

Math 7 Elementary Linear Algebra  
INTRODUCTION TO MATLAB  
4. VECTORS AND MATRICES

Commands learned:

- who/whos
- defining row and column vectors
- linspace
- size of a matrix or vector
- length of a vector
- transpose
- How to create a zero matrix, ones matrix, identity matrix, diagonal matrix
- How to create matrices of random numbers
- Rounding functions: floor, ceiling, arithmetic rounding, truncation
- How to change individual entries in a matrix
- Selecting a row, column, or submatrix of a matrix

This document is intended to teach you some basic MatLab functions and commands through active participation in a MatLab session. It should be read while you are seated at a computer with a MatLab session open. As you read through this document, you should enter and execute the commands given in the examples and take written notes about the results. After completing this document, work and turn in the page of MatLab 4 Exercises.

**MANAGING YOUR SESSION:** You can get a listing of your defined variables in your workspace at any time by issuing the following command at the MatLab prompt.

```
>>who
```

You can get even more information about the variables in your workspace by entering

```
whos
```

You can always get help with a particular command by typing **help** at a prompt followed by the name of the command you want help with. For example, typing

```
help whos
```

will give you information about the **whos** command.

**ROW VECTORS:** You have already learned some basic ways of defining row vectors

1. There are three basic ways to enter a row vector:

Enclose the list of vector entries in square brackets and separate the entries with commas. Spaces between entries are optional

$$x1 = [1, 2, 3, 4, 5]$$

Or, you can dispense with the commas – just be sure you separate the entries with spaces.

$$x1 = [1 2 3 4 5]$$

2. You can create a vector of equally spaced entries in several ways. One method makes use of the colon operator. The general construct is

$$\textit{start} : \textit{increment} : \textit{finish}$$

but you can vary this in several ways. For instance, if you don't supply an increment, MatLab assumes the increment is 1.

**Example:** what happens if you enter

$$x1 = 0:10$$

You can use a negative increment.

**Example:** what happens if you enter

$$x2 = 10:-2:0$$

And you don't have to use integers.

**Example:** what happens if you enter

$$x3 = \text{sqrt}(2) : \text{sqrt}(2) : 6 * \text{sqrt}(2)$$

You can use the linspace command to generate a row vector of equally spaced points. Type

help linspace

at a command prompt to read about this command.

3. You can find the size of a vector (or matrix) by entering

$$\text{size}(\textit{vectorname}) \text{ or } \text{size}(\textit{matrixname})$$

For instance, enter

$$\text{size}(x3)$$

The information returned indicates that the vector  $x3$  has 1 row and 6 columns.

In applications of vectors, we often want to know their length (magnitude). You can use

$$\text{length}(\textit{vectorname})$$

to find the length of a row or column vector. What is the length of  $x3$ ?

**COLUMN VECTORS.** You can create a column vector by entering the entries in a matrix, with a semi-colon separating the entries (the semi-colon signals the end of the row). Square brackets must enclose the list.

```
x1=[1; 2; 3; 4; 5]
```

An alternative way of defining a column vector makes use of the transpose operator. In MatLab, to create the transpose of a matrix you use a single apostrophe. For example, to construct the column vector  $x_1$ , you would enter a row vector and transpose it:

```
x1=[1 2 3 4 5]'
```

(**NOTE:** square brackets must enclose the list.)

**SPECIAL MATRICES.** MatLab has a number of built-in routines to help create and manage matrices.

1. You can use the **zeros** command to create a zero matrix of any size.

For a square matrix of order  $n$  enter: `zeros(n)`

To generate an  $m \times n$  matrix of zeros, enter: `zeros(m,n)`

Try it: use these commands to create a square zero matrix of order 5 and a  $3 \times 5$  zero matrix.

2. You can use **ones** command to create a matrix of ones (square or  $m \times n$ ) as well. The syntax is the same as for the zeros command: `ones(n)` or `ones(m,n)`.
3. Finally, you can generate an identity matrix of any size using the **eye** command: `eye(n)`.

Try it: create an identity matrix of order 4.

4. You can generate diagonal matrices using **diag**. You must specify the numbers which are to appear on the diagonal, separated by spaces (or use commas).

For instance:

```
D=diag([1 2 3 4 5])
```

You can use the increment operator with this command

```
D=diag([1 : 0.5 : 3])
```

5. You can create a matrix of randomly chosen numbers. The **rand** function will create a matrix of uniformly distributed random numbers, each between 0 and 1 (0 included). As with **zeros**, **ones**, and **eye**, the matrix can be square or  $m \times n$ .

Enter the following to see what happens:

```
A=rand(5)
B=rand(3,5)
```

You can use scalar multiplication and any of several commands that round numbers to create matrices of integers.

The next command multiplies each entry of a  $5 \times 5$  matrix of random numbers by the scalar 10, to produce a matrix where at least some entries have a non-zero entry in the ones place.

```
C=10*rand(5)
```

Now, you can produce a matrix of integers by rounding off the entries in  $C$ . There are four different types of “round-off” functions:

- a. The **floor** function rounds each entry to the greatest integer less than or equal to the given number. (Example: the floor of 2.5 is 2 because 2 is the largest integer less than or equal to 2.5; the floor of 3 is 3 because 3 is the largest integer less than or equal to 3.) Try

```
F1=floor(C)
```

Compare  $F1$  to the original matrix  $C$ .

- b. The ceiling function (**ceil**) rounds each entry to the least integer greater than or equal to the given number. (Example: the ceiling of 2.5 is 3 because 3 is the smallest integer greater than or equal to 2.5; the ceiling of 3 is 3 because 3 is the smallest integer greater than or equal to 3.) Try

```
C1=ceil(C)
```

Compare  $C1$  to the original matrix  $C$ .

Do you understand what the floor and ceiling functions do?

- c. The **round** function performs conventional arithmetical rounding. Try

```
R=round(C)
```

- d. The **fix** function truncates a number by taking its integer part (no rounding). Try it:

```
F=fix(C)
```

You can use a nested command structure to generate a random matrix with integer entries by combining the rand function with scalar multiplication by 10 (to move the decimal point) and any of the round-off functions. For instance, to create a  $5 \times 5$  matrix of integers, you could enter

```
round(10*rand(5))
```

Of course, you could use **floor**, **ceil**, or **fix** in place of **round**.

**INDEXING AND SUBMATRICES.** Entries in a matrix are located by means of a double subscript which gives the row and column containing the entry. For instance, for an  $m \times n$  matrix  $A$ , we designate the entries of  $A$  by

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}$$

The entry  $a_{23}$  is in the second row, third column. MatLab uses similar notation to identify the elements of a matrix.

To see how we can use indexing to identify and change entries in a matrix, we will begin by creating a randomly generated matrix  $A$  of integers. Type in

```
A=round(10*rand(4,5))
```

Then enter

```
A(2,3)
```

The resulting output is the value in the second row, third column of the matrix  $A$ . This indexing feature of MatLab can be used in a variety of ways.

1. You can change an entry in a matrix. **Try each of the following and look at what happens.**

$$A(2,3) = -16$$

What happened?

$$A(2,3) = A(4,1)$$

What happened?

$$A(2,3) = -2 * A(1,4)$$

What happened?

$$A(2,3) = -2 * A(1,4)$$

What happened?

2. You can use the indexing structure to pull rows and/or columns out of a matrix. This is a powerful tool which allows you to select a submatrix of a given matrix.

For instance, the entry

$$A([1, 2], [3, 4, 5])$$

creates the submatrix formed using the elements that appear in rows 1 and 2 and in columns 3, 4, and 5 of the matrix  $A$ .

The entry

$$A([1:3], [3:5])$$

produces a submatrix consisting of rows 1 through 3 and columns 3 through 5 of matrix  $A$ .

The rows and/or columns do not have to be selected sequentially. Thus,

$$A([1, 3, 4], [3, 5])$$

builds a submatrix whose entries come from the first, third and fourth rows and third and fifth columns of  $A$ .

3. Using a colon by itself in place of a subscript selects all of the rows (or all of the columns) of a matrix. Thus,

$$A(:, 3)$$

produces the third column of  $A$ , while

$$A(2, :)$$

produces the second row of  $A$ . What do you think  $A(:)$  does? Try it and see!