

Applied Engineering Problem Solving (CHEN.3170)

Part I: Matlab Overview

Lesson 2: Introduction to Linear Algebra & Array and Matrix Operations in Matlab

Lesson Introduction/Overview

The previous lesson introduced Matlab by focusing on evaluating and plotting functions of a single variable. In this unit we introduce arrays and matrix operations in Matlab in much more detail, and illustrate their use in a variety of applications -- where one special emphasis will be on evaluating and plotting functions of two variables.

Although Matlab can treat multidimensional arrays, we will focus our attention at this point on 1-D and 2-D arrays -- which are commonly referred to as vectors and matrices, respectively. As part of the current development, we will introduce some common notation and analytical manipulations from the field of Linear Algebra, and discuss some concepts related to the analytical solutions of systems of equations. The discussion of computer techniques for the solution of linear and nonlinear equations will come later in the semester (Lesson #6). Our interest, at present, is primarily associated with the basic notation of linear algebra, on the fundamentals of matrix addition, subtraction, and multiplication, and on some fundamental concepts concerning the solution of linear systems of equations. And, from a Matlab perspective, we are also interested in how matrix operations are related to the element-by-element arithmetic introduced earlier in Lesson #1.

Note: Notice that the above list of linear algebra operations does not include matrix division. This is because there is no such thing as formal matrix division! Instead, one defines an inverse matrix and does matrix multiplication with the inverse matrix. For example, for an equation containing scalar variables, say $ax = b$, we can write x as

$$x = \frac{b}{a} = \left(\frac{1}{a}\right)b = a^{-1}b$$

where each of these forms are equivalent. However, for a matrix equation, $\mathbf{Ax} = \mathbf{b}$, the only valid way to write the solution vector \mathbf{x} is given by

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$$

where the bold face text implies that the variable is a matrix (upper case) or vector (lower case). Thus, we see that there is no division involved here. This is a confusing point because Matlab, and many of the texts that describe various Matlab matrix operations, routinely refer to matrix division. This is a point that we will discuss later in this lesson and touch on again in Lesson #6. For now, whenever you see the matrix division operator discussed, just think of it as an inverse operation -- and we will add more substance to this statement a little later on...

The subjects associated with this lesson are contained primarily in Chapters 2, 3 and 10 of the Matlab text by Gilat and in Chapters 2 and 8 in your Numerical Methods text by Chapra. In addition, Section 11.1 in Chapra discusses the matrix inverse and Section 13.1 also briefly

introduces the subject of matrix eigenvalues, both of which will be discussed here in Lesson #2. You should read these chapters and individual sections, paying special note to the following:

- Creating and working with arrays in Matlab and obtaining a good understanding of array indexing.
- Element-by-element operations (addition, subtraction, multiplication, division, and exponentiation).
- Various matrix operations (addition, subtraction, multiplication, and the matrix transpose).
- Several concepts from Linear Algebra including the inverse matrix and matrix eigenvalues and eigenvectors.
- Some special matrices and their generation in Matlab.

Note that the whole field of linear algebra was developed to work with, and to find solutions to, systems of algebraic equations. This is an extremely important area -- we plan to study the basic concepts, terminology, and analytical manipulations here in this lesson, and then the practical application of these concepts will be treated in more detail in Lesson #6 after we discuss some computational solution techniques for linear and nonlinear equations. Thus, you will have to wait a little to see the real application of many of the concepts discussed here. However, since we want to stay application-focused throughout this course, it should be noted that another particularly important general application of a matrix or array is simply for storage of information -- and this is the primary application area we will explore further in this lesson.

Array Indexing in Matlab

In general, you should think of a matrix as a two-dimensional array of cells that holds information. Each cell can contain a number, a character, or a sequence of numbers and/or characters. Thus, in general, a matrix element can itself be a matrix, which can contain matrices, and so on. For usage in this lesson, however, we will stick to the normal case where the matrix simply holds a rectangular array of numbers, such as

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 2 & 4 \\ 5 & -1 & 3 & 1 \end{bmatrix}$$

In this case, \mathbf{A} is said to be a 2×4 matrix, with 2 rows and 4 columns. An arbitrary element of the matrix is denoted as a_{ij} , where the i subscript refers to row i and the j subscript denotes column j . Thus, the a_{22} element is -1 and the value of a_{14} is 4, etc..

Now, in Matlab, we could generate this matrix explicitly with

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

where we see that the various rows are separated by a semi-colon within the square brackets. Note that, to form a regular 2-D array, every row must have the same number of entries (i.e. same number of columns).

To address a single element of **A**, we simply specify the appropriate row and column indices within parentheses. For example, the a_{22} element is printed to the screen with the following command:

```
>> A(2,2)
ans =
    -1
```

I can also assign all the elements of the first row of **A** to vector **b** as follows:

```
>> b = A(1,:)
b =
     0     1     2     4
```

where, as we saw in the last lesson, the `:` is the usual repeat operator. The RHS of this Matlab expression says “assign row 1, all columns of **A** to row vector **b**.”

Similarly, I can assign the third column of **A** to column vector **c** by

```
>> c = A(:,3)
c =
     2
     3
```

In general, when referring to specific elements of a 2-D array, we simply need to specify the appropriate row and column indices of the original matrix. For example, what would be the outcome of typing $d = A(:, [1 \ 4])$ or $e = A(1, 1:3:4)$, etc.? Try it and see...

This array indexing capability is important in many applications in Matlab. For example, I could store the homework grades for this class in a matrix. Let's say there will be 12 homework sets and that there are 25 students in the class. Then an array, **G**, with 25 rows and 12 columns could store all the desired grade information. Now, assuming that matrix **G** was available, I could access all 12 grades for Student #5 by typing $G(5,:)$, and compute his or her average homework grade with $\text{sum}(G(5,:))/\text{length}(G(5,:))$. Similarly, I could easily calculate the class average on HW #12 with $\text{sum}(G(:,12))/\text{length}(G(:,12))$.

It is important to note that **G** is a matrix, that $G(5,:)$ and $G(:,12)$ are vectors (row and column vectors, respectively), and that $G(5,12)$ is a scalar (i.e. the grade on HW #12 for Student #5). So when referring to variable **G**, it makes a difference whether or not you specify the whole array or only selected portions of the array.

As seen above, Matlab has many built-in commands for working with arrays. The *sum* and *length* commands that were used above, as well as the *max*, *min*, *size*, *find*, etc. commands, are used quite frequently -- and I suggest that you use the Matlab *help* facility to learn about some of these very useful functions.

Note: If you would like to experiment with the above commands, you can easily generate a 25×12 matrix of random entries with values between 60 and 100 as follows (here I assume no one would get less than a 60 on their HW assignments):

```
>> G = (100 - 60)*rand(25,12) + 60;
```

Now executing the above commands in sequence gives

```
>> G(5,:)
```

```

ans =
95.6520  67.9526  66.0349  84.8524  77.3163  79.9325  60.5145  82.1937  75.7296  67.6447
91.7549  82.6295

>> sum(G(5,:))/length(G(5,:))
ans =
    77.6840

>> sum(G(:,12))/length(G(:,12))
ans =
    83.2343

```

Thus, it looks like Student #5 did okay on his or her HW with about a 78 average. In addition, the whole class seemed to do pretty well on HW #12, with a class average of about 83. Hopefully you will even have better grades than these randomly selected values...

Now, when working with arrays, there are two types of arithmetic operations that can be performed; element-by-element arithmetic and matrix arithmetic. Getting comfortable with these operations is essential for the efficient use of Matlab in real problems, so we will take a little time to be very explicit with the rules of both these types of operations.

Element-By-Element Operations

This type of arithmetic is conceptually simple -- one simply performs the desired operation element-by-element for every term of the array. To do element-by-element operations with two or more arrays, the array sizes must be identical. Using discrete notation, we can explicitly define the various array operations and then show a specific example using the following two arrays:

```

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

```

Addition: **C = A + B** $c_{ij} = a_{ij} + b_{ij}$ (1)

```

>> C = A + B
C =
     3     5     7
     3     5     7

```

Subtraction: **C = A - B** $c_{ij} = a_{ij} - b_{ij}$ (2)

```

>> C = A - B
C =
    -3    -3    -3
     3     3     3

```

Multiplication: **C = A.*B** $c_{ij} = a_{ij} * b_{ij}$ (3)

```

>> C = A.*B
C =
     0     4    10
     0     4    10

```

Division: **C = A./B** $c_{ij} = a_{ij} / b_{ij}$ (4)

```

>> C = A./B
Warning: Divide by zero.
(Type "warning off MATLAB:divideByZero" to suppress this warning.)

```

```
C =
      0      0.2500      0.4000
    Inf      4.0000      2.5000
```

Exponentiation: $\mathbf{C} = \mathbf{A}.^n$ $c_{ij} = a_{ij}^n$ (5)

```
>> C = A.^2      % n = 2, for this example
C =
      0      1      4
      9     16     25
```

where all the results are as expected (notice how gracefully Matlab handles a division by zero -- most other languages simply crash the program). Note here that element-by-element multiplication, division, and exponentiation require the use of “dot arithmetic” as was used extensively in Lesson #1.

Matrix Operations

Matrix addition and subtraction are identical to the element-by-element operations noted above. Thus, the Matlab syntax and the result of these operations are also the same. Because, by definition, matrix addition and subtraction are done element-by-element,

Addition: $\mathbf{C} = \mathbf{A} + \mathbf{B}$ $c_{ij} = a_{ij} + b_{ij}$ (6)

Subtraction: $\mathbf{C} = \mathbf{A} - \mathbf{B}$ $c_{ij} = a_{ij} - b_{ij}$ (7)

the Matlab syntax **does not** include a dot in front of the + or – symbol. Recall that Matlab performs matrix operations by default, so matrix **A** added to matrix **B** is simply $\mathbf{C} = \mathbf{A} + \mathbf{B}$ and, since this also happens to be identical to the above element-by-element operation, no dot is required (or allowed) for either.

Now, **matrix multiplication** is a completely different story!!! One way to introduce the rules for matrix multiplication is by comparison to the notation associated with a system of algebraic equations. To start, let’s write a specific system of three equations and three unknowns,

$$\begin{aligned} 3x_1 - 2x_2 + 2x_3 &= 1 \\ x_1 + 2x_2 - 3x_3 &= 0 \\ 4x_1 + x_2 + 2x_3 &= 0 \end{aligned} \quad (8)$$

where the unknowns are x_1 , x_2 , and x_3 . Now, we can think of each equation as a row, with the coefficients of the three unknowns properly ordered into the corresponding columns or terms of each equation. For example, we can say that the coefficient of the third unknown in the second equation is -3 . If we group all the coefficient information into a matrix (since a matrix is just a rectangular array for storage of information), then $a_{23} = -3$, where the row index corresponds to the equation number and the column index denotes the unknown of interest (third unknown or x_3 in the above example). With this simple one-to-one correspondence, we can write a general 3×3 system of equations as

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad (9)$$

where a_{ij} is the coefficient of x_j in the i^{th} equation, and b_i is the value on the RHS of equation i .

Recognizing that quantities with a single index are stored in vectors and that information cataloged with two subscripts to identify the storage location refer to 2-D arrays (matrices), we can write the above equations, using matrix-vector notation, as

$$\mathbf{Ax} = \mathbf{b} \quad (10)$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \quad \text{and} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

In eqn. (10), the notation, \mathbf{Ax} , implies that vector \mathbf{x} is pre-multiplied by matrix \mathbf{A} . Since we want eqn. (10) to be a shorthand equivalent to eqn. (9), then the rules of matrix-vector multiplication **must** give the result in eqn. (9), or

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} \quad \Rightarrow \quad \begin{aligned} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 &= b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 &= b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 &= b_3 \end{aligned} \quad (11)$$

Basically, the correspondence in eqn. (11) says it all! Focusing, for example, on the first equation in the set, we see that

$$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1$$

can be written as

$$b_1 = \sum_{j=1}^3 a_{1j}x_j \quad (12)$$

In fact, for any element of \mathbf{b} , say for example, b_i , which corresponds to row i of the system of equations, we have

$$b_i = \sum_{j=1}^3 a_{ij}x_j \quad (13)$$

and this is precisely what we mean by **matrix-vector multiplication**, that is

$$\mathbf{Ax} = \mathbf{b} \quad \Rightarrow \quad b_i = \sum_j a_{ij}x_j \quad (14)$$

This is often referred to as the row view of matrix-vector multiplication. The inner product of row i of matrix \mathbf{A} with vector \mathbf{x} gives the scalar b_i . Recall that the inner product of two vectors, \mathbf{x} and \mathbf{y} , gives a scalar, α , or

$$\alpha = \sum_i x_i y_i \quad (15)$$

Thus, matrix-vector multiplication is simply a sequence of inner product operations -- one for each row of the system of equations.

Now, extending this view to matrix-matrix multiplication is quite straightforward. To do this, let's take eqn. (10) and write it for two different RHS vectors, or

$$\mathbf{Ax}_1 = \mathbf{b}_1 \quad \text{and} \quad \mathbf{Ax}_2 = \mathbf{b}_2$$

where \mathbf{x}_1 is the solution to the first set of equations and \mathbf{x}_2 is the vector satisfying the second system -- where the coefficient matrix is the same in both cases but the RHS vectors are different. Now, with our view that a matrix is simply a convenient form for storage of information, let's store the two solution vectors \mathbf{x}_1 and \mathbf{x}_2 in a matrix, or

$$\mathbf{X} = [\mathbf{x}_1 \quad \mathbf{x}_2] = \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix}$$

where the precise notation is important: $\mathbf{X} \rightarrow$ matrix, $\mathbf{x}_j \rightarrow j^{\text{th}}$ solution vector, and $x_{kj} \rightarrow k^{\text{th}}$ component of \mathbf{x}_j .

Writing the two RHS vectors in a similar way, we can form a matrix equation, as follows

$$\mathbf{AX} = \mathbf{B} \tag{16}$$

or,

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_{11} & x_{12} \\ x_{21} & x_{22} \\ x_{31} & x_{32} \end{bmatrix} = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} \tag{17}$$

In general, if the size of \mathbf{A} is $n \times p$ and \mathbf{X} is of order $p \times m$, then the resultant matrix, \mathbf{B} , is of order $n \times m$ -- where we note that, for valid matrix multiplication, the inner matrix dimensions must agree. This says that the number of columns of \mathbf{A} must be equal to the number of rows of \mathbf{X} for the operation, \mathbf{AX} , to be valid. This also implies that, in general, $\mathbf{AX} \neq \mathbf{XA}$ -- that is, the order of operation is absolutely essential!!! In fact, in our above example, $\mathbf{AX} = \mathbf{B}$, we have $(3 \times 3)(3 \times 2) = (3 \times 2)$, so the result will be a 3×2 matrix, as expected. However, note that the operation \mathbf{XA} is not even defined since $(3 \times 2)(3 \times 3)$ has inner matrix dimensions that do not match...

Following the row view of matrix multiplication, we can write the discrete form of eqns. (16) and (17) as

$$b_{ij} = \sum_k a_{ik} x_{kj} \tag{18}$$

where the individual element, b_{ij} , of the resultant matrix, \mathbf{B} , is given as the inner product of row i of matrix \mathbf{A} and column j of matrix \mathbf{X} -- where clearly the number of elements (columns) in each row of \mathbf{A} must be equal to the number of values (rows) in each column of \mathbf{X} (i.e. the inner matrix dimensions must agree). In summary, I like to view the value of b_{ij} simply as "row i of \mathbf{A} into column j of \mathbf{X} ". With this view, the precise definition of **matrix-matrix multiplication** is given by

$$\mathbf{AX} = \mathbf{B} \quad \Rightarrow \quad b_{ij} = \sum_k a_{ik} x_{kj} \tag{19}$$

Well, you are now an expert with matrix multiplication! As a test of your understanding, you should verify, by hand calculation, that the following example cases are indeed correct. My suggestion is to always check the size of the result first $[(n \times p)(p \times m) = n \times m]$ and then do the required inner products to form all the elements of the resultant matrix (and be careful with the arithmetic). Here are some examples:

$$\begin{bmatrix} 1 & 2 & 3 \\ 0 & 1 & -1 \\ -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 10 \\ 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 1 & 0 \\ 2 & 2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 0 & 1 \\ 0 & 1 & -1 & 2 \\ 1 & -2 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 3 & -1 & 4 \\ 3 & 2 & -2 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 2 & 2 \\ 3 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 2 & 2 \\ 3 & 1 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & -1 \end{bmatrix} \begin{bmatrix} 1 & 2 \\ 0 & 1 \\ 1 & -2 \end{bmatrix} = \begin{bmatrix} 0 & 6 \end{bmatrix}$$

$$\begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} \begin{bmatrix} 1 & 2 & -1 \end{bmatrix} = \begin{bmatrix} 3 & 6 & -3 \\ 1 & 2 & -1 \\ 2 & 4 & -2 \end{bmatrix}$$

$$\begin{bmatrix} 3 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix} = 3$$

So, were you able to generate all these by hand?

The purpose for addressing some linear algebra early in the semester is because this is Matlab's strong point -- after all, Matlab stands for **Matrix Laboratory** -- and simple matrix manipulation is very useful in many application areas. Thus, as you might expect, Matlab can do the above matrix multiplication examples rather easily, as we can verify with the following Matlab code:

```
>> format compact
>> A = [1 2 3; 0 1 -1; -1 1 1]; x = [3; 2; 1]; A*x
ans =
    10
     1
     0

>> A = [2 1 0; 2 2 1]; B = [1 1 0 1; 0 1 -1 2; 1 -2 0 0]; A*B
ans =
     2     3    -1     4
     3     2    -2     6
```



```
>> A = [1 0 0; 0 1 0; 0 0 1]; B = [1 0; 2 2; 3 1]; A*B
ans =
     1     0
     2     2
     3     1

>> c = [1 2 -1]; C = [1 2; 0 1; 1 -2]; c*C
ans =
     0     6

>> d = [3 1 2];
>> d'*c
ans =
     3     6    -3
     1     2    -1
     2     4    -2

>> d*c'
ans =
     3
```

Note, in the last two examples, how the transpose operator (a single quote) was used to convert a row vector (which is easier to enter since it does not require any semi-colons) into a column vector for the operations desired in the examples. In general, the matrix transpose simply interchanges the rows and columns of a matrix and it is formally defined by

$$\mathbf{B} = \mathbf{A}^T \quad \Rightarrow \quad b_{ij} = a_{ji} \quad (20)$$

Some Special Matrices

Before leaving this brief demonstration of matrix operations in Matlab, it is important to note that there are many special matrices that are of interest in a variety of applications -- and Matlab makes it very easy to develop and work with these special arrays. For example, a 3×3 identity matrix, a 2×3 matrix full of zeros, a 5×2 matrix with all ones, and a diagonal matrix with 1, 2, -1, and 0 along the diagonal are easily generated via the following Matlab code:

```
>> eye(3)
ans =
     1     0     0
     0     1     0
     0     0     1

>> zeros(2,3)
ans =
     0     0     0
     0     0     0

>> ones(5,2)
ans =
     1     1
     1     1
     1     1
     1     1
     1     1

>> diag([1 2 -1 0])
ans =
     1     0     0     0
     0     2     0     0
     0     0    -1     0
     0     0     0     0
```

In addition, we can also generate sequences of random numbers, such as a 1000×1 column vector with uniformly distributed random numbers between 0 and 1, and a 1×2000 row vector with normally distributed random values having a mean value of $\mu = 10$ and a standard deviation of $\sigma = 2$, or

```
>> x = rand(1000,1);           % uniformly distributed random numbers
>> y = 2*randn(1,2000)+10;     % normally distributed values with mean = 10, std = 2
```

To see these last two quantities, we can plot the uniformly and normally distributed vectors as follows:

```
>> subplot(2,1,1),hist(x)
>> title('1000 Uniformly Distributed Numbers Between 0 and 1')
>> xlabel('Numerical Values'), ylabel('Frequency')

>> subplot(2,1,2),hist(y)
>> title('2000 Normally Distributed Numbers with \mu = 10 and \sigma = 2')
>> xlabel('Numerical Values'), ylabel('Frequency')
```

with the resultant plot given in Fig. 1. Note here, as expected, the *rand* command gives a nearly uniform distribution of random numbers, and the *randn* function gives an approximate Gaussian distribution -- these are clearly shown in the upper and lower subplots, respectively, in Fig. 1.

Note: In general, if there are any Matlab commands used in the examples in these notes that you are not familiar with, you can always type *help command_name* to get the full scoop...

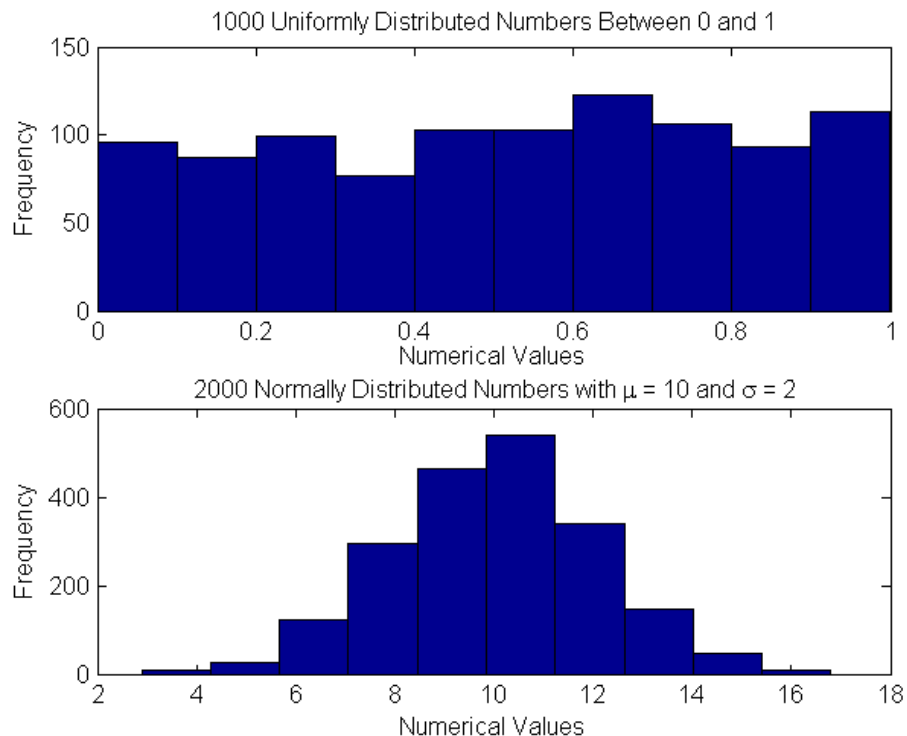


Fig. 1 Visualization of uniformly and normally distributed sequences in Matlab.

Well, this completes our brief introduction to array and matrix operations in Matlab. You should now be comfortable with creating and working with arrays in Matlab and with using array indexing to extract and/or modify information from existing arrays. The differences between dot arithmetic and matrix arithmetic should now be clear, and you should have a good foundation for some additional concepts from the subject of Linear Algebra.

Linear Algebra Concepts and Analytical Manipulations

Some of the notation needed for an introductory treatment of linear algebra has already been used in our discussions from above. This subsection formalizes some of this notation and it also introduces a number of new concepts and analytical manipulations that will assist us in our study of solution methods for linear and nonlinear equations. Here we will only introduce analytical techniques -- we delay the discussion of numerical/computer methods until Lesson #6.

Let's start our introductory treatment of Linear Algebra by defining some formal notation associated with vectors and matrices.

A **vector** is simply an ordered set of numbers or quantities. A column vector is usually written as

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

and a row vector is given by

$$\mathbf{x}^T = [x_1 \quad x_2 \quad x_3]$$

where the length of the vector is equal to the number of elements. The usual notation, without the superscript T to denote the transpose operation, refers to the multiple row, single column format -- thus, the vector quantity is referred to as a column vector. Similarly, the row vector has only one row but multiple columns.

Given two column vectors, $\mathbf{x} = [x_1 \quad x_2 \quad x_3]^T$ and $\mathbf{y} = [y_1 \quad y_2 \quad y_3]^T$, the most common arithmetic operations are defined as follows:

Addition

$$\mathbf{z} = \mathbf{x} + \mathbf{y} = \begin{bmatrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \end{bmatrix} \quad \text{or} \quad z_i = x_i + y_i \quad (21)$$

Multiplication by a Scalar

$$\mathbf{z} = \alpha \mathbf{x} = \begin{bmatrix} \alpha x_1 \\ \alpha x_2 \\ \alpha x_3 \end{bmatrix} \quad \text{or} \quad z_i = \alpha x_i \quad (22)$$

Dot Product (inner product)

$$\alpha = \mathbf{x} \cdot \mathbf{y} = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = x_1 y_1 + x_2 y_2 + x_3 y_3 \quad \text{or} \quad \alpha = \mathbf{x}^T \mathbf{y} = \sum_i x_i y_i \quad (23)$$

Outer Product

$$\mathbf{A} = \mathbf{xy}^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & x_1 y_3 \\ x_2 y_1 & x_2 y_2 & x_2 y_3 \\ x_3 y_1 & x_3 y_2 & x_3 y_3 \end{bmatrix}$$

$$\text{or} \quad \mathbf{A} = \begin{bmatrix} a_{ij} \end{bmatrix} \quad \text{where} \quad a_{ij} = x_i y_j \quad (24)$$

A **matrix** is a regular 2-D array of numbers or quantities and is usually denoted with a bold capital letter,

$$\mathbf{A} = \begin{bmatrix} a_{ij} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} b_{ij} \end{bmatrix}, \quad \text{etc.}$$

where i is the row index and j is the column index. For example, a 3×3 matrix can be written as

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Again, many of the common arithmetic operations with matrices are as follows:

Addition

$$\mathbf{C} = \mathbf{A} + \mathbf{B} \quad \text{or} \quad c_{ij} = a_{ij} + b_{ij} \quad (25)$$

Scalar Multiplication

$$\mathbf{C} = \alpha \mathbf{A} \quad \text{or} \quad c_{ij} = \alpha a_{ij} \quad (26)$$

Matrix Multiplication

$$\mathbf{C} = \mathbf{AB} \quad \text{or} \quad c_{ij} = \sum_k a_{ik} b_{kj} \quad (27)$$

where the number of columns of the first matrix must be equal to the number of rows of the second matrix, or

$$\mathbf{A} \times \mathbf{B} = \mathbf{C}$$

$$(m \times n)(n \times p) = (m \times p)$$

where the notation $m \times n$, for example, implies that the matrix has m rows and n columns.

Matrix-Vector Multiplication

$$\mathbf{y} = \mathbf{Ax} \quad \text{or} \quad y_i = \sum_j a_{ij} x_j \quad (28)$$

Matrix Transpose

$$\mathbf{C} = \mathbf{A}^T \quad \text{or} \quad c_{ij} = a_{ji} \quad (29)$$

Also there are a number of special matrices of interest. For example, some of these matrices include diagonal, triangular, square, and identity matrices, as well as symmetric and skew symmetric matrices, etc.. Most of the names for these matrices are self-explanatory. A square matrix is one with an equal number of rows and columns. A lower triangular matrix is a square matrix with all zero elements above the diagonal elements. An identity matrix, as illustrated above using Matlab's *eye* command, is a diagonal square matrix with ones along the main diagonal and zeros in all the off-diagonal locations (typically denoted as \mathbf{I}). Also, a real symmetric matrix is one that satisfies

$$\mathbf{A}^T = \mathbf{A} \quad \text{or} \quad a_{ji} = a_{ij} \quad (30)$$

and a real skew-symmetric matrix satisfies the relationship

$$\mathbf{A}^T = -\mathbf{A} \quad \text{or} \quad a_{ji} = -a_{ij} \quad (31)$$

Other relationships will be defined as needed in subsequent subsections.

Now, as already emphasized, the major motivation for the matrix/vector notation outlined above is as a shorthand representation for linear systems of algebraic equations. In particular, the system of linear equations

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots a_{nn}x_n &= b_n \end{aligned} \quad (32)$$

can be written in matrix notation as

$$\mathbf{Ax} = \mathbf{b} \quad (33)$$

where explicit definitions of the \mathbf{A} and \mathbf{b} coefficient matrices and solution vector \mathbf{x} are given by

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \vdots & \ddots & \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix} \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad (34)$$

Although we will discuss several computational solution techniques in Lesson #6, the most direct method for solving systems of equations involves a sequence of **elementary row operations**.

These operations represent legal algebraic manipulations that do not alter the basic equality associated with the original equations. ***The purpose of the row operations is to take the original equations and put them into a form that is easier to solve than the original equations.*** There are three row operations that are used to systematically simplify the original system of equations:

1. Interchange two rows
2. Multiply a row by a constant

3. Add a constant times one row to another row

The most well-known method that implements these row operations, called the **Gauss Elimination Method**, takes the original system and converts the matrix into upper triangular form (often called **row echelon form**). In this form, back substitution is used to evaluate the unknown solution vector \mathbf{x} , since there is only one unknown per equation if evaluated sequentially starting with equation n and working backwards to the first equation. The required transformation (i.e. the **elimination step** as it is often called) can be represented symbolically using an augmented matrix notation, where $\tilde{\mathbf{A}} = [\mathbf{A} \ \mathbf{b}]$ is the augmented matrix. For example, a 3×3 system would be transformed as follows:

$$\begin{bmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{bmatrix} \Rightarrow \begin{bmatrix} * & * & * & * \\ 0 & * & * & * \\ 0 & 0 & * & * \end{bmatrix}$$

where the $*$ notation implies a general nonzero entry and the last column in the original matrix contains the right hand side \mathbf{b} vector. Of course, after transformation, the entries in the resultant matrix are different from the original case. However, this new system, with the $n \times n$ part of the augmented matrix in upper triangular form, is an equivalent representation of the original equation. Once in this form, one can easily use back substitution to solve for the unknown \mathbf{x} vector.

As a simple example of this method, consider the following 3×3 system:

$$\begin{matrix} \mathbf{A} & \mathbf{x} & = & \mathbf{b} \\ \begin{bmatrix} 3 & -2 & 0 \\ -1 & 3 & -2 \\ 0 & -1 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} & = & \begin{bmatrix} -7 \\ 9 \\ -5 \end{bmatrix} \end{matrix} \quad (35)$$

Written in augmented matrix form, this becomes

$$\begin{bmatrix} 3 & -2 & 0 & -7 \\ -1 & 3 & -2 & 9 \\ 0 & -1 & 3 & -5 \end{bmatrix}$$

Now, as our first row operation, take $1/3$ times row 1 added to row 2 to give

$$\begin{bmatrix} 3 & -2 & 0 & -7 \\ 0 & 7/3 & -2 & 20/3 \\ 0 & -1 & 3 & -5 \end{bmatrix}$$

Note that only row 2 is modified in this step and that the first element of the row is now zero. In fact, the whole 1st column below the 1,1 element (the **pivot element**) is zero, and we are well on our way to achieving an upper triangular matrix (all zeros below the main diagonal).

Now we choose the 2,2 element as the pivot element. In particular, we multiply row 2 by $3/7$ and add the result to row 3 to give

$$\begin{bmatrix} 3 & -2 & 0 & -7 \\ 0 & 7/3 & -2 & 20/3 \\ 0 & 0 & 15/7 & -15/7 \end{bmatrix}$$

This system is now in row echelon form and can be written as a standard matrix equation,

$$\begin{bmatrix} 3 & -2 & 0 \\ 0 & 7/3 & -2 \\ 0 & 0 & 15/7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -7 \\ 20/3 \\ -15/7 \end{bmatrix} \quad (36)$$

where it is important to emphasize that, since only legal row operations were performed, the system in eqn. (36) is equivalent to the original 3x3 system given by eqn. (35).

Now, since eqn. (36) is written in row echelon form, using back substitution (i.e. evaluating the equations in reverse order) gives

$$\begin{aligned} x_3 &= \frac{-15/7}{15/7} = -1 \\ x_2 &= \frac{3[20/3 + 2x_3]}{7} = \frac{20 - 6}{7} = 2 \\ x_1 &= \frac{[-7 + 2x_2]}{3} = \frac{-7 + 4}{3} = -1 \end{aligned}$$

A quick check shows that $\mathbf{x} = [-1 \ 2 \ -1]^T$ is indeed the correct solution to the original system of equations -- that is, substitution into eqn. (35) gives

$$\begin{aligned} 3(-1) - 2(2) + 0(-1) &= -7 \quad (\text{checks ok}) \\ -1(-1) + 3(2) - 2(-1) &= 9 \quad (\text{checks ok}) \\ 0(-1) - 1(2) + 3(-1) &= -5 \quad (\text{checks ok}) \end{aligned}$$

Thus, the use of the row operations noted above to transform the original system into a simpler form turns out to be a nice way to solve linear systems. However, the hand manipulations performed in the above example become very tedious when there are more than 4 or 5 equations to solve, so we clearly need to formalize this procedure for computer implementation for solution of larger systems. In fact, the so-called **Gauss Elimination Method** is just a formal implementation of the procedure used above and we will discuss this computational approach in further detail in Lesson #6 -- here we show the basic concepts via hand calculations for small systems and in Lesson #6 we will develop a formal algorithm for actual computer implementation.

Another hand manipulation technique for solving small systems of equations involves the formal definition of the so-called **matrix inverse**. The matrix inverse, denoted as \mathbf{A}^{-1} , is a quantity used in the formal manipulation and solution of systems of equations. \mathbf{A}^{-1} is defined such that

$$\mathbf{A}^{-1}\mathbf{A} = \mathbf{A}\mathbf{A}^{-1} = \mathbf{I} \quad (37)$$

This says that a square matrix multiplied by its inverse gives the identity matrix. Also, one should note that the identity matrix operating on a matrix or vector of appropriate size does not

alter the original quantity. These facts can be used to write the formal solution to a system of equations. In particular, given $\mathbf{Ax} = \mathbf{b}$, a formal solution for \mathbf{x} can be developed as follows:

Starting with $\mathbf{Ax} = \mathbf{b}$, *pre-multiply* both sides by \mathbf{A}^{-1} to give

$$\mathbf{A}^{-1}\mathbf{Ax} = \mathbf{A}^{-1}\mathbf{b}$$

but $\mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$, and $\mathbf{Ix} = \mathbf{x}$, therefore we have

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad (38)$$

Note: Recall that the order of matrix operations is important -- that is why we emphasize the word “pre-multiply” in the above development!

This formal solution is very important, since it provides a basis for discussing the uniqueness and existence of solutions and it also allows for various manipulations of matrix equations.

However, the reader should be cautioned that this formulation is not the most efficient procedure for actually computing the solution vector \mathbf{x} . For computer solution of $\mathbf{Ax} = \mathbf{b}$, especially for large systems, other techniques are far more efficient (since the computation of \mathbf{A}^{-1} is more computationally intensive than solving $\mathbf{Ax} = \mathbf{b}$ by other means -- such as by Gauss Elimination).

There are many cases, however, when it is useful to actually evaluate the inverse matrix. There are a variety of ways to do this. In particular, for low-order systems, the following formula is often applied,

$$\mathbf{A}^{-1} = \frac{\text{adj } \mathbf{A}}{\det \mathbf{A}} = \frac{\mathbf{C}^T}{\det \mathbf{A}} \quad (39)$$

where $\text{adj } \mathbf{A}$ is the *adjoint of A*, \mathbf{C} is the matrix whose elements are the *cofactors of A*, and $\det \mathbf{A}$ is the *determinant of A* (note that $\text{adj } \mathbf{A} = \mathbf{C}^T$).

The determinant of a matrix, denoted as $\det \mathbf{A}$ or $|\mathbf{A}|$, appears frequently in applications of matrix equations. It is sometimes thought of as a *measure of the size or magnitude* of a matrix.

Independent of its formal interpretation, it does appear in many formal definitions of other quantities and we must be able to compute $\det \mathbf{A}$ in lots of situations. For hand manipulation of low order systems, *Laplace's expansion* for $\det \mathbf{A}$ is probably the best way to evaluate this quantity (computer computation is done more efficiently using row operations).

Laplace's expansion can be written in terms of an expansion along any row i as

$$\det \mathbf{A} = \sum_j a_{ij}c_{ij} \quad \text{for any } i \quad (40)$$

or down any column j as

$$\det \mathbf{A} = \sum_i a_{ij}c_{ij} \quad \text{for any } j \quad (41)$$

where c_{ij} is the *cofactor* of element a_{ij} . The elements of the cofactor matrix are defined as

$$c_{ij} = (-1)^{i+j} M_{ij} \quad (42)$$

where M_{ij} is referred to as the *minor* of the a_{ij} element. M_{ij} is defined as the determinant of the matrix formed by deleting the i^{th} row and the j^{th} column from the original matrix.

Well, since I expect that all this is quite confusing (because of all the new terminology), let's try to clarify things somewhat with an example. In particular, given a general 3×3 matrix

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

we can evaluate Laplace's expansion by expanding down column 1 [use eqn. (41) with $j = 1$, for example], giving

$$\det \mathbf{A} = a_{11}c_{11} + a_{21}c_{21} + a_{31}c_{31}$$

where

$$c_{11} = (+1)M_{11} = \begin{vmatrix} a_{22} & a_{23} \\ a_{32} & a_{33} \end{vmatrix} = a_{22}a_{33} - a_{23}a_{32}$$

$$c_{21} = (-1)M_{21} = -\begin{vmatrix} a_{12} & a_{13} \\ a_{32} & a_{33} \end{vmatrix} = -a_{12}a_{33} + a_{13}a_{32}$$

$$c_{31} = (+1)M_{31} = \begin{vmatrix} a_{12} & a_{13} \\ a_{22} & a_{23} \end{vmatrix} = a_{12}a_{23} - a_{13}a_{22}$$

Note that this is exactly the same result as if one expands along row 1 (or any other row or column).

A Note about Determinants: The determinant of a matrix may or may not be altered under certain variations to the original matrix. Since determinants are often computed using row operations, several important relations are noted as follows:

1. The $\det \mathbf{A}$ is not altered if the rows are written as columns in the same order. Therefore,
2. If any two rows or columns are interchanged, the value of $\det \mathbf{A}$ is multiplied by -1.
3. The value of $\det \mathbf{A}$ is not altered if the elements of one row are altered by adding any constant multiple of another row to them.
4. The determinant is multiplied by a constant α if any row is multiplied by α .
5. The determinant of a diagonal matrix is simply the product of the diagonal elements. This is also true for triangular matrices. This observation, along with the above statements, establishes a method for computer calculation of the $\det \mathbf{A}$ using row operations by transforming the original matrix into upper triangular form and then taking the product of the diagonal elements -- being careful to account for row interchanges and normalization steps
6. For square matrices,

$$\det (\mathbf{AB}) = \det (\mathbf{BA}) = \det \mathbf{A} \det \mathbf{B} \quad (44)$$

Although eqn. (39) is usually used for finding \mathbf{A}^{-1} for small systems, for larger systems and for automated implementation on the computer, we can also apply elementary row operations to transform the original augmented matrix as follows:

$$\begin{bmatrix} * & * & * & 1 & 0 & 0 \\ * & * & * & 0 & 1 & 0 \\ * & * & * & 0 & 0 & 1 \end{bmatrix} \Rightarrow \begin{bmatrix} 1 & 0 & 0 & * & * & * \\ 0 & 1 & 0 & * & * & * \\ 0 & 0 & 1 & * & * & * \end{bmatrix} \quad (45)$$

With this symbolic notation, we see that the original matrix is augmented with the identity matrix. Extending the notation from before, this says we are trying to evaluate a matrix equation of the form, $\mathbf{AX} = \mathbf{I}$, for the unknown matrix \mathbf{X} . Therefore, we know, from the definition of the inverse matrix, that $\mathbf{X} = \mathbf{A}^{-1}$. We can solve for \mathbf{X} by performing row operations on the augmented matrix $[\mathbf{A} \ \mathbf{I}]$, finally putting it into the form $[\mathbf{I} \ \mathbf{X}]$. This basic technique is often referred to as the **Gauss-Jordan Method**.

Let's illustrate these two methods for finding the inverse matrix [i.e. via eqn. (39) and by the use of row operations] using the following 3x3 matrix:

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 0 \\ -1 & 3 & -2 \\ 0 & -1 & 3 \end{bmatrix}$$

Method I [using eqn. (39)]:

Using eqn. (40) let's first find $\det \mathbf{A}$ by expanding along row 1 (i.e. $i = 1$), or

$$\det \mathbf{A} = 3 \begin{vmatrix} 3 & -2 \\ -1 & 3 \end{vmatrix} - (-2) \begin{vmatrix} -1 & -2 \\ 0 & 3 \end{vmatrix} + 0 \begin{vmatrix} -1 & 3 \\ 0 & -1 \end{vmatrix} = 3(7) + 2(-3) + 0(1) = 15$$

Also, using eqn. (42), the elements of the cofactor matrix are

$$\mathbf{C} = \begin{bmatrix} +7 & -(-3) & +1 \\ -(-6) & +9 & -(-3) \\ +4 & -(-6) & +7 \end{bmatrix}$$

Therefore,

$$\mathbf{A}^{-1} = \frac{\mathbf{C}^T}{\det \mathbf{A}} = \frac{1}{15} \begin{bmatrix} 7 & 6 & 4 \\ 3 & 9 & 6 \\ 1 & 3 & 7 \end{bmatrix} \quad (46)$$

and a quick check on $\mathbf{A}^{-1} \mathbf{A} \stackrel{?}{=} \mathbf{I}$ gives

$$\frac{1}{15} \begin{bmatrix} 7 & 6 & 4 \\ 3 & 9 & 6 \\ 1 & 3 & 7 \end{bmatrix} \begin{bmatrix} 3 & -2 & 0 \\ -1 & 3 & -2 \\ 0 & -1 & 3 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} 15 & 0 & 0 \\ 0 & 15 & 0 \\ 0 & 0 & 15 \end{bmatrix} = \mathbf{I} \quad (\text{checks ok})$$

which shows that all the manipulations have been done correctly!

Method II (using row operations):

For the Gauss-Jordan method, we start with the original matrix augmented with the 3×3 identity matrix, or

$$\begin{bmatrix} 3 & -2 & 0 & 1 & 0 & 0 \\ -1 & 3 & -2 & 0 & 1 & 0 \\ 0 & -1 & 3 & 0 & 0 & 1 \end{bmatrix}$$

Now let's systematically perform a set of row operations to transform this matrix into the desired form. To start, let's normalize row 1 so that the lead coefficient is unity, giving

$$\begin{bmatrix} 1 & -2/3 & 0 & 1/3 & 0 & 0 \\ -1 & 3 & -2 & 0 & 1 & 0 \\ 0 & -1 & 3 & 0 & 0 & 1 \end{bmatrix}$$

Now add row 1 to row 2 to give

$$\begin{bmatrix} 1 & -2/3 & 0 & 1/3 & 0 & 0 \\ 0 & 7/3 & -2 & 1/3 & 1 & 0 \\ 0 & -1 & 3 & 0 & 0 & 1 \end{bmatrix}$$

Normalizing row 2 so that the pivot element (2,2 element) is unity gives

$$\begin{bmatrix} 1 & -2/3 & 0 & 1/3 & 0 & 0 \\ 0 & 1 & -6/7 & 1/7 & 3/7 & 0 \\ 0 & -1 & 3 & 0 & 0 & 1 \end{bmatrix}$$

Now add row 2 to row 3 to give

$$\begin{bmatrix} 1 & -2/3 & 0 & 1/3 & 0 & 0 \\ 0 & 1 & -6/7 & 1/7 & 3/7 & 0 \\ 0 & 0 & 15/7 & 1/7 & 3/7 & 1 \end{bmatrix}$$

Normalizing row 3, we have

$$\begin{bmatrix} 1 & -2/3 & 0 & 1/3 & 0 & 0 \\ 0 & 1 & -6/7 & 1/7 & 3/7 & 0 \\ 0 & 0 & 1 & 1/15 & 3/15 & 7/15 \end{bmatrix}$$

Continuing to perform row operations to eliminate the upper triangular terms, we add 2/3 times row 2 to row 1 to give

$$\begin{bmatrix} 1 & 0 & -4/7 & 3/7 & 2/7 & 0 \\ 0 & 1 & -6/7 & 1/7 & 3/7 & 0 \\ 0 & 0 & 1 & 1/15 & 3/15 & 7/15 \end{bmatrix}$$

Now add 4/7 times row 3 to row 1 to give

$$\begin{bmatrix} 1 & 0 & 0 & 7/15 & 6/15 & 4/15 \\ 0 & 1 & -6/7 & 1/7 & 3/7 & 0 \\ 0 & 0 & 1 & 1/15 & 3/15 & 7/15 \end{bmatrix}$$

Finally, 6/7 times row 3 added to row 2 gives

$$\begin{bmatrix} 1 & 0 & 0 & 7/15 & 6/15 & 4/15 \\ 0 & 1 & 0 & 3/15 & 9/15 & 6/15 \\ 0 & 0 & 1 & 1/15 & 3/15 & 7/15 \end{bmatrix}$$

Therefore,

$$\mathbf{A}^{-1} = \frac{1}{15} \begin{bmatrix} 7 & 6 & 4 \\ 3 & 9 & 6 \\ 1 & 3 & 7 \end{bmatrix}$$

which is the same result obtained from Method I as given in eqn. (46) -- this, of course, is as expected!

Note that the \mathbf{A} matrix used in this example is the same matrix from our previous example for solving $\mathbf{Ax} = \mathbf{b}$ [see eqn. (35)]. Thus, now that we have \mathbf{A}^{-1} , we should be able to compute the solution vector \mathbf{x} via $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$. Doing this as a check gives

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} = \frac{1}{15} \begin{bmatrix} 7 & 6 & 4 \\ 3 & 9 & 6 \\ 1 & 3 & 7 \end{bmatrix} \begin{bmatrix} -7 \\ 9 \\ -5 \end{bmatrix} = \frac{1}{15} \begin{bmatrix} -15 \\ 30 \\ -15 \end{bmatrix} = \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} \quad (47)$$

which agrees perfectly with our previous result (as expected)!

A couple of convenient relationships involving the inverse matrix should also be noted, as follows:

1. If \mathbf{A} is a diagonal matrix (a square matrix with all zeros in the off-diagonal locations), then

$$\mathbf{A}^{-1} = \begin{bmatrix} a_{ii}^{-1} \end{bmatrix} \quad (48)$$

2. The inverse of a product of square matrices is simply the product of the individual inverses in the opposite order, or

$$(\mathbf{AB})^{-1} = \mathbf{B}^{-1}\mathbf{A}^{-1} \quad (49)$$

To show this last relationship (as an example of manipulating matrix equations), we have

$$\begin{aligned} (\mathbf{AB})^{-1} &= \mathbf{C} \\ (\mathbf{AB})(\mathbf{AB})^{-1} &= \mathbf{ABC} = \mathbf{I} \end{aligned}$$

Thus $\mathbf{BC} = \mathbf{A}^{-1}$, or $\mathbf{C} = \mathbf{B}^{-1}\mathbf{A}^{-1}$, which proves the above statement.

Now, with a better understanding of the matrix inverse, determinants, cofactors, etc., we can talk about the existence and uniqueness of the solutions of equations of the form $\mathbf{Ax} = \mathbf{b}$. Within this context, a new term called the **rank** of a matrix is often used. Formally, the **rank** of a matrix is defined as the maximum number of linearly independent rows or columns in the matrix. It is important to note that elementary row operations do not alter the rank of a matrix. Also it should be noted that \mathbf{A} and \mathbf{A}^T have the same rank.

Now, for the usual case of n simultaneous equations with n unknowns, we have

$$\mathbf{Ax} = \mathbf{b} \quad \text{and} \quad \mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \quad \text{where} \quad \mathbf{A}^{-1} = \frac{\mathbf{C}^T}{\det \mathbf{A}}$$

When discussing the existence and uniqueness of solutions, two situations can occur:

I. Non-Homogeneous Problems ($\mathbf{b} \neq \mathbf{0}$):

In this case, $\mathbf{b} \neq \mathbf{0}$, and this system is said to be a non-homogeneous system. For this situation, a single non-trivial solution exists if we have n linearly independent rows, which implies that $\text{rank } \mathbf{A} = n$, that the $\det \mathbf{A}$ is nonzero, and that \mathbf{A}^{-1} exists. When \mathbf{A}^{-1} exists, \mathbf{A} is said to be non-singular, and the product of \mathbf{A}^{-1} and \mathbf{b} gives the unique solution vector \mathbf{x} .

II. Homogeneous Problems ($\mathbf{b} = \mathbf{0}$):

If $\mathbf{b} = \mathbf{0}$, then $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ implies, at first glance, that \mathbf{x} must be the null vector since we are multiplying the inverse matrix and the $\mathbf{b} = \mathbf{0}$ vector. However, if $\det \mathbf{A} = 0$, then \mathbf{A}^{-1} does not exist, and the solution form $\mathbf{x} = \mathbf{A}^{-1}\mathbf{b}$ leads to an indeterminate form, which could lead to a nontrivial solution. In fact, this is indeed the situation, and we can argue that there are nontrivial solutions only if \mathbf{A}^{-1} does not exist. In this case we say that \mathbf{A} is a singular matrix. This happens only if $\det \mathbf{A} = 0$ which implies that at least two of the rows of \mathbf{A} are linearly dependent and that $\text{rank } \mathbf{A} < n$.

The ramifications of the above statements are extremely important when solving linear systems. They say that if $\det \mathbf{A} \neq 0$, then we should be able to find a unique solution to the non-homogeneous problem. However, if $\det \mathbf{A} = 0$, no unique solution exists for the case where $\mathbf{b} \neq \mathbf{0}$ (i.e. the non-homogeneous problem) -- there are either no solutions or an infinite number of solutions. On the other hand, for the homogeneous case (i.e. the $\mathbf{b} = \mathbf{0}$ case), the matrix must be singular for a nontrivial solution and, of course, since the problem has a zero right hand side there will be an infinite number of solutions only differing by an arbitrary normalization (if \mathbf{x} is a solution to a homogeneous equation, then $\alpha\mathbf{x}$ is also a solution, where α is an arbitrary nonzero constant).

To help your visualization of these comments, let's look at a series of four simple examples:

Ex. #1 -- Non-Homogeneous Case with $\text{rank } \mathbf{A} = n$

Given the two equations,

$$3x_1 + 2x_2 = 1 \quad \text{and} \quad x_1 + x_2 = -1$$

the matrix representation is given by

$$\begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

For this non-homogeneous case, $\det \mathbf{A} = 3-2 = 1$; thus, we expect that a single solution exists. Solution of this system via algebraic manipulation gives:

From the 2nd equation: $x_1 = -1 - x_2$

Putting this into the 1st equation: $3(-1 - x_2) + 2x_2 = 1 \quad \text{or} \quad x_2 = -4 \quad \& \quad x_1 = 3$

Thus, we indeed get a unique solution, $\mathbf{x} = [3 \ -4]^T$.

Ex. #2 -- Non-Homogeneous Case with rank $\mathbf{A} < n$

Given the two equations,

$$3x_1 + 2x_2 = 1 \quad \text{and} \quad x_1 + \frac{2}{3}x_2 = -1$$

the matrix representation is given by

$$\begin{bmatrix} 3 & 2 \\ 1 & 2/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \end{bmatrix}$$

For this non-homogeneous case, $\det \mathbf{A} = 2-2 = 0$; thus, the matrix is singular and, from the above discussion, we do not expect that a unique solution exists for this problem.

To see this, notice what happens if the 2nd equation is multiplied by 3. Doing this gives

$$\begin{bmatrix} 3 & 2 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$$

and, clearly, the two equations are inconsistent and no solution can be found.

Notice, however, that if the RHS of the 2nd equation is changed to 1/3 (instead of -1), then the resultant equations are

$$3x_1 + 2x_2 = 1 \quad \text{and} \quad x_1 + \frac{2}{3}x_2 = \frac{1}{3}$$

and the matrix representation is given by

$$\begin{bmatrix} 3 & 2 \\ 1 & 2/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1/3 \end{bmatrix} \quad \text{or} \quad \begin{bmatrix} 3 & 2 \\ 3 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$$

Clearly, in this case, the equations are redundant and there are an infinite number of solutions.

Thus, we see that for the non-homogeneous case with rank $\mathbf{A} < n$, we either get no solution or an infinite number of solutions. The bottom line here is that there are no unique solutions for this class of problems.

Ex. #3 -- Homogeneous Case with rank $\mathbf{A} < n$

Given the two equations,

$$3x_1 + 2x_2 = 0 \quad \text{and} \quad x_1 + \frac{2}{3}x_2 = 0$$

the matrix representation is given by

$$\begin{bmatrix} 3 & 2 \\ 1 & 2/3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This is the same matrix as for Ex. #2, so $\det \mathbf{A} = 0$. From the above discussion, a homogeneous system with a singular matrix should lead to an infinite number of non-trivial solutions (notice that $\mathbf{x} = \mathbf{0}$ always satisfies a homogeneous equation, but this is what we refer to as the *trivial solution*). To find the non-trivial solutions, note that **both** equations lead to

$$x_1 = -\frac{2}{3}x_2$$

and, from here, we see that we can write an infinite number of solutions as long as x_1 and x_2 have this required relationship:

$$\mathbf{x} = \begin{bmatrix} -2/3 \\ 1 \end{bmatrix} \quad \text{or} \quad \mathbf{x} = \begin{bmatrix} -4/3 \\ 2 \end{bmatrix} \quad \text{or} \quad \mathbf{x} = \begin{bmatrix} -5/3 \\ 2.5 \end{bmatrix} \quad \text{etc.}$$

Ex. #4 -- Homogeneous Case with rank $\mathbf{A} = n$

Given the two equations,

$$3x_1 + 2x_2 = 0 \quad \text{and} \quad x_1 + x_2 = 0$$

the matrix representation is given by

$$\begin{bmatrix} 3 & 2 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

This is the same matrix as for Ex. #1, so $\det \mathbf{A} = 1$. From the above discussion, a homogeneous system with a non-singular matrix should only have a trivial solution (i.e. $\mathbf{x} = \mathbf{0}$). Clearly this is the case here, since the two equations are inconsistent -- for $3x_1 + 2x_2 = x_1 + x_2 = 0$ to be valid, both x_1 and x_2 must be zero!!!

Really understanding these simple examples and all the linear algebra terminology that goes along with them (rank of a matrix, linear independence of equations, singular and non-singular matrices, homogenous and non-homogeneous systems, etc.) will be useful in our continuing discussions of solution techniques and in many other areas of mathematics. Thus, make sure you have a good handle on the material in this subsection before continuing...

A Brief Introduction to Eigenvalues and Eigenvectors: Although we will not do much in this course with the so-called *Classical Eigenvalue Problem* (there is simply not enough time), it is important for you to at least be familiar with the concepts and terminology involved here, since this is such an important subject in many areas of practical application. In addition, the introduction of this subject at this point is perfect, since it builds directly upon the uniqueness and existence concepts that were just discussed. Thus, I could not resist at least giving a brief glimpse into this subject.

From the above discussion on systems of linear algebraic equations, we saw that homogeneous equations with n equations and n unknowns require that the system matrix be singular for the existence of nontrivial solutions. The *classical eigenvalue problem* is a special case that falls

into this class of problems and it arises from the general problem given by $\mathbf{Ax} = \mathbf{b}$ when $\mathbf{b} = \lambda\mathbf{x}$ (that is, the right hand side vector is some constant times the solution vector \mathbf{x}). With this substitution, we have

$$\mathbf{Ax} = \lambda\mathbf{x} \quad (50)$$

These systems occur frequently in applications and are usually written as

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{x} = \mathbf{0} \quad (51)$$

which is a homogeneous system of equations (note that the identity matrix is needed here so that both terms inside the parentheses are matrices of the same size). Therefore, for non-trivial solutions, we require that

$$\det(\mathbf{A} - \lambda\mathbf{I}) = 0 \quad (52)$$

which is referred to as the *characteristic equation*. This gives rise to an n^{th} order polynomial in λ which has n roots -- the n eigenvalues of a square matrix of order n .

Note that the eigenvalues may be real and distinct, complex conjugates, repeated, or some combination of these forms. Note also that the sequence $\lambda_1, \lambda_2, \dots, \lambda_n$ is called the *eigenvalue spectrum*, with the magnitude of the largest eigenvalue denoted as the *spectral radius*, or

$$|\lambda_{\max}| = \text{spectral radius}$$

The eigenvector \mathbf{x}_i associated with the i^{th} eigenvalue, λ_i , is found by evaluating the homogeneous equation

$$(\mathbf{A} - \lambda_i\mathbf{I})\mathbf{x}_i = \mathbf{0} \quad (53)$$

where the notation, \mathbf{x}_i , refers to the i^{th} eigenvector, not the i^{th} element of a given vector.

As a specific example, let's find the eigenvalues and eigenvectors of the same matrix that we have used in previous examples:

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 0 \\ -1 & 3 & -2 \\ 0 & -1 & 3 \end{bmatrix}$$

The characteristic equation is given by

$$|\mathbf{A} - \lambda\mathbf{I}| = \begin{vmatrix} 3-\lambda & -2 & 0 \\ -1 & 3-\lambda & -2 \\ 0 & -1 & 3-\lambda \end{vmatrix} = 0$$

Expanding the determinant along row 1 using Laplace's expansion gives

$$\begin{aligned}
|\mathbf{A} - \lambda \mathbf{I}| &= (3 - \lambda) \begin{vmatrix} 3 - \lambda & -2 \\ -1 & 3 - \lambda \end{vmatrix} - (-2) \begin{vmatrix} -1 & -2 \\ 0 & 3 - \lambda \end{vmatrix} \\
&= (3 - \lambda) \left((3 - \lambda)^2 - 2 \right) - 2(3 - \lambda) \\
&= (3 - \lambda) \left((3 - \lambda)^2 - 4 \right) \\
&= (3 - \lambda) (\lambda^2 - 6\lambda + 5) \\
&= (3 - \lambda)(\lambda - 5)(\lambda - 1)
\end{aligned}$$

Thus, we can satisfy the requirement that $|\mathbf{A} - \lambda \mathbf{I}|$ must vanish with

$$\lambda_1 = 1, \quad \lambda_2 = 3, \quad \text{and} \quad \lambda_3 = 5$$

These are the eigenvalues of the given matrix [i.e. the roots of the characteristic equation $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$].

Now, the eigenvector associated with the i^{th} eigenvalue can be determined by solving the matrix equations with the specific eigenvalue inserted into the equation. For example, for $\lambda_1 = 1$, we have $(\mathbf{A} - \lambda_1 \mathbf{I})\mathbf{x}_1 = \mathbf{0}$, or

$$\begin{bmatrix} 2 & -2 & 0 \\ -1 & 2 & -2 \\ 0 & -1 & 2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

which gives three equations

$$\begin{aligned}
2x_1 - 2x_2 &= 0 & \Rightarrow & x_1 = x_2 \\
-x_1 + 2x_2 - 2x_3 &= 0 & \Rightarrow & x_2 = 2x_3 \\
-x_2 + 2x_3 &= 0 & \Rightarrow & x_2 = 2x_3
\end{aligned}$$

where we note that the 2nd and 3rd equations are redundant. This, of course, was expected since we forced the matrix $\mathbf{A} - \lambda \mathbf{I}$ to be singular (has linearly dependent rows).

Now, from these expressions, we see that, with $x_3 = 1$, we have $\mathbf{x}_1 = [2 \ 2 \ 1]^T$ as a valid eigenvector associated with $\lambda_1 = 1$ (normalization is arbitrary here).

Performing the same type of operations with $\lambda_2 = 3$ gives

$$\begin{bmatrix} 0 & -2 & 0 \\ -1 & 0 & -2 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}_2 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

and these equations clearly say that

$$x_2 = 0 \quad \text{and} \quad x_1 = -2x_3$$

Thus, a proper eigenvector for this case, with $x_3 = 1$, is $\mathbf{x}_2 = [-2 \ 0 \ 1]^T$.

Finally, for $\lambda_3 = 5$, doing the same type of manipulations gives $\mathbf{x}_3 = [2 \ -2 \ 1]^T$. You should derive this result yourself as a test of your understanding of these calculations...

In concluding this subsection, you should be aware that the capability to do computations of the type done by hand within the last several pages of these notes (solve $\mathbf{Ax} = \mathbf{b}$, find $\det \mathbf{A}$, compute \mathbf{A}^{-1} , determine the eigenvalues and eigenvectors of a matrix, etc.) is built directly into Matlab and other similar programs and, in practice, automated routines like those in Matlab are used in day-to-day engineering applications as needed (for example, see Table 1 for a listing of the **linademo_lesson2.m** file for the Matlab commands needed to do some of these calculations). However, the student should definitely know the fundamentals behind these numerical algorithms (although the details are not always necessary). By assuring that you can do the above manipulations by hand for low-order systems, you will gain the confidence and experience necessary to intelligently and efficiently use the automated software. Thus, you should make sure you understand the above examples, and be able to perform similar manipulations on small systems as verification of the computer tools that simply automate the procedures.

However, the Matlab commands shown in Table 1 (*det*, *inv*, *eig*, and the backslash operator, \backslash) certainly make doing the required calculations much easier -- and I think you will come to really appreciate the powerful capability that is built into these simple commands. It only took a few minutes to generate the results within Matlab, whereas the hand computations, although straightforward, were much more time consuming -- yet we get the same results as shown in Table 2 (note that Matlab normalizes the eigenvectors so that they have a magnitude of unity, but Matlab's eigenvectors have the same relative internal distribution as our hand calculations that normalized the vectors such that $x_3 = 1$). For larger systems, the hand calculations become impractical and the computer becomes indispensable for working with matrix systems in real applications.

Table 1 Listing of the linademo_lesson2.m file.

```
%
% LINADEMO_LESSON2.M  MATLAB matrix tasks as part a demo in Lesson 2 Lecture Notes
%
% This file just does some simple linear algebra manipulations within Matlab.
% The goal here is to compare with some hand calculations that were done in the
% Lecture Notes to confirm our general understanding.
%
% File prepared by J. R. White, UMass-Lowell (last update:  September 2017)
%

    clear all,    close all
    format compact

%
% define matrices for demo
    disp('Matrices for the demo'),    A = [3 -2 0;-1 3 -2;0 -1 3],    b = [-7 9 -5]'
%
% reproduce some of the hand calculations from the notes using Matlab
    disp('Find det A');    det(A)
    disp('Find inverse of A');    AI = inv(A)
    disp('Solve A*x = b using the backslash operator');    x = A\b
    disp('Solve A*x = b using the inverse operator');    x = AI*b
    disp('Find eigenvalues & eigenvector of A');    [evec,eval] = eig(A)
%
% end of program
```

Table 2 Results from the linademo_lesson2 program.

```

>> linademo_lesson2
Matrices for the demo
A =
     3     -2     0
    -1     3    -2
     0     -1     3
b =
    -7
     9
    -5
Find det A
ans =
    15
Find inverse of A
AI =
    0.4667    0.4000    0.2667
    0.2000    0.6000    0.4000
    0.0667    0.2000    0.4667
Solve A*x = b using the backslash operator
x =
    -1
     2
    -1
Solve A*x = b using the inverse operator
x =
   -1.0000
    2.0000
   -1.0000
Find eigenvalues & eigenvector of A
evec =
   -0.6667   -0.8944    0.6667
    0.6667    0.0000    0.6667
   -0.3333    0.4472    0.3333
eval =
    5.0000         0         0
         0    3.0000         0
         0         0    1.0000

```

We will comment further on the inner workings of the backslash operator that was used to solve the $\mathbf{Ax} = \mathbf{b}$ equation later in the semester (see Lesson #6 Notes). However, at this point, the simple demo given in Tables 1 and 2 are all you need to start being productive with Matlab for a variety of linear algebra applications. Matlab has many more functions for working with systems of equations, and you are certainly encouraged to explore further. However, for now, you already have enough background to solve a variety of real-world applications. Having this knowledge is sufficient to meet our current goals for Lesson #2 but, later in Lesson #6, we will provide more insight into how the computer is used to solve a large system of equations -- and eventually we will get to a point where we can describe some further details of the operations actually performed by the \ operator.

Some Additional Illustrative Applications and Individual Practice

Now, with all this background, it is time to see some illustrative examples where Matlab's capabilities are put to good use to help us visualize and understand some systems and processes of interest. These examples, which are available in separate pdf files, further illustrate the use of 2-D arrays within Matlab (see Lesson #6 for applications involving systems of equations):

sand_pits.pdf -- Sand Pit Utilization

maxwell_2.pdf -- Maxwellian Distribution Revisited

projectile2d_1.pdf -- 2_D Projectile Motion: A Bottle Cap Tossing Simulation

Upon completion of your reading assignment for this lesson and after studying the above examples, you should now be ready to do HW #2 (see **hw2xxx.pdf**). The homework typically involves 3 or 4 problems (sometimes separated into two homeworks, HW2a and HW2b). The first part of the HW deals with matrix manipulation (matrix multiplication, matrix transpose, array indexing, the matrix inverse, row operations, eigenvalues and eigenvectors, etc.), including some hand calculations to make sure you know exactly what Matlab is doing when you ask it to do various matrix operations. The last few problems then usually involve 2-D function evaluation and visualization using various 2-D and/or 3-D plotting techniques in Matlab. The above three illustrative examples and the discussions in this set of Lecture Notes should be useful in helping you accomplish these tasks -- so be sure to spend some quality time with the notes and examples **before** you attempt the HW assignment...

As before, I would prefer that you collect the Matlab m-files, the resultant plots, any hand calculations, and a brief description of the results of each problem in a separate solution package for each problem. Thus, for HW #2, you should prepare several separate solution packs, as needed, and put these together in a professional manner for submittal to me by the HW deadline.

Well, this completes Lesson #2. Working with arrays and matrices for data manipulation and for simple information storage is an essential part of almost every Matlab program. In addition, understanding the basic notation and some analytical manipulations from introductory Linear Algebra will assist you in many different application areas. I hope that you are now more comfortable with these concepts, and that your overall skills and confidence in using Matlab as a problem-solving tool has been significantly enhanced by your reading assignment, the examples from these notes and, of course, your time and effort on HW #2. We still have lots more to do -- but you already have acquired sufficient skills to start using Matlab as an efficient problem-solving tool in a variety of situations. I suggest that you use these new skills, as appropriate, to assist you in many of your HW assignments and projects for your other classes. I think you will find them to be quite useful in many situations!!! Good luck and happy Matlabing...