

Introduction to MATLAB bootcamp

Week 2 Lecture 3

Sandeep Kumar
sk35@princeton.edu

Selecting an element by index in a 2D matrix

```
big_mat_eg=[1 2 3 4; 5 6 7 8; 9 10 11 12];
```

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

```
>> big_mat_eg(:,1)
```

ans =

1
5
9

```
>> big_mat_eg(2,:)
```

ans =

5 6 7 8

```
>> big_mat_eg(:,3:end)
```

ans =

3 4
7 8
11 12

3 Dim matrix (can then be generalized to N Dim)

What will be a good example of a 3 dim matrix?

```
three_dim_mat(:,:,1)=[1 2 3; 4 5 6];  
three_dim_mat(:,:,2)=[11 12 13; 14 15 16];  
three_dim_mat(:,:,3)=[21 22 23; 24 25 26];
```

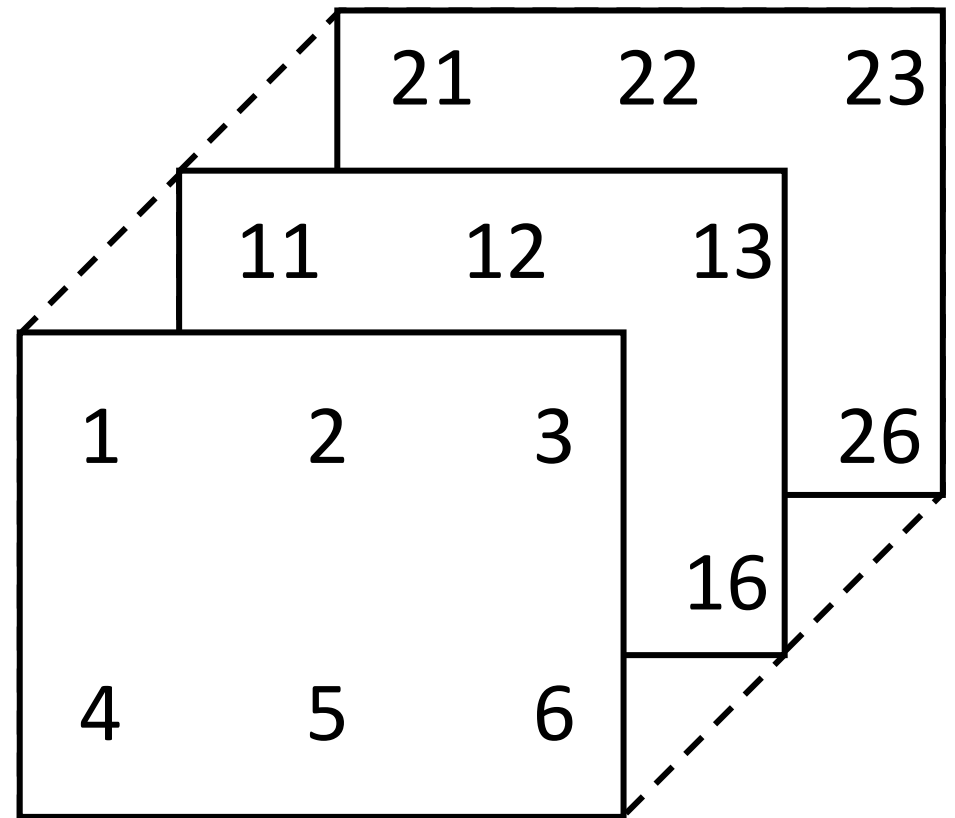
```
>> three_dim_mat(4)    >> three_dim_mat(9)
```

ans =

5

ans =

12

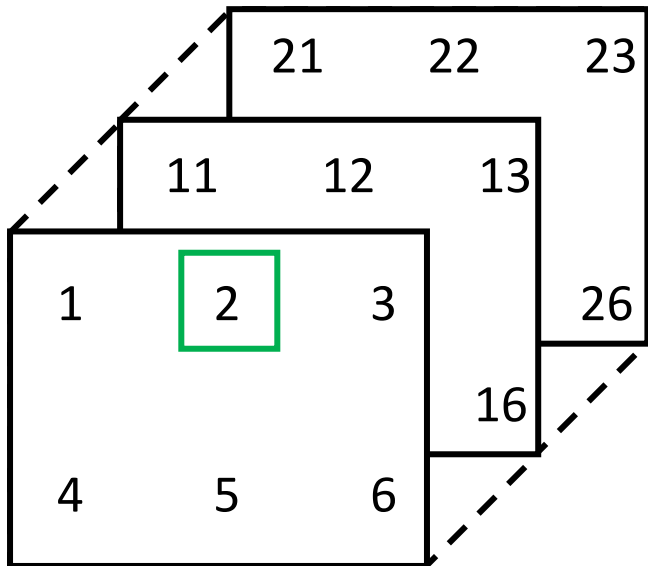


3 Dim matrix (can then be generalized to N Dim)

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

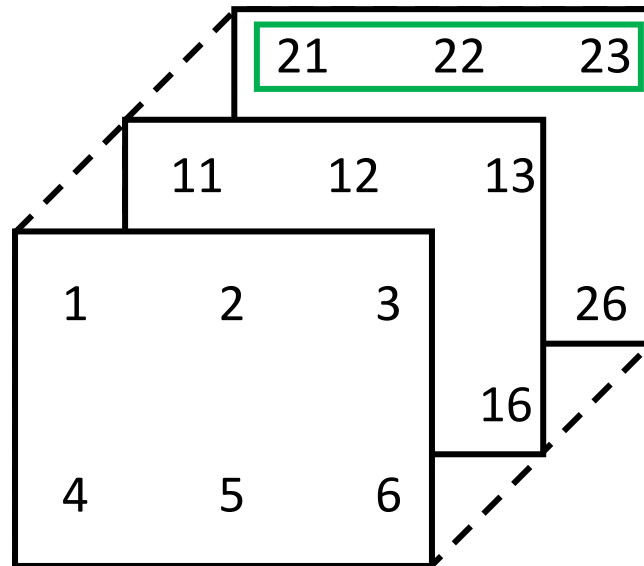
ans =

5



ans =

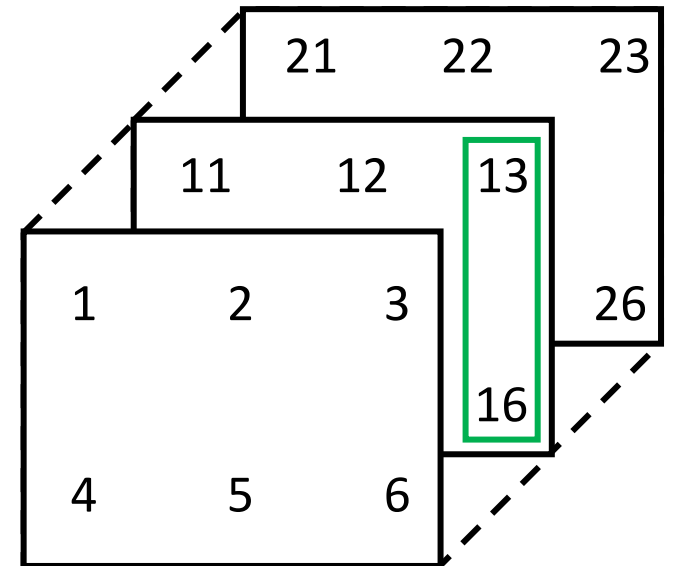
21 22 23



ans =

13

16



Defining matrices in MATLAB

To define a matrix:

`matrix_name=[start_value : step_value : end_value];`

```
>> vector_1=[2:7]
```

```
vector_1 =
```

```
     2     3     4     5     6     7
```

```
>> only_odd_numbers=[1:2:13]
```

```
only_odd_numbers =
```

```
     1     3     5     7     9    11    13
```

```
>> descending_even_numbers=[14:-2:2]
```

```
descending_even_numbers =
```

```
    14    12    10     8     6     4     2
```

Defining matrices in MATLAB

Another way to define a matrix is to use the function linspace/linearly spaced vector

(<https://www.mathworks.com/help/matlab/ref/linspace.html>)

```
matrix_name=linspace(start_num, end_num, num_elements)
```

Here is an example to generate an array with 7 elements between -3 and 3 (including both of them).

```
>> test=linspace(-3,3,7)
```

```
test =
```

```
-3 -2 -1 0 1 2 3
```

Defining matrices in MATLAB

You can use built-in functions to define matrices. For e.g.

1. To generate matrix with all 1s: `ones(num_row, num_column)`
2. To generate matrix with all 0s: `zeros(num_row, num_column)`
3. To generate matrix with all NaNs: `nan(num_row, num_column)`

```
>> ones(2,3)
```

```
ans =
```

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

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

```
ans =
```

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

```
>> nan(2,2)
```

```
ans =
```

```
NaN NaN  
NaN NaN
```

Defining matrices in MATLAB

- To generate matrix with random numbers: `rand(num_row, num_col)`

```
>> rand(2,3)
```

```
ans =
```

```
0.0782 0.1067 0.0046  
0.4427 0.9619 0.7749
```

```
>> rand(2)
```

```
ans =
```

```
0.8173 0.0844  
0.8687 0.3998
```

- To generate matrix with random integers: `randi(max_integer,num_row, num_col)`

```
>> randi(9,2,3)
```

```
ans =
```

```
3 4 2  
8 9 3
```

How will you get random
positive and negative integers?

Matrix operations: Addition and subtraction

```
>> mat_1=[6:2:10;20:-2:16]
```

```
>> mat_2=[3:5;-5:-3]
```

```
mat_1 =
```

```
6 8 10  
20 18 16
```

```
mat_2 =
```

```
3 4 5  
-5 -4 -3
```

```
>> mat_add=mat_1+mat_2
```

```
mat_add =
```

```
9 12 15  
15 14 13
```

```
>> mat_add=mat_1-mat_2
```

```
mat_add =
```

```
3 4 5  
25 22 19
```

Matrix operations: Scalar multiplication

```
>> mat_3=[1:2:5; 5:-2:1]
>> scalar_3=5
>> scalar_mult=mat_3*scalar_3
```

```
>> mat_3=[1:2:5; 5:-2:1]
scalar_3=5
scalar_mult=mat_3*scalar_3
```

mat_3 =

1	3	5
5	3	1

scalar_3 =

5

scalar_mult =

5	15	25
25	15	5

Matrix operations: Matrix (or vector) multiplication

Keep in mind the dimension of the two matrices for vector multiplication

$$[A]_{m \times n} \times [B]_{n \times m} = [C]_{m \times m}$$

For example:

	>> mat_4 =	>> mat_5 =	>> vector_mult =
>> mat_4=[3:5; 5:7]	3 4 5	2 3	52 64
>> mat_5=[2:3; 4:5; 6:7]	5 6 7	4 5	76 94
>> vector_mult=mat_4*mat_5		6 7	

Matrix operations: Element by element multiplication

Both the matrices should be of the same size.

For example:

```
>> mat_6=[3:5; 5:7]
```

```
>> mat_7=[2:4; 4:6]
```

```
>> element_by_element_mult=mat_6.*mat_7
```

```
>> mat_6 =
```

```
3  4  5  
5  6  7
```

```
>> mat_7 =
```

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

```
>> element_by_element_mult =
```

```
6  12  20  
20  30  42
```

Matrix operations: Transpose of a matrix

Flips the dimensions of the matrix

$$[A]_{m \times n} \rightarrow [B]_{n \times m}$$

```
>> mat_8=[3:5; -7:-5]  
>> mat_9=mat_8'
```

```
>> mat_8 =
```

```
3    4    5  
-7   -6   -5
```

```
>> mat_9 =
```

```
3    -7  
4    -6  
5    -5
```

Splitting matrices

- Sometimes you might want to split the matrix and work on a subset of the matrix (for e.g. just a row or column)

```
>> original_mat=[1:7; 21:27; 51:57]
>> split_row=original_mat(2,:)
>> split_column=original_mat(:,4)
```

```
>> split_row =
```

```
21  22  23  24  25  26  27
```

```
>> split_column =
```

```
4
24
54
```

Concatenating matrices: Horizontal

- Combine two matrices horizontally
- If you have 2 matrices A and B then to horizontally concatenate them:
- $C=[A \ B]$ or $C=\text{horzcat}(A, B)$

```
>> mat_10=ones(3,2)*4  
>> mat_11=randi(7,3,2)  
>> horz_cat_1=[mat_10 mat_11]  
>> horz_cat_2=horzcat(mat_10,mat_11)
```

```
>> mat_10 =
```

```
4  4  
4  4  
4  4
```

```
>> mat_11 =
```

```
6  7  
7  5  
1  1
```

```
>> horz_cat_1 =
```

```
4  4  6  7  
4  4  7  5  
4  4  1  1
```

```
>> horz_cat_2 =
```

```
4  4  6  7  
4  4  7  5  
4  4  1  1
```

Concatenating matrices: Vertical

- Combine two matrices vertically
- If you have 2 matrices A and B then to horizontally concatenate them:
- $C=[A; B]$ or $C=\text{vertcat}(A, B)$

```
>> mat_12=ones(2,3)*4
```

```
>> mat_13=randi(7,2,3)
```

```
>> vert_cat_1=[mat_12; mat_13]
```

```
>> vert_cat_2=vertcat(mat_12,mat_13)
```

```
>> mat_12 =
```

```
4 4 4
```

```
4 4 4
```

```
>> mat_13 =
```

```
7 6 3
```

```
4 1 7
```

```
>>vert_cat_1 =
```

```
4 4 4
```

```
4 4 4
```

```
7 6 3
```

```
4 1 7
```

```
>>vert_cat_2 =
```

```
4 4 4
```

```
4 4 4
```

```
7 6 3
```

```
4 1 7
```


mean of an array

```
>> new_matrix=[2:2:16; 10:-2:-5; 3:3:24];
```

```
>> new_matrix =
```

2	4	6	8	10	12	14	16
10	8	6	4	2	0	-2	-4
3	6	9	12	15	18	21	24

By default, MATLAB takes
mean across each column

2	4	6	8	10	12	14	16
10	8	6	4	2	0	-2	-4
3	6	9	12	15	18	21	24

```
mean(new_matrix)
```

```
ans =
```

```
5 6 7 8 9 10 11 12
```

What if you want to find the
mean across each row?

```
mean(new_matrix,2)
```

```
ans =
```

```
9
```

```
3
```

```
13.5
```

2=across each row

1=across each column

2	4	6	8	10	12	14	16
10	8	6	4	2	0	-2	-4
3	6	9	12	15	18	21	24

sort

- To sort an array, you can use the inbuilt MATLAB function sort

- `mat_14=[10 5 2 3 6 7 0 -1 -12 7 6]`

- `sort(mat_14)`

-

`ans =`

`-12 -1 0 2 3 5 6 6 7 7 10`

unique

- Sometimes your goal is to find the unique elements in an array
- For e.g. in the previous example 6 and 7 appeared twice.
- To find the unique elements, use the function 'unique'.
- `mat_14=[10 5 2 3 6 7 0 -1 -12 7 6]`
- `unique(mat_14)`
- `ans =`

-12 -1 0 2 3 5 6 7 10

Note that the 'unique' function also sorts the output

reshape

- Using reshape function, we can reshape a matrix to another size
- `to_reshape_array=[1:1:10]`
- `reshape(to_reshape_array,[5,2])`

• `ans =`

1	6
2	7
3	8
4	9
5	10

end and end-1

```
>> test_array=linspace(21,31,11)
```

```
test_array =
```

```
21 22 23 24 25 26 27 28 29 30 31
```

```
>> test_array(5:end)
```

```
ans =
```

```
25 26 27 28 29 30 31
```

```
>> test_array(5:end-1)
```

```
ans =
```

```
25 26 27 28 29 30
```

nanmean

```
>> test_array=[1 2 3 4 5];
```

```
>>mean(test_array)
```

3

```
>> with_nan_test_array=[1 2 3 4 5 nan];
```

```
mean(with_nan_test_array)
```

NaN

```
>> nanmean(with_nan_test_array)
```

3