ECEN321 Lab 1_1: Introduction to MATLAB

---

1. Meet MATLAB

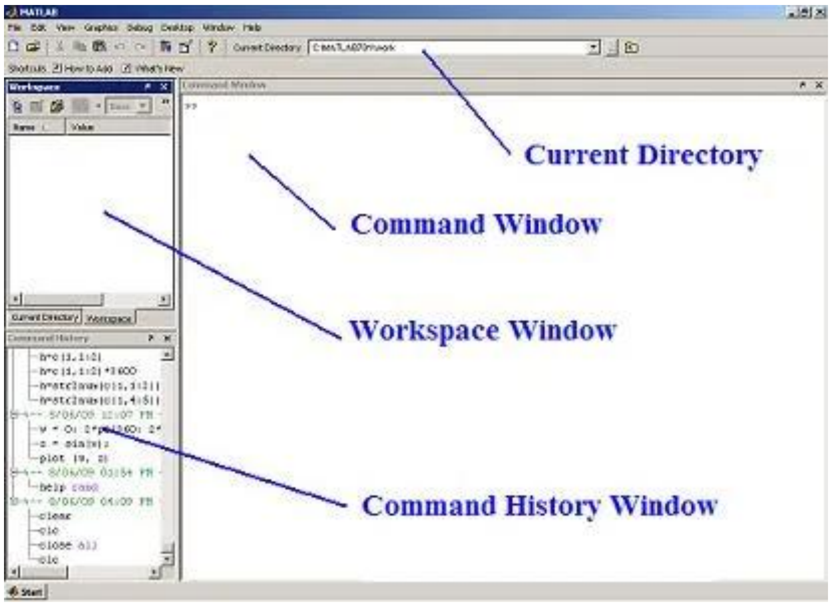
1.1 What is MATLAB?

MATLAB:

- is a high-performance, high-level language.
- stands for ***Matrix Laboratory***
- can be used as a fancy calculator
- allows you to easily work with entire matrices rather than one number at a time
- is useful for anything that requires matrix and vector manipulations such as:
  - Mathematical, scientific, engineering, statistical and financial problems
  - Anything that requires plotting/visualizing and analyzing data
- comes with a basic set of tools for visualizing data and for performing calculations on matrices and vectors

1.2 MATLAB Graphical User Interface

Matlab's graphical interface is written in Java and should be look similar on any OS. It is divided into 4 main parts:



1. **Command Window**—this is where you type commands. Output or error messages usually appear here too.
2. **Workspace window**—as you define new variables, they should be listed here.
3. **Command History window**—this is where past commands are remembered. If you want to re-run a previous command or to edit it you can drag it from this window to the command window or double click to re-run it.
4. **Current Directory window**—shows the files in the Current Directory.

---

2 Basic Operation

2.1 Creating Matrices

Everything in MATLAB is stored as a matrix or an array. To create a 1x1 array, or a **scalar**, you can write:

```
a=5
```

To see that `a` is a 1x1 array, you can execute the command:

```
whos
```

`who` lists all of the variables, and `whos` lists the variables and the size.

To create a 4 x 4 matrix, you can use the following syntax.

```
b=[1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16]
```

or

```
b=[
1  2  3  4
5  6  7  8
9 10 11 12
13 14 15 16
]
```

- opening and closing square brackets are required for matrices larger than 1x1
- entries in a row are separated by white space or commas
- semicolons (;) mark the ends of rows in the matrix
- if you end any of the above statements with a semicolon (;), you will notice that MATLAB does not echo the matrix to the display
- if you want to remind yourself of what is in `b`, you can type `b` ⏎

You can also use MATLAB functions to generate matrices. The following table summarizes a few of these functions:

function	description
`zeros(i,j)`	creates an `i x j` matrix of zeros
`ones(i,j)`	creates an `i x j` matrix of ones
`rand(i,j)`	creates an `i x j` matrix of random elements (between 0 and 1)
`randn(i,j)`	creates an `i x j` matrix of random elements drawn from a normal distribution with mean 0 and standard deviation 1
`eye(i)`	creates an `i x i` identity matrix (a matrix of zeros with ones on the diagonal)

In the above table, `i` is the number of rows and `j` is the number of columns. Be careful, this may confuse you if you are used to thinking in terms of x and y.

If you have a large amount of data, you may want to store your data in a file and load it into MATLAB. When you are typing the contents of the file, each column value is separated by horizontal white space and each row is on a new line. You require the same number of elements in each row. For instance, you can create a text file (called `myData.dat`) with the following contents:

```
     5.0    3.2    6.8
     1.4   12.6    3.9
     7.1   16.4    8.3
     9.3    5.2    4.7
```

To read in the file and store it in a variable called `myData`, you use the following command:

```
  load myData.dat
```

2.2 Basic Matrix operations

Given b (below), the following table summarizes a few basic matrix operations and the results of these functions:

```
  b =
      1    2    3    4
      5    6    7    8
      9   10   11   12
     13   14   15   16
```

Function	Description	Results
`sum(b)`	Sums the values of each column*	`ans =` `28 32 36 40`

`diag(b)`	Produces all of the elements along the diagonal	`ans =`
		`1`
		`6`
		`11`
		`16`

`b'`	Transpose operator. The first column becomes the first row, the second column becomes the second row and so on	`ans =`
		`1 5`
		`9 13`
		`2 6`
		`10 14`
		`3 7`
		`11 15`
		`4 8`
		`12 16`

`max(b)`	Produces the maximum values for each column*	`ans =`
		`13 14`
		`15 16`

`min(b)`	Produces the minimum values for each column*	`ans =`
		`1 2`
		`3 4`

`mean(b)`	Produces the average of each column	`ans =`
		`7 8`
		`9 10`

`mean2(b)`	Produces the average of all the numbers in the matrix	`ans =`
		`8.5000`

*It is great to know the maximum, minimum or sum of each column, but sometimes you want it for the entire matrix. If you have a multidimensional matrix, you can turn it into a single column matrix using the notation b(:) (discussed in the "Colon Operator" section below). For instance, to get the sum of all the values in b, you can write:

```
sum(b(:))
```

which generates the answer below:

```
ans =
      136
```

2.3 Subscripts

Given `b` (from above), if you want to access the element in the first row and the forth column, you can use the notation

```
b(1,4)
```

The results are the following:

```
ans =
      4
```

Notice two things:

- you don't start at index 0
- and the order is in `row,column` format.

You can also use a single index. In this case, MATLAB regards your matrix as one long column that is the concatenation of each of the original columns. So, the first four entries in `b` when you use a single index are: `1 5 9 13`. So to access the same element as above, `b(1,4)`, using a single index, you would write: `b(13)`.

MATLAB provides error checking so that you do read past the bounds of the matrix. For instance, typing: `b(1,5)` results in:

```
??? Index exceeds matrix dimensions.
```

However, you can assign a value to a position that exceeds a matix's dimensions. For example if you type `b(1,5)=17`, you get the following matrix:

```
b =

     1     2     3     4    17
     5     6     7     8     0
     9    10    11    12     0
    13    14    15    16     0
```

In this case, a new (fifth) column has been created and it has been padded with zeros where no values were specified.

2.4 Colon Operator

The colon operator(:) can be used to select ranges or blocks of values.

By typing `b` and hitting return, you will see the values in `b`:

```
b =
       1     2     3     4
       5     6     7     8
       9    10    11    12
      13    14    15    16
```

If you only want a subset of b, for instance, the inner square, you can type: `b(2:3,2:3)`

```
ans =
       6     7
      10    11
```

Let's break down the `b(2:3,2:3)` a little. MATLAB is accessing row two to three (2:3) inclusive and column two to three (2:3) inclusive. Other than that, we are looking at the same `(row,column)` notation that you are used to.

Outside of the subscript notation, the colon operator will create a row vector containing the integers in the range specified. For instance, typing `2:8` results in:

```
ans =
       2     3     4     5     6     7     8
```

You can also specify an increment or decrement (as a middle argument) with the colon operator. For instance, typing `1:2:8` results in:

```
ans =
       1     3     5     7
```

In this case, the increment is 2. Effectively, you are getting all of the odd numbers between 1 and 8.

Just to give you more examples, the following table summarizes the results of using the colon operator both as a subscript to `b` and on its own:

Expression	Result	Description
`5:10`	`ans =` `5` `6 7` `8 9` `10`	Values 5 to 10.
`10:-1:5`	`ans =` `10` `9 8` `7 6` `5`	Values 10 to 5 (using decrement of 1)
`b(1:2,3:4)`	`ans =` `3` `4` `7` `8`	Row 1 to 2, Column 3 to 4 (gets the postage stamp values in the upper right hand corner of b)
`b(5:8)`	`ans =` `2` `6 10` `14`	This is using the single index notation. Remember it is a concatenation of one column after the other. In this case, you are getting the values of the second column.
`b(end)`	`ans =` `16`	**`end` is a special value which indicates the end of the matrix**. This is single index notation. You could have also written `b(end,end)` to specify the ending row and the ending column.
`b(1:4,4)`	`ans =` `4` `8` `12` `16`	This gets the values of the last column (rows 1 to 4 and column 4). Since you are specify all rows and the final column, you could also use the notation: `b(1:end,end)` or the **colon operator on its own, which specifies the entire**

		range of rows `b(:,end)`
`b(end:-1:1,:)`	`ans = 13 14 15 16 9 10 11 12 5 6 7 8 1 2 3 4`	Reverses the order of the rows so that the last row is first and the first row is last. Read this as: Rows last to first with a decrement of 1 (end:-1:1) and all columns(:)
`b(:)`	`ans = 1 5 9 13 2 6 10 14 3 7 11 15 4 8 12 16`	Produces one long column, which is a concatenation of the individual columns. This is useful when using functions such as `max`, `min`, or `sum`

2.5 Variables

- When you are declaring a variable, you do not need to specify the type or the dimensions (think about how this compares to creating an array in C++)
    - If there is another assignment statement using the same variable name, then MATLAB will change the contents and, where necessary, allocate new storage.
- Variable names are case sensitive (`b` and `B` are two separate variables).
- Variable names are a letter followed by any number of letters, digits, or underscores. Notice that there are no special characters allowed, for instance, the dollar sign ($) is not valid in variable names as it is in some other languages

- Variable names can be any length, but only the first `N` characters of the variable will be used. To see what `N` is, you can type: `namelengthmax`
- Every variable declared is an array/matrix. For instance, `myVar=2` creates a 1x1 matrix with a single element (`2`).

2.6 Predefined Functions and Help

The basic MATLAB package provides you with a number of functions that are useful for working with numbers and arrays. All of the following functions are part of the MATLAB base: `sin`, `cos`, `sqrt`, `abs`, `ceil`, `isprime`, `gcd`, `lcm`, `find`, `rot90`, `size`, `length`, `isempty`, and `isequal`.

For a more extensive list of some of the functions available to the basic MATLAB package, you can type the following commands:

`help elfun`	(for more on elementary mathematical functions)
`help specfun`	(for more on specialized mathematical functions)
`help elmat`	(for more on elementary matrices and matrix manipulations)

For a list of the functions available to MATLAB's Image Processing Toolbox, you can type the following command:

`help images`	(for more on Image Processing Toolbox)

If you want more help on a specific function, you can type:

```
help functionName
```

For instance, if you want to know more about the `sin` function. You can type:

```
help sin
```

or better yet:

```
doc sin
```

which opens up an indexed help window with further details about `sin`.

`sin` is a built-in function, which means that it is part of the MATLAB core and you cannot see the code behind this function. By contrast, there are functions such as `acosd` that are implemented in M-files. The file (`acosd.m`) is stored in: `matlabroot\toolbox\matlab\elfun` (where `matlabroot` is the installation directory of MATLAB). You can view this file directly from MATLAB using:

```
type acosd.m  (typing .m, in this case, is optional)
```

or to edit, you can type:

```
edit acosd
```

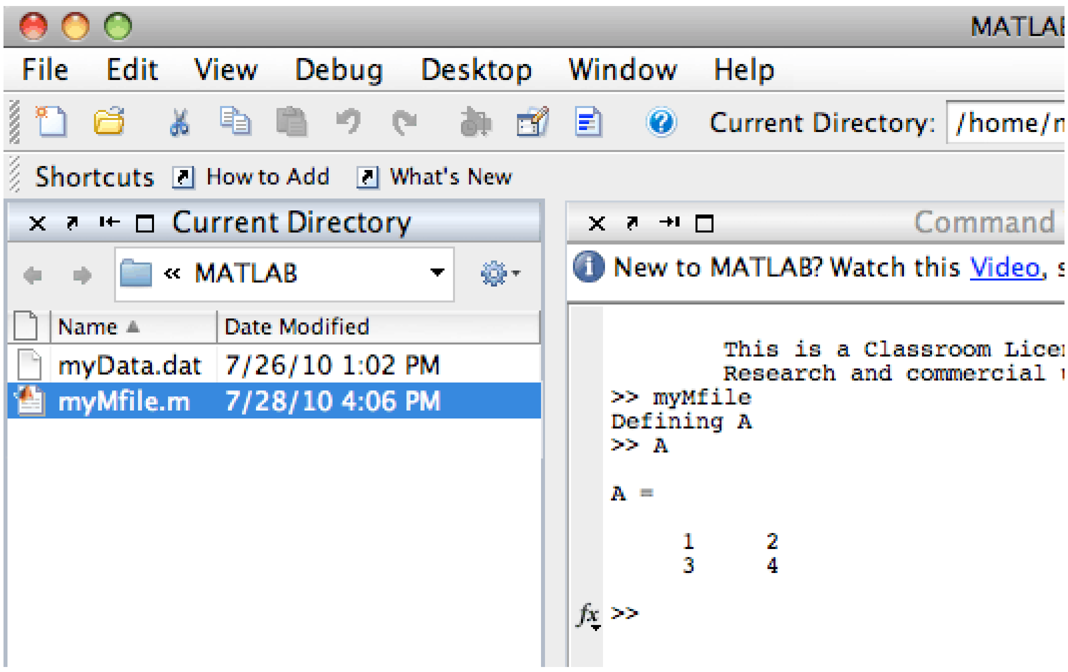
2.7 The M-Files

The edit command in the above section shows you a very basic M-File for the `acosd` function. The following is meant to summarize M-Files:

- M-Files are text files with MATLAB code
- M-Files have a .m extension
- **Exercise:** Create a basic M-File that works as a script to create an array called `A`.
    - Use a basic text editor like Notepad on a PC or Pico, vi, and emacs on a Mac to create the following text file:

      ```
      o  %{
      o     My first M-file
      o     Author: me
      o     Date: today
      o  %}
      o
      o  % Create a Matrix
      o  disp('Defining A');
      o  A=[
      o      1 2
      o      3 4
           ];
      ```

    - Notice the comments.
        - End of line comments start with `%`
        - Block comments are enclosed in `%{ %}` pairs
    - Notice the semicolons at the end of the two statements. Normally a semicolon prevents output as with the assignment, but `disp` works anyway.
    - Save the file as `myMfile.m` in MATLAB's **Current Directory**
    - You should see `myMfile.m` as in the diagram below.
        - **Tip:** Make sure that all your resource files are saved in the Current Directory, Otherwise, MATLAB won't find them when running.


MATLA
File Edit View Debug Desktop Window Help
Current Directory: /home/
Shortcuts How to Add What's New
Current Directory
MATLAB
Command
New to MATLAB? Watch this Video,

Name	Date Modified
myData.dat	7/26/10 1:02 PM
myMfile.m	7/28/10 4:06 PM

```
             This is a Classroom Lice
             Research and commercial
>> myMfile
Defining A
>> A

A =

         1         2
         3         4

>>
```

- run the file by typing:

  ```
  myMfile
  ```

- Voila, `A` should now be created.

2.8 Concatenation

The concatenation operator is indicated with square brackets []. Without even knowing it, you have been concatenating simple 1x1 arrays into multidimensional arrays with statements like the following that creates a 2x2 array called `A`:

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

To create an 4x4 array, which repeats the elements of `A` three times, you can write:

```
[A A; A A]
```

that results in:

```
ans =
       1       2       1       2
       3       4       3       4
       1       2       1       2
       3       4       3       4
```

You can also have two occurances of `A` (one below the other) and at the same time reverse the second occurance of `A` as in:

```
[A;A(end:-1:1,:)]
```

that results in:

```
ans =
     1     2
     3     4
     3     4
     1     2
```

2.9 Modifying Matrices

Let's say you have created `b` as below:

```
b =
     1     2     3     4
     5     6     7     8
     9    10    11    12
    13    14    15    16
```

If you want to change an element, you can do so using subscripts and assignment as in:

```
b(3,1)=3
```

which results in:

```
b =
     1     2     3     4
     5     6     7     8
     3    10    11    12
    13    14    15    16
```

You can change an entire row by specifying:

```
b(3,:)=3
```

which results in:

```
b =
     1     2     3     4
     5     6     7     8
     3     3     3     3
    13    14    15    16
```

You can replace the values in a row or column by using a 1 dimensional array with exactly the correct dimension. For example, to replace the second column of b you can write:

```
b(:, 2) = [1 2 3 4]
```

which results in:

```
b =

     1     1     3     4
     5     2     7     8
     3     3     3     3
    13     4    15    16
```

You can also replace a subset of values in a row or column as follows:

```
b(3,1:3) = [1 2 3]
```

which results in:

```
b =

     1     1     3     4
     5     2     7     1
     1     2     3     3
    13     4    15    16
```

If you decide to delete anything, you can use `[]` (empty square brackets). For instance, to delete the third row of `b`, you can write:

```
b(3,:)=[]
```

which results in:

```
b =

     1     1     3     4
     5     2     7     8
    13     4    15    16
```

However, you cannot partially remove or replace a row or column. For instance, the following statements:

```
b(1:2,2)=[]
b(3,2) = []
```

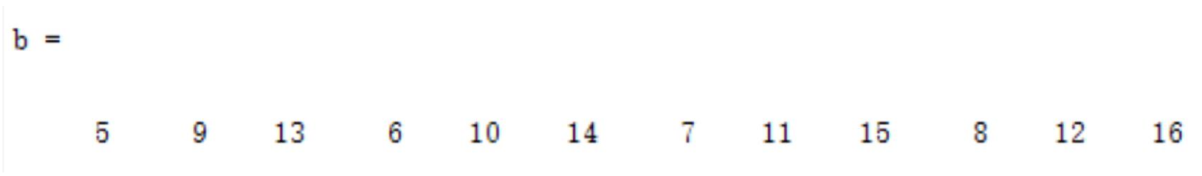
produces this error:

```
??? Subscripted assignment dimension mismatch.
```

You could, however, remove every fourth element and collapse the remaining elements into a row matrix as in:

```
b(1:4:end)=[]
```

which results in:



```
b =

     5     9    13     6    10    14     7    11    15     8    12    16
```

2.10 Operators

In MATLAB, there are both matrix operations and array operations. Addition (+)and subtraction (-) for both arrays and matrices are done element by element. With other operations, for instance multiplication, linear algebra specifies how matrices are multiplied; it is not element by element multiplication.

Sometimes, you may want to use the element by element array operations. MATLAB separates matrix operations from array operations by using a dot notation in front of the operators. The following is a list and description of array operations. The corresponding matrix operations omit the dot.

`A*B`	**Matrix multiply.** If one side is a scalar all elements of the other side are multiplied by the scalar. Otherwise, a matrix multiplication is performed and the number of columns in A must match the number of rows in B. See: help mtimes
`A.*B`	**Array multiply** If one side is a scalar all elements of the other side are multiplied by the scalar. Otherwise corresponding entries in each side are multiplied together and both sides must be matrices with identical dimensions. See: help times
`A/B`	**Right matrix divide** If B is a scalar all elements of A are divided by the scalar. Otherwise, if possible, the equivalent of the following multiplication is performed: AB^{-1} Specifically this operation is performed: (B' \ A') ' See: help mrdivide
`A./B`	**Right array divide** If B is a scalar all elements of A are divided by the scalar. Otherwise entries in A are divided by the corresponding entry in B and both sides must be matrices with identical dimensions. See: help rdivide
`A\B`	**Left matrix divide** If A is a scalar all elements of B are divided by the scalar. Otherwise, if possible, the equivalent of the following multiplication is performed: $A^{-1}B$. A warning will be printed if the result is likely to be unreliable. See: help mldivide
`A.\B`	**Left array divide** If A is a scalar all elements of B are divided by the scalar. Otherwise entries in B are divided by the corresponding entry in A and both sides must be matrices with identical dimensions. See: help ldivide

`A^B`	**Matrix power** If A is a matrix, it must be square and B must be a scalar. Then `A^3` would be equivalent to `AAA`. If B is a matrix, it must be square and A must be a scalar. Then A is raised to the matrix power B. See: help mpower
`A.^B`	**Array Power** If A and B are scalars, A is raised to the power B. If A is a scalar and B is a matrix, a matrix results where the elements are A raised to the power of the equivalent entry in B. If B is a scalar and A is a matrix, a matrix results where the elements are the elements of A raised to the power of B. If A and B are matrices, both must be the same size and a matrix results where the elements are the elements of A raised to the equivalent element of B. See: help power
`A'`	**Complex conjugate transpose** Array elements are flipped across the diagonal. In effect, in a 2D matrix the indices for an element are exchanged. The value of the imaginary portion of complex numbers is negated—the complex conjugate is taken for all entries in the matrix. See: help ctranspose
`A.'`	**Transpose** Array elements are flipped across the diagonal. In effect, in a 2D matrix the indices for an element are exchanged. Complex numbers are merely transposed, their values are not affected. See: help transpose

The following page provides some examples of the basic difference between matrix operations and array operations: arrayVersusMatrix.html.

For a complete list of MATLAB operators type `help ops` into the command window. You can get a description of the behaviour of an individual operator by typing `help` *`operator_mnemonic`*.

2.11 Moving Around

You may have already noticed that MATLAB provides you ways of cycling through commands you have already typed and editing those commands. A list of commands is on the bottom, left hand side, you can double click on any of those commands to repeat them.

In addition, you can use the up arrow or down arrow to move through previously typed commands.

There is a summary table of commands for editing on the command line available through MATLAB Help. You can also find it online by clicking on the following link and clicking on the links in the sequence indicated below: http://www.mathworks.com/access/helpdesk/help/techdoc/matlab.html

- Getting Started

- Matrices and Arrays
- Controlling Command Window Input and Output

2.12 Using Semicolons (;)

In MATLAB, semicolons “;” have two main uses. They:

- define the end of row, etc. A=[1,2;3,4].
- tell Matlab not to display any output when you end a line with a semicolon (;). This is particularly useful when you generate large matrices or perform long scripted computations.

3. Exercise

Write a script in an M-file called `matrices.m` that reproduces the results shown below. Matrix results should appear below one another, not beside as shown. The exercise is progressive, so create each array in the order shown. Use a header comment and in-line comments as you would in any other language.

1. Create the following three matrices in MATLAB:

```
A =                B =                                      C =
    22     5             17     9     2     1     3              1     0     0
     4    11              6     4    11     9    16              0     1     0
    13     2                                                     0     0     1
```

2. Use A to create the following matrices:

```
A =                 D =                 E =
    22     5            66    15            22     1    13
     1    11             3    33             5    11     2
    13     2            39     6
```

3. Use B to create the following matrices:

```
B =                                        F =
    17     9     3     1     3                  3     3    17
     6     4     3     9    16                 16     3     6
```

4. Use C to create the following matrices:

```
C =                 C =                 G=                    *
     1     0             1     0             22    -5
     0     1             0    -1              1   -11
                                             13    -2
```

*Hint: Matrix G uses A and C together

Remember to provide identifying information and adequate comments in your code.