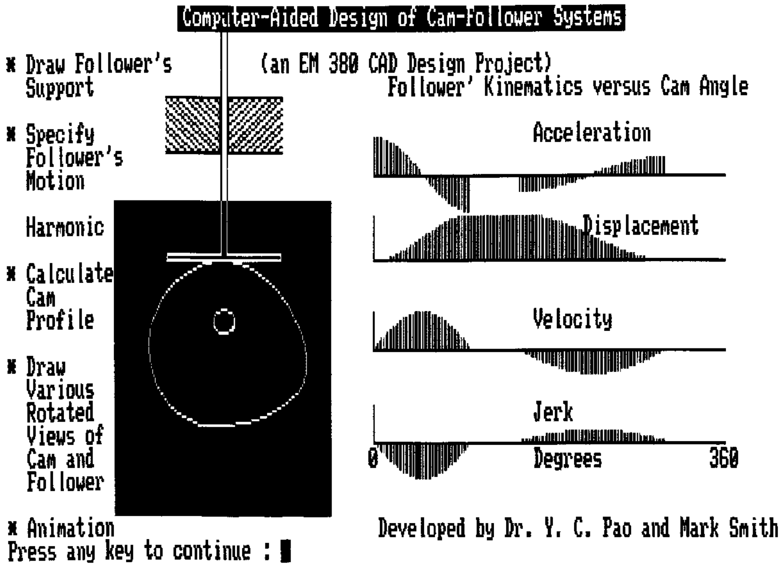
FIGURE 9. Problem 20.



Oscillation of a Compressed Spring

```
Enter number of swings : 3
Enter Amount of compression
  (between 10 and 40) : 20
        Enter the speed of your processor, e.g., enter 50 for 50MHz : 50
```

FIGURE 10. Problem 21.



Computer-Aided Design of Cam-Follower Systems

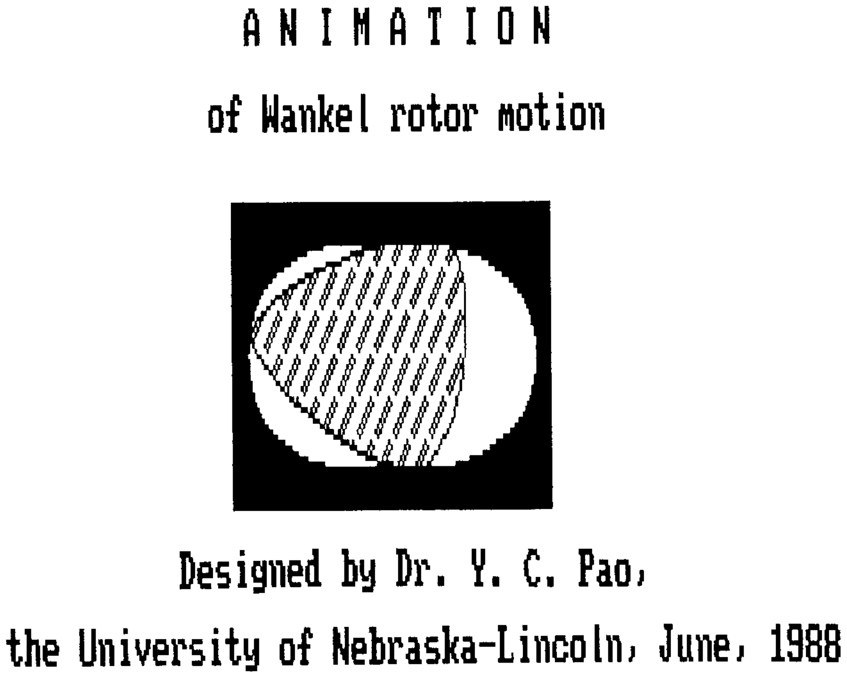
(an EM 380 CAD Design Project)

Follower' Kinematics versus Cam Angle

```
* Draw Follower's
  Support

* Specify
  Follower's
  Motion

  Harmonic

* Calculate
  Cam
  Profile

* Draw
  Various
  Rotated
  Views of
  Cam and
  Follower

* Animation
Press any key to continue :
```

Developed by Dr. Y. C. Pao and Mark Smith

FIGURE 11. Problem 22.


A N I M A T I O N

of Wankel rotor motion

Designed by Dr. Y. C. Pao,

the University of Nebraska-Lincoln, June, 1988

FIGURE 12. Problem 23.

1.7 REFERENCE

1. Y. C. Pao, "On Development of Engineering Animation Software," in *Computers in Engineering*, edited by K. Ishii, ASME Publications, New York, 1994, pp. 851–855.