Principles of Technical Computing

Lecture 4 – User-defined Functions and Function Files in Matlab

Matylda Jabłońska-Sabuka

Lappeenranta University of Technology

Overview of Week 5

- Generally about user-defined functions and function files
- Structure of a function file
- Calling (using) a user-defined function
- Anonymous functions
- Function functions
- Subfunctions and nested functions
- Examples

Generally about user-defined functions and function files

Functions can be defined either as separate files, or at the end of a script file.

The way how functions are structured is very important.

A function consists of:

- function definition line
- function body
- `end`-command if the function is within a script file or if it's a nested/subfunction.

Structure of a function file

A function consists of:

- function definition line
- function body
- `end`-command if the function is within a script file or if it's a nested/subfunction.

Function **definition line**

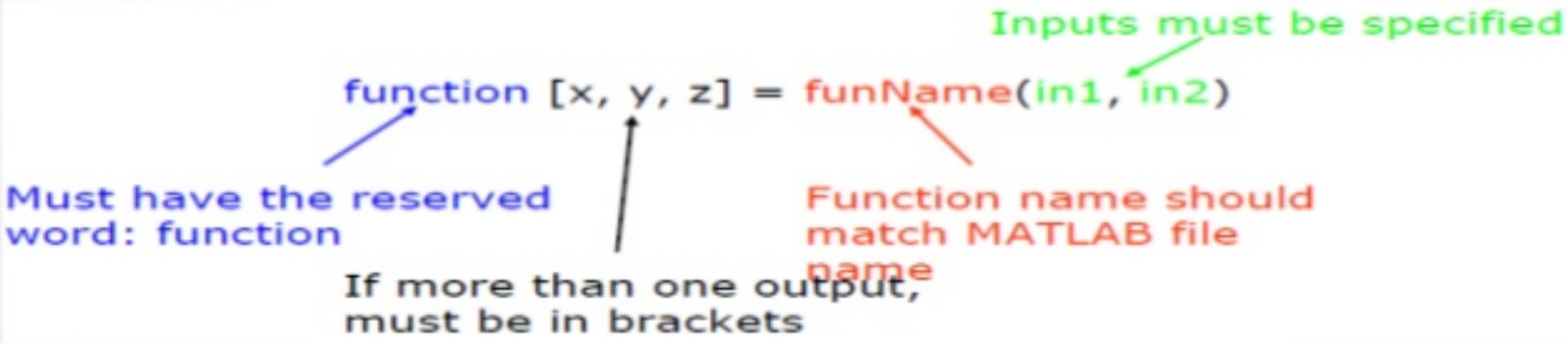
- defines the file as a function file
- defines the name of the function
- defines the number and order of the input and output arguments

MATLAB Programing

User-defined Functions:

Functions look exactly like scripts, but for ONE difference

Functions must have a function declaration.



No need for return :

MATLAB 'returns' the variables whose names match those in the function declaration.

Comparison between script and function files

- both are saved with extension `.m`
- first line of a function file has to be the function definition line
- the variables in a function are always local, unless otherwise specified by the programmer; the variables in a script file are recognized in a Command Window
- script files can use variables that have been defined in the workspace
- function files can accept data through input arguments or have variables defined directly in itself, and can return data through output arguments

```
function [output_arg] = function_name(input_arg)
```

- the word `function` must be the first word, always in lower-case letters
- `function_name` should always be the same as function file name
- the rules for naming functions are the same as for naming variables
- the order of input and output arguments matters while calling the function

Input and output arguments

The input and output arguments can be multiple.

- `function [mpay,tpay] = loan(amount,rate,years)`
  - three input arguments, two output arguments
- `function [A] = RectArea(a,b)`
  - two input arguments, one output argument
- `function A = RectArea(a,b)`
  - same as above
- `function [V,S] = SphereVolArea(r)`
  - one input argument, two output arguments
- `function trajectory(v,h,g)`
  - three input arguments, no output arguments

Function **body**

The function body contains the computer program (code) that actually performs the computations.

The code can use all MATLAB programming features, which you would normally use in script files. This includes calculations, assignments, any built-in or user-defined functions, flow control (conditional statements and loops), comments, blank lines, and interactive input/output.

Calling (using) a user-defined function

Using a user-defined function

Assume that we want to write a function that will calculate the account balance after we have deposited the amount of money m for a period of n years with interest year r.
The MATLAB function `myMoney.m` would look as follows:

```
function balance = myMoney(m,n,r)
balance = m*(1+r)^n;
```

where `balance` is the function output.

To call the function for $m = 10000$, $n = 15$ and $r = 4\%$, and save the result as variable B we write:

```
m = 10000; n = 15; r = 0.04
B =myMoney(m,n,r)
```

What is wrong in this code?

```
m = 10000; n = 15; r = 0.04
B = myMoney(m,n,r)

function myMoney(m,n,r)
balance = m*(1+r)^n;
end

??? Error using ==> myMoney
Too many output arguments.
```

What is wrong in this code?

```
B0 = 0; m = 10000; n = 15; r = 0.04;
B = myMoney(B0,m,n,r)

function balance = myMoney(m,n,r)
balance = m*(1+r)^n;
end

???  Error using ==> myMoney
Too many input arguments.
```

Local and global variables

All the variables in the function file are local (the input and output arguments and any other variables that are assigned within the function file).

These variables are then used in the calculations within the function file and will never be confused with variables called with the same name in the main script file or other functions.

It is possible, however, to make a variable common (recognized) in several different function files, and perhaps in the workspace too. This is done with declaring the variable global, both in the function file (after the definition line) and in the script file:

```
global variable_name
```

Anonymous functions

Anonymous functions

An anonymous function is a simple (one-line) user-defined function that is defined without creating a separate function file. Anonymous functions can be constructed in the Command window, within a script file, or inside a regular user-defined function.

An anonymous function is created as follows:

```
name = @(arglist) expr
```

A simple example is the balance calculation function:
Assume that we want to write a function that will calculate the account balance after we have deposited the amount of money m for a period of n years with interest year r:

```
myMoney = @(m,n,r) m*(1+r)^n;
```

To get the result we call the function same way as if it was in classical form:

```
m = 10000; n = 15; r = 0.04;
B = myMoney(m,n,r)
```

or directly

```
B = myMoney(10000,15,0.04)
```

Function functions

Function functions

There are many situations where a function (Function A) works on another function (Function B). This means that when Function A is executed it has to be given Function B. A function that accepts another function is called in MATLAB a function function.

A function function, which accepts another (imported) function, includes in its arguments a name that represents the imported function. The imported function name is used for the operations in the program (code) of the function function.

Using function handles for passing functions into functions

Assume we want to find minimal value of a given function of one variable, for instance:

$$f(x) = x^2 - 3x + 2$$

Matlab has a ready function which can be used for minimum search `fminsearch`.

```
x = fminsearch(fun,x0)
```

`x` – is the found x-coordinate of the minimum point
`fun` – is the function (mathematical formula) of which we are searching the minimum
`x0` – is the starting point for Matlab to start the search (initial guess)

We need to define our formula to be a function file, for instance

```
function y = myfunction(x)
y = x.^2 - 3*x + 2
```

Then in the script file we call

```
xmin = fminsearch(@myfunction,4)
```

and Matlab returns `xmin = 1.5`

We could have also defined the formula as an anonymous function directly in the script, and then call it within `fminsearch` without the handle:

```
f = @(x) x.^2 - 3*x + 2
xmin = fminsearch(f,4)
```

Subfunctions and nested functions

Subfunctions

- A function file can contain more than one user-defined function.
- The functions are typed one after another.
- Each function begins with a function definition line.
- The first function is called the primary function and the rest of the functions are called subfunctions.
- The name of the function file should correspond to the primary function name.
- Each of the functions in the file can call any other function in the file.

```
function [me SD] = stat(v)          <- primary function
n=length(v);
me=AVG(v);
SD=StandDiv(v,me);

function av=AVG(x)                  <- subfunction
av=sum(x)/length(x)

function Sdiv=StandDiv(x,xAve)      <- subfunction
n=length(x);
xdif=x-xAve;
xdif2=xdif.^2;
Sdiv=sqrt(sum(xdif2)/(n-1));
```

Nested functions

A nested function is a user-defined function that is written inside another user-defined function. The portion in the code that corresponds to the nested function starts with a function definition line and ends with an `end` statement. An `end` statement must be also entered at the end of the function that contains the nested function.

```
function y=A(a1,a2)
........
    function z=B(b1,b2)
    ........
    end
........
end
```

```
function [me SD]=statNest(v)    <- the primary function
n=length(v);
me=AVG(v);

    function av=AVG(x)                  <- nested function
    av=sum(x)/n;
    end

    function Sdiv=StandDiv(x)           <- nested function
    xdif=x-me;
    xdif2=xdif.^2;
    Sdiv=sqrt(sum(xdif2)/(n-1));
    end

SD=StandDiv(v);
end
```

Example 1

Write a function which removes elements greater than `t` from an arbitrary vector `x`, that is such that "throws away" elements equal to `t` and returns a new same or shorter vector.

For instance, for `t=5` and array `x = [1 3 5 3 4 7 5 6 5 4]` it would return `y = [1 3 5 3 4 5 5 4]`.

Example 2

Write another function which for the same inputs `t` and `x` replaces with `t` those elements of `x` which are less than `t`.

For instance, for the same inputs `t` and `x` it would return `y = [5 5 5 5 5 7 5 6 5 5]`