A Brief Introduction to MATLAB

September 1, 2005, 12:01

Professor: Jeffrey A. Fessler (originally by Robert Nickel)

MATLAB is a technical computing environment for high-performance numeric computation and visualization. MATLAB integrates numerical analysis, matrix computation, signal processing (via the Signal Processing Toolbox), and graphics into an easy-to-use environment where problems and solutions are expressed just as they are written mathematically, without much traditional programming. The name MATLAB stands for *matrix laboratory*.

We will use MATLAB to illustrate concepts with *numerical examples*. Some or all homework assignments will include some problems that require MATLAB solutions. Wise students will quickly realize that MATLAB can often be used to check solutions to other problems as well. This is perfectly legal, in fact, encouraged. But you still must turn in your analytical solutions for the pencil-and-paper problems.

MATLAB is available on CAEN on UNIX, PC, and Mac platforms. You start MATLAB by double-clicking on the MATLAB-Icon (MAC, PC) or by typing **`matlab`** on the UNIX command line. You should then see the MATLAB prompt, denoted "**`>>`**".

Some assignments may require you to use MATLAB version 5.0 (or later). When you first start MATLAB, the version number is printed. Make sure it is 5.0 or higher. In CAEN you may need to use the **`swselect`** command to choose the latest version of MATLAB if it is not the default already.

This document is by no means a complete reference. There are tutorial and reference manuals for MATLAB at the Media Union. MATLAB has lots of on-line help. CAEN offers MATLAB tutorials each semester. The CAEN Hotline can help with questions about printing etc.

- **Getting Help**
  For a list of all the available help topics, type
  ```
  >> help
  ```
  For help on a particular topic or command, such as **`plot`** type:
  ```
  >> help plot
  ```

- **Loading and Saving Data**
  MATLAB's **`load`** and **`save`** commands allow reading **`.mat`** files from disk into MATLAB, or saving a MATLAB variable to disk. The command **`whos`** lists the current variables.

- **Definition of Vectors/Signals**
  MATLAB provides several commands for generating vectors. We use vectors to represent arrays of samples of signals. The following are all equivalent commands for generating a vector.
  ```
  >> x = [4 6 8 10 12 14]
  >> x = 4 + 2*[0:5]
  >> x = 4:2:14
  >> x = linspace(4,14,6)
  ```
  We only use the first one for very short vectors. For more information on the "**`:`**" operator type:
  ```
  >> help ops
  >> help colon
  ```

- **Making long vectors**
  You can form long vectors (*e.g.* signals) by concatenating two shorter vectors.
  ```
  >> x = 4:2:14
  >> y = [x, x];
  ```
  The comma concatenator works for row vectors. For column vectors use the semicolon concatenator:
  ```
  >> x = [4:2:14]'
  >> y = [x; x];
  ```

- **How to Suppress Display of Results**
  Append a semicolon "**`;`**" to the *end of a line* to suppress the display of the results. For example:
  ```
  >> x=1:50;
  ```

- **Plotting a Signal**
  The easiest way to graphically display a continuous-time signal is with the **`plot`** command:
```
>> t = linspace(0, 10, 100);
>> x = sin(t);
>> plot(t, x)
```
  For a discrete-time signal, we usually use the **`stem`** command:
```
>> n = 0:20;
>> x = cos(pi*n/3);
>> stem(n, x)
```
  Be sure to label the axes of your graphs using the **`xlabel`**, **`ylabel`**, and **`title`** commands. You can put some mathematical symbols in these labels, for example
```
title('Frequency response from -\pi to \pi')
```
  produces a plot title that reads "Frequency response from $-\pi$ to π." You can adjust the graph axes using the **`axes`** command.

- **Images**
  The commands **`image`** and **`imagesc`** are useful for displaying images. However, be cautioned that in Matlab the array column index various fastest, whereas in most image processing software the row index varies fastest. A simple solution is to always display the *transpose* of the array when using **`imagesc`**.

- **Generating Matrices**
  MATLAB also provides several commands to generate matrices. Try out the following.
```
>> A = [1 2 3; 4 5 6; 7 8 9]
>> B = eye(3)
>> C = ones(2,3)
>> D = zeros(3,2)
>> E = rand(1,5)
>> F = randn(5,1)
```
  The last two commands generate random vectors, i.e. "random signals".

- **Matrix Manipulation**
  The following commands are useful for matrix/vector manipulations. You can transpose a matrix, flip the matrix from left to right or up and down, concatenate two matrices and so forth. Use the matrices defined above and type:
```
>> P = A.'
>> P = fliplr(A)
>> P = flipud(A)
>> Q = [A D]
>> R = [A; B]
```

- **Complex Numbers and Constants**
  Both **`i`** and **`j`** are defined by default to be $\sqrt{-1}$, unless you assign them to some other value. Thus MATLAB can handle complex numbers. It also has many built-in variables:
```
>> x=2+3*j
>> y=pi
```
  If you want to use **`i`** and **`j`** for other purposes, you can always still form complex numbers as follows:
```
>> x=2+3j
>> y=2+3i
```

- **Functions**
  Functions are evaluated *element wise*, as in the **`sin(t)`** example above. For example, if we type
  `>> t = linspace(0,4pi,9);` then **`t`** is a vector containing 9 time samples. If we then type
  `>> x = sin(t)`, then MATLAB creates the vector **`x`** with 9 values corresponding to the **`sin`** of each of the elements of the vector **`t`**, *i.e.* **`x = 0 1 0 -1 -0 1 0 -1 0`** *Think carefully about why I used 9 rather than 8 samples in this example. This is known as the "picket fence" problem (ask me why) and is a common error in* MATLAB. To get a list of available functions, type:

```
>> help elfun
```

- **Operators**
  Since MATLAB generally deals with matrices, you must be careful when using operators like "`*`" or "`/`". If you want these operators to operate in an element-by-element fashion you have to denote this by a leading period, e.g."`.*`" and "`./`" ! Try the following examples:
  ```
  >> x=1:5
  >> y=x+x
  >> y=x.*x
  >> y=x-x
  >> y=x./(x+x)
  ```
  Note: "*" and "/" without "." are matrix multiplication and "matrix division" (special functions of MATLAB). For example:
  ```
  >> A = eye(3) * rand(3,2)
  ```
  A special case applies for scalar multiplication:
  ```
  >> y=2*x
  ```
  So for scalar multiplication or division, you do not need the extra leading period.

- **Writing Programs in MATLAB**
  You can also write your own programs in MATLAB using any regular ASCII text editor. Simply open a file with the extension **`.m`** (which is called an *m-file*) and edit line-by-line the sequence of commands you want to include in your program. Save the file and execute the program by typing the name of the file (without the **`.m`**) on your MATLAB command line.
  EXAMPLE: Invoke a text editor (*e.g.* **`emacs`** on UNIX or **`notepad`** on PCs) and edit the following lines:
  ```
  % This is a program that generates a noisy signal
  x=linspace(0,10*pi,200);
  % compute the signal
  y=sin(x);
  % compute the noise
  z=0.3*rand(1,200);
  y=y+z;
  % plot the signal
  plot(y)
  ```
  Save this program in a file named **`noisy.m`** in MATLAB's current working directory. (Use MATLAB's **`cd`** command to change MATLAB's current working directory. Print MATLAB's current working directory with MATLAB's **`pwd`** command.) To execute your program, type its name at the prompt:
  ```
  >> noisy
  ```
  You can also write your own functions in MATLAB. Check out:
  ```
  >> help function
  ```

- **Control Flow in M-Files**
  MATLAB also provides the usual programming language commands *for-end*, *if-else-break-end* and *while-end*. For example, try the following:
  ```
  >> for c=1:2:12; disp(c); end;
  ```
  See the command line help for more information.

- **Unit step function**
  MATLAB's **`inline`** function is convenient for creating the unit step function $u(t)$ or $u(n)$.
  Try:
```
>> u = inline('t >= 0');
>> t = linspace(-2, 10, 100);
>> plot(t, u(t-3) + u(t-5))
```

- **Kronecker delta function** $\delta[n]$
  MATLAB's **`inline`** function is convenient for creating the Kronecker delta function $\delta[n]$. (But note that MATLAB cannot implement the Dirac delta function $\delta(t)$.) Try:
```
>> delta = inline('n == 0');
>> n = 0:10;
>> x = delta(n-1) + 2 * u(n-3);
>> stem(n, x)
```

- **Miscellaneous**
  The following commands are important in DSP. The sooner you get acquainted with them the better ...
```
>> help conv
>> help filter
>> help roots
>> help fft
>> help sum % summing vectors
```

- **Printing**
  In Unix, you probably must type **`setenv PRINTER printername`** *before* starting MATLAB to get plots to print out locally.

- **Symbolic Integration** In this course, you are *always* allowed to use MATLAB to perform tedious integration.
```
>> help sym/int
```
  For example, to compute $\int_1^\infty at\,\mathrm{e}^{-t}\,\mathrm{d}t$ you simply type the following.
```
>> int('a * t * exp(-t)', 't', 1, inf)
```

- **Sound**
  On machines that have sound cards, MATLAB can use the **`sound`** command to send discrete-time signals to those cards to be converted to analog audio signals, which can be heard using headphones. Here is an example that generates a 1kHz sinusoidal signal of 0.5 second duration at a 8192Hz sampling rate.
```
>> fs = 8192;
>> f = 1000;
>> n = 1:(0.5*fs);
>> x = sin(2*pi*f*n/fs);
>> sound(x, fs)
```
  On SUN workstations, you must start **`/usr/demo/SOUND/bin/gaintool`** before you can replay audio signals. PC's probably have audio control panels too.
  Caution: Undoubtably CAEN has policies against playing sounds from workstation speakers in the labs. Use headphones!
  Some machines (Suns?) may not support sampling frequencies other than 8192Hz.

- **For those who have way too much time ...**
  MATLAB has a lot to offer. Also a lot of "fun" stuff to play with. Just type:
```
>> demo
```
  There is book from Prentice Hall called Mastering Matlab 5 by Duane Hanselman and Bruce Littlefield that you may find helpful.