

Introduction to MATLAB

Yuichiro Waki

July 24, 2007

Why do we use MATLAB?

- Relatively easy.
- You have an access to it in this computer lab.
- A free, similar software is available (Octave).
- But for serious research, need higher-level languages e.g. fortran 90, C++.

Introduction to MATLAB

- Starting MATLAB
- MATLAB command window
- Basic MATLAB programming

Fundamental operations and built-in variables/functions

- Fundamental operations

mathmatics	MATLAB
$1 + 2$	<code>1 + 2</code>
$1 - 2$	<code>1 - 2</code>
1×2	<code>1 * 2</code>
$1 \div 2$	<code>1 / 2</code>
2^2	<code>2^2</code>

- There are built-in functions like `sqrt`, `log`, `exp`, `sin` etc.
- There are built-in variables like `pi`, etc.

Some examples

```
>> pi^2
```

```
ans =
```

```
9.8696
```

```
>> sqrt(ans)
```

```
ans =
```

```
3.1416
```

The result of calculation is stored with the name `ans`, but is replaced in each calculation.

Creating variables

You can create new variables with your favorite names.

```
>> x = pi^2
```

```
x =
```

```
9.8696
```

```
>> X = sqrt(x)
```

```
X =
```

```
3.1416
```

(Look at the workspace, and you'll find variables `ans`, `x`, and `X`.)

Some examples

When you are not interested in intermediate results, you can suppress results using semicolons.

```
>> x = pi^2;  
>> X = sqrt(x);  
>> x/X
```

```
ans =
```

```
3.1416
```

Vectors and Matrices

- The name MATLAB is taken from "MATrix LABoratory." It is good at dealing with matrices.
- Actually all variables in MATLAB are matrices. (Scalars are 1-by-1 matrices and vectors are N-by-1 (or 1-by-N) matrices.)
- You can see this by executing `>> size(x)`.

Creating matrices: Some examples

You can manually create matrices:

```
>> a = [2,1]
```

```
a =
```

```
     2     1
```

```
>> b = [1; 2]
```

```
b =
```

```
     1
```

```
     2
```

```
>> A = [1, 2, 4; 9, 11, 21]
```

```
A =
```

```
     1     2     4
```

```
     9    11    21
```

Creating matrices: Some examples

You can assign new value to an element of matrix:

```
>> A(1,1) = 3;
```

```
>> A
```

```
A =
```

```
     3     2     4
     9    11    21
```

Creating matrices: Some examples

Some built-in functions create special kinds of matrices:

```
>> B = zeros(2,4)
```

```
B =
```

```
    0    0    0    0
    0    0    0    0
```

```
>> D = ones(2,5)
```

```
D =
```

```
    1    1    1    1    1
    1    1    1    1    1
```

```
>> C = eye(2)
```

```
C =
```

```
    1    0
    0    1
```

Creating matrices: Some examples

```
>> c = linspace(2,20,5)
```

```
c =
```

```
    2.0000    6.5000   11.0000   15.5000   20.0000
```

```
>> d = 2:2:11
```

```
d =
```

```
    2    4    6    8   10
```

```
>> e = 3:5
```

```
e =
```

```
    3    4    5
```

3:5 is equivalent to 3:1:5.

Creating matrices: Some examples

You can append a matrix to another.

```
>> [e, d]
```

```
ans =
```

```
    3    4    5    2    4    6    8   10
```

```
>> [c;d]
```

```
ans =
```

```
 2.0000    6.5000   11.0000   15.5000   20.0000  
 2.0000    4.0000    6.0000    8.0000   10.0000
```

Matrix operations

```
>> a*b
```

```
ans =
```

```
4
```

```
>> b*a
```

```
ans =
```

```
2    1
```

```
4    2
```

```
>> a+b'
```

```
ans =
```

```
3    3
```

Matrix operations

```
>> c  
c =  
    2.0000    6.5000   11.0000   15.5000   20.0000
```

```
>> c(1,3)  
ans =  
    11.0000
```

```
>> c(1,[1,3,5])  
ans =  
    2.0000   11.0000   20.0000
```

```
>> C(:,1)  
ans =  
    1
```

Warning!

Avoid using existing function's (and built-in variable's) name for matrices you create:

```
>> log(1)
ans =
     0
```

```
>> log = [1,2]
log =
     1     2
```

```
>> log(1)
ans =
     1
```

Clearing variables

To use log function again, you need to clear the variable "log."

```
>> clear log  
>> log(1)
```

```
ans =
```

```
0
```

Check the workspace (or execute the command `whos`) to find there is no variable "log" any longer.

Character Strings

```
>> ex = 'This is an example.'  
ex =  
This is an example.
```

Note that `ex` is a row vector. To see this,

```
>> size(ex)  
ans =  
     1     19
```

```
>> ex(1:6)  
ans =  
This i
```

Array operations

Element-by-element operations are possible.

```
>> a
a =
     2     1
>> a.*a
ans =
     4     1
>> a./a
ans =
     1     1
>> a.^a
ans =
     4     1
```

Vectorization and Plotting

```
>> c
c =
    2.0000    6.5000   11.0000   15.5000   20.0000

>> f = log(c)
f =
    0.6931    1.8718    2.3979    2.7408    2.9957

>> plot(c,f)
```

Did a figure window open?

Need help?

When you know the name of a built-in function/variable but don't know what it is/does, use `help functionname`.

```
>> help sqrt
```

When you don't know the name of a built-in function/variable but have a keyword in your mind, use `lookfor keyword`.

```
>> lookfor square
```

m-files

- For most purposes you need to execute a long, complicated series of commands. m-files facilitate that.
- File > New > M-File
- First write a series of commands on an m-file and then execute the file.

Using loops

- for loop

```
for k = 1:100
    g(k) = log(k);
end
```

- while loop

```
k = 1;
while k<=100
    h(k) = log(k);
    k = k+1;
end
```

User-defined functions

User-defined functions are useful for the following purposes:

- To write subroutines which are general and applicable to many problems (e.g. numerical differentiation),
- To make a program detail-free, i.e. free from a specification you are currently using (e.g. return function in a dynamic programming problem)

User-defined functions

A user defined function is an m-file `functionname.m` written in the following syntax:

```
function [outputlist] = functionname(inputlist)
```

```
output1 = ...
```

```
output2 = ...
```

User-defined functions: examples

OLS estimates $\hat{\beta}$ and variance estimates s^2 for given data (y, X) :

```
function [beta,s2] = ols(y,X)
```

```
[N,K] = size(X);  
beta = (X'*X)\(X'*y);  
mu = X*beta;  
s2 = (y-mu)'*(y-mu)/(N-K);
```

Save this as `ols.m`. In the command window do the following:

```
y = rand(10,1);  
X = rand(10,2);  
[beta,s2] = ols(y,X)
```

`rand(n,m)` generates n-by-m matrix with i.i.d. r.v.'s $\sim U[0,1]$.
(You may need to change the working directory to where you saved the file.)

User-defined functions: examples

A utility function $u(c; \sigma) = (c^{1-\sigma} - 1)/(1 - \sigma)$:

```
function util = u(c,sigma)
```

```
util = (c^(1-sigma)-1)/(1-sigma);
```

When $\sigma = 1$, we want to use `log`, but the above code gives an error. (Division by 0.)

If statement

You can avoid it using if statement

```
function util = u(c,sigma)

if sigma == 1
    util = log(c);
else
    util = (c^(1-sigma)-1)/(1-sigma);
end
```

If statements: syntax

```
if (condition 1)
    (operation a)
elseif (condition 2)
    (operation b)
else
    (operation c)
end
```

Use comments

Whatever written in the same line after % is not executed (ignored). You can exploit this for putting comments. To avoid confusion, use comments effectively.

```
function [beta,s2] = ols(y,X)
```

```
% This function ols computes OLS estimates  
% for beta (coefficients vector)  
% and s^2 (variance) given data (y,X).
```

```
[N,K] = size(X);  
beta = (X'*X)\(X'*y); % OLS estimates for coef. vector  
% beta = inv(X'*X)*X'*y; % alternative way  
mu = X*beta; % fitted value  
s2 = (y-mu)'*(y-mu)/(N-K); % estimates for sigma^2
```

Making output neat

Use `disp` command. Compare

```
>> A
```

and

```
>> disp('A is')
```

```
>> disp(A)
```