Introduction to Matlab

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 02

Matlab variable types

Matlab variable types

- integer
  - 123, -345, 0

Matlab variable types

- integer
  - 123, -345, 0
- real or float
  - 12.2344
  - 5.445454
  - engineering notation
    - `4.2323e-9` $= 4.2323 \times 10^{-9}$

Matlab variable types

- integer
  - 123, -345, 0
- real or float
  - 12.2344
  - 5.445454
  - engineering notation
    - `4.2323e-9` $= 4.2323 \times 10^{-9}$
- complex
  - $i = \sqrt{-1} =$ `1i`
  - `34.23+21.21i`
  - `(1+1i)*(1-1i) = 2`

Matlab variable types

- integer
  - 123, -345, 0
- real or float
  - 12.2344
  - 5.445454
  - engineering notation
    - `4.2323e-9` $= 4.2323 \times 10^{-9}$
- complex
  - $i = \sqrt{-1} =$ `1i`
  - `34.23+21.21i`
  - `(1+1i)*(1-1i) = 2`
- strings (put your words inside apostrophes)
  - handy for file names and messages
  - `'programming is fun'`
  - `s='Williamsburg'`

Some built in constants and functions

- $\pi = 3.14159265358979323846264338327950 2\cdots$
  - use `pi`
- trigonometry functions

By default angle is in radians

  - `sin`, `cos`, `tan`, `cot`
  - `asin`, `acos`, `atan`, `acot`

`sin(pi/2)=1`

But can be done in degrees

  - `sind`, `cosd`, `tand`, `cotd`
  - `asind`, `acosd`, `atand`, `acotd`

`sind(90)=1`

- hyperbolic functions
  - `sinh`, `cosh`, `tanh`, `coth`
  - `asinh`, `acosh`, `atanh`, `acoth`
- logarithms
  - natural `log`
  - base of 10 `log10`
- power
  - x^y use `x^y` or alternatively `power(x,y)`
  - e^y use `exp(y)`

Assignment operator

```
x = 1.2 + 3.4
```

Assignment operator

```
x = 1.2 + 3.4
```

Despite the look = is not an equality operator.
= is an assignment operator.

Assignment operator

```
x = 1.2 + 3.4
```

Despite the look `=` is not an equality operator.
`=` is an assignment operator.
The expression above should be read as

- evaluate expression at the right hand side of equality symbol
- assign the result of the RHS to the variable on the left hand sign
- now variable `x` holds the value `4.6`

We are free to use the **value** of the variable `x` in any further expressions

```
> x + 4.2
ans = 8.8
```

Efficient editing - Tab-completition

Once you typed some expressions in “Command window”

- type couple of first symbols of variable or function name
- hit tab and you will get
  - either fully typed name (if it is uniq)
  - or little chart with choices
    - use <up> or <down> arrows to choose
    - alternatively <Ctrl-p>, <Ctrl-n>
    - then hit <enter> to make your choise

Help related commands

These are the most important commands

- `docsearch word`
  - will search for `word` in the help files and show up matched help files
  - example: `docsearch trigonometry`
- `help name`
  - output short help text into "Command window" about function/method named `name`
  - example: `help sin`
- `doc name`
  - show a reference page about function/method named `name` in the help vrowser
  - usually has more information compare to `help name`
  - example: `doc sin`

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

```
-80 + (1)^2
```

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

```
-80 + (1)^2 = -80 + 1
```

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

```
-80 + (1)^2 = -80 + 1= -79
```

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

```
-80 + (1)^2 = -80 + 1= -79
```

Rule of thumb: if not sure use extra parentheses ()

Operator Precedence

Look at the following Matlab expression

```
-2^4*5 + tan(pi/8+pi/8)^2
```

Guess the answer.

```
- (2^4)*5 + (tan( (pi/8+pi/8) ))^2
```

```
- (16)*5 + (tan( (pi/4) ))^2
```

```
-80 + (1)^2 = -80 + 1= -79
```

Rule of thumb: if not sure use extra parentheses ()

- Read more by executing `doc precedence`
- or searching for 'precedence' in the help browser.

Matrices

Recall that Matlab stands for **Mat**rix **Lab**oratory

- So deep inside **everything** is a **matrix** (array)
- a number is the case of 1×1 matrix

Matrices

Recall that Matlab stands for **Mat**rix **Lab**oratory

- So deep inside **everything** is a **matrix** (array)
- a number is the case of 1×1 matrix

Let's create a 3×5 matrix (3 rows and 5 columns)

```
>> Mz=zeros(3,5)


Mz =
 0     0     0     0     0
 0     0     0     0     0
 0     0     0     0     0
```

This is not the only way, but it is one which make sure that matrix is filled with zeros
Note: it is possible to have more than 2 dimensional arrays.

Matrix elements assignment

```
>> Mz(2,4)=1  % 2nd row, 4th column

Mz =


  0      0      0      0      0
  0      0      0      1      0
  0      0      0      0      0
```

Matrix elements assignment

```
>> Mz(2,4)=1   % 2nd row, 4th column

Mz =

     0     0     0     0     0
     0     0     0     1     0
     0     0     0     0     0
```

```
>> Mz(3,5)=4   % 3rd row, 5th column

Mz =

     0     0     0     0     0
     0     0     0     1     0
     0     0     0     0     4
```

Alternative way to assign a matrix

- comma separates column elements
- semicolon separates row elements

```
>> Mz=[ ...
0, 0, 0, 0, 0; ...
0, 0, 0, 1, 0; ...
0, 0, 0, 0, 4]

Mz =

 0     0     0     0     0
 0     0     0     1     0
 0     0     0     0     4
```

Notice $\cdots$ mark, which means that input continues on the next line

Strength of Matlab

Native matrix operations

```
Mz =
0  0  0  0  0
0  0  0  1  0
0  0  0  0  4
```

```
>> Mz+5
ans =
5     5     5     5     5
5     5     5     6     5
5     5     5     5     9
```

Strength of Matlab

Native matrix operations

```
Mz =
0  0  0  0  0
0  0  0  1  0
0  0  0  0  4
```

```
>> Mz+5
ans =
5     5     5     5     5
5     5     5     6     5
5     5     5     5     9
```

```
>> Mz*2
ans =
0     0     0     0     0
0     0     0     2     0
0     0     0     0     8
```

More example on matrices operations

```
Mz  =
0  0  0  0  0
0  0  0  1  0
0  0  0  0  4
```

```
>> Mz+Mz
ans  =
0      0      0      0      0
0      0      0      2      0
0      0      0      0      8
```

More example on matrices operations

```
Mz =
0  0  0  0  0
0  0  0  1  0
0  0  0  0  4
```

```
>> Mz+Mz
ans =
0     0     0     0     0
0     0     0     2     0
0     0     0     0     8
```

Matrix multiplication according to the linear algebra rules

```
>> Mz*Mz'
ans =
0     0     0
0     1     0
0     0    16
```

Here Mz' corresponds to transposed matrix Mz, i.e. $Mz'(i,j) = Mz(j,i)$

Matrix as a function argument

A function can take a matrix as the function argument, it will evaluate the value of the function for each matrix element

```
Mz =
0  0  0  0  0
0  0  0  1  0
0  0  0  0  4
```

```
>> sin(Mz)
ans =
0         0         0         0         0
0         0         0    0.8415         0
0         0         0         0   -0.7568
```

Vectors and column vector

A special case of the matrix is it has only one dimension.
Such matrices generally called vectors

- $m \times 1$ column vector
- $1 \times m$ just a vector

Vectors and column vector

A special case of the matrix is it has only one dimension.
Such matrices generally called vectors

- $m \times 1$ column vector
- $1 \times m$ just a vector

To create a vector

```
>> % use comma to separate column elements
>> v=[1, 2, 3, 4, 5, 6, 7, 8]
v =
 1      2      3      4      5      6      7      8
>> % alternatively you can use spaces
>> v=[1 2 3 4 5 6 7 8];
>> % or mix of these two notations (NOT RECOMMENDED)
>> v=[1 2 3, 4, 5, 6 7 8]
v =
 1      2      3      4      5      6      7      8
```

Column vector

Construction of column vector

```
>> vc=[1; 2; 3]
% use semicolon to separate row elements

vc =

 1
 2
 3
```

Yet one more way to create matrix

If you have prearranged vectors or column vectors you can use them

```
>> vc=[1; 2; 3];
>> % note that ; after a statement suppresses output
>> Mc=[vc, vc, vc]
Mc =
 1      1      1
 2      2      2
 3      3      3
```

Yet one more way to create matrix

If you have prearranged vectors or column vectors you can use them

```
>> vc=[1; 2; 3];
>> % note that ; after a statement suppresses output
>> Mc=[vc, vc, vc]
Mc =
 1     1     1
 2     2     2
 3     3     3
```

```
v =
 1     2     3     4     5     6     7     8
>> Mv=[v; 2*v; 3*v]
Mv =
 1     2     3     4     5     6     7     8
 2     4     6     8    10    12    14    16
 3     6     9    12    15    18    21    24
```

Colon (:) operator

The `:` operator is extremely useful to create vectors or matrix indexes
It usually take form `start:increment:stop`
and creates a vector with following values

```
[ start, start+increment, ... , start+m*increment]
```

where

`min(start,stop)` $\leq$ `m*increment` $\leq$ `max(start,stop)`

Colon (:) operator

The `:` operator is extremely useful to create vectors or matrix indexes
It usually take form `start:increment:stop`
and creates a vector with following values

`[ start, start+increment, ... , start+m*increment]`

where
`min(start,stop)` $\le$ `m*increment` $\le$ `max(start,stop)`

```
>> v=5:2:11


v =
     5     7     9    11
```

Colon (:) operator

The `:` operator is extremely useful to create vectors or matrix indexes
It usually take form `start:increment:stop`
and creates a vector with following values

`[ start, start+increment, ... , start+m*increment]`

where
`min(start,stop)` $\le$ `m*increment` $\le$ `max(start,stop)`

```
>> v=5:2:11

v =
     5     7     9    11
```

It is also possible to have negative `increment`

```
>> v2=12:-3:1

v2 =
    12     9     6     3
```

Colon (:) operator continued

Another form `start:stop` in this case `increment = 1`

```
>> v1=1:5

v1 =

     1     2     3     4     5
```

Colon (:) operator continued

Another form `start:stop` in this case `increment = 1`

```
>> v1=1:5

v1 =

     1     2     3     4     5
```

Notice that

```
>> v3=5:1

v3 =

   Empty matrix: 1-by-0
```

Produce somewhat unexpected result, since default increment is positive

Slicing matrices

It is handy to choose a subset (block) from the matrix
We have a matrix Mv with size 3×8 and we want to choose all elements from columns 2,5,6

```
>> Mv
Mv =
1     2     3     4     5     6     7     8
2     4     6     8    10    12    14    16
3     6     9    12    15    18    21    24

>> Mv(:,[2,5,6])
ans =
2     5     6
4    10    12
6    15    18
```

The meaning of the `:` now is choose all. Notice also that we use vector to specify desired columns

Plotting

Suppose you have a vector with values of x coordinates and we want to plot $\sin(x)$.

```
>> x=linspace(0,2*pi,10)
x =
0     0.6981    1.3963    2.0944    2.7925    3.4907
4.1888 4.8869    5.5851    6.2832
>> y=sin(x)
y =
0     0.6428    0.9848    0.8660    0.3420   -0.3420
-0.8660   -0.9848   -0.6428   -0.0000
>> plot(x,y,'o') %  alternatively plot(x,sin(x),'o')
>> % every plot MUST have title, x and y labels
>> xlabel('x (radians)')
>> ylabel('sin(x)')
>> title('Plot of sin(x)')
```

Saving plots

Now we want to save the figure, use `print`

```
>> print('-dpdf', 'sin_of_x')
```

This will generate file *sin_of_x.pdf* notice automatic fileextension addition.

Saving plots

Now we want to save the figure, use `print`

```
>> print('-dpdf', 'sin_of_x')
```

This will generate file *sin_of_x.pdf* notice automatic fileextension addition.
The '-d' switch stands for output format ('pdf', 'ps', 'eps', 'png"...)

Saving plots

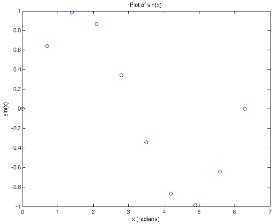
Now we want to save the figure, use `print`

```
>> print('-dpdf', 'sin_of_x')
```

This will generate file *sin_of_x.pdf* notice automatic fileextension addition.
The '-d' switch stands for output format ('pdf', 'ps', 'eps', 'png"...)
To generate 'png' file

```
>> print('-dpng', '-r100',  'sin_of_x')
```

By default figure size is 8×6 inches, the '-r' switch tells the figure resolution in dpi (dots per inch). In this case it is 100 dpi so resulting image will be 800×600 pixels.



For 3D plots, please see help files for `plot3`, `mesh`, `surf`

Special array arithmetic operators

There are special arithmetic operators which applied to the elements of matrices (disregard linear algebra rules), they start with `.`

- `.*`

```
>> x=1:3
x = 1      2      3
>> x*x  % will generate an error
>> x.*x % equivalent to x.^2 (see below)
ans = 1      4      9
```

Special array arithmetic operators

There are special arithmetic operators which applied to the elements of matrices (disregard linear algebra rules), they start with `.`

- `.*`

```
>> x=1:3
x = 1      2      3
>> x*x  % will generate an error
>> x.*x % equivalent to x.^2 (see below)
ans = 1      4      9
```

- `.^`

```
>> x.^2
ans = 1      4      9
```

Special array arithmetic operators

There are special arithmetic operators which applied to the elements of matrices (disregard linear algebra rules), they start with `.`

- `.*`

```
>> x=1:3
x = 1      2      3
>> x*x  % will generate an error
>> x.*x % equivalent to x.^2 (see below)
ans = 1      4      9
```

- `.^`

```
>> x.^2
ans = 1      4      9
```

- `./`

```
>> x./x
ans = 1      1      1
```

Special array arithmetic operators continued

```
>> m=[1,2,3;4,5,6;7,8,9]
m =
1       2       3
4       5       6
7       8       9
```

Linear algebra rules

```
>> m*m
ans =
30      36      42
66      81      96
102     126     150
```

Element wise operation

```
>> m.*m
ans =
1       4       9
16      25      36
49      64      81
```

Special array arithmetic operator .^

```
>> m=[1,2,3;4,5,6;7,8,9]
m =
1       2       3
4       5       6
7       8       9
```

Linear algebra rules

```
>> m^m % undefined
```

Element wise operation

```
>> m.^m
ans =
1           4           27
256         3125        46656
823543      16777216    387420489
```

Special array arithmetic operator ./

```
>> m=[1,2,3;4,5,6;7,8,9]
m =
1       2       3
4       5       6
7       8       9
```

Linear algebra rules

```
>> m/m % unity matrix
ans =
1       0       0
0       1       0
0       0       1
```

Element wise operation

```
>> m./m %matrix of ones
ans =
1       1       1
1       1       1
1       1       1
```