Introduction to Matlab

Will Fox
25 September, 2006

Contents:

Interacting with Matlab

```
% Matlab evaluates expressions
5+5

% Matlab can assign the result to a variable
a = 5 + 5;

% note that the semi-colon suppresses output.  To get the answer
a

% what variables does Matlab know about?
who

% this is the best way to find out about a without displaying it
size(a)

% Answer: a is a matrix, with 1 row and 1 column, i.e. 1 elements
% Everything in matlab is a matrix


% some other simple operations:
b = 2*a

% This does exponentiation:
c = a^2

% some built-in functions, too, and

% Matlab has a couple built-in constants
pi
i

% and built-in functions
a = pi/2
d = sin(a)


% Chapter 1 is done
clear
```

Arrays, aka Vectors

```
% create arrays directly with the [ ] operations
fib = [1 1 2 3 5 8 13];

% getting the array length
length(fib)

% getting the size  (surprise here!)
size(fib)

% Access elements of the array with ( )
fib(3)
% For those who care: Matlab does unit-offset indexing of arrays


% use the colon (:) operator to construct simple arrays simply
oneto10 = [1:10]
odds = [1:2:11]
zeroto1 = [0:0.05:1]


% Language idea: the for loop.
fib = [1 1];                            % initialize array
for k=3:15
        fib(k) = fib(k-1)+fib(k-2);
end

% Matlab also lets us concatenate arrays
fibfib = [fib fib]

% main point
[ ] is for array creation
( ) is for array subscripting  (and for function calls, see later)
```

Thinking in Matlab: Vectorized Subscripting

% Getting the first three elements of fib

% the C way to do it

```
fib3 = [];
for k=1:3
     fib3(k) = fib(k)
end
```

% the Matlab way to do it

```
ind = [1 2 3]
fib(ind)
```

% or

```
fib([1 2 3])
```

% get odd elements

```
fib(1:2:9)
```

% better!

```
fib(1:2:end)
```

% reverse

```
fib(end:-1:1)
```

Thinking in Matlab: Vectorized Math

% want to compute squares

% The C way

```
fibsq = []
for k=1:9
     fibsq(k) = fib(k) * fib(k);
end
```

% matlab also allows

```
fibsq = []
for k=1:9
     fibsq(k) = fib(k)^2;
end
```

% Better!

```
fibsq = fib.^2
```

% why .^ ?
% _MAT_ lab - originally for matrices,
% so ^ is reserved for strict matrix multiplication

Thinking in Matlab: Tests and Vectorized Tests

% First, we need to know about tests and test operations

% remember assignment operations

```
a = 5
```

% Now test. Watch what Matlab returns for true and false

```
a > 3
a < 2
a ~= 8
```

% Matlab returns 1 for true, 0 for false

Note that == is not the same is =

```
a == 5
```

% if, then, else construct

```
b = 0
if (a == 5)
     b = 1;
else
     b = 2;
end
b
```

% Let's grab the elements of fib that are greater than 5

```
fibg5 = [];
for k = 1:length(fib)
     if fib(k) > 5
          fibg5 = [fibg5 fib(k)];
     end
end
fibg5
```

% Let's think in Matlab now

% First, check out vectorized tests

```
fib > 5
fib == 5
```

% Second, check out find.

```
find(fib == 5)
find(fib < 5)
```

% Find returns the indices of the elements that are true.

```
find([ 1 0 1 1 0] )
```

% Elements of fib > 5, in Matlab

```
indfibg5 = find (fib > 5)

fib(indfibg5)

fibg5 = fib(indfibg5)
```

% Nice

Built-in functions and Plotting

```
% some functions
sin, cos, tan, atan, exp, ...

% to get a list of what matlab can do, try
help

help sin
help elmat


% Simple
sin(pi/2)

% It's Matlab, so we like vectorized functions

x = pi *  [ 0:4]

sin(x)
cos(x)


% Demo plotting routines


% open a new figure window
figure

x = [0:0.01:2*pi];
plot(x, sin(x))

plot(x, cos(x))

% Gettting multiple plots on the same graph
hold on

plot(x, sin(x), 'r')


% Clearing in the figure
clf
```

For longer tasks...

% We need to load and save files, and generally interact with the file system

% Print the working directory
```
pwd
```

% change the working directory
```
cd c:\will
```

% show contents of the directory
```
ls
```

% use the matlab editor
```
edit
```

% Loading and saving your work to .mat files

```
save Sep23talk.mat fib

clear
ls

load Sep23talk
```

% Check out `help load, help save` for more options

% Matlab lets you write scripts and functions

% Script

```
% fibscript.m
N = 50;
fib = [1 1];                    % initialize array
for k=3:N
        fib(k) = fib(k-1)+fib(k-2);
end
```

```
% try it out
clear
who
fibscript
who

% Things to watch out for:
path
which fibscript
```

```
% Now, do the same as a function.
% fibfunction.m

function fib = fibfunction(N)
fib = [1 1];                    % initialize array
for k=3:N
     fib(k) = fib(k-1)+fib(k-2);
end
```

```
% try it out
clear
fib = fibfunction(50)
fib = fibfuncton(100)
```

Notes on scripts vs. functions

Scripts

Just a list of matlab commands in a file.
Executed in the *present* workspace

Functions

Have *private* workspace
Have input and output parameters

What you want depends on your task

Scripts are for one-off things -

You don't always want to be generic - sometimes you want to be *very* particular about what you did, especially when you're getting back to a problem after 6 mos.

I like to save lots of scripts with descriptive names and dates

The ideal is to be able to open up matlab, cold call a script, and have something to play with

Example – here is part one of my directories

...
loadAndFtData9Sep.m
loadAndFtOverData24Aug.m
loadAndFtOverData30Aug.m
loadAndFtOverData31Aug.m
loadAndFtOverData31AugB.m
loadAndFtOverData7Sep.m
loadAndOverBdata10Nov.m
loadAndOverBdata9Nov.m
loadAndOverData11Sep.m
loadAndOverTe21Aug.m
....

Functions are for repetitive tasks

Example: one of my directories has

fourierdelay.m
fouriercoefs.
fourierint.m
fouriersum.m

Continuing Education

Other Resources

www.mathworks.com
www.octave.org

Things I haven't covered

more language elements	while…end, break, case, try, …
exotic plotting,	loglog, semilogx, …
3D plotting	contour, surface
exotic 3D plotting	quiver…
strings,	help strfun
displaying	disp, sprintf
file access	fopen, fread, fwrite
graphical user interface creation	
matrix math	try help matfun
interfacing to external (e.g. C) code	
signal processing	fft, specgram, …
…	

Alcator-Specific functions

% open "LH" tree for "shot shot_number"
```
stat = mdsopen('alcdaq::LH',shot_number);
```

% retrieve data from the tree and assign it to variable
```
x1=mdsvalue('\LH::TOP.HXR.RESULTS.COUNTRATE:CH01');
```

% close tree
```
mdsclose
```

Done with Matlab?

```
exit
```

Essential Matlab commands

Interacting with the Matlab interpreter

`help`	ask Matlab about something
`help <command>`	
`<Ctrl-C>`	try to stop a command
`<Up Arrow>`	cycle back through command history
`who`	show variables in memory
`whos`	. . . with more detail on size
`clear`	clear all variables from memory
`size(<array>)`	return dimensions of `<array>`

Operation

`=`	assignment
`.*, .^, ./,`	remember to use "dot" for element-by-element ops
`+, -`	automatically element-by-element!
`==, ~=, <, etc`	tests (equal, not equal, less than, etc)
`[ ]`	array creation operators
`( )`	array subscripting. (N.B. also function call!)
`find`	return indices of elements that are true (i.e. not zero)

Language

`for <ind>=<arr> ...`	loop over each element of `<arr>`
`end`	

```
if <1st cond> ...
elseif <2nd cond>...
else ...
end
```

if/then/else blocks

Plotting

`plot`	regular 2-D plot. Tons of options – try `help plot`
`figure`	open a new figure window
`figure(<fig>)`	bring window `<fig>` to the foreground
`hold on`	add next plot to the figure, instead of clearing it
`hold off`	clear figure before doing next plot (this is the default!)
`clf`	clear the figure

Alcator

`mdsopen`	Open the tree
`mdsvalue`	Get a value

`mdsclose`	Close the tree

Multi-day tasks

`%`	start comment – Matlab will ignore the rest of the line
`edit`	bring up Matlab's nice editor to edit .m files
`save`	save variables to a .mat file
load	load variables from a .mat file
`path`	show and set the path
`which (<command>)`	show the path to *<command>*
`cd`	change directory
`pwd`	print working directory
`ls`	list files in working directory

Defining your own functions

This should be the first line of your .m file
(except if you put comments above... try `help help` or `type help.m`)

`function` *<output>* `=` *<fun_name>* `(` *<inputs>* `)`

<output>	name the variable that will be returned as output, set it somewhere in the routine
<fun_name>	what you are going to call the function. Note Matlab's convention is to name the file *<fun_name>*`.m`
<inputs>	input variables to the function. Note that these are "passed by value", so caller won't know if these get changed inside the routine.