

## Math 338: m-files and programming

### 1. Vectorized if statements

Using **if** can be slow. It is more efficient to use vector and matrix multiplication if possible. In example chap03ex09.m, for example, the **for...end** loop and the **if...end** can both be replaced by the following line of code:

```
y = (x>c).*sin(x);
```

The first part of this statement ( $x>c$ ) creates a vector of 1's and 0's. The 1's correspond to each entry where  $x>c$  and the 0's are all the other possibilities. The  $\sin(x)$  statement evaluates  $\sin$  at all of the values of  $x$ . When the two statements are multiplied together, we end up with values of 0 whenever  $x\leq c$  and  $\sin(x)$  whenever  $x>c$ .

*Example:* The following function m-file uses a “vectorized” if and also makes use of the **global** command.

```
function y = chap03ex10
% This is chap03ex10.m
% This is a function m-file with no inputs.
% Instead it uses a global value for x and a

global x a % this says it can use x and a if
           % they are defined in the workspace.
           % If they are not defined then they
           % are initialized as empty sets.

if isempty(x)
    x = [0:0.01:1];
end
if isempty(a)
    a = 0.5;
end

y = (x<a).*sin(x) + (x>=a).*cos(x);
```

Note that this m-file does not have any inputs. It can be called as follows:

```
>> y1=chap03ex10;
>> global x
>> x=[-1:0.2:5];
>> y2=chap03ex10;
```

Note that  $y_1$  and  $y_2$  will have different values. For  $y_1$ , the default value of  $x$  will be used in the m-file. For  $y_2$ , the global value of  $x$  will be used.

## 2. Using **feval** and **eval**

Since strings can be used as inputs for m-files, the next step is actually using a string that represents a function name as an input. This is where the **feval** command is useful.

*Example:* Consider the following simple function m-file that defines the function  $y = \sin(x) + x\cos(x)$ .

```
function y = chap03ex11(x)
% This is chap03ex11.m
```

```
y = sin(x) + x.*cos(x);
```

We can use the **feval** command to evaluate this function.

```
>> x=[0:0.1:1];
>> y=feval('chap03ex11', x);
```

This gives the same result as the command  $y = \text{chap03ex11}(x)$  so why is it so special? Consider the following example.

*Example:* a script m-file with a function name as an input

```
% This is chap03ex12.m
% It should prompt the user for the name of a function
% and then plot that function over the interval [0,1].
```

```
clear % this clears all of the variables
```

```
% Try inputting chap03ex11 here.
funname = input(['What is the name of the function',...
' you would like to plot? '], 's');
```

```
x = [0:0.1:1];
y = feval(funname, x); % here we define y using the name
                       % that has been input into
                       % funname.
```

```
plot(x,y)
title(['This is your plot of ', funname])
```

Note that when prompted for **funname** you can input any function name that you want which takes an input of  $x$ . For example, you could enter  $\sin$ ,  $\cos$ , or  $\exp$  since

these are all m-files that Matlab has defined. Notice that `sin + cos` would not work since there is no m-file called `sin + cos.m` that Matlab has defined.

The **feval** command can also be used with functions that have more than one input.

*Example:* Consider the following simple function m-file that has two inputs:

```
function y = ch3fun13(x, a)
% ch3fun13.m
% inputs: x (vector) and a (scalar)
y = (abs(x)<a).*exp(-a^2./(a^2 - x.^2));
```

Then **feval** can call this function in the following way:

```
>> x=[-1:1:1]; a=10;
>> y=feval('ch3fun13',x,a);
>> plot(x,y)
```

The **eval** command works in a similar fashion except that it will evaluate any string that you give it. This can be useful for giving different variable names within a loop.

*Example:* script m-file using **eval**.

```
% This is chap03ex14.m
% It is a simple loop giving an example
% of the eval command.
clear
y = 1;

for i = 1:5
    y = y*i;
    eval(['factorial0',num2str(i),' = y'])
end
```

### 3. Subfunctions

Just as function m-files contain local variables which are not retained in the general workspace, they may also contain local functions or subfunctions which cannot be accessed from the main workspace (or at least directly).

*Example:* This is a function m-file which can be used to calculate either  $y = \sin(x)$  or  $y = \cos(x)$ .

```
function y = ch3fun15(x, funtype)
% ch3fun15.m -- gives either [1] sin(x) or [2] cos(x)
% inputs: x (vector) and funtype (either 1 or 2)
```

```
%
if nargin < 2
    warning('Enter a function type either 1 or 2')
end
if (funtype~=1)&(funtype~=2)
    warning('funtype should be either 1 or 2')
end

eval(['y = subfun',num2str(funtype),'(x);'])
% either y = subfun1(x) or y = subfun2(x)
% notice the placement of the semi-colon inside the string

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% subfunctions
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function y = subfun1(x)
y = sin(x);

function y = subfun2(x)
y = cos(x);
```