Getting Started with MATLAB

Table of Contents

Document compiled by Axel Brandt and Devon Sigler for the CU Denver Graduate Student Seminar

Overview

MATLAB, short for MATrix LABoratory, is designed to efficiently execute matrix computations. Among the many advantages of MATLAB include the MATLAB debugger, extensive help documentation, plotting and visualization, extensive libraries of predefined functions, and compatability.

The CU Denver mathematics department maintains a license that allows graduate students to install MATLAB. Visit the CCM website (http://ccm.ucdenver.edu/sysdocs/how_to_get_and_use_matlab.html) for installation instructions.

To benefit from this starter guide, the reader should execute the included code. For a greater benefit, the reader should alter the included code and explore coding on their own. For the greatest benefit, the reader should access documentation for the MATLAB functions included below. This documentation is available online and by typing `help` followed by the function name into the Command Window.

Basic Computations

In the most basic of operations, MATLAB functions as a calculator. For example, the expression $2(3+5)-\left(\frac{12}{5}\right)^4$ is evaluated and stored as the variable *x* by the following code:

```
x = 2*(3+5)-(12/5)^4
```

Once variables are defined, they are stored in the workspace and their value can be used by using the variable name. For instance,

```
y = x-1
```

Outside of the basic operations, MATLAB has preloaded functions to evaluate the absolute value, square root, exponentials, and logarithms.

```
y = abs(x)
```

```
y = sqrt(x)
```

```
y = exp(x)
```

```
y = log(y)
```

Note that the last statement 'updates' the value of the variable *y* by operating on its stored value. MATLAB also recognizes some constants such as pi and *i*.

```
y = pi + i
```

Many preloaded MATLAB functions take arrays as inputs. To define a vector/array, elements are specified across rows delimited by either a space or comma, and the rows are delimited by a semicolon all within square brackets.

```
m = [1,2,3;4 5 6]
```

Vectors can also be defined using the colon operator to specify the elements of a vector via start:increment:end. The increment defaults to 1 if it is omitted.

```
v = 1:2:10
```

To automatically generate evenly spaced data values in a vector, the linspace function can be used: linspace(start,end,number_of_values). The default number of values is 100 should the function argument be omitted.

```
v = linspace(0,1,5)
```

MATLAB can also generate commonly used arrays.

```
m = zeros(2)
```

```
m = ones(2,3)
```

```
m = eye(3,2)
```

```
m = diag(v)
```

```
m = rand(2,3)
```

There are functions that generate random data according to specific distributions (normal, sparse, integers, etc)

Arrays can be edited by redefining specific elements. Note that arrays are indexed starting with 1, opposed to 0 which is used in numerous other programming languages.

```
m(1,2) = i
```

Subarrays can also be accessed and edited if desired. To specify an entire row or column, use the colon operator.

```
m(2,:)
```

The `sortrows()` function sorts the rows of an array. The `find()` function identifies the nonzero elements of an array; this is commonly used in combination with logical operations on an array. The transpose and conjugate transpose of an array can be obtained as follows.

```
m.'
```

```
m'
```

Ending lines of code with a semicolon suppresses the output of that line.

Plotting Data

In newer versions of MATLAB, data can be plotted simply by selecting the arrays containing the data and then selecting the desired plot from the 'plots' tab. Note that the order in which variables are selected from

the workspace is the order they appear as arguments in the automatically generated function call. In lieu of this feature, we cover commands useful for more advanced plotting.

Plots appear in figure windows. The `hold` command allows for multiple plots to appear on the same axes. For example:

```
hold on

plot(v)

plot(v2)

hold off
plot(v+v2)
```

Notice that the third plot replaced the first set of axes. Use the `figure()` function before plotting to specify the figure in which the plot appears.

Once a figure window is open, the 'Show Plot Tools and Dock Figure' button in the figure toolbar allows for customizing the plot. Common options are colors, titles, point shapes, names. Customizing these features is possible through commands but are omitted in this starter guide. One command we will mention is `legend()`, which adds a legend to the graph.

To organize multiple axes in the same figure window, use the `subplot()` command. `subplot(m,n,p)` divides the figure into an `mxn` grid and labels positions left to right, top to bottom. The vector `p` specifies a rectangular area within the grid for a set of axes.

```
figure(1)
subplot(3,3,1)
plot(v)
title('Subplot 1')

subplot(3,3,2:3)
plot(v)
title('Subplot 2')

subplot(3,3,[4,7])
plot(v)
title('Subplot 3')

subplot(3,3,9)
plot(v)
title('Subplot 4')
```

The single quotations as input for `title()` denote character strings rather than variables or functions. To generate values for 3D plots, the combination of `linspace()` and `meshgrid()` is useful. For example:

```
x = linspace(-2,2,20);
y = linspace(-2,2,20);
[X,Y] = meshgrid(x,y);
Z = X .* exp(-X.^2 - Y.^2);
subplot(1,2,1)
mesh(X,Y,Z)
subplot(1,2,2)
surf(X,Y,Z)
```

Plotting a graph (the discrete structure) is possible using `gplot()` and the desired adjacency matrix of a graph as input.

Programming

Although programming is possible in the Command Window, it is often more useful to save code for future use as a script or function.

When writing scripts, sections of code can be distinguished using %% and then executed individually within the script. Interface with the user is possible using the `input()` command, where a character string as input into `input()` appears in the Command Window as a prompt to the user.

While scripts access and use the Workspace for variable storage, functions create their own (separate) workspace during execution, a feature that helps avoid confusion when accessing variables. With the capability for inputs and outputs, functions can be applied in numerous projects without concern to matching variable names. While both scripts and functions are saved in `.m` files, functions are specified by the following code:

```
function [ output_args ] = my_fxn( input_args )
%MY_FXN Summary of this function goes here
%   Detailed explanation goes here

    _code_
end
```

Notice that the use of a single % comments that line of text. Ample comments describing how to use and what is occuring in your code is a great programming practice! Functions can also be defined in a single line as an Anonymous Function and referred to using Function Handles (see documentation for more details).

Logical operators return either 1 or 0 for true or false, respectively. The 'and', 'or', and 'not' operators are denoted in the following lines of code:

```
0 & 1
1 | 0
~1
```

Doubling the 'and' and 'or' operators allows for short-circuiting, in which testing the logical operator stops once the outcome is known. The operators >,|<|,|>=|,|<=|,|==|,|~=| represent tests for greater than, less than, greater than or equal to, less than or equal to, equal, and not equal, respectively. Note that a single = assigns values to variables rather than testing for equality.

Branching statement syntax appears below.

```
if logical_expression_1
    _code_
elseif logical_expression_2
    _code_
    ...
else
    _code_
end

switch variable
    case _first_
        _code_
```

```
    case _second_
        _code_
        ...
    otherwise
        _code_
end
```

Overloading functions (allowing multiple uses) is possible using branching statements and other logical or numerical values to determine the number and type of variables given as inputs when the function was called. For example, `nargin` and `nargout` return the number of input variables and output variables, respectively, in the function call.

Examples of overloaded functions include `plot()` and `nchoosek()`. When `plot()` is called with one vector of values, the values are plotted as the *y* coordinate with their index as the *x* coordinate. However, we have seen `plot()` called with two vectors (of equal size) to specify specific `(x,y)` coordinates. When `nchoosek()` is called with two scalar inputs, it returns the binomial coefficient. However when it is called with a vector input `v` followed by a scalar input `c`, it returns a matrix containing all possible combinations of `v` of `c` elements.

Syntax or loops appears below.

```
for _variable_ = _range_
    _code_
end

while logical_expression
    _code_
end
```

Miscellaneous

As you explore MATLAB and are trying to find MATLAB functions that may be helpful, the `lookfor` command followed by a search keyword may help you. Once you become more familiar with MATLAB functions but may not remember the order for inputs, the `help` command followed by the function name will display the commented description in that function's `.m` file header. This works with user defined functions as well.

If you are in the middle of calculations and (for whatever reason) need to leave, the `save` command will save your current workspace, both variables and values, which can be loaded when you come back.

The function `latex()` will generate LaTeX representaiton of a symbolic expression.

When working in the Command Window, the command `clc` will clear the text that currently appears in the Command Window, and the command `clear` will clear the variables in the Workspace.

Scripts and functions can display formatted text to the Command Window using `fprintf()`.

The newer versions of MATLAB include a publishing feature, which can generate a .pdf file of a formatted `.m` file. This `.pdf` was created this way!

Published with MATLAB® R2014b