Functions in MATLAB

Function Definitions

There are different ways to organize scripts and functions, but for now every function that we write will be stored in a separate M-file, which is why they are commonly called M-file functions.

A function in MATLAB that returns a single result consists of

- The function header (the first line); this has

– the reserved word function

– since the function returns a result, the name of the output argument followed by the assignment operator =

– the name of the function (Important: This should be the same as the name of the M-file in which this function is stored in order to avoid confusion)

– the input arguments in parentheses; these correspond to the arguments that are passed to the function in the function call

- A comment that describes what the function does (this is printed if help is used)
- The body of the function, which includes all statements and eventually must assign a value to the output argument

The general form of a function definition for a function that calculates and returns one value looks like this:

functionname.m

function outputargument = functionname(input arguments)

% Comment describing the function

Statements here; these must include assigning a value to

the output argument For example, the following is a function called calcarea, which calculates and returns the area of a circle; it is stored in a file called calcarea.m.

calcarea.m

```
function area = calcarea(rad)
% This function calculates the area of a circle
area = pi * rad * rad;
```

A radius of a circle is passed to the function to the input argument rad; the function calculates the area of this circle and stores it in the output argument area. In the function header, we have the reserved word function, then the output argument area followed by the assignment operator =, then the name of the function (the same as the name of the M-file), and then the input argument rad, which is the radius. Since there is an output argument in the function header, somewhere in the body of the function we must assign a value to this output argument. This is how a value is returned from the function. In this case, the function is simple and all we have to do is assign to the output argument area the value of the built-in constant pi multiplied by the square of the input argument rad.

The function can be displayed in the Command Window using the type command.

```
>> type calcarea
function area = calcarea(rad)
% This function calculates the area of a circle
area = pi * rad * rad;
```

Calling a Function

Here is an example of a call to this function in which the value returned is stored in the default variable ans:

>> calcarea(4)

ans =

50.2655

Technically, calling the function is done with the name of the file in which the function resides. In order to avoid confusion, it is easiest to give the function the same name as the filename, so that is how it will be presented in this lectures. In this example, the function name is calcarea and the name of the file is calcarea.m. The result returned from this function can also be stored in a variable in an assignment statement; the name could be the same as the name of the output argument in the function itself but that is not necessary; for example, either of these assignments would be fine:

>> area = calcarea(5)

area =

78.5398

>> myarea = calcarea(6)

myarea =

113.0973

The value returned from the calcarea function could also be printed using

either disp or fprintf:

>> disp(calcarea(4))

50.2655

>> fprintf('The area is %.1f\n', calcarea(4))

The area is 50.3

Notice that the printing is not done in the function itself; rather, the function returns the area and then a print statement can print or display it.

Using help with the function displays the contiguous block of comments under the function header:

>> help calcarea

This function calculates the area of a circle

Calling a User-Defined Function from a Script

Now, we'll modify our script that prompts the user for the radius and calculates the area of a circle, to call our function calcarea to calculate the area of the circle rather than doing this in the script.

script3.m

```
% This script calculates the area of a circle
% It prompts the user for the radius
radius = input('Please enter the radius:');
% It then calls our function to calculate the
% area and then prints the result
area = calcarea(radius);
fprintf('For a circle with a radius of %.2f,',radius)
fprintf('the area is %.2f\n',area)
```

So, the program consists of the script *script3* and the function calcarea. Running this will produce the following:

>> script3

Please enter the radius: 5

For a circle with a radius of 5.00, the area is 78.54

Passing Multiple Arguments

In many cases it is necessary to pass more than one argument to a function. For example, the volume of a cone is given by

$V=1/3\ \pi r^2 h$

where r is the radius of the circular base and h is the height of the cone. Therefore, a function that calculates the volume of a cone needs both the radius and the height:

conevol.m

```
function outarg = conevol(radius, height)
% Calculates the volume of a cone
outarg = (pi/3) * radius * radius * height;
```

Since the function has two input arguments in the function header, two values must be passed to the function when it is called. The order makes a difference.

The first value that is passed to the function is stored in the first input argument (in this case, radius), and the second argument in the function call is passed to the second input argument in the function header.

This is very important: The arguments in the function call must correspond one-to-one with the input arguments in the function header. Here is an example of calling this function. The result returned from the function is simply stored in the default variable ans.

```
>> conevol (4,6.1)
ans =
```

102.2065

In the next example, the result is printed instead with a format of two decimal places.

>> fprintf('The cone volume is %.2f\n', conevol(3, 5.5))

The cone volume is 51.84