Notes for Matlab Review

Multimedia and Video Communications (SSY150)

Prepared by: Yixiao Yun, yixiao@chalmers.se
Department of Signals and Systems,
Chalmers University of Technology,
41296, Gothenburg, Sweden

March 22, 2016

1 Introduction

This note covers basic commands and useful functions in MATLAB for SSY150 Multimedia and Video Communications. It is meant to serve as a quick way to get familiar with MATLAB and a quick reference to the commands that are used in this course. For more detailed information, you are recommended to consult the official MATLAB documentation.

The student version of MATLAB is available for download from Chalmers software server (`http://studentfile.portal.chalmers.se/library/Matlab/`). The page is only accessible from computers connected to Chalmers network, so you may need to use VPN client to enable off-campus access.

2 Basic Operations

A. Definition of Variables

Variables are assigned numerical values by typing the expression directly. For example, typing

```
a = 1+2
```

yields:

```
a =
     3
```

The answer will not be displayed when a semicolon is put at the end of an expression. For example, type

```
a = 1+2;
```

MATLAB utilizes the following arithmetic operators:

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
∧	power operator
`'`	conjugate transpose
`.'`	non-conjugate transpose

A variable can be assigned using a formula that utilizes these operators with either numbers or previously defined variables. For example, since `a` has been defined previously, the following expression is valid

```
b = 2*a;
```

To determine the value of a previously defined quantity, type the quantity by itself:

```
b
```

yields:

```
b =
    6
```

If your expression does not fit on one line, use an ellipsis (three or more periods at the end of the line) and continue on the next line.

```
c = 1+2+3+...
5+6+7;
```

There are several predefined variables which can be used at any time, in the same manner as user-defined variables:

`i`	sqrt(-1)
`j`	sqrt(-1)
`pi`	3.1416...

For example,

```
y = 2*(1+4*j)
```

yields:

```
y =
   2.0000 + 8.0000i
```

There are also a number of predefined functions that can be used when defining a variable. Some functions that are commonly used are:

`abs`	magnitude of a number (absolute value for real numbers)
`angle`	angle of a complex number, in radians
`cos`	cosine function, assumes argument is in radians
`sin`	sine function, assumes argument is in radians
`exp`	exponential function

For example, with `y` defined as above,

```
c = abs(y)
```

yields:

```
c =
    8.2462
```

```
c = angle(y)
```

yields:

```
c =
    1.3258
```

With `a = 3` as defined previously,

```
c = cos(a)
```

yields:

```
c =
   -0.9900
```

```
c = exp(a)
```

yields:

```
c =
   20.0855
```

Note that `exp` can be used on complex numbers. For example, with `y = 2+8i` as defined above,

```
c = exp(y)
```

yields:

```
c =
  -1.0751 + 7.3104i
```

which can be verified by using Euler's formula:

$$c = e^2\cos(8) + je^2\sin(8)$$

B. Definition of Vectors and Matrices

MATLAB is based on matrix and vector algebra, where even scalars are treated as 1×1 matrices. Vectors can be defined in two ways. The first method is used for arbitrary elements:

```
v = [1 3 5 7];
```

creates a 1×4 vector with elements 1, 3, 5 and 7. Note that commas could have been used in place of spaces to separate the elements. Additional elements can be added to vector:

```
v(5) = 8;
```

yields the vectors `v = [1 3 5 7 8]`. Previously defined vectors can be used to define a new vector. For example, with `v` defined above

```
a = [9 10];
b = [v a];
```

creates the vector `b = [1 3 5 7 8 9 10]`.

The second method is used for creating vectors with equally spaced elements:

```
t = 0:.1:10;
```

creates a 1×101 vector with the elements 0, .1, .2, .3, ..., 10. Note that the middle number defines the increment or decrement. An example of creating vector with decreasing elements is

```
t = 10:-1:0;
```

which generates a 1×11 vector with the elements 10, 9, 8, ..., 0.

If only two numbers are given, then the increment is set to a default of 1:

```
t = 0:10;
```

creates a 1×11 vector with the elements 0, 1, 2, ..., 10.

Matrices are defined by entering the elements row by row:

```
M = [1 2 4; 3 6 8];
```

creates the matrix

$$\mathtt{M} = \begin{bmatrix} 1 & 2 & 4 \\ 3 & 6 & 8 \end{bmatrix}$$

where the semicolon in between square brackets is used to separate rows.

There are a number of special matrices that can be defined:

null matrix:	`M = [];`
`m`×`n` matrix of zeros:	`M = zeros(m,n);`
`m`×`n` matrix of ones:	`M = ones(m,n);`
`n`×`n` identity matrix:	`M = eye(n);`

A particular element of a matrix can be assigned:

```
M(1,2) = 5;
```

places the number 5 in the first row, second column.

Operations and functions that are defined for scalars in the previous section can also be used on vectors and matrices. For example,

```
a = [1 2 3];
b = [4 5 6];
c = a + b
```

yields:

```
c =
    5 7 9
```

Functions are applied element by element. For example,

```
t = 0:10;
x = cos(2*t);
```

creates a vector `x` with elements equal to `cos(2t)` for `t` = 0, 1, 2, ..., 10.

Operations that need to be performed element-by-element can be accomplished by preceding the operation by a ".". For example, to obtain a vector `x` that contains the elements of `x(t) = t·cos(t)` at specific points in time, you cannot simply multiply the vector `t` with the vector `cos(t)`. Instead you multiply their elements together:

```
t = 0:10;
x = t.*cos(t);
```

C. General Information

MATLAB is case sensitive so "`a`" and "`A`" are two different names.

Comment statements are preceded by a "`%`". For example,

```
M = eye(3); % to create a 3×3 identity matrix
```

On-line help for MATLAB can be reached by typing `help` for the full menu or typing `help` followed by a particular function name or M-file name. For example, `help cos` gives help on the cosine function.

The number of digits displayed is not related to the accuracy. To change the format of the display, type `format short e` for scientific notation with 5 decimal places, `format long e` for scientific notation with 15 significant decimal places and `format bank` for placing two significant digits to the right of the decimal.

The commands `who` and `whos` give the names of the variables that have been defined in the workspace.

The command `length(x)` returns the length of a vector `x` and `size(x)` returns the dimension of the matrix `x`.

The command `clc` clears the command window and `clear` removes items from workspace, freeing up system memory.

After using MATLAB, you may wish to leave the program but save the vectors and matrices you have defined in workspace. To save the data file to current working directory, type

```
save filename
```

where "`filename`" is a name of your choice. To retrieve the saved data, type

```
load filename
```

3 Scripts

A. M-files and Functions

M-files are macros of MATLAB commands that are stored as ordinary text files with the extension "m", that is *filename*.m. An M-file can be either a function with input and output variables or it can be a list of commands. All laboratory assignments of this course are supposed to be written in M-files.

MATLAB requires that the M-file must be stored in a directory that is specified in the MATLAB path list. For example, to access a user-defined M-file stored in a directory called "`\MATLAB\MFILES`", the directory should be added to the path by clicking "File" then clicking "Set Path" and following directions.

To create an M-file, click on "File" then click on "New" then click on "M-file" and an editor window will appear. Type in your MATLAB commands and then save the file to a directory that is in the path. The M-file is executed by typing the name of the M-file from the MATLAB command window (without the .m extension). For example, suppose an M-file named `example.m` is located in the path. Then typing `example` from the command prompt runs that M-file.

As example of an M-file that defines a function, create a file in your working directory named `yplusx.m` that contains the following commands:

```
function z = xplusy(x,y)
z = x + y;
```

The following commands typed from within MATLAB demonstrate how this M-file is used:

```
x = 2;
y = 3;
z = xplusy(x,y)
```

B. Flow Control Statements

Loops and `if` statements are available in MATLAB, but should be used sparingly since they are computationally inefficient. An example of the use of the command `for` is

```
for k = 1:10
  x(k) = cos(k);
end
```

An alternative way is to use the command `while`:

```
k = 1;
while (k <= 10)
  x(k) = cos(k);
  k = k + 1;
end
```

Both creates a 1×10 vector `x` containing the cosine of the positive integers from 1 to 10. This operation is performed more efficiently with the commands

```
k = 1:10
x = cos(k);
```

which utilizes a function of a vector instead of a `for` or `while` loop.

The `if` statement can be used to define conditional statements. An example is

```
if (a <= 2)
  b = 1;
elseif (a >= 4)
  b = 2;
else
  b = 3;
end
```

The allowable comparisons between expressions are `>=` (greater or equal), `<=` (less or equal), `<` (less than), `>` (greater than), `==` (equal), and `~=` (not equal).

Conditions can be combined by using logical operators, which are `&&` (and), `||` (or), ~ (not), and `&` (and), `|` (or) for element-wise operation. For example,

```
if (a >= 2) && (a <= 4)
  b = 3;
end
```

4 Plotting

A. Plotting of 1D Data

The command most often used for 1D plotting is `plot`, which creates linear plots of vectors and matrices; `plot(t,y)` plots the vector `t` on the x-axis versus vector `y` on the y-axis. There are options on the line type and the color of the plot which are obtained using `plot(t,y,'option')`. The line type options are `'-'` solid line (default), `'--'` dashed line, `'-.'` dot dash line, `':'` dotted line. The points in `y` can be left unconnected and delineated by a variety of symbols: `'+' '.' '*' 'o' 'x'`. The following lists some available color options:

`'r'`	red
`'b'`	blue
`'g'`	green
`'w'`	white
`'k'`	black

For example, `plot(t,y,'--')` uses a dashed line, `plot(t,y,'*')` uses `*` at all the points defined in `t` and `y` without connecting the points, and `plot(t,y,'g')` uses a solid green line. The options can also be used together. For example, `plot(t,y,'g:')` plots a dotted green line.

To plot two or more graphs on the same set of axes, use the command `plot(t1,y1,t2,y2)`, which plots `y1` versus `t1`, and `y2` versus `t2`.

To label your axes and give the plot a title, type

```
xlabel('Time')
ylabel('Magnitude')
title('My Plot')
```

Finally, add a grid to your plot to make it easier to read. Type

```
grid
```

The problem that you will encounter most often in plotting is that MATLAB scales the axes in a way that is different from how you want them to appear. You can easily override the auto-scaling of the axes by using the `axis` command after plotting:

```
axis([xmin xmax ymin ymax]);
```

where `xmin`, `xmax`, `ymin`, and `ymax` are numbers corresponding to the limits you desire for the axes. To return to the automatic scaling, simply type `axis`.

For discrete-time signals, use the command `stem` which plots each point with a small open circle and a straight line. To plot `y[k]` versus `k`, type

```
stem(k,y)
```

To plot more than one graph in a figure window, use the command `subplot(m n p)` which partitions the window into an `m`×`n` grid where `p` determines the position of the particular graph counting the upper left corner as `p = 1`. For example,

```
subplot(2 1 1); plot(t1,y1);
subplot(2 1 2); plot(t2,y2);
```

plots `y1` versus `t1` on top, and `y2` versus `t2` below. Titles and labels can be inserted immediately after each plot. To return to a full window plot, type `subplot(1 1 1)`.

B. Visualization of 2D Images

When you encounter an image you want to work with, it is usually in form of a file. For example, if you download an image from the web, it is usually stored as a JPEG file. To read and load a 2D image in MATLAB workspace, the `imread` command is used.

For example, suppose an image file `your_image.jpg` has been stored in the path, typing

```
I = imread('your_image.jpg');
```

loads the image data to workspace and store it in matrix `I`. Make sure to use semicolon, otherwise you will get lots of number scrolling in your command window.

For displaying the image represented as the matrix `I`, the command `imshow` is recommended. Simply type

```
imshow(I)
```

and the image will show in a figure window.

When you are about to process an image (that is performing mathematical operations on an image), you should convert it into `double` precision. Typing

```
I = im2double(I);
```

converts an image named `I` from `uint8` to `double`.

Sometimes you may need to convert an RGB color image to grayscale. This can be done by typing

```
I = rgb2gray(I);
```

To make sure that the range of pixel values is between 0.0 and 1.0, we can run the following command

```
I = mat2gray(I);
```

Before you store a processed image, you are recommended to format it as a `uint8` image since this requires far less memory than `double`. Then typing

```
I = im2uint8(I);
```

converts an image named `I` from `double` to `uint8`.

Once we finish processing an image, we may want to write it back to e.g. a JPEG file. This is done by using the `imwrite` command. Type

```
imwrite(I,filename);
```

where the first argument `I` is the name of the image matrix you have worked with, and the second argument `filename` is the name of the file and format that you want to write the image to. For example,

```
imwrite(I,'my_image.jpg');
```

5 Exercises

Exercise 1 Find a short MATLAB expression to build the matrix

$$\mathtt{B} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 \\ 9 & 7 & 5 & 3 & 1 & -1 & -3 \\ 4 & 8 & 16 & 32 & 64 & 128 & 256 \end{bmatrix}$$

Exercise 2 Use MATLAB to plot the following signals.

(a) `x(t) = 3cos(4t) + sin(`π`t)`

(b) `x(t)` $= \begin{cases} 0 & \text{if } \mathtt{t<-4} \\ \mathtt{t+2} & \text{if } \mathtt{-4} \leq \mathtt{t<3} \\ \mathtt{t-2} & \text{if } \mathtt{3} \leq \mathtt{t} \end{cases}$

Scale your time axis so that a sufficient amount of the signal is being plotted. Use `subplot` to give the 2 plots. Label your plots with 'Time (sec)' on the x-axis, and '`x(t)`' on the y-axis. The title should be the problem number, e.g. '(a)'.

Exercise 3 Generate a set of random numbers using the `rand` command, then compute the sum, the mean, and the standard deviation of the data set using `for` or `while` loop. Also verify the result of your loop by applying the `sum`, the `mean`, and the `std` functions.

Exercise 4 Read 4 different color images and show them in grayscale in one figure window using `subplot` function.

6 References

[1] Edward Kamen, Bonnie Heck; MATLAB tutorial as a supplement to the textbook "Fundamentals of Signals and Systems Using the Web and Matlab", 3rd edition, Prentice Hall.

[2] Image Processing Toolbox$^{\text{TM}}$: User's Guide, The MathWorks Inc.