Digital Signal Processing

Digital audio processing tutorial 1

### Introduce to MATLAB programming for Audio

By Yonghao Wang

Centre of Digital Media Technology

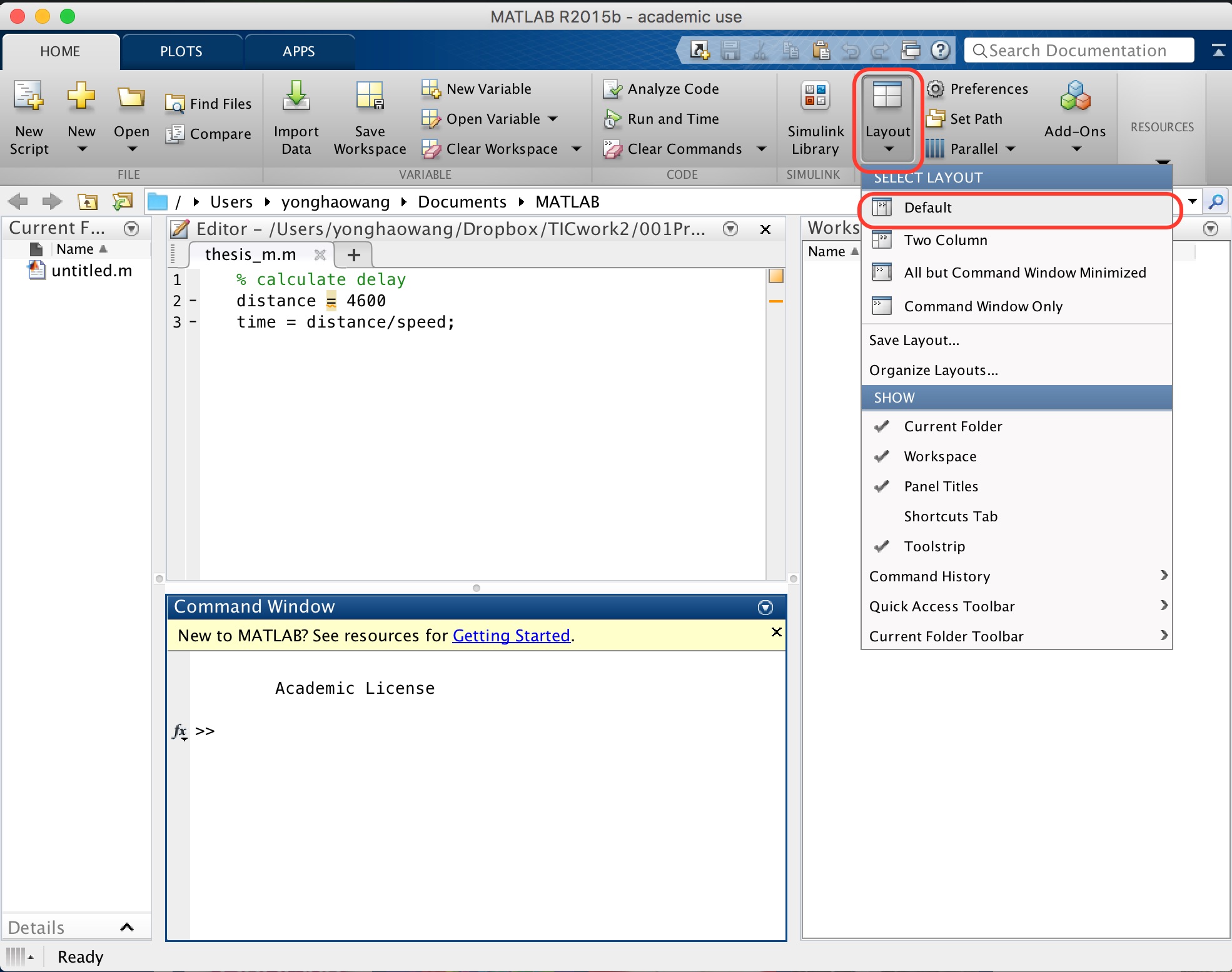


Note:

In this tutorial, text in “Courier New” fixed size fonts are the commands you can directly input into Matlab environment for execution. The text in *“Italic”* are the example of syntax and normally need to be understood and modified to be used in execution.

1. **MATLAB Interface layout**

Matlab interface is extremely flexible and configurable. It is easy to lose the track. You can always turn it back to default layout by following the click (Desktop -> Desktop Layout -> Default) as shown in the below diagram:



**Figure 1 reset Matlab interface to Default layout**

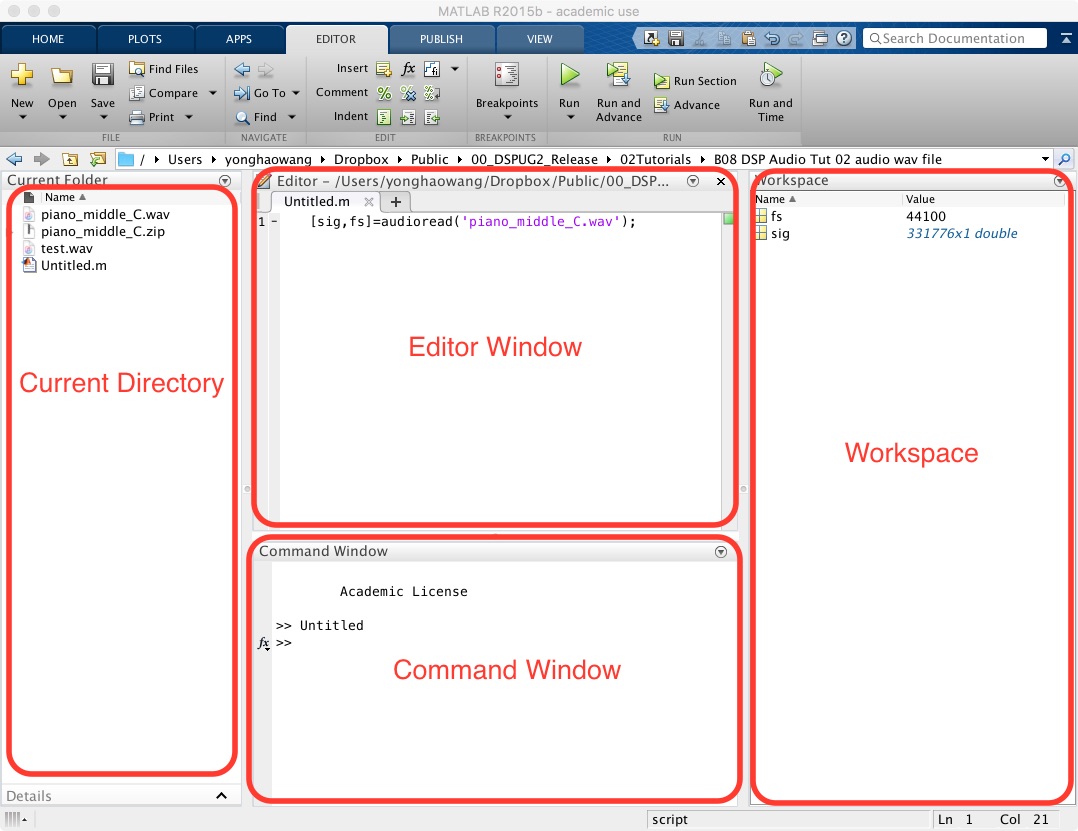
1. **MATLAB Working Areas**

As shown in following picture, the MATLAB **Command Window** is the main window where you type commands directly to the MATLAB interpreter.  The MATLAB **Editor Window** is a simple text editor where you can load, edit and save complete MATLAB programs.  The Editor window also has a menu command (Debug/Run) which allows you to submit the program to the command window.

The **Current Directory** window on left side shows which directory the Matlab is working on at that moment. The outputs and inputs files will be generated or expected at that directory.

The **Workspace** shows all the memory information of the variables as you program along. It is very useful for debugging and finding errors.

The MATLAB help system gives you access to a great deal of useful information about the MATLAB language and MATLAB computing environment.  It has a number of example programs and tutorials which is also available online.



**Figure 2 Matlab working areas**

1. **A first program**

The build-in commands: Matlab recognises a set of commands which performs certain tasks which can be simple operation or complex mathematic functions. For example, type “**clc**” will clear the command window, and type “**clear**” will clear the Workspace window as well as all the variables shown in the workspace. Now type “clc” and “clear” to clean both working areas.

**Step 1**

Type the following line in command window:

A = 3

You will get the feedback from the command window looks like the following:

A =

3

It tells you that Matlab has accepted your command which stores number three in variable “A”. Have a look into workspace, what information is available there?

**Step 2**

Type the following line in command window:

B = 5;

Note this time there no feedback information from command window. Because this time you have put semicolon at end of command, the Matlab command window shall hide the extra information. However you can still find the information of variable “B” in workspace.

**Step 3**

Type the following command:

C = A+B;

disp(C);

You will get the feedback from the command window looks like the following:

8

The command “disp” is a build-in command to display the value of a variable or argument in bracket on the command window. In this example, you put number 3 in variable “A”. Then put number 5 in variable “B”. Then you let Matlab to calculate the result of “A” + ”B” and store the result in variable “C”. Finally you display the content of “C “ on the command window.

Try

disp(2\*3+1);

or

disp(’Hello World!’);

1. **Programming concept**

Although the first program is an extremely easy one, it demonstrates all the necessary elements of a complete software code. That is

* Assign the values to variables
* Let Matlab to perform the certain operation to deal with the defined variables, for example: addition.
* Utilise the build-in command to interactive with Matlab, for example display the values, draw diagrams.

*disp(argument);* displays the value of the argument.  This can be a number, a string in single quotes, or an expression.  For simple numbers, the arithmetic operators are: **+, -, \*, /** and **^.**

Variables are named locations in memory where numbers, strings and other elements of data may be stored while the program is working.  Variable names are combinations of letters and digits, but must start with a latter.  MATLAB does not require you to declare the names of variables in advance of their use.  This is actually a common cause of error, since it allows you to refer accidentally to variables that don’t exist.  To assign a variable a value, use the **assignment statement**.  This takes the form

*variable*=*expression*;

for example

a=6;

or

name=’Mark’;

To display the contents of a variable, use

disp(*variable*);

1. **Arrays**

MATLAB is particularly powerful in the way it deals with tables of data, called **arrays**.  An array is simply a variable that can contain a number of values arranged in tabular form.  Arrays may be one dimensional (like a list), two dimensional (like a table), or have more dimensions.

The concept of Array is extremely important in digital audio processing. Normally you can regard one channel (mono) audio as one dimensional array. The two channel audio (stereo) can be regarded as two dimensional array.

For example:

table = [1 1 1 1];

Shall create an one dimensional array call “table” with 4 elements inside that array. Each element stores value 1.

To set the value of one **element** of a one dimensional array, use the notation

*variable*(*index*)=*expression*;

for example

table(1)=3;

table(2)=6;

Note that indexes must be expressions evaluating to positive integers.  The smallest index is 1.  To access one element from a one dimensional array, use the notation

*variable*(*index*)

for example

a=table(2);

disp(table(2));

For two dimensional arrays, use

*variable*(*index*,*index*)=*expression*;

to set the value and

*variable*(*index*,*index*)

to retrieve its value.  You can store strings in tables, but each string occupies a row, and all rows must be the same length (think of a two-dimensional array of characters).

You can assign a whole array in one operation using a notation involving square brackets: for example:

*array* = [ 1 1 1; 2 2 2];

This will create two dimensional array in row mode. The ‘;’ marks the end of a row.

You can generate arrays containing sequences very easily with the ‘:’ operator.

Try

disp(1:10);

disp(1:2:10);

You can also select sub-parts of the array with the ‘:’ operator.  For example,

x(3:5)

represents the array consisting of the third through fifth elements of x.  Also

y(2:2:100)

represents the array containing the even number elements of y below index 100.

The expression

*start***:***stop*

generates a sequence of integers from start to stop.  The expression

*start***:***increment***:***stop*

generates a sequence from start to stop with the specified increment.

You can also add subtract, multiply and divide arrays of data using the operators we’ve mentioned previously.  However MATLAB makes a difference between operations that work on a **cell-by-cell** basis (so-called “dot” operations) as opposed to operations that work on the arrays as a whole. For example, if you want to multiply two arrays of equal size to give a third array in which each cell contains the product of the corresponding cells in the input, then you need to use the “dot-multiply” operator **.\*** for example

C = A.\*B;

Finally, the array is directional you can **transpose** rows and columns of a matrix with the **' (apostrophe**) operator, for example

disp(A**'**)

1. **Demo code**

If you are going to write a complex code, it is better to use the “Editor” area. You can type your program in the editor window and save to “filename.m” format. To run the code, you should click the “play” button on the editor window.

The filename must end with “.m”. It must not be the build-in command. It must not contain space and cannot begin with numbers. The convention of filename is specified so that the Matlab files can be portable between different operating systems and backwards compatible.

Copy and paste the following code into editor window and save it into your working directory with a proper filename such as “gensin.m”. Note the file name cannot have space.

Amp=0.9;

f=441;

fs = 44100;

ts= 1/fs;

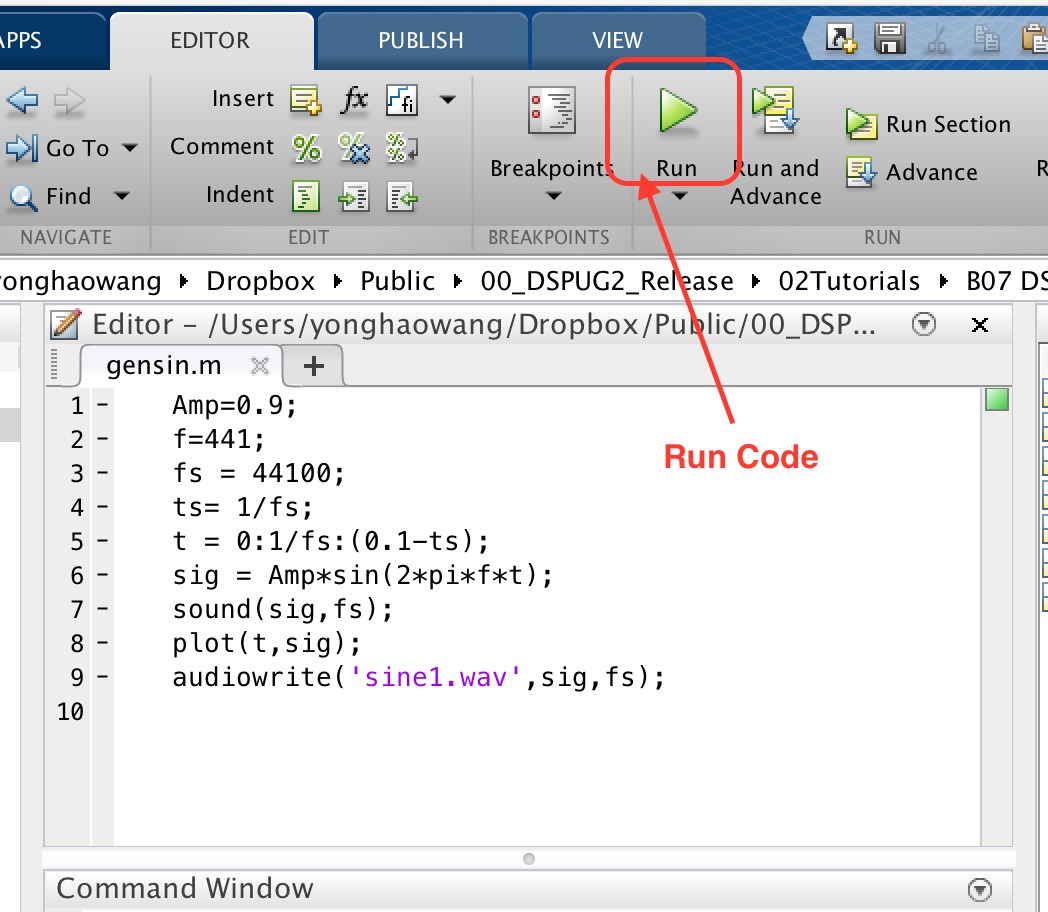
t = 0:1/fs:(0.1-ts);

sig = Amp\*sin(2\*pi\*f\*t);

sound(sig,fs);

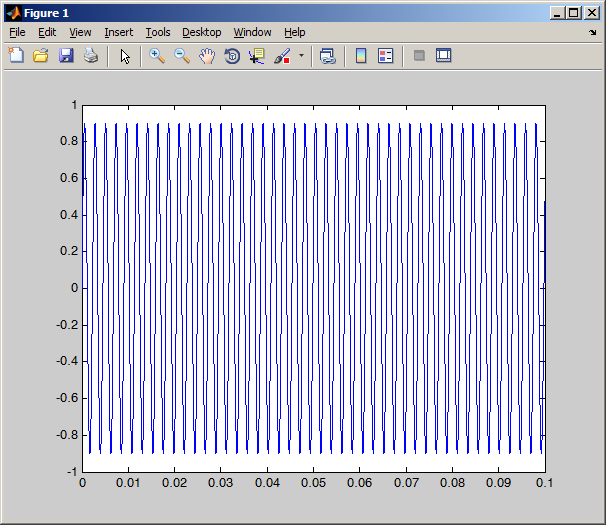
plot(t,sig);

audiowrite('sine1.wav',sig,fs);



**Figure 3 Editor window and Run button**

Save the code in your current directory. Run the code and you should get the plot of sine wave and an audio file “sine1.wav” in the current directory. This demo code demonstrates how to use Matlab to generate the audio signal and output it into the audio file. The plotting should look like the following diagram:



**Figure 4 Sine Wave generated by Matlab**

The full details of how this code works will be discussed in the future Labs. At this moment let us concentrate how to plot the diagram:

1. **Basic Plotting**

To create XY graphs, it is easiest to form your data into two row vectors, one for the x co-ordinates, and one for the y co-ordinates.  The command

plot(x,y)

If signals are discrete typically for digital signals, the command to use is

stem(x, y)

will then create a figure with points at each y value for each matching x value.  You can control the style of any line drawn through the points by a third string argument to the plot command:

plot(x,y,style);

where style is made up from characters as follows:

* Color strings are 'c', 'm', 'y', 'r', 'g', 'b', 'w', and 'k'. These correspond to cyan, magenta, yellow, red, green, blue, white, and black.
* Linestyle strings are '-' for solid, '--' for dashed, ':' for dotted, '-.' for dash-dot, and none for no line.
* The marker types are '+', 'o', '\*', and 'x' and the filled marker types 's' for square, 'd' for diamond, '^' for up triangle, 'v' for down triangle, '>' for right triangle, '<' for left triangle, 'p' for pentagram, 'h' for hexagram, and none for no marker.

For example:

x = [ 1 2 3 4 ];

y = [ 10 15 20 25 ];

figure(1);

plot(x,y,'g-\*');

figure(2);

stem(x,y,'g-\*');

You can give the graph a title with the

title(*‘plot of xy’*);

command, where label is a character string.  Likewise you can add labels to the X and Y axes with

xlabel(*‘index of x’*);

and

ylabel(*‘value of y’*);

You can add a legend with

legend(*label1*,*label2*,*label3*,…);

1. **Built in functions**

Generation

|  |  |
| --- | --- |
| zeros() | matrix of specified size filled with zeros |
| ones() | matrix of specified size filled with ones |
| rand() | generate pseudo random number(s) between 0 and 1 |

Arithmetic

|  |  |
| --- | --- |
| rem() | remainder after integer division |
| abs() | absolute value (also character -> number) |
| fix() | truncate a value to its integer part (towards zero) |
| round() | round a value to nearest integer. |
| sqrt() | square root |
| sin() | sine (angle in radians) |
| cos() | cosine (angle in radians) |
| exp() | exponential |
| log() | natural logarithm |
| log10() | logarithm base 10 |

Status

|  |  |
| --- | --- |
| length() | length of a vector (longest dimension of matrix) |
| size() | size of a matrix [nrows, ncols] |

Miscellaneous

|  |  |
| --- | --- |
| sum() | sum the elements of a vector |
| mean() | find mean of elements of a vector |
| sort() | sort the elements of a vector in increasing size |
| clock() | returns date and time as a vector [year month day hour minute seconds] |
| date() | returns date as a string dd-mmm-yyyy |

1. **Generating Waveforms**

Waveforms are just long vectors with one number per amplitude sample. Usually they are best kept scaled so that each amplitude is between –1 and 1.  To generate a sinewave, first generate a time sequence t representing the times of each sampling instant; for example:

t = 0:0.0001:2;

would generate a two second sequence with a sampling interval of 0.1ms (i.e. 10,000Hz).  You can then generate a sinewave at frequency F with the following expression, in this case F is 10Hz:

F = 10;

y = sin(2\*pi\*F\*t);

plot(t, y)

1. **Reading**

"Getting Started: Expressions" in MATLAB Help.

“Introduction to Computer Programming with MATLAB [Online]”

<http://www.phon.ucl.ac.uk/courses/spsci/matlab/>

1. **Exercises**

For these exercises, use the editor window to enter your code, and save your answers to files under your account on the central server.  To run the programs, use the Debug/Run menu option or cut and paste the code into the command window.  When you save the files, give them the file extension of ".m". Solution of Question 1) is given below. Solutions will be available at end of tutorial session.

1. Write a program to assign the following expressions to a variable A and then to print out the value of A.

            a)         http://www.phon.ucl.ac.uk/courses/spsci/matlab/lect2_files/image001.gif

            b)        http://www.phon.ucl.ac.uk/courses/spsci/matlab/lect2_files/image002.gif

            c)       http://www.phon.ucl.ac.uk/courses/spsci/matlab/lect2_files/image003.gif

            d)       (0.0000123 + 5.67×10-3) × 0.4567×10-4

% ex11.m solution

A = (3+4)/(5+6);

disp(A);

A=2\*pi\*pi;

disp(A);

A=sqrt(2);

disp(A);

A=(0.0000123+5.67E-3)\*0.4567E-4;

disp(A);

1. Celsius temperatures can be converted to Fahrenheit by multiplying by 9, dividing by 5, and adding 32.  Assign a variable called C the value 37, and implement this formula to assign a variable F the Fahrenheit equivalent of 37 Celsius.

% ex22.m solution

C=37;

F=9\*C/5+32;

disp(F);

1. Modify the demo code in gensin.m to plot sine wave with X,Y labels and give the figure a meaningful title. The output is as below:



1. Create the following periodic signals with sampling frequency 10 KHz and plot 500 points of each of them. Signal 1 with 20Hz; signal 2 with 200Hz and Signal 3 with 1000Hz. Signals shall last 1 second and with amplitude 0.5. Using *subplot()* command to create nice plots in one diagram. The output can look like below figure:

