Introduction

The purpose of this tutorial is to provide an introduction to MATLAB, a powerful software package that performs numeric computations. MATLAB is extremely useful for generating and analyzing sound files and it will be used throughout the course, so an investment of time now to become familiar with MATLAB will pay dividends throughout the semester. As you follow this tutorial you should try all of the examples. If you are unfamiliar with matrices don't worry about that part of this tutorial, in sound analysis we usually deal with simple one-dimensional arrays. The best way to learn MATLAB is through examples - so let's get started.

Using the Command Window

At startup MATLAB opens several windows. You can customize the space and decide which windows to keep open. However, you will mostly use

- Editor window - to write m-files;
- Command window – to type a command wait for the prompt sign “>>”.

You can start MATLAB by double clicking on the MATLAB icon that should be on the desktop of your computer. This brings up the Command Window. In the Command Window the user can enter simple commands one at a time and execute them as they are entered.

For example, type (we will use bold for the things that you should type)

2+2

and then press the <enter key>. You should see

ans = 4

This is the simplest way to use MATLAB, simply as a calculator. I'm not going to keep telling you to press the <enter> key after typing each command, you probably have caught on to that by now.

You can also introduce variables and assign values to them, for example type

a = 2

b = 3

Now you have two variables, a and b, in the “workspace” and you can use them in further calculations. For example type

a + b

you should see

ans = 5

MATLAB also knows a lot of functions and constants, such as π (pi) and sin among many others, for example you can type

fun = sin(pi/4) and MATLAB returns

fun = 0.7071

Note that the results of these computations are saved as variables with the names chosen by the user. If a variable is needed during your current MATLAB session, then you can obtain its
value by typing the variable name and pressing the Enter key. For example, at any time you can type

    fun

and MATLAB will return

    fun = 0.7071

This variable will appear in the Workspace, another one of the Windows that should have opened when you started MATLAB, until you clear it. You can clear any variable by typing clear followed by the variable name. For example

    clear fun

will clear the value stored for fun. You can clear all of the variables by typing simply

    clear

Variable names always have to begin with a letter, followed by more letters, numbers or underscores (no spaces). MATLAB recognizes only the first 31 characters of a variable name so don’t get too carried away making long names.

As you have just seen, Matlab expressions typed by the user are interpreted and immediately evaluated by the MATLAB program. If a MATLAB statement ends with a semicolon, MATLAB evaluates the statement but suppresses the display of the results.

So if you were to type

    fun = sin(pi/4);

Matlab will just return the prompt symbol ‘>>’, however fun is stored in memory with the value 0.7071.

**Arrays and Matrices**

Matlab is optimized to work with arrays and matrices. If you are unfamiliar with matrices to not worry, we’ll make the rules for manipulating them clear as we go along. Arrays (or matrices) are just blocks of numbers that can be manipulated according to specific rules.

A matrix may be entered as follows:

    A = [1 2 3; 2 3 4; 3 4 5]

Upon pressing <enter>, MATLAB will return

    A =
         1   2   3
         2   3   4
         3   4   5

Note that the matrix entries are enclosed by square brackets [ ] with the elements in a row separated by spaces or by commas. The end of each row, with the exception of the last one, is indicated by a semicolon. The matrix A can also be entered across three input lines as
A = [ 1 2 3
 2 3 4
 3 4 5];

In this case, the carriage returns replace the semicolons. A row vector B with five elements can be entered as

B = [6 9 12 15 18]

or if you like,

B = [6,9,12,15,18]

For readability, it is better to use spaces rather than commas between the elements. The row vector B can be turned into a column vector by transposition, which is obtained by typing a single quote (’ ) after the variable, such as

C = B’

The above results in

C =
 6
 9
 12
 15
 18

Other ways of entering the column vector C are

C = [6
 9
 12
 15
 18]

or

C = [6;9;12;15;18]

MATLAB is case sensitive in naming variables, commands and functions. Thus b and B are not the same variable. If you do not want MATLAB to be case sensitive, you can use the command casesen off. To obtain the size of a specific variable, type size(). For example, to find the size of matrix A, you can execute the following command:

size(A)

will return

ans = 3 3

The result will be a row vector with two entries. The first is the number of rows in A, the second the number of columns in A.
HELP!!!

One of the nicer features of MATLAB is its help system. To learn more about a function you would like to use, say ‘soundsc’ a command we will use a lot in this course, type in the Command Window

```
help soundsc
```

and you will see this.

SOUNDSC Autoscale and play vector as sound.
SOUNDSC(Y,...) is the same as SOUND(Y,...) except the data is scaled so that the sound is played as loud as possible without clipping. The mean of the data is removed.

SOUNDSC(Y,...,SLIM) where SLIM = [SLOW SHIGH] linearly scales values in Y in the range [SLOW, SHIGH] to [-1, 1]. Values outside this range are not clipped. By default, SLIM is [MIN(Y) MAX(Y)].

See also sound, wavplay, wavrecord.

Reference page in Help browser
doc soundsc

If you do not remember the exact name of a function you want to learn more about use the command ‘lookfor’ followed by the incomplete name of a function in the Command window. For example if you type

```
lookfor soun
```

after a while (it takes MATLAB some time to complete the search) you will receive a list of commands that contain anything to do with the string ‘soun’; the commands sound, soundsc and a lot of other things! This is a nice to explore MATLAB, if you think that it would be nice if a certain function existed, let’s say ‘plot’, just type

```
help plot  or
lookfor plot
```

and you will find it. You can also type

```
helpwin
```

which opens a new help window that you can follow until you find what you are looking for.

You can go directly to the help text of any function, for example by typing `helpwin` followed by the function name. For example, executing the following command

```
helpwin zeros
```

tells you all about the zeros command which allows you to create an array of zeros of any size and shape.

MATLAB also provides a web-browser-based help utility. In order to access these help files click on ‘MATLAB Help’ in the main ‘Help’ menu.

To explore MATLAB’s capabilities you can find many demo programs in the help utility. One of the best ways to learn MATLAB is by following examples.
As was mentioned above all variables used in the current MATLAB session are saved in the Workspace. You can view the contents of the Workspace by going to the ‘Desktop’ menu and selecting ‘Workspace’. You can also check contents of the Workspace by typing ‘whos’ in the Command Window.

To enter a statement that is too long to be typed in one line, use three periods, ... , followed by <enter>. For instance,

\[
x = \sin(1) - \sin(2) + \sin(3) - \sin(4) + \sin(5) - \ldots
\]

\[
\sin(6) + \sin(7) - \sin(8) + \sin(9) - \sin(10)
\]
gives
\[
x = 0.7744
\]

A Few Other Commonly Used Commands

% The comment sign. Everything appearing on the same line after the % symbol is not executed. Use lots of comments in your own programs, I guarantee that you will find it helpful in the future!

clear Clears the variables or functions from the workspace
clc Clears the command window during a session
clf Clears all figures and graphs

length Length of an array, largest dimension if the array is multi-dimensional

Matrix Operations

The basic matrix operations are addition (+), subtraction (-), multiplication (*), and conjugate transpose (') of matrices, and exponentiation (^). In addition to the above basic operations, MATLAB has two forms of matrix division: the left inverse operator \ or the right inverse operator /.

Matrices of the same dimension may be subtracted or added. Thus if E and F are entered in MATLAB as

\[
E = \begin{bmatrix} 7 & 2 & 3; 4 & 3 & 6; 8 & 1 & 5 \end{bmatrix};
\]

\[
F = \begin{bmatrix} 1 & 4 & 2; 6 & 7 & 5; 1 & 9 & 1 \end{bmatrix};
\]

and

\[
G = E - F
\]

\[
H = E + F
\]

then, matrices G and H will be returned as the element by element difference and sum of E and F.
A scalar (1-by-1 matrix) also may be added to or subtracted from a matrix. In this particular case, the scalar is added to or subtracted from each element. For example,

\[
J = H + 1
\]
gives

\[
J = \\
9 7 6 \\
11 11 12 \\
10 11 7
\]

Matrix multiplication is defined provided the inner dimensions of the two matrices are the same. Thus if \(X\) is an \(n\)-by-\(m\) matrix and \(Y\) is a \(i\)-by-\(j\) matrix, \(X*Y\) is defined provided \(m = i\), i.e., the number of columns of \(X\) equals the number of rows of \(Y\). Since \(E\) and \(F\) are 3-by-3 matrices, the product

\[
Q = E*F
\]
results in

\[
Q = \\
22 69 27 \\
28 97 29 \\
19 84 26
\]

Any matrix can be multiplied by a scalar. For example,

\[
2*Q
\]
gives

\[
ans = \\
44 138 54 \\
56 182 58 \\
38 168 52
\]

Remember that if a variable name and the “\(=\)” sign are omitted, a variable name \(ans\) is automatically created.

Matrix division can be the left division operator \(\backslash\) or the right division operator \(/\). The right division \(A/B\), for instance, is algebraically equivalent to \(A*inv(B)\), while the left division \(A\backslash B\) is
equivalent to \( \text{inv}(A) \times B \), where \( \text{inv}(A) \) is the matrix inverse of \( A \), i.e., it gives \( \text{inv}(A) \times A = I \) where \( I \) is the identity matrix with ones along the diagonal and all other elements equal to zero. You have to be careful, some matrices do not have inverses, such matrices are said to be singular.

There are MATLAB functions that can be used to produce special matrices. Examples are given in the following table.

Function Description

- \( \text{ones}(n,m) \): Produces \( n \)-by-\( m \) matrix with all the elements being unity
- \( \text{eye}(n) \): Gives \( n \)-by-\( n \) identity matrix
- \( \text{zeros}(n,m) \): Produces \( n \)-by-\( m \) matrix of zeros
- \( \text{diag}(A) \): Produce a vector consisting of the elements on the diagonal of a square matrix \( A \)

Array Operations

Array operations refer to element-by-element arithmetic operations on matrices or arrays. Preceding the multiplication (\( * \)) or division operation (\( / \)), by a period (\( . \)) indicates an array or element-by-element operation. If \( A1 \) and \( B1 \) are matrices of the same dimension, then \( A1 \times B1 \) denotes an array whose elements are products of the corresponding elements of \( A1 \) and \( B1 \).

Thus if,

\[
A1 = \begin{bmatrix} 2 & 7 & 6; 8 & 9 & 10 \end{bmatrix}; \\
B1 = \begin{bmatrix} 6 & 4 & 3; 2 & 3 & 4 \end{bmatrix}; 
\]

then

\[
C1 = A1 \times B1 
\]

results in

\[
C1 = \\
12 28 18 \\
16 27 40 
\]

An array operation for left and right division also exists. The expressions \( A1 \div B1 \) give the quotient of element-by-element division of matrices \( A1 \) and \( B1 \). The statement

\[
D1 = A1 \div B1 
\]

gives the result

\[
D1 = \\
0.3333 1.7500 2.0000 \\
4.0000 3.0000 2.5000 
\]

The array operation of raising to the power is denoted by (\( .^\) ). The statement will be of the form:

\[
q = r1.^s1 
\]
If \( r1 \) and \( s1 \) are matrices of the same dimensions, then the result \( q \) is also a matrix of the same dimension. For example, if
\[
\begin{align*}
  r1 &= [7 \ 3 \ 5]; \\
  s1 &= [2 \ 4 \ 3];
\end{align*}
\]
then
\[
q1 = r1.^s1
\]
gives the result
\[
q1 =
\begin{bmatrix}
  49 & 81 & 125 \\
\end{bmatrix}
\]
which has the elements \([7^2 \ 3^4 \ 5^3]\). One of the operands can be a scalar, just a single number. For example the expression
\[
q2 = r1.^2
\]
will give the result
\[
q2 =
\begin{bmatrix}
  49 & 9 & 25 \\
\end{bmatrix}
\]
and
\[
q3 = 2.^s1
\]
will give back
\[
q3 =
\begin{bmatrix}
  4 & 16 & 8 \\
\end{bmatrix}
\]
Note that when one of the operands is a scalar, the resulting matrix will have the same dimension as the matrix operand.

**Complex Numbers**
MATLAB also allows operations involving complex numbers. Complex numbers are entered using either \( i \) or \( j \) to represent the square root of \(-1\), \( \sqrt{-1}\). For example, a number
\[
z = 2 + j2 \text{ may be entered in MATLAB as}
\]
\[
z = 2+2*i
\]
or
\[
z = 2+2*j
\]
Also, a complex number can be entered into MATLAB as complex exponential functions.
\[
za = 2*sqrt(2)*exp(j*pi/4)
\]
Matrices with complex elements are also possible, for example typing
\[
w = [1+j \ 2-2*j; 3+2*j \ 4+3*j]
\]
will produce the result

\[
\begin{bmatrix}
1.0000 + 1.0000i & 2.0000 - 2.0000i \\
3.0000 + 2.0000i & 4.0000 + 3.0000i
\end{bmatrix}
\]

If the entries in a matrix are complex, then the “prime” (’) operator also takes the complex conjugate of each element in addition to transposing the matrix. Thus typing

\[
w' = w
\]

will produce

\[
w = \\
\begin{bmatrix}
1.0000 & 3.0000 - 2.0000i \\
2.0000 + 2.0000i & 4.0000 - 3.0000i
\end{bmatrix}
\]

For the unconjugated transpose of a complex matrix, we can use the point transpose (.’) command. For example,

\[
w' = w
\]

will yield

\[
w = \\
\begin{bmatrix}
1.0000 + 1.0000i & 3.0000 + 2.0000i \\
2.0000 - 2.0000i & 4.0000 + 3.0000i
\end{bmatrix}
\]

There are three kinds of numbers most commonly used in MATLAB:

- integers
- real numbers
- complex numbers

In addition to the classes of numbers mentioned above, MATLAB has three symbols that represent non-numbers for lack of a better label:

- \(-\text{Inf}\)
- \(\text{Inf}\)
- \(\text{NaN}\)

The symbols \(-\text{Inf}\) and \(\text{Inf}\) are for negative and positive infinity, respectively. Infinity is generated by overflow or by the operation of dividing by zero. The \(\text{NaN}\) stands for the not-a-number and is obtained as a result of mathematically undefined operations such as \(0.0/0.0\) or \(\infty - \infty\).

**The Colon Symbol (:)**
The colon symbol (:) is one of the most useful operators in MATLAB. It can be used to create vectors and matrices, to specify sub-matrices and vectors, and to perform iteration. The statement

\[ t1 = 1:6 \]

will generate a row vector containing the numbers from 1 to 6 with unit increment. MATLAB produces the result

\[ t1 = 1\ 2\ 3\ 4\ 5\ 6 \]

Non-unity (positive or negative) increments may be specified. For example, the statement

\[ t2 = 3:-0.5:1 \]

will result in

\[ t2 = 3.0000\ 2.5000\ 2.0000\ 1.5000\ 1.0000 \]

The statement

\[ t3 = [(0:2:10);(5:-0.2:4)] \]

will result in a 2-by-4 matrix

\[ t3 = \\
0.0000\ 2.0000\ 4.0000\ 6.0000\ 8.0000\ 10.0000 \\
5.0000\ 4.8000\ 4.6000\ 4.4000\ 4.2000\ 4.0000 \]

Other MATLAB functions for generating vectors are linspace and logspace. Linspace generates linearly evenly spaced vectors, while logspace generates logarithmically evenly spaced vectors. The usage of these functions is of the form:

\[ \text{linspace (i\_value, f\_value, np)} \]
\[ \text{logspace (i\_value, f\_value, np)} \]

where i\_value is the initial value, f\_value is the final value, and np is the total number of elements in the vector.

For example,

\[ t4 = \text{linspace (2,6,4)} \]

will generate the vector

\[ t4 = 2.0000\ 3.3333\ 4.6667\ 6.0000 \]

Individual elements in a matrix can be referenced with subscripts inside parentheses. For example, t4(4) is the fourth element of vector t4. Also, for matrix t3, t3(2,3) denotes the entry in the second row and third column. Using the colon as one of the subscripts denotes all of the
corresponding row or column. For example, \( t3(:,4) \) is the fourth column of matrix \( t3 \). Thus, the statement

\[
t5 = t3(:,4)
\]

will give

\[
t5 =
\begin{pmatrix}
6.0000 \\
4.4000
\end{pmatrix}
\]

Also, the statement \( t3(2,:) \) is the second row of matrix \( t3 \). That is the statement

\[
t6 = t3(2,:)
\]

will result in

\[
t6 =
\begin{pmatrix}
5.0000 \\
4.8000 \\
4.6000 \\
4.4000 \\
4.2000 \\
4.0000
\end{pmatrix}
\]

If the colon exists as the only subscript, such as \( t3(;) \), the latter denotes the elements of matrix \( t3 \) strung out in a long column vector.

**Graphing Signals**

In MATLAB both plot and stem can be used to plot a signal. In general, stem is used to plot short discrete-time signals, and plot is better suited for sampled approximations of continuous-time signals or for very long discrete-time signals. The difference between these functions can be seen by typing the following example in the Command window:

\[
x = [-5:5];
\]

\[
y = \sin(x);
\]

\[
\text{subplot}(211), \text{stem}(x,y)
\]

\[
\text{subplot}(212), \text{plot}(x,y)
\]

Here is a slightly more relevant example of generating a sinusoidal signal \( x(t) \), with a frequency of 200 Hz, sampled at a sampling rate \( R = 10,000 \) times per second. Here is a simple sequence of commands to generate this signal, plot it and then listen to it.

\[
R = 10000; \quad \% \text{ sampling rate in samples per second}
\]

\[
f = 200;
\]

\[
t = 0:1/R:2; \quad \% \text{ Create a time array 2 seconds long}
\]

\[
x = \sin(2*\pi*f*t);
\]

\[
\text{plot}(t(1:200),x(1:200)) \quad \% \text{Only plot the first 200 points}
\]

\[
\text{soundsc}(x,R)
\]

Either type this into the Command window, or copy and paste it, to see what it does.
M-Files and Scripts

More advanced computations often require execution of several lines of MATLAB code. Rather than typing all of the commands in the Command Window you should create what is known as an m-file. Each time that you will wish to repeat a computation just run the m-file again. Another advantage of using m-files is the ease in modifying the commands. There are two kinds of m-files: script files and function files. Both script and function files contain a sequence of commands. However, function files take arguments and return values and can be called by other script files, or even other function files.

To list all of the m-files in the current directory on your disk, you can use the MATLAB command what. The MATLAB command, type, can be used to show the contents of a specified file.

Creating Script files

To make an m-file click on File, select New and click on M-File from the pull-down menu. You will be presented with the MATLAB Editor/Debugger screen into which you will type your commands. Once you are done typing, select File, in the MATLAB Editor/Debugger screen and select Save As... . Chose a name for your file, e.g., firstgraph.m and click on Save. Make sure that your file is saved in the directory that is in MATLAB's search path.

If you have two files with the same name, then the one that occurs first in MATLAB's search path will be executed. To open the m-file from within the Command Window type

edit firstgraph

and then press the <enter> key. You can open a previously created m-file by selecting ‘Open ...’ in the ‘File’ menu. Choose the desired m-file and open it. You should see a new window that contains your code.

To illustrate the use of an m-file, type the series of commands in the previous example into the Matlab Editor and save it as “firstgraph”.

    % My first MATLAB m-file
    % This creates a sine wave and then plots it and plays it as a sound
    clear
    close all
    R = 10000; % sampling rate in samples per second
    f = 200;
    t = 0:1/R:2; % Create a time array 2 seconds long
    x = sin(2*pi*f*t);
    plot(t(1:200),x(1:200)) % Only plot the first 200 points
    soundsc(x,R)

To run the script, go to the Debug menu and select Run. You can also choose to press F5 (on a PC) to run your script.

Statements in a script file operate globally on the workspace data. Normally, when

M-files are running, the commands are not displayed on screen. The MATLAB echo command can be used to view m-files while they are executing.
Functions

Functions are m-files that are used to create new MATLAB functions. Variables defined and manipulated inside a function file are local to the function, and they do not appear in the workspace. However, arguments may be passed into and out of a function file.

The general form of a function file is:

```matlab
function variable(s) = function_name(arguments)
% help text in the usage of the function
command lines
end
```

To illustrate the use of function files and rules for writing m-file functions, let’s look at an example. Write a function file to generate a sinusoidal sum, for given input frequencies, sampling rate, and desired length in seconds

Solution

```matlab
function signal_add = u(f1,f2,R,tMax)
% f-input frequencies, R- sampling rate, tMax – max duration
t = 0:1/R:tMax;
signal_add = sin(2*pi*f1*t) + sin(2*pi*f2*t);
end
```

Save this m-file as signal_add.m It is very important that you give the function file the same name as the function itself. It is not necessary to compile the function file. Be sure the folder that you saved the function file to is in the same folder as the m-file that calls the function. Now suppose we want to find the sum of two sinusoids whose frequencies are 210 and 200 Hz, sampled at 2000 Hz, for 2 seconds. The following statements can by typed in the MATLAB command window to apply the function to a specific set of inputs and then to plot the output.

```matlab
x = u(210,200,2000,2);
plot(x)
```

The Panic Button

To interrupt a running program either press the Ctrl and ‘c’ keys or the Ctrl and ‘.’ together. Sometimes you have to do this multiple times to halt execution of your program. This is not a recommended way to exit a program, however, in certain circumstances it is a necessity. For instance, a poorly written program can put MATLAB into an unending loop and this would be your only way to escape the program short of shutting down the computer.