Center for Teaching, Research and Learning
Research Support Group
American University, Washington, D.C.
Hurst Hall 203
rsg@american.edu
(202) 885-3862

# Introduction to MATLAB

---

## WORKSHOP OBJECTIVE:

This course provides students with basic tools needed to work with MATLAB.

## LEARNING OUTCOMES:

1. Orient to MATLAB interface and help functions
2. Using numbers and variables
3. Using matrices and arrays
4. Basic MATLAB programming

## I. STARTING MATLAB

### Windows system

You can start MATLAB by double-clicking on the MATLAB icon or invoking the application from the Start menu of Windows. The main MATLAB window, called the MATLAB Desktop (command window), will then pop-up.

## II. OBTAINING HELP

- Help and information on Matlab commands can be found in several ways:
  - from the command line by using the "help topic" command
  - from the separate Help window found under the Help menu or
  - from the Matlab helpdesk stored on a disk or CD-ROM
- Demonstrations are available as well and there is a comprehensive set of demos available by typing the command: `demo`
- Note however that this command will clear values of all current variables.

## III. DESKTOP VIEW

- The desktop manages tools differently from documents. The Command History and Editor are examples of tools, and an M-file is an example of a document, which appears in the Editor tool.
- Desktop menu allows for resizing, opening and closing, docking, grouping and other operations of the tools.
- It also contains some of the predefined layouts. For example default Matlab appearance is restored by going to: Desktop > Desktop Layout > Default

## IV. NUMBERS AND FORMATS

- Matlab recognizes several different kinds of numbers: integer, real, complex, Inf (result of dividing by zero), and NaN – **N**ot **a** **N**umber (result of 0/0)
- Number formats are controlled by the "format" command. Full list is available by typing >>help format

## V. VARIABLE NAMES

- Variable names consist of any number of letters, digits or underscores. Spaces are not allowed. Also, note that Matlab is case sensitive; so "Beta" and "beta" are not the same variable.
- Text can be assigned to variables by using single quotes around the text you want to assign to a variable.
  - Ex: `>>b = 'this is a variable'`
- Note: MATLAB provides three basic types of variables:
  - Local Variables
  - Global Variables
  - Persistent Variables
- Please refer to the Help menu in Matlab for further discussion on use and difference between these types.
- If the text includes a single quote, use two single quotes within the definition.

```
otherText = 'You''re right'

otherText =
'You're right'
```

## VI. MATRICES AND ARRAYS

- *MATLAB* is an abbreviation for "matrix laboratory." While other programming languages mostly work with numbers one at a time, MATLAB® is designed to operate primarily on whole matrices and arrays.
- 
- All MATLAB variables are multidimensional *arrays*, no matter what type of data. A *matrix* is a two-dimensional array often used for linear algebra.
- 
- A matrix is a two-dimensional numeric array that represents a linear transformation. Informally, the terms matrix and array are often used interchangeably.
- Arithmetic operations on arrays are done element by element. Thus, addition and subtraction are the same for arrays and matrices, but multiplication is different. Matlab notation dealing with it discussed below.

### Generating matrices

To enter a matrix into Matlab we can type it in row by row: 3

```
>> A = [1 3 5
4 5 6]
```

Or by separating rows by semi-colons rather than a new line:

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

There are four functions that generate basic matrices:

`zeros` - All zeros

```
X = zeros(n)
```

`ones` - All ones

```
Y = ones(n)
Y = ones(m,n)
```
`rand` - Uniformly distributed random elements
```
W= rand(n)
```
`randn` – Normally distributed random elements
```
Y = randn(n)
Y = randn(m,n)
```

## Array operations

The list of operators includes:
+ Addition
- Subtraction
 .* Element-by-element multiplication
./ Element-by-element division
.\ Element-by-element left division
.^ Element-by-element power
.' Unconjugated array transpose

To transpose a matrix, use a single quote (`'`):

```
a'
```

```
ans =

     1     4     7
     2     5     8
     3     6    10
```

Inverse

Inverse(a)

To perform element-wise multiplication rather than matrix multiplication, use the `.*` operator:

```
p = a.*a
```

```
p =

     1     4     9
    16    25    36
    49    64   100
```

The matrix operators for multiplication, division, and power each have a corresponding array operator that operates element-wise. For example, raise each element of `a` to the third power:

```
a.^3
```

```
ans =

           1           8          27
          64         125         216
         343         512        1000
```

‘ Matrix/Array Conjugated transpose

## Concatenating matrices

Matrix concatenation is the process of joining one or more matrices to make a new matrix. The brackets [] operator discussed earlier in this section serves not only as a matrix constructor, but also as the MATLAB concatenation operator. The expression C = [A B] horizontally concatenates matrices A and B. The expression C = [A; B] vertically concatenates them.
This example constructs a new matrix C by concatenating matrices A and B in a vertical direction:

## Concatenation

*Concatenation* is the process of joining arrays to make larger ones. In fact, you made your first array by concatenating its individual elements. The pair of square brackets `[]` is the concatenation operator.

```
A = [a,a]
A =

     1     2     3     1     2     3
     4     5     6     4     5     6
     7     8    10     7     8    10
```

Concatenating arrays next to one another using commas is called *horizontal* concatenation. Each array must have the same number of rows. Similarly, when the arrays have the same number of columns, you can concatenate *vertically* using semicolons.

```
A = [a; a]

A =

     1     2     3
     4     5     6
     7     8    10
     1     2     3
     4     5     6
     7     8    10
```

```
A = ones(2, 5)*6;    % 2-by-5 matrix of 6's
B = rand(2, 5);      % 2-by-5 matrix of random values
C = [A; B]           % Vertically concatenate A and B
C =
6.0000 6.0000 6.0000 6.0000 6.0000
6.0000 6.0000 6.0000 6.0000 6.0000
0.9501 0.4860 0.4565 0.4447 0.9218
0.2311 0.8913 0.0185 0.6154 0.7382
```

Some built in functions to concatenate matrices are:
`cat` - Concatenate matrices along the specified dimension

Sytnax:
`C = cat(dim, A, B)` where `dim=1` is for columns and `dim=2` for rows
Example:
```
C = cat(1, A, B)
```
`horzcat` - Horizontally concatenate matrices
Sytnax:
```
C = horzcat (A1, A2, ...)
```
`vertcat` - Vertically concatenate matrices
Sytnax:
```
C = vertcat(A1, A2, ...)
```
`repmat` - Create a new matrix by replicating and tiling existing matrices
Syntax:
`repmat(M, v, h)` : Matlab will replicate matrix M v times vertically and h times horizontally
`blkdiag` - Create a block diagonal matrix from existing matrices

## The colon operator

The colon operator (first:last) generates a 1-by-n matrix (or vector) of sequential numbers from the first value to the last. The default sequence is made up of incremental values, each 1 greater than the previous one:

```
A = 10:15
A =
10 11 12 13 14 15 5
```

The numeric sequence does not have to be made up of positive integers. It can include negative numbers and fractional numbers as well:

```
A = -2.5:2.5
A =
-2.5000 -1.5000 -0.5000 0.5000 1.5000 2.5000
```

By default, MATLAB always increments by exactly 1 when creating the sequence, even if the ending value is not an integral distance from the start:

```
A = 1:6.3
A =
1 2 3 4 5 6
```

Also, the default series generated by the colon operator always increments rather than decrementing. The operation shown in this example attempts to increment from 9 to 1 and thus MATLAB returns an empty matrix:

```
A = 9:1
A =
```

Empty matrix: 1-by-0

## Using the colon operator with a step value

To generate a series that does not use the default of incrementing by 1, specify an additional value with the colon operator (first:step:last). In between the starting and ending value is a step value that tells MATLAB how much to increment (or decrement, if step is negative) between each number it generates.
To generate a series of numbers from 10 to 50, incrementing by 5, use
A = 10:5:50
A =
10 15 20 25 30 35 40 45 50
You can increment by noninteger values. This example increments by 0.2:
A = 3:0.2:3.8
A =
3.0000 3.2000 3.4000 3.6000 3.8000

To create a sequence with a decrementing interval, specify a negative step value: 6
A = 9:-1:1
A = 9 8 7 6 5 4 3 2 1

## Accessing single elements

To reference a particular element in a matrix, specify its row and column number using the following syntax, where A is the matrix variable. Always specify the row first and column second:

```
A(row, column)
```

Every variable in MATLAB® is an array that can hold many numbers. When you want to access selected elements of an array, use indexing.

For example, consider the 4-by-4 magic square `A`:

```
A = magic(4)
```

```
A =

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

There are two ways to refer to a particular element in an array. The most common way is to specify row and column subscripts, such as

```
A(4,2)
```

```
ans =

    14
```

```
    test = A(4,5)
```

```
Index exceeds matrix dimensions.
```

However, on the left side of an assignment statement, you can specify elements outside the current dimensions. The size of the array increases to accommodate the newcomers.

```
A(4,5) = 17
```

```
A =

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

The colon alone, without start or end values, specifies all of the elements in that dimension. For example, select all the columns in the third row of `A`:

```
A(3,:)
```

```
ans =

     9     7     6    12     0
```

The colon operator also allows you to create an equally spaced vector of values using the more general form `start:step:end`.

```
B = 0:10:100

B =

     0    10    20    30    40    50    60    70    80    90   100
```

To refer to multiple elements of an array, use the colon operator, which allows you to specify a range of the form `start:end`. For example, list the elements in the first three rows and the second column of `A`:

```
A(1:3,2)

ans =

     2
    11
     7
```

### Specifying all elements of a row or column

The colon by itself refers to all the elements in a row or column of a matrix. Using the following syntax, you can compute the sum of all elements in the second column of a 4-by-4 magic square A:

```
>> sum(A(:,2))
ans =
12
```

### Sorting matrix elements

The sort function sorts matrix elements along a specified dimension. The syntax for the function is

`sort(matrix, dimension)`

To sort the columns of a matrix, specify 1 as the dimension argument. To sort along rows, specify dimension as 2.

### Sorting the data in each row

Use `issorted` to sort data in each row. Using the example above, if you sort each row of A in descending order, `issorted` tests for an ascending sequence. You can flip the vector to test for a sorted descending sequence:

### Sorting the data in each column

The `sortrows` function keeps the elements of each row in its original order, but sorts the entire row of vectors according to the order of the elements in the specified column. 7

## VII. BASIC PROGRAMMING COMPONENTS

### MATLAB expressions for string evaluation

`eval`

The eval function evaluates a string that contains a MATLAB expression, statement, or function call. In its simplest form, the eval syntax is

```
eval('string')
```

`feval`

The feval function differs from eval in that it executes a function rather than a MATLAB expression. The function to be executed is specified in the first argument by either a function handle or a string containing the function name.

Example:

>> sin(5)
>> feval(@sin,5)

```
fun = {@sin; @cos; @log};
k = input('Choose function number: ');
x = input('Enter value: ');
feval(fun{k}, x)
```

## VIII. PROGRAM CONTROL STATEMENTS

### If, else, ifelse

if evaluates a logical expression and executes a group of statements based on the value of the expression. In its simplest form, its syntax is
`if` logical_expression
statements
`end`
`if` evaluates a logical expression and executes a group of statements based on the value of the expression. In its simplest form, its syntax is
`if` logical_expression
statements
`end`

### Loop control – for, while, continue, break

The **for** loop executes a statement or group of statements a predetermined number of times. Its syntax is:

for index = start:increment:end
statements
end
The default increment is 1. You can specify any increment, including a negative one. For positive indices, execution terminates when the value of the index exceeds the end value; for negative increments, it terminates when the index is less than the end value.
For example, this loop executes five times.

```
for n = 2:6
x(n)  = 2 * x(n - 1);
end
```

The **while** loop executes a statement or group of statements repeatedly as long as the controlling expression is true (1). Its syntax is
while expression
statements
end

If the expression evaluates to a matrix, all its elements must be 1 for execution to continue. To reduce a matrix to a scalar value, use the `all` and `any` functions.
For example, this while loop finds the first integer n for which n! (n factorial) is a 100-digit number.

```
n = 1;
while prod(1:n) < 1e100
n = n + 1; end
```

## IX. FUNCTIONS AND SCRIPTS

A Matlab script is an ASCII text file that contains a sequence of Matlab commands (expressions and assignments). To execute all of the lines of the file sequentially just type the filename at the command prompt.
Any text editor or program editor can be used to create scripts, but you must make sure that scripts are saved as simple text documents. Matlab has a built-in 9 text editor which will automatically save files as ASCII text file. Also, when naming script files suffix ".m" is needed (ex. "myprogram.m").
M-files can be scripts that simply execute a series of MATLAB statements, or they can be functions that also accept input arguments and produce output.
MATLAB scripts:
-Are useful for automating a series of steps you need to perform many times.
-Do not accept input arguments or return output arguments.
-Store variables in a workspace that is shared with other scripts and with the MATLAB command line interface.
MATLAB functions:
-Are useful for extending the MATLAB language for your application.
-Can accept input arguments and return output arguments.
-Store variables in a workspace internal to the function.

### Basic parts of an m-file

This simple function shows the basic parts of an M-file. Note that any line that begins with % is not executable:

`function f = fact(n)` Function definition line
% Compute a factorial value. H1 line
% FACT(N) returns the factorial of N, Help text
% usually denoted by N!
% Put simply, FACT(N) is PROD(1:N). Comment
`f = prod(1:n);` Function body

***Both functions and scripts can have all of these parts, except for the function definition line which applies to functions only.***
Note:
1) If a MATLAB command is followed by a semicolon the output associated is suppressed. 2) The variables defined in the script remain in the workspace even after the script finishes running.

### Scripts

Scripts are the simplest kind of M-file because they have no input or output arguments. They are useful for automating series of MATLAB commands, such as computations that you have to perform repeatedly from the command line. 10
Scripts share the base workspace with your interactive MATLAB session and with other scripts. They operate on existing data in the workspace, or they can create new data on which to operate. Any variables that scripts create remain in the workspace after the script finishes so you can use them for further computations. You should be aware, though, that running a script can unintentionally overwrite data stored in the base workspace by commands entered at the MATLAB command prompt.
Simple Script Example
These statements calculate rho for several trigonometric functions of theta, then create a series of polar plots:

```
% An M-file script to produce % Comment lines
```

```
% "flower petal" plots
theta = -pi:0.01:pi;  % Computations
rho(1,:) = 2 * sin(5 * theta) .^ 2;
rho(2,:) = cos(10 * theta) .^ 3;
rho(3,:) = sin(theta) .^ 2;
rho(4,:) = 5 * cos(3.5 * theta) .^ 3;
for k = 1:4
polar(theta, rho(k,:)) % Graphics output
end
```

Try entering these commands in an M-file called petals.m. This file is now a MATLAB script. Typing petals at the MATLAB command line executes the statements in the script.

After the script displays a plot, press Enter or Return to move to the next plot. There are no input or output arguments; petals creates the variables it needs in the MATLAB workspace. When execution completes, the variables (i, theta, and rho) remain in the workspace. To see a listing of them, enter „whos" at the command prompt.

### Functions

Functions are program routines, usually implemented in M-files, that accept input arguments and return output arguments. They operate on variables within their own workspace. This workspace is separate from the workspace you access at the MATLAB command prompt. 11

Each M-file function has an area of memory, separate from the MATLAB base workspace, in which it operates. This area, called the function workspace, gives each function its own workspace context.

While using MATLAB, the only variables you can access are those in the calling context, be it the base workspace or that of another function. The variables that you pass to a function must be in the calling context, and the function returns its output arguments to the calling workspace context. You can, however, define variables as global variables explicitly, allowing more than one workspace context to access them.

You can also evaluate any MATLAB statement using variables from either the base workspace or the workspace of the calling function using the evalin function. See Extending Variable Scope for more information.

### Simple function example

The average function is a simple M-file that calculates the average of the elements in a vector:

```
function y = vector_average(x)
% VECTOR_AVERAGE Mean of vector elements.
% VECTOR_AVERAGE(X), where X is a vector, is the mean of vector
% elements. Nonvector input results in an error.
[m,n] = size(x);
if (~((m == 1) | (n == 1)) | (m == 1 & n == 1))
error('Input must be a vector')
end
y = sum(x)/length(x); % Actual computation
```

Try entering these commands in an M-file called `average.m`. The average function accepts a single input argument and returns a single output argument. To call the average function, enter

```
z = 1:99;
vector_average(z)
ans =50
```

## X. OTHER NOTES

MATLAB is available on computers in the Hurst 202, 203 labs and in Anderson Computing Labs.

For a full list of our other workshops, go to http://www.american.edu/ctrl/rsgevents.cfm

Assistance with MATLAB is also available in the CTRL lab during normal business hours. For more information, go to http://www.american.edu/ctrl/lab.cfm