FF505

Computational Science

MATALB: Functions

Marco Chiarandini (marco@imada.sdu.dk)

Department of Mathematics and Computer Science (IMADA)
University of Southern Denmark

Outline

Outline

User-Defined Functions

The first line in a function file distinguishes a function M-file from a script M-file. Its syntax is as follows:

```
function [output variables] = name(input variables)
```

The function name should be the same as the file name in which it is saved (with the .m extension).

Example

```
function z = fun(x,y)
% the first line of comments is accessed by lookfor
% comments immediately following the definition
% are shown in help
u = 3*x;
z = u + 6*y.^2;
```

```
q = fun(3,7)
q =
   303
```

⇝ variables have local scope

Variable Scope

Local Variables: do not exist outside the function.

```
function z = fun(x,y)
u = 3*x;
z = u + 6*y.^2;
```

```
>> x = 3; y = 7;
>> q = fun(x,y);
>> x
x =
3
>> y
y =
7
>> u
??? Undefined function or variable ’u’.
```

The variables x, y and u are local to the function fun

Local Variables

Local variables do not exist outside the function

```
>>x = 3;y = 7;
>>q = fun(x,y);
>>x
x =
3
>>y
y =
7
>>u
??? Undefined function or variable ’u’.
```

Local Variables

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

In fun.m

```
function z = fun(x,y)
x=x+1; %we increment x but x is local and
       will not change globally
z=x+y;
```

At prompt

```
>> x=3;
>> z=fun(x,4)
>> x
x =
    3
```

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.

Global Variables

The `global` command declares certain variables global: they exist and have the same value in the basic workspace and in the functions that declare them global.

```
function h = falling(t)
global GRAVITY
h = 1/2*GRAVITY*t.^2;
```

```
>> global GRAVITY
>> GRAVITY = 32;
>> y = falling((0:.1:5)');
```

Programming style guidelines recommend against using them.

Parameters and Arguments

Arguments passed by position Only the order of the arguments is important, not the names of the arguments:

```
>> x = 7; y = 3;
>> z = fun(y, x)
z =
   303
```

The second line is equivalent to z = fun(3,7).

Inside the function variables `nargin` and `nargout` tell the number of input and output arguments involved in each particular use of the function

One can use arrays as input arguments:

```
>> r = fun(2:4,7:9)
r =
   300 393 498
```

A function may have no input arguments and no output list.

```
function show_date
clear
clc
today = date
```

Function Handles

- A function handle is an address to reference a function.
- It is declared via the `@` sign before the function name.
- Mostly used to pass the function as an argument to another function.

```
function y = f1(x)
y = x + 2*exp(-x) - 3;
```

```
>> plot(0:0.01:6, @f1)
```

Example: Finding zeros and minima

`X = FZERO(FUN,X0)` system function with syntax:

```
fzero(@function, x0) % zero close to x0
fminbnd(@function, x1, x2) % min between x1 and x2
```

```
fzero(@cos,2)
ans =
   1.5708
>> fminbnd(@cos,0,4)
ans =
   3.1416
```

Ex: plot and find the zeros and minima of $y = x + 2e^x - 3$

To find the minimum of a function of more than one variable

```
fminsearch(@function, x0)
```

where `@function` is the handler to a function taking a vector and x_0 is a guess vector

Other Ways

```
>> fun1 = ’x.^2-4’;
>> fun_inline = inline(fun1);
>> [x, value] = fzero(fun_inline,[0, 3])
```

```
>> fun1 = ’x.^2-4’;
>> [x, value] = fzero(fun1,[0, 3])
```

```
>>[x, value] = fzero(’x.^2-4’,[0, 3])
```

Types of User-Defined Functions

- The primary function is the first function of an M-file. Other are subroutines not callable.
- Subfunctions placed in the file of the primary function, not visible outside the file
- Nested functions defined within another function. Have access to variables of the primary function.
- Anonymous functions at the MATLAB command line or within another function or script

```
% fhandle = @(arglist) expr
>> sq = @(x) (x.^2)
>> poly1 = @(x) 4*x.^2 - 50*x + 5;
>> fminbnd(poly1, -10, 10)
>> fminbnd(@(x) 4*x.^2 - 50*x + 5, -10, 10)
```

- Overloaded functions are functions that respond differently to different types of input arguments.
- Private functions placed in a `private` folder and visible only to parent folder

Outline

Function Arguments

Create a new function in a file named addme.m that accepts one or two inputs and computes the sum of the number with itself or the sum of the two numbers. The function must be then able to return one or two outputs (a result and its absolute value).