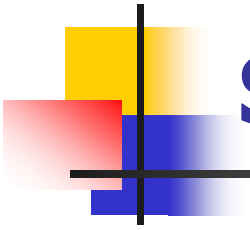
Chapter 2: Basic MATLAB

Yunong Zhang (张雨浓)

Email: zhynong@mail.sysu.edu.cn

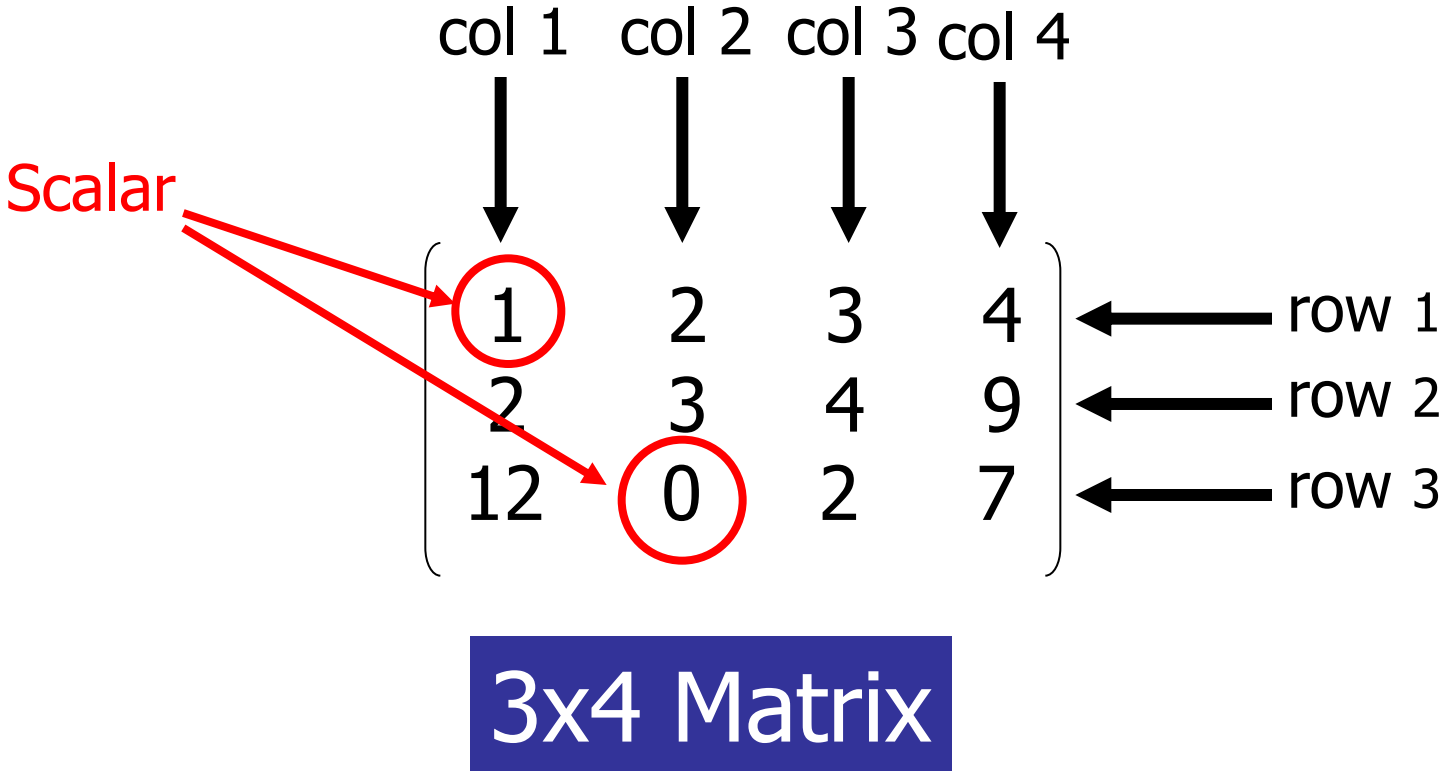


Some facts for a first impression

- **Everything in MATLAB is an array !**
- **again: MATLAB is an interpreted language, no compilation needed (but possible)**
- **again: MATLAB does not need any variable declarations, no dimension statements, has no packaging, no storage allocation, no pointers**
- **Programs can be run step by step, with full access to all variables, functions etc.**

Matrix

One major difference between university-level and high-school-level knowledge!!



3x4 Matrix

It is a collection of data values (scalars) organized into rows and columns.

Matrix (cont.)

- Size / Dimension

  Specified by the number of rows and the number of columns

3x4 Matrix

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 9 \\ 12 & 0 & 2 & 7 \end{pmatrix}$$

The number of elements of a matrix
= (number of rows) $\times$ (number of columns)

Multidimensional Arrays

- Matlab can generate arrays with as many dimensions as you want.

Array c: 2×3×2

```
>> c(:,:,1)=[1 2 3;4 5 6];
>> c(:,:,2)=[7 8 9;10 11 12];
>> whos c
  Name      Size                Bytes  Class

  c         2x3x2                  96  double array

Grand total is 12 elements using 96 bytes
```

cf. who c

Multidimensional Arrays (Cont.)

```
>> c(:,:,1)=

     1     2     3
     4     5     6

>> c(:,:,2)

     7     8     9
    10    11    12
```

Matrix / Array Storage

```
>> a=0;
>> who a

Your variables are:

a

>> whos a
  Name        Size                Bytes  Class

  a           1x1                     8  double array

Grand total is 1 elements using 8 bytes
```

How many bytes does a 1000x1000 matrix need?

How many bytes does a 10000x10000 matrix need?

How many bytes does a 20000x20000 matrix need?

Matrix / Array Storage

In standard implementations,

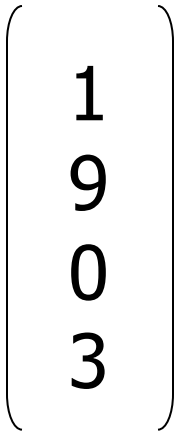
- the **computational complexity is O(N^3) operations,**
- the **storage complexity is O(N^2) operations**.

Table 1: Memory requirement of a matrix in double precision versus its dimension N

Dimension N	1000	3000	5000	7000
Storage (MB)	7.7	68.7	190.8	373.9

...	9000	11000	13000	15000	17000
	618.0	923.2	1289.4	1716.6	2204.9

Vector



Column Vector
4 X 1

Row Vector
1 X 3

Naming variables

- Variable's name **MUST** begin with a letter, followed by any combination of letters, numbers, and the underline (_) character.

- The length of a variable's name **SHOULD** be equal or less than 31.

  Only the first 31 characters are significant.

  If more than 31 characters are used, the extra characters will be ignored.

Examples of Variable Names

- Illegal:

  4ab, _ab

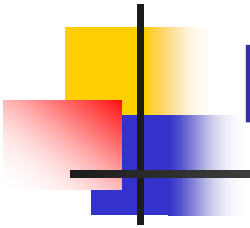
- Legal:

  ab4, ab_

Good Programming Practice

A descriptive and easy-to-remember name is preferred.

Adding comments in the header of a program is preferred.

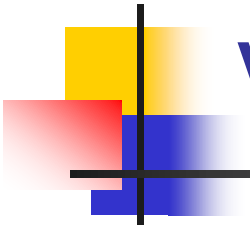


Remember in mind

- The Matlab language is case-sensitive.

Uppercase and lowercase are not the same.

Variables *name, Name,* and *NAME* are all different.



Variables' Initialization

- **Three** common methods to initialize a variable

1. Input data into the variable from the keyboard;
2. Assign data the variable in an assignment statement;
3. Read data from a file.

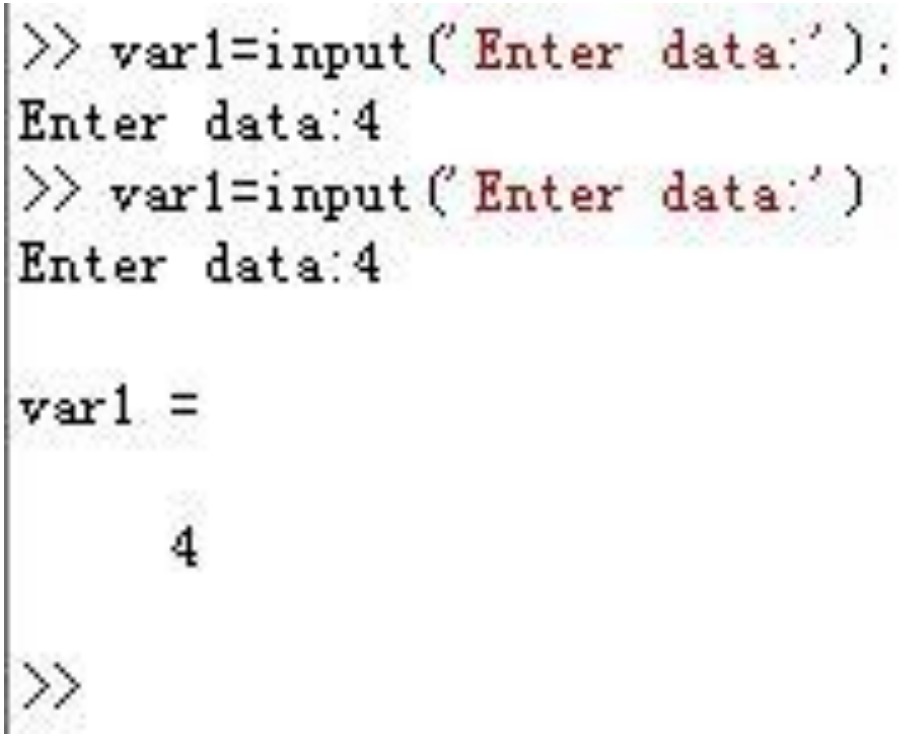
Initializing Variables via Keyboard

- *input* function

```
>> var1=input('Enter data:');
   Enter data:4
>> var1

var1 =

     4
```



Initializing Variables via Keyboard (cont.)

```
>> var2=input('Enter data:','s');
    Enter data:It is OK!
>> var2

var2 =

It is OK!
```

Initializing Variables via the Assignment Statements

- An assignment statement has the general form:

var=expression

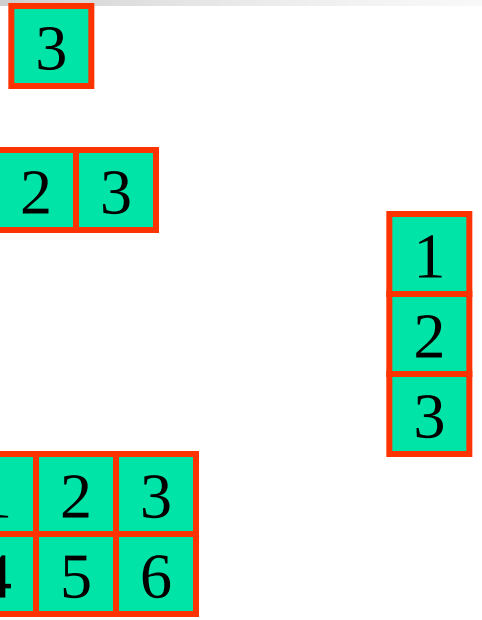
```
X=1;
Y=2;
Var=X/2;
Array=[1 2 3 4];
```

What if there is no semicolon?

comma, colon: semicolon; period.

Initializing Variables via the Assignment Statements (Cont.)

- Array=[3];
- Array=[1 2 3];
- Array=[1,2,3];
- Array=[1;2;3];
- Array=[1,2,3;4 5 6];



- Array=[1,2,3
  4 5 6];

which creates a 2x3 matrix. The end of the first line terminates the first row naturally.

Initializing Variables via the Assignment Statements (Cont.)

- Array=[1,2,3;4 5]; What will happen?

Illegal!

```
>> Array=[1,2,3;4 5 ];
??? Error using ==> vertcat
All rows in the bracketed expression must have the same
number of columns.
```

- Note:

  All rows have the same number of columns.

  All columns have the same number of rows.

Vertical concatenation

Initializing Variables via Shortcut Expressions

- Set all elements of an m*n matrix to zero

```
m=2;
n=3;
array(2,3)=0;
```

```
>> m=2;
   n=3;
   array(2,3)=0;
>> array

array =

   0   0   0
   0   0   0
```

```
>> m=2;
   n=3;
   array(2,3)=1;
>> array

array =

   0   0   0
   0   0   1
```

Initializing Variables via Shortcut Expressions (Cont.)

- Colon operator (:)

start_value:step_increment:end_value

```
>> x=1:2:10

x =

     1     3     5     7     9
```

Initializing Variables via Shortcut Expressions (Cont.)

- angles=0:1/3*pi:2*pi;
- angles=0:(1/3*pi):(2*pi);
- Note that, if the step_increment is 1, it may be omitted.

```
>> x=1:5

x =

     1     2     3     4     5
```

Initializing with Built-in Functions

- zeros(n); Generates an n*n matrix of zeros
- zeros(m,n); Generate an m*n matrix of zeros
- ones(n); Generate an n*n matrix of ones
- ones(m,n); Generate an m*n matrix of ones
- eye(n); Generate an n*n identity matrix
- size(array); Returns the numbers of rows and columns of input-argument *array*

Square matrix vs. rectangular matrix

Updating Arrays

Array2 =

```
    1     2     3     4
    5     6     7     8
    9    10    11    12
```

>>Array2+2

Vague usage!

```
ans =
     3     4     5     6
     7     8     9    10
    11    12    13    14
```

Updating Arrays (Cont.)

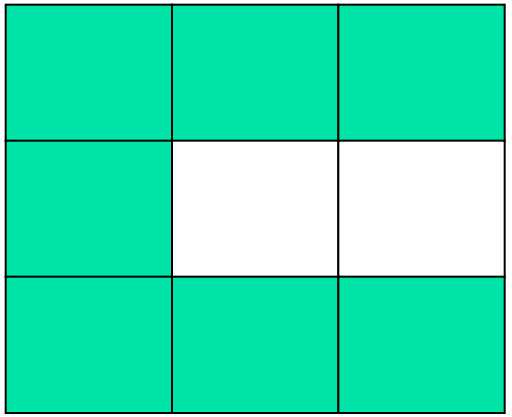
```
>> Array2=[20 21;22 23];

Array2 =

    20    21
    22    23
```

Sub-Arrays

- Select a portion of a matrix



Array =

1 2 3
4 5 6

Array =

1 2 3
4 5 6

Array(:,1)

Array =

1 2 3
4 5 6

Array(1,:)

The *end* Function

- It is used to generate array subscripts.

```
Array=[1 2 3 4 5 6];
Array(3:end)                                    3    4    5    6
Array2=[1 2 3 4;5 6 7 8;9 10 11 12];
Array2(2:end,2:end)
```

```
Array2 =
     1     2     3     4
     5     6     7     8
     9    10    11    12
```

```
ans=
     6     7     8
    10    11    12
```

What if Array2(1:end,1:end)?

Vague usage!

Updating Sub-Arrays

Array2 =

1	2	3	4
5	6	7	8
9	10	11	12

20	21
22	23

Array2(1:2,[1 4])=[20 21;22 23];

Array2 =

20	2	3	21
22	6	7	23
9	10	11	12

Updating Sub-Arrays (Cont.)

- Assigning a scalar to a sub-array

Array2 =

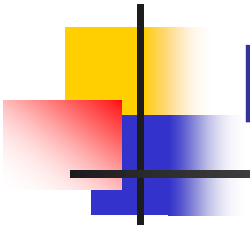
1	2	3	4
5	6	7	8
9	10	11	12

1

```
>> Array2(1:2,[1 4])=[1 1;1 1];
>> Array2(1:2,[1 4])=1;
```

Vague usage!

1	2	3	1
1	6	7	1
9	10	11	12



Remember in mind

- When updating a sub-array, only those involved values are updated, and all other values in the array remain unchanged.

- When updating an array, the entire information of the array are deleted and replaced by the new one.

Sincere Thanks!

- Using this group of PPTs, please read

- [1] Yunong Zhang, Weimu Ma, Xiao-Dong Li, Hong-Zhou Tan, Ke Chen, MATLAB Simulink modeling and simulation of LVI-based primal-dual neural network for solving linear and quadratic programs, Neurocomputing 72 (2009) 1679-1687

- [2] Yunong Zhang, Chenfu Yi, Weimu Ma, Simulation and verification of Zhang neural network for online time-varying matrix inversion, Simulation Modelling Practice and Theory 17 (2009) 1603-1617