

Programming in Mathematics

Mili I. Shah

Starting Matlab

Go to

<http://workspace.loyola.edu/>

and login with your Loyola name and password...

Matlab has eight main windows:

Command Window	Main window, enter variables, runs programs
Figure Window	Contains output from graphic commands
Editor Window	Creates and debugs script and function files
Help Window	Provides help information
Launch Pad Window	Provides access to tools, demos, and documentation
Command History Window	Logs commands entered in the Command Window
Workspace Window	Provides information about the variables that are used
Current Directory Window	Shows the files in the current directory

Command Window

- To type a command the cursor must be placed next to the command prompt (`>>`).
- Press **Enter** for the command to be executed. Multiple commands can be typed by typing a comma (,) between them.
- A semicolon (;) at the end of a command suppresses the screen output.
- Upper and lower case characters are not equivalent.
- The up and down arrow keys can be used to scroll through previous commands. Also an old command can be recalled by typing the first few characters followed by the up arrow.
- Type `help topic` to access online help on the command, function, or symbol `topic`.
- Type `clc` to clear the screen
- Type `exit` or `quit` to quit Matlab.

Built-In Functions

<code>sqrt(x)</code>	Square root
<code>exp(x)</code>	Exponential (e^x)
<code>abs(x)</code>	Absolute value
<code>log(x)</code>	Natural logarithm
<code>sin(x)</code>	Sine of x
<code>cos(x)</code>	Cosine of x
<code>tan(x)</code>	Tangent of x
<code>cot(x)</code>	Cotangent of x
<code>pi</code>	π

```
>> sqrt(4)
```

```
ans =
```

```
2
```

```
>> pi
```

```
ans =
```

```
3.1416
```

Defining Scalar Variables

Variable = Numerical value or computable expression

- = is the **assignment operator** which assigns a value to a variable
- Left-hand side can include only **one** variable name
- Right-hand side can be a number or an expression made up of numbers and/or variables previously assigned numerical values
- Variables must begin with a letter
- Press **Enter** to make the assignment
- `ans` is the value of the last expression that is not assigned

Remember:

- Use semicolon (;) to suppress screen output
- Multiple commands can be typed by typing a comma (,) between them.

Defining Scalar Variables

Example: Assign the number 3 to variable **a** and 4 to variable **b**. Print out $\sqrt{a^2 + b^2}$ and assign the solution to the variable **c**.

```
>> a=3; b=4; c = sqrt(a^2+b^2)
c =
5
```

Example: Verify

$$\cos^2 \frac{x}{2} = \frac{\tan x + \sin x}{2 \tan x}$$

by calculating each side of the equation for $x = \pi/5$.

```
>> x = pi/5;
>> LHS = cos(x/2)^2, RHS = (tan(x)+sin(x))/(2*tan(x))
LHS =
0.9045
RHS =
0.9045
```

Arrays

- Used to store and manipulate numbers
- Arranged in rows or columns

One-Dimensional Array (Vector)

- Represents point in n -dimensional space
Ex: (x, y) in 2D and (x, y, z) in 3D
- **Row Vector** (Use space or comma between numbers)

```
>> x = [1 2 3]
x =
1 2 3
```

- **Column Vector** (Use semicolon between numbers)

```
>> x = [1; 2; 3]
x =
1
2
3
```

- **Constant Spaced Vectors:**

- From m spaced by q to n

```
variable = [m : q : n]
```

```
>> x = [1:2:7]
```

```
x =
```

```
1 3 5 7
```

- From m to n with q elements

```
variable = linspace(m,n,q)
```

```
>> x = linspace(0,1,5)
```

```
x =
```

```
0 0.2500 0.5000 0.7500 1.0000
```


Two-Dimensional Array (Matrix)

- Can store information like a table
- Solve systems of equations such as

$$2x + 3y + z = 4$$

$$x - 5y + 3z = 3$$

$$4x - 2y + 3z = 2$$

variable = [1st row; 2nd row; ...; last row]

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

```
x =
```

```
2   3   1
```

```
1  -5   3
```

```
4  -2   3
```

- **Vector:**

- $ve(k)$ picks the k th element of ve
- $ve(m:n)$ picks the m th through n th elements of ve

```
>> ve = [1 5 2 6 8 7]
```

```
ve =
```

```
1 5 2 6 8 7
```

```
>> ve(5)
```

```
ans =
```

```
8
```

```
>> ve(2:4)
```

```
ans =
```

```
5 2 6
```

Addressing Elements

- **Matrix:**

- `mat(m,n)` picks the (m, n) th element of `mat`
- `mat(m:n,p:q)` picks the $(m : n) \times (p : q)$ submatrix of `mat`

```
>> mat = [1 4 2 3; 3 6 9 2; 1 4 9 7; 2 5 1 8]
```

```
mat =
```

```
1 4 2 3
```

```
3 6 9 2
```

```
1 4 9 7
```

```
2 5 1 8
```

```
>> mat(2,3)
```

```
ans =
```

```
9
```

```
>> mat(2:4, 1:3)
```

```
3 6 9
```

```
ans = 1 4 9
```

```
2 5 1
```

Adding Elements

- Can add elements by using the variable within vector/matrix
- Must be of appropriate size

```
>> mat = [1 4 2 3; 3 6 9 2; 1 4 9 7]
```

```
mat =
```

```
1 4 2 3
```

```
3 6 9 2
```

```
1 4 9 7
```

```
>> [mat; 2 5 1 8]
```

```
ans =
```

```
1 4 2 3
```

```
3 6 9 2
```

```
1 4 9 7
```

```
2 5 1 8
```

Deleting Elements

- Delete elements by assigning nothing to these elements

```
>> ve = [1 5 2 6 8 7]
```

```
ve =
```

```
1 5 2 6 8 7
```

```
>> ve(2:4) = []
```

```
ve =
```

```
1 8 7
```

```
>> mat = [1 4 2 3; 3 6 9 2; 1 4 9 7]
```

```
mat =
```

```
1 4 2 3
```

```
3 6 9 2
```

```
1 4 9 7
```

```
>> mat(2:3,:) = []
```

```
mat =
```

```
1 4 2 3
```

Helpful Tips for Arrays

<code>length(A)</code>	Returns number of elements in the vector A
<code>size(A)</code>	Returns size of matrix A
<code>reshape(A,m,n)</code>	Rearranges A to have m rows and n columns (arranged column-wise)

Matrix Operations

- Matrix operations can be performed by using (+, -, *, /, ^)

```
>> A = [1 4 2; 1 6 3; 1 3 2]
```

```
A =
```

```
1 4 2
```

```
1 6 3
```

```
1 3 2
```

```
>> B = [1 2; 1 5; 3 6]
```

```
B =
```

```
1 2
```

```
1 5
```

```
3 6
```

```
>> A*B
```

```
ans =
```

```
11 34
```

```
16 50
```

```
10 29
```

```
>> B*A?
```

Systems of Equations

- Can solve systems of equations with `inv` or the backslash (`\`)

```
>> A = [1 4 2; 1 6 3; 1 3 2]
```

```
A =
```

```
1 4 2
```

```
1 6 3
```

```
1 3 2
```

```
>> b = [1; 1; 3]
```

```
b =
```

```
1
```

```
1
```

```
3
```

```
>> inv(A)*b;
```

```
>> A\b;
```

- See the answer for both is $[1, -2, 4]^T$

Element-wise operations

- Element-wise matrix operations can be performed using $(.*, ./, .^)$

```
>> A = [1 4 3; 2 5 6]
```

```
A =
```

```
1 4 3
```

```
2 5 6
```

```
>> B = [4 6 3; 7 3 5]
```

```
B =
```

```
4 6 3
```

```
7 3 5
```

```
>> A.*B
```

```
ans =
```

```
4 24 9
```

```
14 15 30
```

- Can create random matrices using the following commands
 - `rand`: Generates a single random number between 0 and 1
 - `rand(1,n)`: Generates an n element row vector of random numbers between 0 and 1
 - `rand(n)`: Generates an $n \times n$ matrix with random numbers between 0 and 1
 - `rand(m,n)`: Generates an $m \times n$ matrix with random numbers between 0 and 1
 - `randn`: Generates normally distributed numbers with mean 0 and standard deviation 1 (inputs same as `rand`)
 - `randperm(n)`: Generates a row vector with n elements that are random perturbations of integers 1 through n

Many different type of 2D plots

- `plot`: Basic 2D plot (x - y plane)
- `loglog`: Plot with logarithmically scaled axes
- `semilogx`: Plot with logarithmically scaled x -axis
- `semilogy`: Plot with logarithmically scaled y -axis
- `bar`: Bar graph
- `hist`: Histogram
- `polar`: Polar graph

Basic Plots

- Basic connect-the-dots command

```
plot(x,y,string)
```

```
>> x = linspace(-2,2);  
>> y = x.^2;  
>> plot(x,y)
```

Now for a red curve with the markers as stars

```
>> plot(x,y,'r*')
```

For a full list of options type `help plot`

- `loglog` is the same as `plot` except the axes are logarithmic
- `semilogx` is the same as `plot` except the x-axis is logarithmic
- `semilogy` is the same as `plot` except the y-axis is logarithmic
- `hold on` Holds the graph so you can add more
- `xlabel`, `ylabel`, `title`, `legend` Labels for the graph
- `axis auto`, `axis equal`, `axis square`, `axis tight`
- `axis([xmin xmax ymin ymax])`

(Not so) Basic Plots

```
% This produces one of those killer spirographs
% Adapted from:
% http://www.starchamber.com/paracelsus/matlab/
r1 = rand;
r2 = rand;
r3 = rand;

a1= 0:0.15:314;
a2 = a1 * r1/r2;

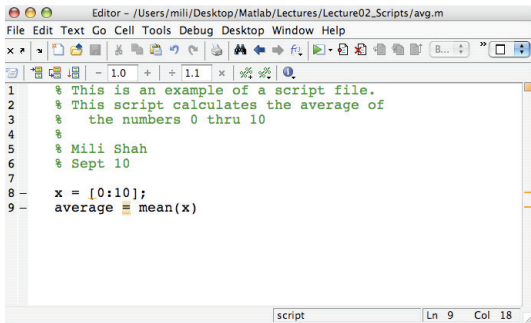
x = (r1 - r2) * cos(a1) - cos(a2) * r3;
y = (r1 - r2) * sin(a1) - sin(a2) * r3;

plot(x,y)
```

Try axis equal, axis square, grid on, axis off, hold

Scripts

- A **script file** is a sequence of commands also called a program.
- The output is displayed in the Command Window
- A script file is convenient because it can be edited and executed many times
- Script files can be typed and edited in any text editor and then pasted into the Matlab editor
- Also called an m-file because the extension `.m` is used



The screenshot shows a MATLAB script editor window titled "Editor - /Users/mili/Desktop/Matlab/Lectures/Lecture02_Scripts/avg.m". The window contains the following code:

```
1 % This is an example of a script file.
2 % This script calculates the average of
3 % the numbers 0 thru 10
4 %
5 % Mili Shah
6 % Sept 10
7
8 x = [0:10];
9 average = mean(x)
```

The status bar at the bottom of the window indicates "script" and "Ln 9 Col 18".

Script Output: disp

`disp`: Displays output on the screen

`disp(name of variable)` or `disp('text as string')`

```
>> A = [1 4 3; 2 5 6];  
>> disp(A)  
1 4 3  
2 5 6  
>> disp('Matlab is GREAT');  
Matlab is GREAT
```

- Note: `disp('')` displays an empty line
- Note: If printing a table, spaces may have to be added for tables to line up

Script Output: fprintf in Command Window

fprintf: Displays output on the screen or saves output to file

```
fprintf('text typed in as a string')
```

```
>> fprintf('Matlab is GREAT');  
Matlab is GREAT>>
```

- Note: New line is not created. Use `\n` for a new line
- Note: Use `\t` for a horizontal tab
- Can be used to display a mix of text and numerical data

```
fprintf('text %-5.2f additional text %-5.2f', var1,var2)
```

- Formatting numerical data:

`%Flag Precision(1st).Precision(2nd) Conversion`

- Flag: (-) Left justifies, (+) Prints sign, (0) Adds zeros if short
- Precision: (1st) is field width, (2nd) is # of digits rt of decimal
- Conversion: (e) lower exp, (E) upper exp, (f) fixed, (i) integer

Script Output: fprintf in Command Window

Example:

```
% This script prints out the surface area and volume  
% of a cylinder with a given radius and height
```

```
radius = 5;  
height = 2;
```

```
surfarea = 2*pi*radius^2+ 2*pi*radius*height;
```

```
vol = pi*radius^2*height;
```

```
fprintf('For a cylinder with radius %i and height %e, the surface area  
is %5.2f and the volume is %5.3f', radius, height, surfarea, vol)
```

Script Output: fprintf to file

Three Steps

- 1 Opening a file with `fopen`

```
fid = fopen('filename','permission')
```

- permission: (r) reading, (w) writing, (a) appending

- 2 Writing output to open file with `fprintf`

```
fprintf(fid,'text')
```

- 3 Closing the file with `fclose`

```
fclose(fid)
```

Tips

- Saved in current directory
- Can write to several files using different `fid`, e.g. `fid1`, `fid2`
- Is vectorized \Rightarrow the command repeats itself until all elements are displayed (column-wise)

Script Output: fprintf to file

Example:

```
% This script creates a chart of inches to centimeters
```

```
inches = [1:10];
```

```
centim = 2.54*inches;
```

```
% Creates table to take care of vectorization
```

```
TBL = [inches;centim];
```

```
fid = fopen('in2cm.txt','w');
```

```
fprintf(fid, 'Inches to Centimeters Table\n\n');
```

```
fprintf(fid, 'Inches \t Centimeters\n');
```

```
fprintf(fid, '%i \t \t %3.2f\n', TBL);
```

```
fclose(fid);
```