# Introduction to MATLAB

Week 2 Lecture 2

Polina Cherepanova
cherepanova@princeton.edu

# 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_sub=mat_1 - mat_2
```

```
mat_sub =

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

$$[A]_{mxn} \times [B]_{nxm} = [C]_{mxm}$$

For example:

```
>> mat_4=[3:5; 5:7]
>> mat_5=[2:3; 4:5; 6:7]
>> vector_mult=mat_4*mat_5
```

```
>> mat_4 =

3   4   5
5   6   7
```

```
>> mat_5 =

2   3
4   5
6   7
```

```
>> vector_mult =

52   64
76   94
```

# Matrix operations: Element-wise 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

$$[A]_{mxn} \rightarrow [B]_{nxm}$$

```
>> 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
```

# 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=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=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
```

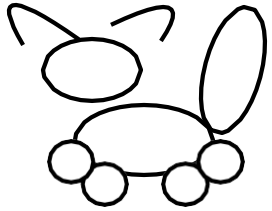
```
>>vert_cat_1 =

   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
```

cat(DIM, A,B)



# 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?

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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,2)

ans =

9
3
13.5
```

2=across each row

1=across each column

# 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]

Can sort in descending order with: sort(mat_14, ‘descend’)

# 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=[1:10]

- reshape(to_reshape_array, [5, 2])

- ans =

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

# Functions in MATLAB

- You have already used many in-built MATLAB functions such as:

- sort
- length
- mean
- unique
- disp

- You can also make your own functions in MATLAB pretty easily

# Syntax for functions

func_name.m*

```
function func_name(input_to_function)
%%% Add some info about your function

%%% Main body of your function

end
```

Important:

1) Rules for naming a function are same as rules for naming a variable
2) Functions go at the bottom of MATLAB scripts

# Example: basic_average

basic_avg.m

```
function basic_avg(array)
    avg=mean(array);
end
```

Drawback: The output of this function is not stored as a separate variable, and so we can’t access it after running the function.

# Store output of the function as a variable

better_avg.m

```
function avg=better_avg(array)
    avg=mean(array);
end
```

```
>> avg_result = better_avg([9 2 4 6 7 10 11 1 0]);
```

# Exercise 1: Write a function to calculate the circumference of a circle

$$\text{circumference} = 2 * \pi * \text{radius}$$

Make a function that takes in the radius of a circle and returns its circumference

```
>> circumference = calc_circ(4.6);
>> circumference

circumference =

    28.90
```

# Custom functions can call built-in MATLAB functions

```
function avgx3 = mean_times_three(array)

    avg = mean(array);
    avgx3 = avg * 3;

end
```

# Exercise 2: Write the following function:

Make a function that takes in a vector of elements and returns the sum of its unique elements

```
>> output = sum_unique([1 2 2 3 5 1 10 9]);
>> output

output =

    30
```

# Functions can generate multiple outputs

min_max.m

```
function [min_num,max_num]=min_max(array)
    min_num=min(array);
    max_num=max(array);
end
```

```
>> [min_n, max_n] = min_max([1 2 3 5 6 1 10 7]);

>> [~, max_n] = min_max([1 2 3 5 6 1 10 7]);
```

# Exercise 3: Write a function to return the results of five operations

Make a function that takes in a vector of elements and returns its mean, standard deviation, min, max, and unique elements, all as separate variables.

```
>> [avg, stand_dev, mini, maxi, uniq] = operations([1 2 2 3 5 1 10 9]);
>> stand_dev

mini =

     1
```

# Exercise 4: Temperature conversion

$$C = \frac{5}{9}(F - 32)$$

$$K = C + 273.15$$

Write a function that takes in a temperature in Fahrenheit and returns two outputs: the converted temperature in Celsius and the converted temperature in Kelvin

# Custom functions can call other custom functions

```
function avgx3 = mean_times_three(array)

    avg = mean(array);
    avgx3 = avg * 3;

end
```

```
function avgx3_plus5 = mean_times_three_plus_five(array)

    avgx3 = mean_times_three(array);
    avgx3_plus5 = avgx3 + 5;

end
```

# Exercise 5:

Write a function that takes in an array of temperatures in Fahrenheit and returns both the average and the range of these temperatures in Celsius.

Hint: call the temperature conversion function you just wrote when coding this new function

```
>> [avg_C, range_C] = temp_operations([65 22 89]);
```