Lecture 4 for Math 398 Section 952: Scripts and Functions

Thomas Shores
Department of Math/Stat
University of Nebraska
Fall 2002

Some Background

What they're about.

Script files and function files are both text files which end in the suffix ".m". Think of a script as analogous to a program consisting of sequence of Matlab commands that one could execute at the command line if one wanted to. Function files, on the other hand, are analogous to subroutines that return a value. As a matter of fact, many of the "built-in" Matlab functions are themselves function files written in Matlab.

As text files, both scripts and function files can be manipulated by any text editor. In particular, Matlab has a handy built-in editor. Let's try it out on a script and a function. Start Matlab and type

```
> edit classRK4
```

Then use the window commands to open a script file from the MatlabGuideFiles directory.

Relational and logical operators.

Matlab recognized Boolean values "true" which has value 0 and "false" which has value "1" (actually anything not 0).

Relational operators:

```
==  ~=  <  >  <=  >=
```

Logical operators:

```
&  |  ~  xor  any  all
```

Enter these commands and interpret the results:

```
> A = [1, -2; 3, 4]
> B = [1 0; -5 4]
> isequal(A,B)
(A < B) | (A < abs(B))
A < B & A < abs(B)
all(A == B)
any(A == B)
~(A == B)
```

Control structures.

They are:

```
if (expression) ...  {elseif}...  {else} end
for (expression) ...  end
while (expression) ...  end
switch (variable) ...{case} {otherwise} end
```

Issue these commands at the command prompt to explore the if command:

```
> x = 25
> if (x > 10)
disp('x is larger than 10.')
else
disp('x is not larger than 10.')
end
```

Repeat these commands after changing x to 5. Next, the for command. First make a vector.

```
> v = A(:)
> for (j = 1:length(v))
if v(j) > 10
disp ('Bigger than 10'), disp(v(j))
end
end
>
```

Next the while command. Here we will use it to simulate a “for” command.

```
> j = 1, lnth = length(v)
> while (j <= lnth)
v(j)
j = j+1;
end
>
```

Next and example of the switch command.

```
> switch lnth
case 1
disp('x is scalar number')
case 2
disp('x is 2-d vector')
case 3
disp('x is a big vector')
end
>
```

Already we can see the need for these control structures to live in a script file, where we won’t have to retype them constantly.

Functions

Function files.

Remember that a function file must have as its first executable statement a function statement of the form

```
   function retval = fname({parameter list})
```

Also, the name of the function must match the name of the file, which in this case would be "fname.m". Notice that the file has a parameter list of variables (really copies thereof) that will be passed to the function. There is a way for a function to return a variable number of values, too. Just use a vector list of values on the left. If you don't want the function to return any value, delete the "retval = " portion of the definition. There is even a way to have a variable number of arguments, plus mandatory arguments. For example,

```
   function [retval1, retval2] = fname(x, y, varargin)
```

To learn more, get help on varargin.

It is even possible to do recursion, that is, a function can call itself. If you use this feature, be very careful. It is too easy to get caught in an infinite sequence of recursive calls.

Built-ins.

Many of the Matlab function files are themselves written in Matlab. To find them, all you need to do is to use the edit command, since they live on your default search path. For example, the command "hilb(n)" returns a Hilbert matrix of size n. Try the following:

```
   > hilb(2)
   > hilb(3)
   > edit hilb % matlab assume a default extension of ".m"
```

Now examine the hilb file. You won't be able to modify it (which is good!) since it is a system file.

Here's a little tip: it's always nice to have a head start on a problem, so when I need to write a script or function file, I try to find one that I already have, load it up and, right away, use Save As to save this file to a different name. Now I have a template to modify.

Inline: easy functions.

Matlab has a handy facility for creating simple functions on the fly, without having to resort to the editor, etc, etc. But it has to be a "one-liner" expression that Matlab can interpret. Try the following:

```
   > f = inline('x^2 + y - sin(x*y)') % what does this define?  see
below
   > f(1)
   > f(1,2)
   > f(2,1)
   > f = inline('x^2 + y - sin(x*y)', 'y', 'x') % what does this change?
see below
   > f(1,2)
```

About subfunctions.

There is only one exception to the "one function, one file" rule. You can create subfunctions in a function file that are visible only to the function in whose file it lives. Sometimes this is handy.

About variables and scope.

Variables defined in the workspace are visible to any command entered in the workspace. Thus, they are visible to any script file that you run. However, they are not visible to functions, which have their own private variables, local in scope. So how can you create a truly "global" variable, visible to the workspace and calling function? You have to make the declaration in both the workspace and in the calling function that the variable is global. Use the command "global varname;". Functions, by contrast, are global. In other words, an accessible function can be called from anywhere: the workspace or within any other function.

Scripts

Input and output.

Matlab has fairly extensive input/output routines, most of which we will not examine. In particular, there are file manipulation routines much in the style of C programming: commands "fopen", "fprintf" and "fclose". You can get help on these if you're interested. We'll just examine the very simple screen input/output functions. Try these commands:

```
   > clear
   > x
   > x = input('Input a value for x:  ');
   > x
   > disp('The value of x is:  '), disp(x)
   > fprintf('%6.3f\n', pi) % what does this do?
   > % now for a different kind of input
   > z = 0:.1:1;
   > plot(z, sin(2*pi*z));
   > [x, y] = ginput(3); %try clicking on the graph at three differ-
ent spots.
   > x,y % so what did this command do?
```

Debugging and profiling.

Matlab has a debugger. We leave exploration of it the the reader. It also has a nice profiler utility. The command “membrane” is a Matlab built-in, written in Matlab. Try the following:

```
> profile on
> A = membrane(1,50);
> profile report
> profile off
```

End of Lesson