Introduction to MATLAB

Mike Renfro

February 8, 2011
Introduction to MATLAB

Starting MATLAB, Exiting, Getting Help
Starting and Exiting MATLAB
Getting Help

Major Elements of the MATLAB Environment
Command Window
Workspace
Command History
Current Directory Selector

MATLAB as a Calculator
Real Values
Complex Values
Trigonometric Functions
Function Reference
Variables
Writing MATLAB Functions

Writing Functions

Function Example: Projectile Motion
Rules for Writing Functions
Functions With Multiple Returned Variables

“Homework”

Other Function Examples

Distance
Root-Mean-Square
Fast Fourier Transform
Part I

Introduction to MATLAB
Starting MATLAB

- In the CAE domain: double-click the MATLAB icon on the desktop for your preferred version. Here, we’ll use MATLAB R2010b, but the instructions should generally apply for any version of MATLAB.

- In the PCLAB domain: Start → Programs → Engineering → MATLAB → (version number) → MATLAB (version number)

- Other domains: ask whoever manages them.
Exiting MATLAB

File → Exit or type `exit` and press Enter at the MATLAB command prompt.
Help → MATLAB Help
This is a very good starting point for the most commonly-used MATLAB documentation. Major topic areas include:

- Getting Started
- Using MATLAB
- Examples
- MATLAB Functions Listed by Category
- MATLAB Functions Listed Alphabetically
Command Window

For these notes, if you see a >> symbol, it’s something you type in the command window. Otherwise, it’s often something you type into an editor.
Workspace

The image shows a screenshot of the MATLAB 7.4.0 (R2007a) interface with the Workspace window open. The table in the Workspace window displays the following variables and their values:

- **Area**: 314.1593
- **r**: 10

The values for Min and Max are not shown in the table.
Command History

%-- 1/29/08 6:59 PM --%

r=10; % in
Area=pi*r^2;
The first thing you should do after starting MATLAB is to set the current directory to either your F: drive, U: drive, your flash drive, or somewhere else you can save files permanently. Some labs may start MATLAB in C:\TEMP or some other temporary area. Others may start MATLAB in an unwritable folder in drive C:.
Basic Mathematical Operators

The symbols +, -, *, and / act the same as in most languages (addition, subtraction, multiplication, and division).

```
>> 1+3
ans =
   4
>> 2-5
ans =
  -3
>> 1.5*2.5
ans =
3.7500
>> 4.5/3
ans =
1.5000
>>
```
The caret (^) is used for raising a number to a power: \( 2^3 \) in MATLAB is the same as \( 2^3 \) in mathematics.

\[
\begin{align*}
\text{>>} & \quad 2^3 \\
\text{ans} & = 8 \\
\text{>>} &
\end{align*}
\]

The parentheses symbols, ( and ), are used to group mathematical terms together.

\[
\begin{align*}
\text{>>} & \quad 1+2*3 \\
\text{ans} & = 7 \\
\text{>>} & \quad (1+2)*3 \\
\text{ans} & = 9 \\
\text{>>} &
\end{align*}
\]
MATLAB handles real and complex numbers equally easily. C, at least, does not have any standard facility for complex math. C++ and FORTRAN handle complex numbers as easily as they handle real numbers.

```
>> i*i
ans =
   -1
>> 2i-3j
ans =
     0 - 1.0000i
>> 1+exp(i*pi)
ans =
     0 +1.2246e-016i
```
Basic Trigonometric Functions

All arguments to MATLAB trigonometric functions are in units of radians, not degrees.

\[ \sin(), \cos(), \tan(), \cot(), \sec(), \csc() \]

\[
\begin{align*}
\text{>> sin(0)} & \\
\text{ans} & = 0 \\
\text{>> tan(pi/4)} & \\
\text{ans} & = 1.0000 \\
\text{>> cos(60*(pi/180))} & \\
\text{ans} & = 0.5000 \\
\end{align*}
\]
Inverse Trigonometric Functions

All results from MATLAB inverse trigonometric functions are in units of radians, not degrees.

\[ \text{asinx}, \text{acosx}, \text{atanx}, \text{acotx}, \text{atan2x}, \text{asecx}, \text{acscx} \]

\[
\begin{align*}
\text{>> asin (.5)} \\
\text{ans =} \\
0.5236 \\
\text{>> asin (.5)*(180/pi)} \\
\text{ans =} \\
30.0000 \\
\text{>> atan2(-1,-1)} \\
\text{ans =} \\
-2.3562 \\
\text{>>}
\end{align*}
\]
Hyperbolic and Inverse Hyperbolic Trigonometric Functions

\[
\text{sinh()}, \text{cosh()}, \text{tanh()}, \text{coth()}, \text{sech()}, \text{csch()}, \text{asinh()}, \\
\text{acosh()}, \text{atanh()}, \text{acoth()}, \text{asech()}, \text{acsch()}
\]
Reminder: MATLAB Function Reference

Help → MATLAB Help → MATLAB Functions Listed By Category
Help → MATLAB Help → MATLAB Functions Listed Alphabetically
Variable Usage

The first thing we’ll show that separates MATLAB from a basic scientific calculator is its ability to store numbers as variables for later usage. This lets you start building up equations out of meaningful names instead of less-obvious numbers.

- MATLAB variable names must begin with a letter, which may be followed by any combination of letters, digits, and underscores.
- MATLAB distinguishes between uppercase and lowercase characters, so A and a are not the same variable.
- MATLAB only uses the first 63 characters of a variable name, so make sure your variable names are unique within the first 63 characters.
Variable Example

```matlab
>> circleRadius = 2
circleRadius =
    2
>> circleArea = pi * circleRadius ^ 2
circleArea =
    12.5664
```
Detour: What to Name Variables?

- The names of variables should document their meaning or use.
- Variable names should be in mixed case starting with lower case. (linearity, qualityOfLife)
- Variables with a large scope should have meaningful names. Variables with a small scope can have short names. Variables for temporary values can keep a short name (x, k).
- The prefix n should be used for variables representing the number of objects (nFiles, nSegments).
- Pluralize consistently. Having two variables with names differing only by a final letter s should be avoided (point, points). Use something like point and pointArray or point and pointList instead.

(Johnson, “MATLAB Programming Style Guidelines”)
Part II

Writing MATLAB Functions
MATLAB functions are reusable bits of MATLAB code that perform a series of calculations or other tasks.

Normally, a function takes one or more input arguments and returns one or more results into output variables.

A function is saved in its own file with a ‘.m’ extension, such as ‘computeProjectilePosition.x.m’
Function Example: Projectile Motion

Given: a projectile is launched with an initial velocity $v_0$ and initial angle $\theta$.
Calculate: the projectile’s horizontal and vertical positions relative to launch at a given time $t$.
Governing equations:

\[ x(t) = (v_0 \cos \theta) t \]
\[ y(t) = (v_0 \sin \theta) t - \frac{1}{2} gt^2 \]
Mathematics to MATLAB Translation

\[ x(t) = (v_0 \cos \theta) t \]

`compute projectile position x.m`

```matlab
function x = compute projectile position x(v0, theta, t)  
% Compute the relative horizontal position  
% of a projectile  
%  
% Usage: compute projectile position x(v0 ,theta ,t)  
x = (v0 * cos(theta)) * t;
```
Mathematics to MATLAB Translation

\[ y(t) = (v_0 \sin \theta) t - \frac{1}{2} gt^2 \]

\begin{verbatim}
compute projectile positiony.m

function y = compute projectile positiony(v0, theta, t)
    % Compute the relative vertical position
    % of a projectile
    %
    % Usage: compute projectile positiony(v0, theta, t)
    g = 9.808; % m/s^2
    y = (v0 * sin(theta)) * t - 0.5 * g * t^2;
\end{verbatim}
Procedure for Creating Functions

- Change directories to somewhere you can save files permanently (F:\, your flash drive, etc.)
- At the MATLAB command prompt, type 'edit computeprojectilepositionx.m', and click the 'Yes' button when it asks you if you want to create a new file.
- Enter the six lines of MATLAB code into the editor window that pops up. Save the file, and make sure you save it as 'computeprojectilemotionx.m'
- Repeat the previous two steps for 'computeprojectilemotiony.m'
function x = compute-projectile-position-x(v0, theta, t)
    \% Compute the relative horizontal position
    \% of a projectile
    \%
    \% Usage: compute-projectile-position-x(v0, theta, t)
    x = (v0 * cos(theta)) * t;

function y = compute-projectile-position-y(v0, theta, t)
    \% Compute the relative vertical position
    \% of a projectile
    \%
    \% Usage: compute-projectile-position-y(v0, theta, t)
    g = 9.808; \% m/s^2
    y = (v0 * sin(theta)) * t - 0.5 * g * t^2;
Test The Functions

Test out how your functions work with a variety of inputs:

```matlab
>> help computeprojectilepositionx
>> computeprojectilepositionx(5,pi/4,0)
>> computeprojectilepositionx(5,pi/4,2)
>> computeprojectilepositiony(5,pi/4,0.1)
>> computeprojectilepositiony(5,pi/4,0.25)
>> computeprojectilepositiony(5,pi/4,1)
```

Do all the results make sense?
Rules for Writing Functions

- The first part of the function filename must match the name of the function given on line 1.
- The variable on the left-hand side of the equal sign on line 1 must be given a value before the function ends.
- In general, the function cannot use any predefined variables except for the ones listed in parentheses on line 1. There are exceptions to this rule, but you have to take special steps to get around it.
- The names of the variables listed in parentheses don’t have to match anything previously defined, but their meanings have to match.
MATLAB functions can return as many results as needed, instead of the single return variable shown in previous examples.

The only change required is for you to specify multiple return variables on the left hand side of the equal sign on line 1 of your function.
function [x,y]=compute(projectileposition(v0,theta,t)
% Compute the relative horizontal and vertical % position of a projectile
% % Usage: [x,y]=compute(projectileposition(v0,theta,t)
g=9.808; % m/s^2
x=(v0*cos(theta))*t;
y=(v0*sin(theta))*t-0.5*g*t^2;

Test this as follows:

>> [x,y]=compute(projectileposition(5,pi/4,0.25)
Write MATLAB functions to calculate results related to projectile motion as follows:

- Calculate $x$ and $y$ positions relative to a fixed origin $(0, 0)$ assuming the projectile is fired from a position $(x_0, y_0)$ according to the equations

  \[
  x(t) = x_0 + v_0 \cos \theta t \\
  y(t) = y_0 + v_0 \sin \theta t - \frac{1}{2}gt^2
  \]

- Calculate $x$ and $y$ components of velocity according to the equations

  \[
  v_x(t) = v_0 \cos \theta \\
  v_y(t) = v_0 \sin \theta - gt
  \]
Other Function Examples: Distance

The distance $d$ between two points $(x_1, y_1)$ and $(x_2, y_2)$ is defined as

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

```matlab
function d = distance(x1, y1, x2, y2)
    d = sqrt((x1-x2)^2 + (y1-y2)^2);
```
Other Function Examples: RMS (Root-Mean-Square)

The RMS value of a set of data \( x = x_1, x_2, \cdots, x_N \) is defined as

\[
RMS = \sqrt{\frac{\sum_{i=1}^{N} x_i^2}{N}}
\]

function out = rms(x)
%
% rms: root-mean-square value of a vector
% (mwr, 10/96)
%
% Usage: rms(x)
out = sqrt(sum(x.^2)/length(x));
The FFT of an arbitrary periodic function \( x(t) \) is a way of identifying the magnitudes and frequencies of simpler periodic functions that sum up to that function. MATLAB has an FFT function built in, but it doesn’t format its results in a particularly useful way for mechanical vibrations work.

```matlab
function [X,f]=myfft(x,t)
% myfft - An FFT function with usable output
%
% Usage: [X,f] = myfft(x,t)
%
% Where: x is some time-domain signal
% t is the time axis for x
%
% Note: t must be an evenly-spaced vector
% f is in units of Hz
```
Other Function Examples: FFT (Fast Fourier Transform)

% FFT only runs quickly on vectors of length $2^k$
\n% n_points = $2^{(\text{floor}(\log(\text{length}(x))/\log(2)))}$
\n% MATLAB FFT output needs to be scaled according to
% number of points
\nfftX = 2*fft(x, n_points)/n_points;
\n% First value in fftX is DC offset. Not useful
% in my application, so delete it.
\nfftX(1) = [];
\n% Define the Nyquist frequency and use that to
% make a useful frequency vector.
\ndt = t(2) - t(1);
\nnyquist = 1/2;
\nf = (1:n_points/2)/(n_points/2)*nyquist/dt;
\nX = fftX(1:n_points/2);
Magnitude of FFT for $x(t) = 0.8 \sin(2\pi(25)t) + 0.4 \sin(2\pi(75)t)$