Matrices and Matrix Operations in MATLAB

Lecture 2

Dr. Görkem Saygılı

Department of Biomedical Engineering
Ankara University

Introduction to MATLAB, 2017-2018 Spring

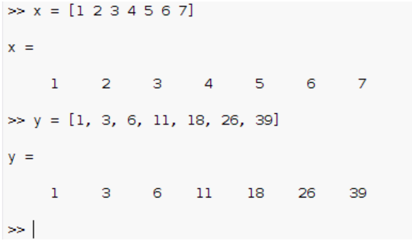
MATLAB Capability:

Although we can use MATLAB for programming purposes like C/C++ or JAVA, it has some advantages over these languages in numerical calculations.

MATLAB is abbreviation for Matrix Laboratory (not Mathematics Laboratory). It is capable of doing fast matrix calculations.

MATLAB 1-D Array:

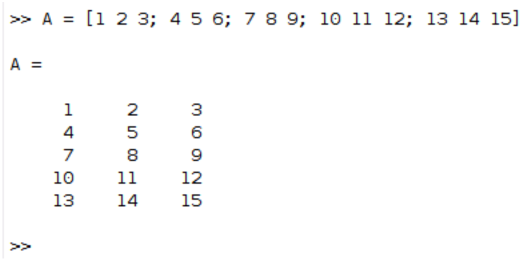
Let's start with a list of numeric values. From the last lecture, we saw how to define a vector (1-D array of numerical values) in MATLAB:



```
>> x = [1 2 3 4 5 6 7]

x =

     1     2     3     4     5     6     7

>> y = [1, 3, 6, 11, 18, 26, 39]

y =

     1     3     6    11    18    26    39

>> |
```

Matrix:

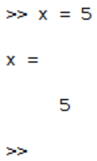
Matrix is a rectangular array of numbers. The following is a rectangular matrix with size 5×3:



```
>> A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]

A =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15

>>
```

<u>Scalar</u>:

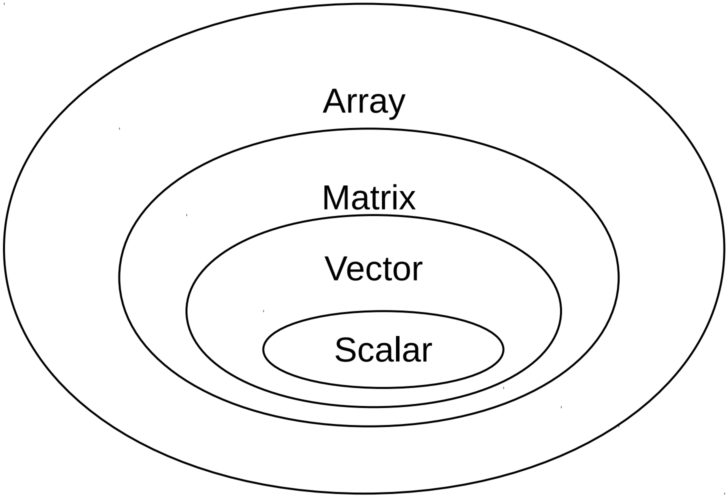
In comparison to 1-D Arrays and Matrices, a scalar is a single value:



```
>> x = 5

x =

     5

>>
```

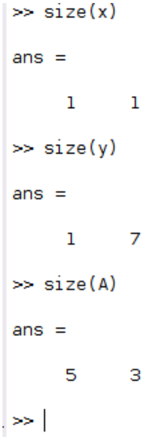
A scalar can be thought as 1×1 1-D array.

Array vs Matrix vs Vector vs Scalar:



Size of a Variable:

There is a built-in function in MATLAB to check the size of a variable, size():



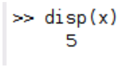
```
>> size(x)

ans =

     1     1

>> size(y)

ans =

     1     7

>> size(A)

ans =

     5     3

>> |
```

Functions & Arguments:

A function in Mathematics is an operation that gives the same result for the same input. We have also functions in programming where we operate on an input to produce a desired output.

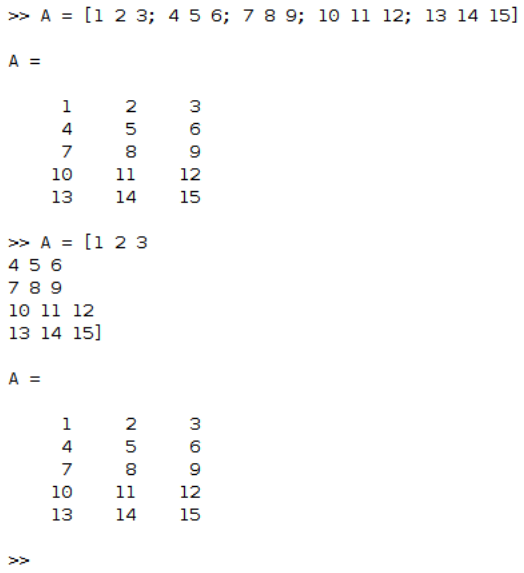
However, a function may not always give the same output for the same input in programming (we will examples of this).

A function in general requires one or multiple input(s). These inputs are called arguments of that function. For example:



```
>> disp(x)
     5
```

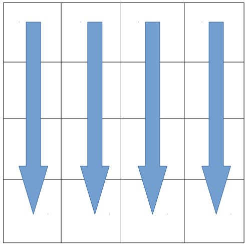
Different Ways of Initializing Matrices:



```
>> A = [1 2 3; 4 5 6; 7 8 9; 10 11 12; 13 14 15]

A =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15

>> A = [1 2 3
4 5 6
7 8 9
10 11 12
13 14 15]

A =

     1     2     3
     4     5     6
     7     8     9
    10    11    12
    13    14    15

>>
```

Column-major Order:

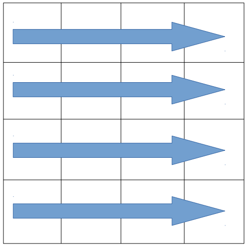
Elements are stored in the memory column-wise. In a way, you can think about it as a matrix that is constructed by concatanation of 1-D column vectors.



!Attention: You need to keep in mind column-major order if you want to access matrix elements with their linear indices. Also when you are using C/C++ (Mex Coding) with MATLAB.

Row-major Order:

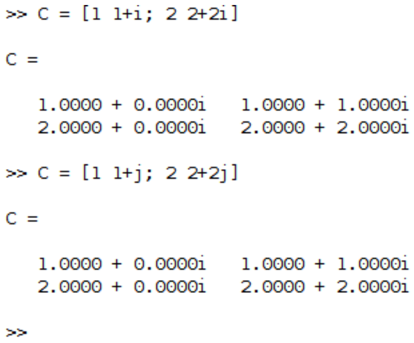
Elements are stored in the memory row-wise. In a way, you can think about it as a matrix that is constructed by concatanation of 1-D row vectors.



!Attention: In C/C++ matrices are stored as row-major order. You should keep this in mind while accessing matrix elements.

Complex Numbers:

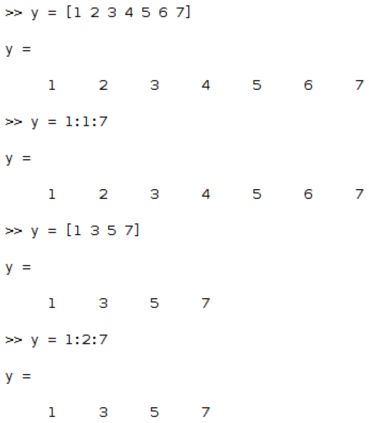
You can use complex numbers in MATLAB (inside matrices as well). The following shows how you can initialize a matrix with a complex element:



```
>> C = [1 1+i; 2 2+2i]

C =

   1.0000 + 0.0000i   1.0000 + 1.0000i
   2.0000 + 0.0000i   2.0000 + 2.0000i

>> C = [1 1+j; 2 2+2j]

C =

   1.0000 + 0.0000i   1.0000 + 1.0000i
   2.0000 + 0.0000i   2.0000 + 2.0000i

>>
```

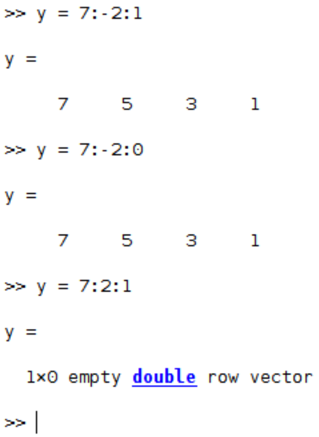
Different Ways of Defining Arrays:

In MATLAB, we can use ":" operator while creating arrays with a specific ordering:



```
>> y = [1 2 3 4 5 6 7]

y =

     1     2     3     4     5     6     7

>> y = 1:1:7

y =

     1     2     3     4     5     6     7

>> y = [1 3 5 7]

y =

     1     3     5     7

>> y = 1:2:7

y =

     1     3     5     7
```

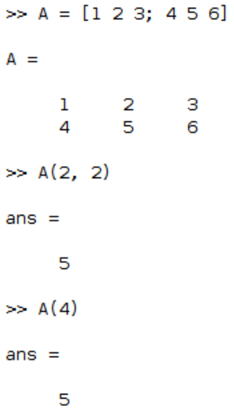
Different Ways of Defining Arrays:



```
>> y = 7:-2:1

y =

     7     5     3     1

>> y = 7:-2:0

y =

     7     5     3     1

>> y = 7:2:1

y =

  1×0 empty double row vector

>> 
```

Accessing Matrix Elements:

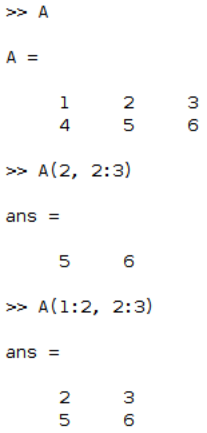
Since a matrix is a 2-D array of numbers, you can reach each of its elements by referring to its row and column position (index). Furthermore, you can use linear indexing to access each element by using just a single index as following:



```
>> A = [1 2 3; 4 5 6]

A =

     1     2     3
     4     5     6

>> A(2, 2)

ans =

     5

>> A(4)

ans =

     5
```

Accessing Parts of a Matrix:

We have seen how to access to a single element in a matrix. We can also access to a part of a matrix:



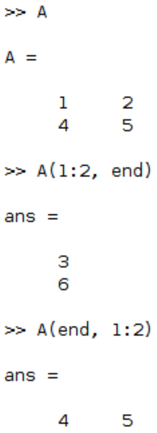
```
>> A

A =

     1     2     3
     4     5     6

>> A(2, 2:3)

ans =

     5     6

>> A(1:2, 2:3)

ans =

     2     3
     5     6
```

Accessing Parts of a Matrix:

```
>> A

A =

     1     2     3
     4     5     6

>> A([2 1], [2 1])

ans =

     5     4
     2     1

>> A(1, :)

ans =

     1     2     3

>> A(:, 2)

ans =

     2
     5
```

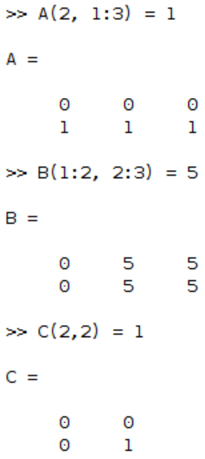
“end” Keyword:

We can use a keyword to let MATLAB know that we want to access to the last row or column. That keyword is “end”:



```
>> A

A =

     1     2     3
     4     5     6

>> A(1:2, end)

ans =

     3
     6

>> A(end, 1:2)

ans =

     4     5
```

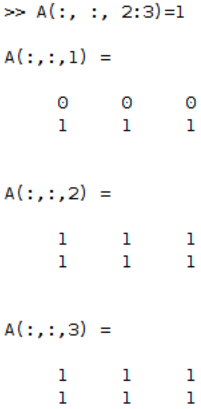
Creating a Matrix with Indexing:



```
>> A(2, 1:3) = 1

A =

     0     0     0
     1     1     1

>> B(1:2, 2:3) = 5

B =

     0     5     5
     0     5     5

>> C(2,2) = 1

C =

     0     0
     0     1
```

3-D Arrays:

We can also create 3-D arrays:



```
>> A(:, :, 2:3)=1

A(:,:,1) =

     0     0     0
     1     1     1


A(:,:,2) =

     1     1     1
     1     1     1


A(:,:,3) =

     1     1     1
     1     1     1
```

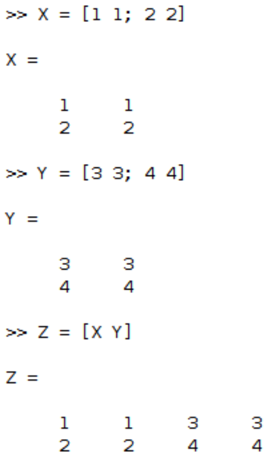
3rd Dimension:

The first dimension is called row, the second is called column. The third dimension is called a page in general. However, when we are processing medical images like 3-D CT scans, we call the third dimension as slice.

MATLAB displays a 3-D array page by page as we saw in the previous slide.

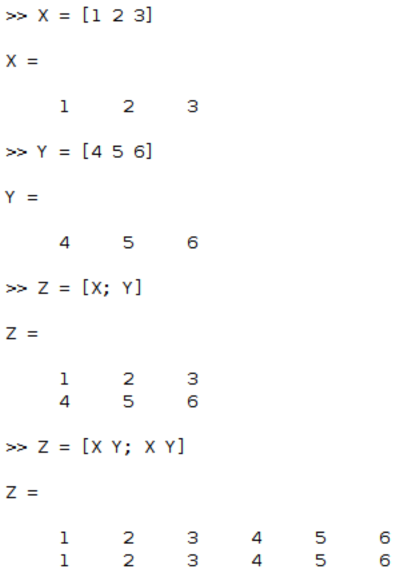
Combining Matrices:

Matrices with the same number of rows can be cascaded as follows:



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

X =

     1     1
     2     2

>> Y = [3 3; 4 4]

Y =

     3     3
     4     4

>> Z = [X Y]

Z =

     1     1     3     3
     2     2     4     4
```

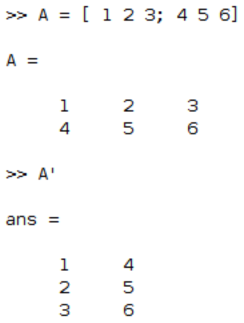
Combining Matrices:



```
>> X = [1 2 3]

X =

     1     2     3

>> Y = [4 5 6]

Y =

     4     5     6

>> Z = [X; Y]

Z =

     1     2     3
     4     5     6

>> Z = [X Y; X Y]

Z =

     1     2     3     4     5     6
     1     2     3     4     5     6
```

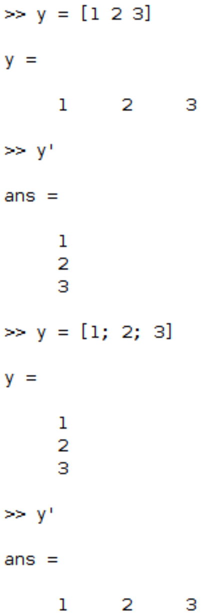
Matrix Transpose:

To take the transpose of a matrix, we use " ' ", apostrophe operator:



```
>> A = [ 1 2 3; 4 5 6]

A =

     1     2     3
     4     5     6

>> A'

ans =

     1     4
     2     5
     3     6
```

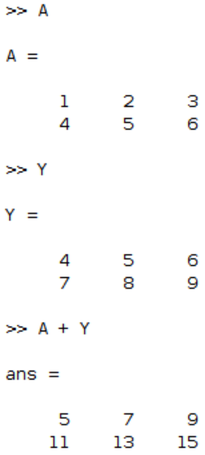
Vector Transpose:



```
>> y = [1 2 3]

y =

     1     2     3

>> y'

ans =

     1
     2
     3

>> y = [1; 2; 3]

y =

     1
     2
     3

>> y'

ans =

     1     2     3
```

Matrix Addition:

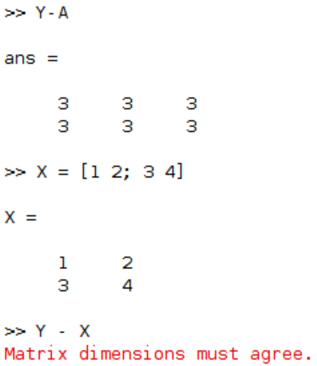
Calculation operators that are used with scalars can be also used with arrays. However, you need to be carefull about the sizes of the arrays that you apply these operators:



```
>> A

A =

     1     2     3
     4     5     6

>> Y

Y =

     4     5     6
     7     8     9

>> A + Y

ans =

     5     7     9
    11    13    15
```

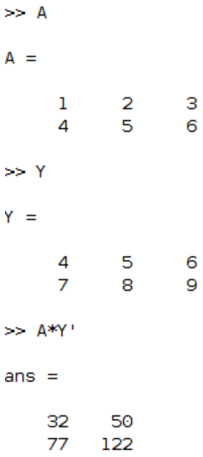
Matrix Subtraction:

Matrix subtraction is similar to matrix addition. Again you need to be carefull about size of matrices:



```
>> Y-A

ans =

     3     3     3
     3     3     3

>> X = [1 2; 3 4]

X =

     1     2
     3     4

>> Y - X
Matrix dimensions must agree.
```

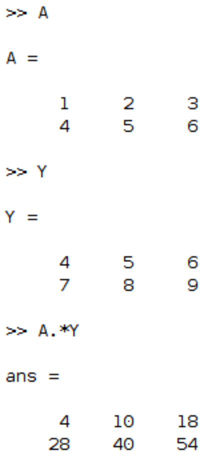
Matrix Multiplication:



```
>> A

A =

     1     2     3
     4     5     6

>> Y

Y =

     4     5     6
     7     8     9

>> A*Y'

ans =

    32    50
    77   122
```

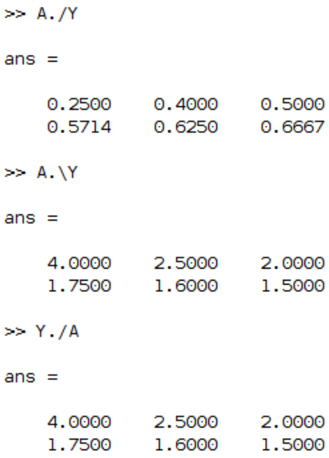
Array Multiplication (Element by Element):

If you use “ .* ” instead of “ * ”, the operation becomes element by element multiplication:



```
>> A

A =

     1     2     3
     4     5     6

>> Y

Y =

     4     5     6
     7     8     9

>> A.*Y

ans =

     4    10    18
    28    40    54
```

Matrix Division:



```
>> A./Y

ans =

    0.2500    0.4000    0.5000
    0.5714    0.6250    0.6667

>> A.\Y

ans =

    4.0000    2.5000    2.0000
    1.7500    1.6000    1.5000

>> Y./A

ans =

    4.0000    2.5000    2.0000
    1.7500    1.6000    1.5000
```

Taking Powers of a Matrix:

```
>> X = [1 2; 3 4]

X =

     1     2
     3     4

>> X^2

ans =

     7    10
    15    22

>> X * X

ans =

     7    10
    15    22
```

Scalar Multiplication:

```
>> A

A =

     1     2     3
     4     5     6

>> 5*A

ans =

     5    10    15
    20    25    30

>> 5 - A

ans =

     4     3     2
     1     0    -1
```

Next Lecture:

In the next lecture, we will see some built-in functions in MATLAB for matrices and start learning about writing functions in MATLAB.

In this week's lab, after finishing lab exercise, you can start practicing the following functions:

circshift()
permute()
repmat()
reshape()