

# Functions and scripts

Eugeniy E. Mikhailov

The College of William & Mary



Lecture 05

# Matlab functions

Used for separating often called code

```
function [out1, out2, ..., outN] = func_name (arg1, arg2, ..., argN)
    function body
    set of expressions
end
```

```
function h=hypotenuse(cathetus1, cathetus2)
% calculates hypotenuse for a right angle triangle
% inputs are the length of the catheti:
% cathetus1 and cathetus2
    h=sqrt(cathetus1^2+cathetus2^2);
end
```

We can either type it in the Command Window but better save it to the file *hypotenuse.m*

```
>> c=hypotenuse(3,4)
c =
    5
```

# Function self documentation

```
function h=hypotenuse(cathetus1, cathetus2)
% calculates hypotenuse for a right angle triangle
% inputs are the length of the catheti:
% cathetus1 and cathetus2
    h=sqrt(cathetus1^2+cathetus2^2);
end
```

```
>> help hypotenuse
```

```
calculates hypotenuse for a right angle triangle
inputs are the length of the catheti:
cathetus1 and cathetus2
```

# Function with multiple output

```
function [pos,neg]=pos_neg_sum(x)
% calculates sum of positive and negative elements
% of the input vector
    pos=sum(x(x>0));
    neg=sum(x(x<0));
end
```

```
>> v=[1,2,-2,3,-5]
v =
     1     2    -2     3    -5
```

```
>> [p,n]=pos_neg_sum(v)
```

```
p =
     6
n =
    -7
```

```
>> y=pos_neg_sum(v)
```

```
y =
     6
```

# Local space of variables in functions

```
function [pos,neg]=pos_neg_sum(x)
% calculates sum of positive and negative elements
% of the input vector
pos=sum(x(x>0));
neg=sum(x(x<0));
end
```

```
>> pos=23;
>> x=[1,-1,-1];
>> v=[1,2,-2,3,-5];
```

```
[p,n]=pos_neg_sum(v)
```

```
p =
    6
n =
   -7
```

```
>> pos
pos =
    23
```

```
>> x
x =
    1  -1  -1
```

# Recursion: function calls itself

Canonical example: factorial

$$N! = N \times (N - 1) \times (N - 2) \cdots 3 \times 2 \times 1$$

We can rewrite it as

$$N! = N \times (N - 1)!$$

Notice that  $0! = 1$

# Recursion for factorial

```
function f=myfactorial(N)
% calculated factorial of the input. N!=N*(N-1)!
% input must be integer larger or equal to zero

% ALWAYS sanitize the input !!!
if ( N<0 )
    error('wrong input, input must be >= 0');
end
if ( N~=floor(N) )
    error('input is not integer number');
end

% -----
if ( N==0 )
    f=1; return;
end
f=N*myfactorial(N-1);
end
```

# Scripts

Script is the sequence of the matlab expressions written in the file.

```
N=1:N_max;  
M=0*(N);  
for i=N  
    M(i)=(1+x/i)^i;  
end  
plot(N,M,'-');  
xlabel('N, number of payments per year');  
ylabel('Money grows');  
title('Money grows vs number of payments per year');
```

Let's save it to the file

*money\_grows.m*

Now we can assign any N\_max  
and x, then execute the script

```
>> N_max=4; x=.5;  
>> money_grows;  
>> M  
M =  
1.5000    1.5625  
1.5880    1.6018
```



# Scripts variable space

Unlike functions **scripts modify Workspace variables**

```
N=1:N_max;  
M=0*(N);  
for i=N  
    M(i)=(1+x/i)^i;  
end  
plot(N,M,'-');  
xlabel('N, number of payments per year');  
ylabel('Money grows');  
title('Money grows vs number of payments per year');
```

```
>> M=123; x=.5;  
>> N_Max=2; money_grows;  
>> M
```

```
M =  
1.5000    1.5625
```

Think about script as it is a keyboard macro. Calling script is equivalent to typing the scripts statements from the keyboard.

# Saving your results

Let's say you have calculated some intermediate results and want to save them.

Not surprisingly it is done with `save` command. It can be called in several different ways.

- command form

```
save 'filename.mat'
```

- functional form

```
save ('filename.mat' )
```

- saves all workspace variables to the file 'filename.mat'

To save only var1, var2, and var3

- `save 'filename.mat' var1 var2 var3`

- `save ('filename.mat' , 'var1' , 'var2' , 'var3');`

- `fname='saved_variables.mat' ; save (fname, 'var1' , 'var2' , 'var3');`

notice the use of apostrophes

i.e. `save` as a function expect strings for the arguments.

# Loading your results

Now you want your results back to the workspace

It is done with `load` command. It can be called in several different ways.

- command form

```
load 'filename.mat'
```

- functional form

```
load ('filename.mat' )
```

- loads all workspace variables to the file 'filename.mat'

To load only var1, var2, and var3

- `load 'filename.mat' var1 var2 var3`

- `load ('filename.mat' , 'var1' , 'var2' , 'var3');`

- `fname='load_variables.mat' ; load (fname, 'var1' , 'var2' , 'var3');`

notice the use of apostrophes

i.e. `load` as a function expect strings for the arguments.