Matrices and plotting.

Eugeniy E. Mikhailov

The College of William & Mary

Lecture 03

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

```
>> 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 ··· mark, which means that input continues on the next line

Strength of Matlab

Native matrix operations

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

More example on matrices operations

```
>> 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

```
>> 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

```
>> 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]

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];
>> 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];
>> 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 `start+m*increment` $\leq$ `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 `start+m*increment` $\leq$ `stop`

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

v =
     5     7     9    12
```

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 `start+m*increment` $\leq$ `stop`

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

v =
     5     7     9    12
```

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') %  other way 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

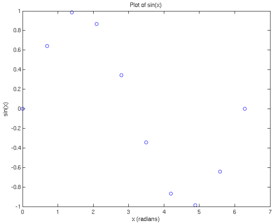
Now we want to save the figure, use `print`

```
>> 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 $\times$ 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 $\times$ 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)

- .*

```
>> x=1:3
x = 1      2      3
>> x*x % will generate an error
>> 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)

- `.*`

```
>> x=1:3
x = 1      2      3
>> x*x % will generate an error
>> x.*2
ans = 1      4      9
```

- `./`

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

Special array arithmetic operators

There are special arithmetic operators which applied to the elements of matrices (disregard linear algebra rules)

- `.*`

```
>> x=1:3
x = 1      2      3
>> x*x % will generate an error
>> x.*2
ans = 1      4      9
```

- `./`

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

- `.^`

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

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
ans =
1        0        0
0        1        0
0        0        1
```

Element wise operation

```
>> m./m
ans =
1        1        1
1        1        1
1        1        1
```