Chapter 1
Matrix Algebra and MATLAB

1.1 Introduction

In the middle of the 19th century the English mathematician James Sylvester (1814-1897) used the term *matrix* with reference to a rectangular array of numbers. The name was quickly adopted by mathematicians who recognized the advantages of a concise notation in the analysis of various problems. Matrices, and the related algebra that soon followed, remained largely in the province of mathematics. Beginning in the mid 1920's physicists studying quantum mechanics and engineers analyzing vibration problems realized the importance of matrices and matrix algebra in the solution of physical problems.

Matrices are used in the solution of systems of linear algebraic equations, differential equations, economic input-output studies and in an ever expanding list of different applications. Current computer software makes extensive use of matrices.

MATLAB, short for *matrix laboratory*, is a mathematical software product written to perform various mathematical operations on matrices. Programs called M-files, provided as part of MATLAB, may be combined by the user to perform complicated numerical computations in the solution of various problems using matrices. With this in mind, it is appropriate to begin with a brief introduction to matrix algebra and, at the same time, demonstrate some of the features of MATLAB.

Since one of the primary uses of matrices focuses on the representation and solution of systems of linear algebraic equations, perhaps it is best to begin by considering a simple example of two linear equations and three unknowns.

$$\begin{aligned} 3x - 4y + 9z &= -4 \\ -2x + 1y + 7z &= 7 \end{aligned}$$

Clearly, the two rectangular arrays

$$\begin{bmatrix} 3 & -4 & 9 \\ -2 & 1 & 7 \end{bmatrix} \text{ and } \begin{bmatrix} -4 \\ 7 \end{bmatrix}$$

containing the numbers in the equations display all of the important data. The unknowns or variables x, y and z could be any other symbols, for example, s, t and u, or x_1, x_2 and x_3. The rectangular arrays are called matrices. In general, it is not necessary that the entries in a matrix be only real numbers. Words, functions or any quantities that can be arranged in rows and columns may be used.

For mathematical purposes, a matrix is defined to be a rectangular array of real or complex numbers displayed in rows and columns. To specify the size, or *order*, of a matrix it is necessary to state the number of rows and the number of columns. Convention requires that the

number of rows be mentioned first; thus, a general matrix, say $\boldsymbol{A}$, of order r by c, denoted $r \times c$, will have r rows and c columns and is given by

$$\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1c} \\ a_{21} & a_{22} & \cdots & a_{2c} \\ & & a_{ij} & \\ \vdots & & & \vdots \\ a_{r1} & a_{r2} & \cdots & a_{rc} \end{bmatrix}$$

where the entry a_{ij} appears in row i and column j. Before we describe how to enter matrices, a few introductory comments about MATLAB.

- The *prompt* symbol, >> , appears in the Command Window to indicate that MATLAB is ready to accept input. Similar prompts are used in all versions.
- In general, MATLAB is case sensitive.
- To execute a command, press Enter or ↵.
- MATLAB allows the user to set many preferences for the Command Window and Editor. Preferences may be set using the Windows interface or MATLAB commands. For example, >> **format compact** will suppress blank lines.
- MATLAB has an extensive help feature. To obtain information about **format,** execute the command, >> **help format**.
- Number formats will influence screen display not MATLAB computations. For most purposes, the **short e** format, >> **format short e**, a floating point number with 5 digits **1.2345e+006**, is satisfactory. Use caution with the default **short** format, a displayed **-0.0000** may not be exactly zero.
- To stop a runaway MATLAB program use CTRL+c.
- Exit MATLAB with >> **exit,** >> **quit** or use the standard Windows features.

As we proceed, many more MATLAB commands and features will be used.

To see how MATLAB employs matrices we continue with the two equations given above. The 2×3 matrix, $\begin{bmatrix} 3 & -4 & 9 \\ -2 & 1 & 7 \end{bmatrix}$, may be entered into MATLAB by typing **a** $= [3, -4, 9; -2, 1, 7]$ after the prompt. The input and MATLAB's confirmation are as follows:

```
>> a = [3,-4,9;-2,1,7]
a =
   3   -4    9
  -2    1    7
>>
```

Commas or a blank space separate entries in a row and a semicolon begins a new row. The output reflects the **compact** format. Since all entries in **a** are integers the **short e** format does not display. The ending prompt, >>, indicates that MATLAB is ready for a new command. Changing the last entry in **a** to 7.1 will cause the all entries to be displayed in the **short e** format. In addition, if the symbol **a** is omitted, MATLAB will provide a default symbol of **ans**. See >> **help ans**.

```
>> [3 -4 9;-2 1 7.1]
ans =
  3.0000e+000 -4.0000e+000  9.0000e+000
 -2.0000e+000  1.0000e+000  7.1000e+000
```

Matrix Concepts

It is often the case that the number of rows, r, is identical to the number of columns, c. With $c = r$ the matrix $\boldsymbol{A}$ is called a *square* matrix of order r. If $\boldsymbol{A}$ is square, the elements a_{11}, $a_{22}, \ldots, a_{rr}$ are said to lie on the *main diagonal* of $\boldsymbol{A}$. The main diagonal of a square matrix serves as a convenient reference for square matrices.

In fact, if all the entries of a square matrix which lie below the main diagonal are zeros, the matrix is called *upper triangular*. Likewise, a *lower triangular* matrix will have all zeros above the main diagonal. A *diagonal* matrix will have non zero entries only on the main diagonal. For example, the matrix $\begin{bmatrix} 1 & 2 & 3 \\ 0 & 4 & \pi \\ 0 & 0 & 5 \end{bmatrix}$ is upper triangular and $\begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{bmatrix}$ is diagonal. A special diagonal matrix is the *identity* matrix, usually denoted by $\boldsymbol{I}$, which has ones on the main diagonal and zeros elsewhere. $\boldsymbol{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ is a 2×2 or second order example of the identity matrix. MATLAB has reserved **eye** for the identity matrix.

```
>> eye(2)
ans =
   1   0
   0   1
```

In certain instances, it is necessary to interchange all the corresponding rows and columns of a matrix. In other words, row i of $\boldsymbol{A}$ will become column i in a new matrix called the *transpose* of $\boldsymbol{A}$, denoted mathematically as $\boldsymbol{A}^{\mathrm{T}}$. Using $[\, a_{ij} \,]$ for $\boldsymbol{A}$,

$$\boldsymbol{A}^{\mathrm{T}} = [\, a_{ij}^{T} \,] = [\, a_{ji} \,].$$

For example, if $\boldsymbol{A} = \begin{bmatrix} 1 & 2 & 0 \\ 3 & -7 & 5 \end{bmatrix}$, then $\boldsymbol{A}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & -7 \\ 0 & 5 \end{bmatrix}$. MATLAB has a special symbol to compute the transpose of a matrix. For matrices containing real numbers MATLAB uses a prime to denote the transpose, for example, the transpose of **a** is simply **a'**. Do not confuse the

transpose notation with the standard symbol for the derivative of a function in the calculus. See >> **help transpose** for more details.

Before selecting a MATLAB variable name it is usually a good procedure to determine if MATLAB has reserved use of the name, such as **eye** for the identity matrix. If we wish to use the symbol **atran** for the transpose of **a**, simply type **atran** at the prompt.

```
>> atran
??? Undefined function or variable 'atran'.
```

The response indicates that **atran** is undefined and may be used without conflict.

Terminating an assignment command, >> **a** $=$, with a semicolon suppresses screen output for that variable so that **a** itself is not displayed.

```
>> a = [1 2 0;3 -7 5];
>> atran = a'
atran =
    1    3
    2   -7
    0    5
```

Two other types of matrices are frequently encountered. A matrix of order 1 by c consists of one row and is called a *row* matrix. For example, $[1 \quad 3 \quad 4 \quad \pi]$ would be called a row matrix of order 4, where 4 indicates the number of columns. Likewise, a matrix of order r by 1 is called a *column* matrix of order r. It is conventional to refer to row or column matrices as *vectors* of the appropriate order. Usually, any mention of the order is simply omitted.

To take the place of the number zero in the algebra of real numbers, we define the *null* matrix, $\boldsymbol{\Phi}$, wherein all entries are zeros. A null matrix will have an order consistent with its use. For example, $\boldsymbol{\Phi}$ may be a square matrix in one instance and a column matrix in another. MATLAB reserves the command **zeros** for the null matrix. See >> **help zeros** plus >> **help ones** for yet another special matrix containing all ones.

1.2 Matrix Algebra

In a study of matrix algebra it is important to realize that a matrix, a rectangular array of numbers, is a new quantity – an entity all by itself. Thus, it becomes necessary to develop an algebra which will allow for the manipulation of matrices in a useful and orderly fashion. By analogy with the algebra of real numbers, the concepts of equality, addition, subtraction, multiplication by a constant, multiplication of matrices themselves and all of the various laws of algebra must be defined. As a counterpart of division, the inverse of a matrix will be defined.

Many definitions will be based on intuition and a desire to analyze systems of linear algebraic equations using matrices. As a general rule, if matrices are of order 1×1, the definitions constructed for matrix algebra should be identical to the definitions in the algebra of real numbers.

With these ideas in mind, it is intuitive to define equality of matrices as follows: Two matrices, $\boldsymbol{A}$ and $\boldsymbol{B}$, both of order $r \times c$ are said to be equal if $a_{ij} = b_{ij}$ for $i = 1, 2, 3, \ldots, r$ and $j = 1, 2, 3, \ldots, c$. In other words, the equality $\boldsymbol{A = B}$ requires that (1) matrices are of the same order and (2) all corresponding entries are identical.

Addition and subtraction of matrices require, intuitively, that the matrices involved have the same order. Thus, the sum $\boldsymbol{S = A + B}$ is defined as $s_{ij} = a_{ij} + b_{ij}$ for all i and j. Clearly, $\boldsymbol{A + B = B + A}$. The difference, $\boldsymbol{D = A - B}$, is defined as expected, $d_{ij} = a_{ij} - b_{ij}$ for all i and j.

The concept of a multiple of a matrix, scalar multiplication, is also easy to define. Given $\boldsymbol{A}$, the multiple $\alpha\boldsymbol{A}$ (α is a scalar) requires that each entry in $\boldsymbol{A}$ be multiplied by α. Thus $\alpha\boldsymbol{A}$=α $[\, a_{ij} \,] = [\, \alpha a_{ij} \,]$. It should be noted that $\alpha\boldsymbol{A}$ = $\boldsymbol{A}\alpha$ and that $\alpha(\boldsymbol{A} + \boldsymbol{B}) = \alpha\boldsymbol{A} + \alpha\boldsymbol{B}$.

MATLAB implements these matrix operations as defined. Note that more than one command may be entered on a line separated by either commas or semicolons. For example

```
>> a = [1 2;-4 6];b = [3 1;5 -11];
>> s = a+b
s =
    4    3
    1   -5
>> d = a-b
d =
   -2    1
   -9   17
>> 4*a
ans =
    4    8
  -16   24
```

Matrix Multiplication

For this important concept, intuition fails. In other words, given two matrices of the same order, the product is not obtained by simply multiplying all corresponding entries. In an attempt to motivate matrix multiplication, recall the pair of equations

$$\begin{aligned} 3x - 4y + 9z &= -4 \\ -2x + 1y + 7z &= 7 \end{aligned}$$

and the two matrices, now called $\boldsymbol{A}$ and $\boldsymbol{b}$,

$$\boldsymbol{A} = \begin{bmatrix} 3 & -4 & 9 \\ -2 & 1 & 7 \end{bmatrix} \text{ and } \boldsymbol{b} = \begin{bmatrix} -4 \\ 7 \end{bmatrix}.$$

The remaining unknown quantities, x, y and z, are usually placed in a column matrix $\boldsymbol{x} = [\,x \quad y \quad z\,]^T = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$. Remember, the transpose of a row matrix will be a column matrix. The pair of equations is now denoted by the simple matrix equation $\boldsymbol{Ax} = \boldsymbol{b}$.

The left hand side of the first equation may be obtained from the first row of $\boldsymbol{A}$ and the column matrix $\boldsymbol{x}$ as follows:

$$a_{11}x + a_{12}y + a_{13}z = 3x - 4y + 9z$$

Informally stated, the three entries in the first row of $\boldsymbol{A}$ are multiplied by the three entries in the column $\boldsymbol{x}$ with a sum of products accumulated to yield the first equation. In a similar manner, the second row of $\boldsymbol{A}$ and the column $\boldsymbol{x}$ may be used to produce the left hand side of the second equation.

$$a_{21}x + a_{22}y + a_{23}z = -2x + 1y + 7z$$

In matrix form:

$$\boldsymbol{Ax} = \begin{bmatrix} 3 & -4 & 9 \\ -2 & 1 & 7 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 3x - 4y + 9z \\ -2x + 1y + 7z \end{bmatrix}$$

In general, the product $\boldsymbol{AB} = \boldsymbol{P}$ will be defined provided the number of columns of $\boldsymbol{A}$ equals the number of rows of $\boldsymbol{B}$. If this is true, we say that the matrices are *compatible*. With $\boldsymbol{A}$ $r \times n$ and $\boldsymbol{B}$ $n \times c$, the ij entry in $\boldsymbol{P}$, p_{ij}, is defined by the expression

$$p_{ij} = a_{i1}b_{ij} + a_{i2}b_{2j} + a_{i3}b_{3j} + \cdots + a_{in}b_{nj}.$$

In other words, p_{ij} results from using row i of $\boldsymbol{A}$ and column j of $\boldsymbol{B}$, accumulating the sum of n products. With $1 \leq i \leq r$ and $1 \leq j \leq c$, the product $\boldsymbol{P}$ will be of order $r \times c$.

MATLAB easily implements the product of two matrices. For example, with $\boldsymbol{A}$ 3×2 and $\boldsymbol{B}$ 2×2,

```
>> a = [1 2;3 4;5 6];b = [3 4;-1 2];
>> p=a*b
p =
    1    8
    5   20
    9   32
>> q=b*a
??? Error using ==> *
Inner matrix dimensions must agree.
```

For the second product, **b*a**, MATLAB has provided an error message indicating that the product is undefined. Why?

At the risk of confusion, MATLAB does provide for the intuitive multiplication of two matrices of the same order. For example, if both $\boldsymbol{A}$ and $\boldsymbol{B}$ are 2 by 3, the matrix product $\boldsymbol{AB}$ is

undefined; however, the intuitive *array* product wherein all corresponding entries are multiplied together does exist. Special notation is needed to distinguish the two products. MATLAB places a period or dot in front of the multiplication symbol, .* , to identify the array operation. Consider two 2×3 matrices wherein the matrix product is undefined.

```
>> a = [1 2;3 4;5 6];b = [3 4;-1 2;0 1];
>> a*b
??? Error using ==> *
Inner matrix dimensions must agree.
```

The array product is show below. Convince yourself that the array product is correct.

```
>> a.*b
ans =
    3    8
   -3    8
    0    6
```

We shall discover that the intuitive array operations of multiplication, division and exponentiation, .* , ./ and .^ , have important applications. Keep in mind that array multiplication and divison require matrices of the same order. Please note that matrix addition and subtraction are array operations by definition. Many of MATLAB's built-in functions are designed to operate in the array mode. For example, the trigonometry command **sin** gives

```
>> alpha = [0 pi/6 pi/4 pi/3 pi/2];
>> sin(alpha)
ans =
  0  5.0000e-001  7.0711e-001  8.6603e-001  1.0000e+000
```

MATLAB's **ans** contains familiar results for the sine function. Since **alpha** is a row vector, **sin**(**alpha**) is also a row vector. Note MATLAB's use of **pi** for π.

Matrix Inversion

Associated with a square matrix, say $\boldsymbol{A}$, is a special matrix called the *inverse* of $\boldsymbol{A}$, denoted by the symbol $\boldsymbol{A}^{-1}$. $\boldsymbol{A}^{-1}$ is just a symbol, it does not mean the reciprocal of $\boldsymbol{A}$. The inverse of $\boldsymbol{A}$ is a matrix with the property

$$\boldsymbol{A}^{-1}\boldsymbol{A} = \boldsymbol{A}\boldsymbol{A}^{-1} = \boldsymbol{I}.$$

Not all matrices have inverses. If $\boldsymbol{A}^{-1}$ exists, then $\boldsymbol{A}$ is called *nonsingular* (or sometimes *invertible*); if $\boldsymbol{A}^{-1}$ does not exist, $\boldsymbol{A}$ is said to be *singular*. Later in this section another, easy to use, definition of a singular matrix will be given.

If A^{-1} exists, the solution to the square linear system $Ax = b$ may be obtained by multiplying both sides of the equation by the inverse of A, keeping the inverse on the left.

$$A^{-1}Ax = A^{-1}b \;\Rightarrow\; Ix = A^{-1}b \;\Rightarrow\; x = A^{-1}b$$

As a practical matter, computation of the inverse of a square matrix is seldom needed as other numerical procedures are used to solve $Ax = b$; however, MATLAB does provide for the computation of inverses using the **inv** command. The actual numerical procedure involves topics to be treated later.

The inverse of $\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ may be computed, using the **long e** format, as follows:

```
>> format long e
>> a = [1 2;3 4],ainv = inv(a)
a =
    1    2
    3    4
ainv =
  -2.000000000000000e+000    9.999999999999998e-001
   1.500000000000000e+000   -4.999999999999999e-001
```

To test the result we compute **a*ainv** which should be **eye(2)**.

```
>> a*ainv
ans =
   1.000000000000000e+000                    0
   8.881784197001252e-016    9.999999999999996e-001
```

Within the accuracy of MATLAB, see Chapter 2, the numerically computed product **a*ainv** is the identity matrix.

MATLAB provides for a symbolic mode of operation using Maple, a product of Waterloo Maple, Inc. The matrix A may be inverted symbolically, see >> **help sym,** as follows:

```
>> a
a =
    1    2
    3    4
>> ainvs = inv(sym(a))
ainvs =
[   -2,    1]
[  3/2, -1/2]
>> a*ainvs
ans =
[ 1, 0]
[ 0, 1]
```

As you can see, symbolic matrices are displayed in a different format using brackets. The command **>> help symbolic** will provided a list of symbolic operations. It is important to understand the difference between actual numerical computations using various algorithms and symbolic manipulations using the rules of mathematics. The inverse, **ainvs**, is one example of a symbolic operation. Side by side comparison shows the differences.

```
ainv =                                                    ainvs =
  -2.000000000000000e+000   9.999999999999998e-001       [  -2,   1]
   1.500000000000000e+000  -4.999999999999999e-001       [ 3/2, -1/2]
```

Determinants

Associated with a square matrix, $\boldsymbol{A}$, there is a number called the determinant of $\boldsymbol{A}$, denoted det($\boldsymbol{A}$), which will prove to be very important in several respects.

Definitions for three cases are as follows:

If $\boldsymbol{A}$ is 1×1, $\boldsymbol{A} = [\, a_{11}]$ then det($\boldsymbol{A}$) $= a_{11}$.

If $\boldsymbol{A}$ is 2×2, $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ then det($\boldsymbol{A}$) $= a_{11}a_{22} - a_{12}a_{21}$.

Simply stated, the determinant of a 2×2 matrix is the product of the entries on the main diagonal minus the product of the entries off the main diagonal.

If $\boldsymbol{A}$ is 3×3, $\boldsymbol{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$ then

$$\det(\boldsymbol{A}) = a_{11}(a_{22}a_{33} - a_{23}a_{32}) - a_{12}(a_{21}a_{33} - a_{23}\,a_{31}) + a_{13}\,(a_{21}\,a_{32} - a_{22}\,a_{31}).$$

The 3×3 definition shows the complexity of the determinant. The definition given seems to stress the first row $-$ note the leading multipliers. With rearrangement of the six terms any row or column may play the leading role. Fortunately, there are various theorems about determinants that provide a systematic procedure for numerical computation. A full discussion of determinants requires the introduction of additional topics. Virtually all textbooks treating elementary linear algebra provide the necessary mathematics.

Computation of the determinant of a square matrix is seldom needed; however, MATLAB does provide for the computation of determinants using the **det** command. The actual numerical procedure involves topics to be treated later.

The determinant of the matrix $\begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 1 \\ 1 & 0 & 1 \end{bmatrix}$ may be computed as follows:

```
>> det([1 1 1;2 3 1;1 0 1])
ans =
   -1
```

The inverse of a square matrix and the determinant of that matrix are intimately related. If det($\boldsymbol{A}$) $= 0$, $\boldsymbol{A}^{-1}$ does not exist and, as noted above, $\boldsymbol{A}$ is said to be singular. Frequently, MATLAB provides a warning message that a matrix is singular to working precision. The implications of the warning message will be discussed later.

Determinants have many interesting properties including

$$\det(\ \alpha\ \boldsymbol{A}\ ) = \alpha^n \det(\ \boldsymbol{A}\ ) \text{ for an } n \times n \text{ matrix}$$

$$\det(\ \boldsymbol{A}\ \boldsymbol{B}\ ) = \det(\ \boldsymbol{A}\ ) \det(\ \boldsymbol{B}\ )$$

One unfortunate result of the complicated definition is the loss of a linear relationship, in other words, $\det(\ \boldsymbol{A}+\boldsymbol{B}\ ) \neq \det(\ \boldsymbol{A}\ ) + \det(\ \boldsymbol{B}\ )$. Frequently, the determinant of $\boldsymbol{A}$ is denoted by $|\boldsymbol{A}|$. Do not confuse this shorthand notation with absolute value.

1.3 MATLAB Management

MATLAB provides three very important commands which will allow the user to save all or selected variables present in the workspace, to retrieve those variables at a later date as needed, and to save the text displayed on the screen for subsequent reporting efforts. The commands are **save**, **load**, and **diary**. It is a good idea to read the help screens for each of these commands.

The command

```
>> save file1
```

will save all workspace variables to a file called file1.mat in the work directory. Also, **>> save file2 x y z** will save only the variables x y z to a file called file2.mat in the work directory. Note that commas are not used to separate the variables. Without thinking, it is rather easy to unintentionally delete previously saved information with the **save** command. For example, a subsequent **>> save file2 x** will indeed save x; however, the data in y and z will be lost. In some installations of MATLAB it may be necessary to specify a path before the filename. See also the related command **addpath**.

A saved file may be retrieved with the **load** command

```
>> load file1
```

which will retrieve all the variables saved in file1. The commands **who** or **whos** will allow the MATLAB user to view all active variables in the workspace.

The **diary** command allows the activity of the command window to be saved to a text file for editing. This feature allows the user to easily incorporate MATLAB results in reports

using the cut, copy, and paste features of most word processing software. **diary** functions much like a tape recorder.

The command

>> **dairy fileout**

will open a file called fileout and begin recording MATLAB activity. To suspend recording, issue the command >> **diary off** , start recording again with >> **diary on**. The file is saved in the work directory unless a path is specified. If you prefer, the MATLAB information in the Command Window may be selected and copied to another application. It is perhaps best to experiment with the **save**, **load**, and **diary** commands to assess their features.

Finally, there are cases where one wishes to make a minor change to a rather large matrix in the workspace. A particular location, say the entry found in row 4 and column 5, may be changed using, >> **m(4,5)** = the new value. An alternative procedure is make use of

>> **open m**

which will open the matrix **m** in an array editor where changes may be made in spreadsheet fashion. To make changes in the matrix, select a cell, type the new value and press enter. The new values are now saved in the workspace matrix **m**.

1.4 Problems

1-1. Given $\boldsymbol{A} = \begin{bmatrix} -3 & 1 \\ 4 & 2 \end{bmatrix}$ $\boldsymbol{B} = \begin{bmatrix} -1 & 1 & 5 \\ 3 & 4 & 10 \end{bmatrix}$

Compute each of the following by hand and verify the results with MATLAB. If the computations are not possible explain why.
a) $\boldsymbol{A} + \boldsymbol{B}$, b) $\boldsymbol{AB}$, c) $\boldsymbol{BA}$, d) $\boldsymbol{B}^{\mathrm{T}}\boldsymbol{A}$, e) $\boldsymbol{A}$ **.*** $\boldsymbol{A}$, f) $\boldsymbol{B}$ **.**^2

1-2. Let $\boldsymbol{A} = \begin{bmatrix} 3 & -3 \\ 0 & 1 \\ -2 & 2 \end{bmatrix}$ and $\boldsymbol{B} = \begin{bmatrix} 2 & -1 & 4 \\ 3 & 5 & -4 \\ 1 & -2 & 3 \end{bmatrix}$

Compute each of the following by hand and verify the results with MATLAB. If the computations are not possible explain why.
a) $\boldsymbol{B} - \boldsymbol{A}$, b) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}$, c) $\boldsymbol{A}^{\mathrm{T}}\boldsymbol{B}\boldsymbol{A}^{\mathrm{T}}$, d) $\boldsymbol{B}$**./**$\boldsymbol{B}$, e) $\boldsymbol{A}^{\mathrm{T}}$**.*** $\boldsymbol{B}$

1-3. The definition of the product of two matrices leads to some unexpected results. Consider the matrix equation $\boldsymbol{AB} = \boldsymbol{\Phi}$. Find two non-null matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ whose product is equal to $\boldsymbol{\Phi}$. Use 2×2 matrices. In other words if $\boldsymbol{AB} = \boldsymbol{\Phi}$ we may not infer that either $\boldsymbol{A}$ or $\boldsymbol{B}$ is equal to a null matrix $\boldsymbol{\Phi}$.

1-4. Another unexpected result: $\boldsymbol{AC} = \boldsymbol{BC}$ does not imply that $\boldsymbol{A} = \boldsymbol{B}$. Explain why. Hint: $\boldsymbol{AC} = \boldsymbol{BC}$ leads to $(\boldsymbol{A} - \boldsymbol{B})\boldsymbol{C} = \boldsymbol{\Phi}$.

1-5. Write the MATLAB command to enter the matrix $\begin{bmatrix} \pi & -2 \\ 3 & e \end{bmatrix}$, call it **m**, into MATLAB without displaying the matrix on the screen.

1-6. Execute the MATLAB command >> **a = [1 3;2 4]; b = a(:)**
What is displayed on the screen? Investigate the use of the colon symbol in MATLAB using the help feature. Give examples.

1-7. a. Use the MATLAB help feature to investigate **rot90.**
b. Let $\boldsymbol{P}$ = **rot90(eye(**5**))**. If $\boldsymbol{M}$ is any 6×6 matrix, compute the products $\boldsymbol{PM}$ and and $\boldsymbol{MP}$. Describe the results.

1-8. Let $\boldsymbol{U}$ be any $n \times n$ upper triangular matrix and $\boldsymbol{D}$ be any $n \times n$ diagonal matrix.
a. Is the product $\boldsymbol{UD}$ an upper triangular or diagonal matrix? Explain.
b. How many products are needed to compute the product $\boldsymbol{UD}$? Find the maximum number. Note: 0 times something does not count as a product.

1-9. Let $\boldsymbol{M}$ be any 3×3 matrix. What is the result of the MATLAB products $\boldsymbol{M}*$**ones**(3) and **ones**(3)$*\boldsymbol{M}$? Experiment and explain.

1-10. What is the purpose of the MATLAB command **trace**?

1-11. The product of two matrices is one of most important concepts in Chapter 1. In your own words, what is the motivation for the non-intuitive row times column definition.

1-12. Define a, b, c and d to be symbolic variables in MATLAB with the statement
>> **syms a b c d**
Find the symbolic inverse of $\boldsymbol{M} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$.

1-13. A MATLAB statement may be continued to the next line by entering an ellipsis of three periods, **...** , followed by enter. In addition, MATLAB accepts $.5e{-}6$ for 0.0000005. Consider the following lower triangular matrix **w** due to Wilkinson:

```
>> w = [.9143e-4,0,0,0; ...
.8762,.7156e-4,0,0; ...
.7943,.8143,.9504e-4,0; ...
.8017,.6123,.7165,.7123e-4]
```

Use MATLAB to compute **det**(**w**), **inv**(**w**)$*$**w**, and **w**$*$**inv**(**w**). Comment on the results.