

Functions and scripts

Eugeniy E. Mikhailov

The College of William & Mary



Lecture 04

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,'-'); set(gca,'FontSize',24);
xlabel('N, number of payments per year');
ylabel('M_n, return on investment'); % note M_n use
title('Return on investment vs number of payments');
```

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,'-'); set(gca,'FontSize',24);  
xlabel('N, number of payments per year');  
ylabel('M_n, return on investment'); % note M_n use  
title('Return on investment vs number of payments');
```

Let's save it to the file

```
return_on_investment.m
```

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,'-'); set(gca,'FontSize',24);  
xlabel('N, number of payments per year');  
ylabel('M_n, return on investment'); % note M_n use  
title('Return on investment vs number of payments');
```

Let's save it to the file

return_on_investment.m

Now we can assign any N_{max}
and x , then execute the script

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,'-'); set(gca,'FontSize',24);  
xlabel('N, number of payments per year');  
ylabel('M_n, return on investment'); % note M_n use  
title('Return on investment vs number of payments');
```

Let's save it to the file

return_on_investment.m

Now we can assign any N_{max}
and x , then execute the script

```
>> N_max=4; x=.5;  
>> return_on_investment;  
>> M  
M =  
1.50    1.56    1.58    1.60
```

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,'-'); set(gca,'FontSize',24);  
xlabel('N, number of payments per year');  
ylabel('M_n, return on investment'); % note M_n use  
title('Return on investment vs number of payments');
```

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

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,'-'); set(gca,'FontSize',24);  
xlabel('N, number of payments per year');  
ylabel('M_n, return on investment'); % note M_n use  
title('Return on investment vs number of payments');
```

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

```
M =  
1.5000    1.5625
```

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

Matlab functions

Used for separation of a meaningful chunk of code

```
function [out1, out2, ..., outN] = func_name (arg1, arg2, ..., argN)  
    % optional but strongly recommended function description  
    set of expressions of the function body  
  
end
```


Matlab functions

Used for separation of a meaningful chunk of code

```
function [out1, out2, ..., outN] = func_name (arg1, arg2, ..., argN)  
    % optional but strongly recommended function description  
    set of expressions of the function body
```

```
end
```

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

Function must be saved into separate name with filename matching function name and extension `m`. In our case it is `hypotenuse.m`

Matlab functions

Used for separation of a meaningful chunk of code

```
function [out1, out2, ..., outN] = func_name (arg1, arg2, ..., argN)  
    % optional but strongly recommended function description  
    set of expressions of the function body
```

```
end
```

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

Function must be saved into separate name with filename matching function name and extension *m*. In our case it is *hypotenuse.m*

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

Function self documentation

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

Function self documentation

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

```
>> help hypotenuse
```

Function self documentation

```
function h=hypotenuse(cathetus1, cathetus2)
% Calculates hypotenuse of 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 of 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
```

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)
```

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
```


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
```

If you ask for less it will return the first in the list value i.e. pos

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

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
```

If you ask for less it will return the first in the list value i.e. pos

```
>> 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
```

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)
```

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
```

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
```

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

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

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)
% Calculates factorial of the input. N!=N*(N-1)!
% Input must be an integer larger or equal to zero.

if ( N < 0 ) % ALWAYS sanitize the input !!!
    error('wrong input, input must be >= 0');
end
if ( N ~= floor(N) )
    error('input is not an integer number');
end
% Once input is good we can calculate the factorial
if ( N==0 )
    f=1; return; % return stops the evaluation
end
f=N*myfactorial(N-1);
end
```

Saving your results

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

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'

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.

Saving: couple words of wisdom

By default Matlab saves into a binary format specific to Matlab. If you work with Matlab only it is fine.

But I personally do not like formats which are not human (simple text editor/viewer) readable.

To generate human readable format you can use `-ascii` switch when saving but such notation **drops the variable name** from the file.

So do not use `-ascii` switch to save multiple variables, **save only one variable per file**

This can be done in multiple ways

- `save -ascii 'filename.mat' var1`
- `save('filename.mat', '-ascii', 'var1')`
- `fname='saved_var1.mat';`
`save(fname, '-ascii', 'var1');`

Loading your results

Now you want your results back to the workspace

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.

To load all saved variables from the file `'filename.mat'`

- command form

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

- functional form

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


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.

To load all saved variables from the file `'filename.mat'`

- command form

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

- functional form

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

To load only `var1` and `var3`

- `load 'filename.mat' var1 var3`

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

- `fname='variables.mat';`

```
load(fname, 'var1', 'var3');
```

notice the use of apostrophes, `load` as a function expects **strings** for its arguments.

Loading your results from ASCII files

If you use my advise and save human readable files. Than loading is a bit more evolved since you would need to provide a variable to assign by value stored in the file. But it is easy

- `var1=load('-ascii', 'file_with_var1.mat')`
- or you could be fancy and assign stored data to another variable
`counter=load('-ascii', 'file_with_var1.mat')`

Nice side effect is that you can prepare such data file yourself or by the other software and than load it into Matlab.

Data import of complex data formats

Often you need to import data from other sources. One example would be to read Excel data files which your less tech savvy colleagues might send to you.

- `load` is often smart enough
- Otherwise right click on a data file in the `Current Folder` tab and chose `Import Data`.
 - Notice handy check mark `Generate Matlab code` for the case where you have many similarly structured files to be imported.

This is general recipe, though Matlab has some built-ins to read Excel files in particular.