Lecture 4

User Defined Functions

and

Files

Getting Help for Functions

You can use the `lookfor` command to find functions that are relevant to your application.

For example, type

```
lookfor imaginary
```

to get a list of the functions that deal with imaginary numbers. You will see listed

```
imag   Complex imaginary part
i      Imaginary unit
j      Imaginary unit
```

User-Defined Functions

```
function [output variables] = name(input variables)
.
.
.
.
.
.
Function body
.
.

end
```

Example

Write a user defined function for

$\boldsymbol{e^{-2x} sin(x+2) - 0.1}$

Solution

```
function [y] = f1(x)
y=exp(-2*x).*sin(2*x);
end
```

Note the use of a semicolon at the end of the lines. This prevents the values of y from being displayed.

Note also the use of the array exponentiation operator (`.^`). This enables the function to accept `y` as an array.

(continued …)

```
>> x=0:0.1:1;
>> y=f1(x);
>> [x' y']
ans =
         0         0
    0.1000    0.1627
    0.2000    0.2610
    0.3000    0.3099
    0.4000    0.3223
    0.5000    0.3096
    0.6000    0.2807
    0.7000    0.2430
    0.8000    0.2018
    0.9000    0.1610
    1.0000    0.1231
```

Example

Write a function to compute area A and circumference C of a circle given its radius r as input

```
function [ A,C] = circle( r )
% r-radius of circle
% A-area of circle
% C-circumference of circle
A=pi*r.^2;
C=2*pi*r;
end
```

(continued …)

TEST:

This function is called as follows:

```
>> clear

>> [A,C]=circle(5)

A =

   78.5398

C =

   31.4159
```

Example

Write a function that computes a falling object’s velocity and distance dropped.

Method 1

```
function[d,v]=drop(t)
%d-distance taveled by falling object (m)
%v-velocity of falling object in (m/sec)
%v0=initial velocity in (m/sec)
% g – acceleration of gravity (m/sec^2);
%t-time in (sec)
% compute velocity of falling object after
t seconds
```

(continued …)

```
v0=3;
g=9.81;
v=v0+g*t;
% compute the distance covered
by the falling object
d=v0*t+0.5*g*t.^2;
end
```

TEST

```
>> [d,v]=drop(5)
d =
  137.6250
v =
   52.0500
```

Method-2

The variable names used in the function definition may, but need not, be used when the function is called:

```
function[d,v]=drop(v0,g,t)
%d-distance taveled by falling object
%v-velocity of falling object
%v0=initial velocity
% g –local acceleration of gravity ;
%t-time in seconds
```

(continued)

```
% compute velocity of falling object after t seconds
v=v0+g*t;
% compute the distance covered by the falling object
d=v0*t+0.5*g*t.^2;
end
```

TEST

(continued …)

```
>> v0=3;
>> t=5;
>> g=9.81;

>> [d,v]=drop(v0,g,t)

d =
  137.6250
v =
   52.0500
```

2. The input variables need not be assigned values outside the function prior to the function call:

```
>> [d,v] = drop(9.81,10,5)

d =
  174.0500

v =
   59.8100
```

3. The inputs and outputs may be arrays:

```
>> t=0:1:4
t =
     0     1     2     3     4
>> [d,v] = drop(9.81,10,t)
d =
      0   14.8100   39.6200   74.4300  119.2400
v =
   9.8100   19.8100   29.8100   39.8100 49.8100
```

Local Variables

The names of the input variables given in the function definition line are local to that function.

This means that other variable names can be used when you call the function.

All variables inside a function are erased after the function finishes executing, except when the same variable names appear in the output variable list used in the function call.

Example

Consider the

```
function[d,v]=drop(t)
```

Here d,v,v0,g,t are local variables to that function. Thay are not stored in the workspace.

Global Variables

The `global` command declares certain variables global, and therefore their values are available to the basic workspace and to other functions that declare these variables global.

The syntax to declare the variables `a`, `x`, and `q` is

```
global a x q
```

Any assignment to those variables, in any function or in the base workspace, is available to all the other functions declaring them global.

EXAMPLE

```
function[d,v]=drop(t)
%d-distance taveled by falling object
(m)
%v-velocity of falling object in (m/sec)
%v0=initial velocity in (m/sec)
% g -acceleration of gravity (/sec^2);
%t-time in (sec)
 global g
```

(continued …)

```
global v0
% compute velocity of falling
object after t seconds
v=v0+g*t;
% compute the distance covered
by the falling object
d=v0*t+0.5*g*t.^2;
end
```

TEST

(continued …)

```
>> clear
>> t=1:1:3;
>> global g
>> g=9.81;
>> global v0
>> v0=10;
>> [d,v]=drop(t)
d =
   14.9050   39.6200   74.1450
v =
   19.8100   29.6200   39.4300
```

Function Handles

You can create a function handle to any function by using the *at* sign, `@`, before the function name. You can then use the handle to reference the function. To create a handle to the function log(x) , define the following function :

```
>> ln=@(x) log(x);
```

(continued …)

The @ symbol alerts MATLAB that

ln is a function

Example

```
>> ln(10)
ans =
    2.3026
```