MM 220 Autumn 2020
Tutorial 1: MATLAB Preliminaries

Anirban Patra, Amrita Bhattacharya
Department of Metallurgical Engineering and Materials Science, IIT Bombay

This tutorial provides a brief introduction to MATLAB and its capabilities.

MATLAB stands for MATrix LABoratory. It is a proprietary software developed by MathWorks. MATLAB is a programming language and computing environment used widely in scientific computing. It provides easy use of matrix and vector operations and may be used for linear algebra calculations as well as differential equation solving, numerical integration, and 2D and 3D plotting. MATLAB programming uses the MATLAB scripting language and also provides interface to other programming languages such as C, C++, Fortran and Python.

In this tutorial, we will learn how to use MATLAB for simple mathematical operations and plotting. In this tutorial, all commands/functions for MATLAB are written using the `Courier` font.

Lesson 0: *Getting familiarized with the MATLAB environment*

`L0.1:` A typical MATLAB environment has multiple panels: Command Window, Command History, Workspace, and Current Folder. Panels may be added or removed by selecting from the Layout tab. All MATLAB commands are to be written at the command prompt (`>>`) in the Command Window. Command History shows all the previously used commands. Workspace shows all the variables and data used in the current session. Current Folder shows all the files in the folder.

`L0.2:` MATLAB is well documented. Help for all commands can be obtained by:

```
>> help <command>
```

where `<command>` is to be replaced by the specific command/function. And of course there is Google for everything else!

`L0.3:` Basic calculator functions can be performed at the command prompt in MATLAB. For example,

```
>> sin(pi/4)*exp(pi)
```

gives the output

```
ans =
   16.3629
```

Here, `sin()` is the function for calculating the sine of a number, `exp()` is the function for calculating the exponential of a number, and `pi` is π. Functions for most of the common mathematical operations exist in MATLAB and can be found from the documentation.

`L0.4:` Before beginning a new session/writing new code in MATLAB, it is always good practice to clear existing data (unless of course we want to work with existing data). This can be done using:

```
>> clear all;
```

For clearing the Command Window, use:

```
>> clc;
```

Note: It is always a good practice to end MATLAB commands with a semicolon (`;`), especially when writing long pieces of code. This suppresses the output of the current command.

Lesson 1: *Creating vectors and matrices and assigning values*

Unlike programming languages such as C, C++ and Fortran, variables need not be defined a priori in MATLAB, although it is always a good practice to do so, especially for multi-dimensional matrices.

`L1.1:` Use:

```
>> a = 5;
```

to create a 1x1 matrix called `a` with a value of `5` assigned to it. The value of `a` can be queried using:

```
>> a
a =
     5
```

`L1.2:` Now, we create a vector `b` of dimension 5 with values (10 20 30 40 50) using:

```
>> b = [10 20 30 40 50];
>> b
b =
    10    20    30    40    50
```

Note that when defining vectors or matrices, we use the square brackets `[]` in MATLAB. The columns of the vector are separated by blank spaces or tabs ( ).

`L1.3:` If we want to query a specific element of this vector, we use:

```
>> b(2)
ans =
      20
```

Note that we use round brackets `()` when querying the value of an element in a vector. The index `2` refers to the 2nd element of the vector. In MATLAB, the matrix index starts from `1`. This is unlike C/C++, where the matrix index starts from `0`.

`L1.4:` If we want to modify or assign another value to an element of the vector, this can be simply done using:

```
>> b(2) = 25;
>> b
b =
    10    25    30    40    50
```

`L1.5:` Now, we create a 2D matrix `c` of dimension 5x4.

```
>> c = [1 2 3 4; 5 6 7 8; 9 10 11 12; 13 14 15 16; 17 18 19 20];
>> c
c =
    1    2    3    4
    5    6    7    8
    9   10   11   12
   13   14   15   16
   17   18   19   20
```

`c` is a matrix with 5 rows and 4 columns. Columns are separated by blank spaces or tabs (`   `), while rows are separated by semi-colons (`;`).

We can query the value of an element in `c` using:

```
>> c(5,1)
ans =
     17
```

`L1.6:` We can use the function `size` to query the dimensions of any vector or matrix:

```
>> size(a)
ans =
     1   1
>> size(b)
ans =
     1   5
>> size(c)
ans =
     5   4
```

Note that all variables are defined as matrices in MATLAB. For example, the variable `a` has dimension 1x1.

`L1.7:` There are various ways to define/initialize matrices in MATLAB.

For example, we can define a vector `b2` with the value of the first element as `10` and the last element as 50, and having elements with equal spaced intervals of 10:

```
>> b2 = 10:10:50;
>> b2
b2 =
    10   20   30   40   50
```

Note that the elements of vector `b2` have the same values as vector `b` initialized in `L1.2`.
To initialize a 1x5 matrix with all elements having the value `0`, we use:

```
>> b3 = zeros(1,5);
```

To initialize a 5x4 matrix with all elements having the value `0`, we use:

```
>> c2 = zeros(5,4);
```

To initialize a 5x4 matrix with all elements having the value `1`, we use:

```
>> c3 = ones(5,4);
```

Note that if only one dimension is provided, a square matrix is initialized, i.e.,

```
>> d = zeros(3);
>> d
d =
    0   0   0
    0   0   0
    0   0   0
```

Another way to initialize an equally spaced vector is:

```
>> b3 = linspace(10,50,5);
```

Here, `b3` has `10` as the first element and `50` as last element, and a total of equally spaced `5` elements.

Lesson 2: *Performing mathematical operations*

MATLAB provides a variety of functions for matrix operations. This is the main benefit of using MATLAB over other programming languages such as C, C++ or Fortran.

For the lessons in this Section, let us first define a 3x3 square matrix d2 as:

```
>> d2 = [1 2 3; 4 1 6; 7 8 1];
```

`L2.1:` The transpose of a matrix can be obtained by:
`>> d3 = transpose(d2);` or simply `>> d3 = d2’;`

`L2.2:` The inverse of `d2` can be obtained by:

```
>> d4 = inv(d2);
```

`L2.3:` Two matrices can be multiplied by:

```
>> d5 = d2 * d3;
```

`L2.4:` Instead of matrix multiplication, if we simply want to multiply an element of one matrix with the corresponding element of another matrix, we use:

```
>> d6 = d2 .* d3;
```

Note that the `.` operator can lead to unwanted results if not used carefully. IT IS NOT TO BE CONFUSED WITH MATRIX MULTIPLICATION.

`L2.5:` All elements of the matrix can be multiplied by a scalar using:

```
>> d7 = 5 * d6;
```

`L2.6:` Two matrices can be added or subtracted using:

`>> d8 = d7 + d6;` and `>> d9 = d7 - d6;`

`L2.7:` Similar to scalar multiplication, trigonometric, exponential and logarithmic operations can be performed on matrices, i.e.,

```
>> d9 = sin(d8*pi/4);
```

This is only a small list of useful functions and operations. Refer to the documentation for more advanced functions in MATLAB!

`L2.8:` Now we will learn how to use conditional statements in MATLAB. Let us use the `b` vector defined in `L1.2`. We will try to find the number of elements in the vector that have a value greater than or equal to `30`.

```
>> counter = 0;
>> b = [10 20 30 40 50];
>> for i = 1:length(b)
     if (b(i) >= 30)
       counter = counter + 1;
     end
   end
```

Here, the variable `counter` counts the number of elements having a value greater than or equal to `30`. We use the `for` loop to iterate over all the elements of the vector. Note the function `length`, which can be used to find the number of elements in a vector. We loop over the elements using `i` and the conditional `if` statement checks if the value of `b(i)` is greater than or equal to `30`. Both the `for` loop and the `if` block need to have an `end` statement to indicate the end of the respective blocks.

Lesson 3: *Plotting data*

MATLAB can be used for different types of plotting, both 2D and 3D. Here we will go through a few of them.

`L3.1:` We will first start with creating a 2D line plot of a trigonometric function.

```
>> x = [-pi:pi/20:pi];
>> y1 = sin(x);
>> plot(x,y1,'Color','r');
>> xlabel('x');
>> ylabel('sin(x)');
>> xlim([-pi pi]);
>> ylim auto;
```

Here, `plot(x,y1,'Color','r')` is used for plotting a 2D line plot of `x` vs. `sin(x)` with a red (`'r'`) line. `xlabel` and `ylabel` are used for specifying the axis labels, while `xlim` and `ylim` are used for specifying the axis limits.

Now, try this:

```
>> hold on;
>> y2 = cos(x);
>> plot(x,y2,'Color','g');
```

`hold on;` retains the current plot and axis properties. `plot(x,y2,'Color','g');` plots x vs. y2 in green (`'g'`) color.

Finally, `>> legend('sin(x)','cos(x)');` adds the legend for the two lines to the plot.

`>> print('plot1','-depsc');` saves the plot to plot1.eps file. `'-depsc'` specifies the format (.eps). This may be replaced with `'-dpng'` to save to a .png file. We just created our first plot in MATLAB!

`L3.2:` Now we will create a 3D line plot of a helix.

The equation of a helix is given as: $x(t) = cos(t)$; $y(t) = sin(t)$; $z(t) = t$. We can implement this in MATLAB as:

```
>> hold off;
>> clear all;
>> t = [0:pi/10:5*pi];
>> x = cos(t);
>> y = sin(t);
>> z = t;
>> plot3(x,y,z);
>> xlabel('cos(t)');
>> ylabel('sin(t)');
>> zlabel('t');
>> print('plot2','-depsc');
```

Here, `plot3` is the function used for creating 3D line plots.

`L3.3:` Now we will learn how to create a surface plot in MATLAB. For this example, we will plot the equation: $z = x^2 + y^2$.

```
>> hold off;
>> clear all;
>> x1 = [0:0.1:1];
>> y1 = [0:0.1:2];
>> [x,y] = meshgrid(x1,y1);
>> z = x.^2 + y.^2;
>> surf(x,y,z);
>> xlabel('x');
>> ylabel('y');
>> zlabel('x^2 + y^2');
```

Here, we use a new function `meshgrid` to create a 2D grid with the two axes, `x` and `y`, represented by the two vectors `x1` and `y1`, respectively. Note the use of the `.` operator. We then plot `z` using the `surf` function to create a surface plot. If we want a mesh plot instead of a surface plot, we can use `mesh(x,y,z)`.

Lesson 4: *Creating MATLAB functions/scripts*

All MATLAB commands written at the terminal can also be written in the form of a script and executed. This allows reusability of the code for future applications. As an example, we put together the code written in `L3.3` in a function and then execute it. We create a file called `example_plot.m`:

```
function example_plot(deltax,xmax,deltay,ymax)
% First MATLAB function
% Name:
% Roll no.:
% Define grid
x1 = [0:deltax:xmax];
y1 = [0:deltay:ymax];
[x,y] = meshgrid(x1,y1);
% Define function
z = x.^2 + y.^2;
% Plot z
surf(x,y,z);
% Label axes
xlabel('x');
ylabel('y');
zlabel('x^2 + y^2');
```

The name of the function is `example_plot` and we pass the grid parameters `deltax`, `xmax`, `deltay` and `ymax` to the function. Also note that any line or statement that starts with a `%` is considered as a comment in MATLAB. It is not executed during run time.

We execute this code from the MATLAB Command Window by typing:

```
>> hold off;
>> clear all;
>> example_plot(0.1,1,0.1,2)
```

We just created our first function in MATLAB. Now we can play around with the grid parameters and see how this alters the surface plot!

Assignments

Assignment 1:
Given the function: $z = sin(x)cos(y)/x$. Plot z in the range: $x \in [-5\pi : 5\pi]$, $y \in [-5\pi : 5\pi]$. Use `xmin`, `xmax`, `deltax`, `ymin`, `ymax` and `deltay` as input parameters. Use `deltax` = `deltay` = $\pi/10$. Use the `surf` function in MATLAB for plotting this. Label your axes properly.
Make sure that you use the variable names given in the instructions, else your tests will not pass.

Assignment 2:
Create a MATLAB script to calculate the trajectory (height, `h`) of a projectile motion which is generally described by the equation:

$$h = ut - \frac{1}{2}gt^2$$

where, `u` = 30 m/s is the initial velocity, g = 9.8 m/s^2 is the acceleration due to gravity, and `t` is the time in seconds. Calculate the maximum height (`hmax`) and the time (`tmax`) at which the projectile hits the ground again (Do not use the derivative of the above equation for solving). Use `deltat` = 0.1 s and use a maximum time of 15 s for your calculations. Also plot the trajectory of the projectile with time on the x-axis. (Hint: Compare the height at one instant of time with that the next time step. Use for loop and if statement to accomplish this.) You will be evaluated for both the MATLAB functions and the plots. Label your axes properly.

For the variables stated in these questions, use the same variable names as indicated in the `highlighted text`.

These assignments will be automatically graded by MATLAB grader. If you do not use the variable names and functions given here, your tests will fail and you will end up losing marks.