

ENG 346

Data Structures and

Algorithms for Artificial

Intelligence

Matrix Operations and Numpy

Dr. Mehmet PEKMEZCI

mpekmezci@gtu.edu.tr

<https://github.com/mehmetpekmezci/GTU-ENG-346>

ENG-346 Teams code is **0uv7jlm**

Codes

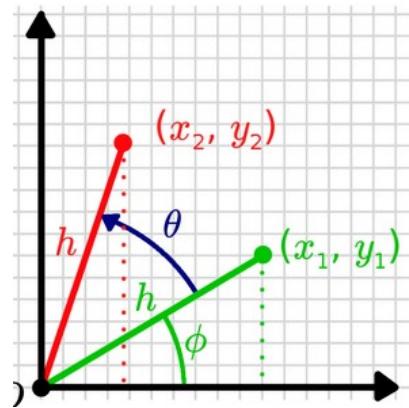
- <https://github.com/mehmetpekmezci/GTU-ENG-346/tree/main/matrix-numpy-codes>

Matrices

- Matrices can be used to store data
 - Image data ($n \times m$) matrix
 - Sound data ($n \times 1$) matrix
 - Any Log data ($n \times m$) matrix,
 - e.g. : log of humidity, temperature, ... values in a room
 - Geometric data
- Matrices can be used for Linear Transformations (Linear Function)
 - Rotation
 - Translation
 - Stretching

Matrices – Geometric Meaning

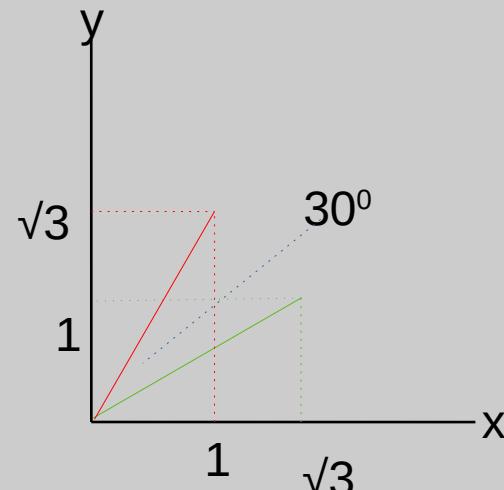
- Rotation



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

NOTE: 3D rotation matrix is a 3x3 matrix, not a 2x2x2 tensor

Example :



$$\begin{bmatrix} 1 \\ \sqrt{3} \end{bmatrix} = \begin{bmatrix} \cos(\pi/6) & -\sin(\pi/6) \\ \sin(\pi/6) & \cos(\pi/6) \end{bmatrix} \begin{bmatrix} \sqrt{3} \\ 1 \end{bmatrix}$$

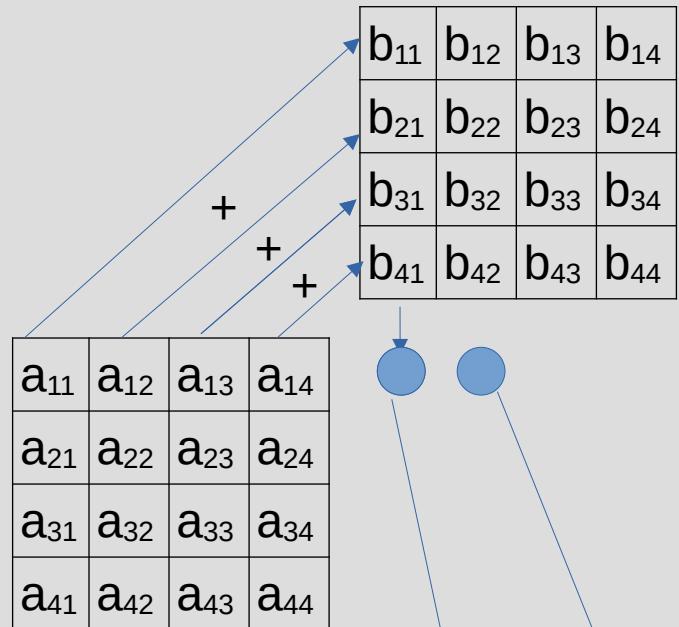
```
import numpy as np
theta = np.radians(30)
c, s = np.cos(theta), np.sin(theta)
R = np.array(((c, -s), (s, c)))
green_vector=np.array([np.sqrt(3),1])
print(green_vector)
red_vector=np.dot(R,green_vector)
print(red_vector)
```

<https://articulatedrobotics.xyz/tutorials/coordinate-transforms/rotation-matrices-2d/>

Multiplication of Matrices

Example : 4x4 matrix A=

A.B =



$$a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} + a_{14} \cdot b_{42}$$

$$a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + a_{14} \cdot b_{41}$$

- We can multiply A and B
If $A(N,M) \cdot B(M,K)$
- $A(N,M) \cdot B(M,K) = C(N,K)$

Matrix Example

- Example :

$$x + 2y = 4$$

$$3x - 4y = 5$$

$$\begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$A \cdot X = B$$

$$\begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix} + \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

x+2y
3x-4y

$x + 2y = 4$
 $3x - 4y = 5$

Determinants and Inverse of a Matrix

Example : 4x4 matrix A=

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

$$\det(A) = |A| = a \cdot \det \left(\begin{array}{ccc} & & \\ f & g & h \\ j & k & l \\ n & o & p \end{array} \right) - b \cdot \det \left(\begin{array}{ccc} & & \\ e & g & h \\ i & k & l \\ m & o & p \end{array} \right)$$

$$+ c \cdot \det \left(\begin{array}{ccc} & & \\ e & f & h \\ i & j & l \\ m & n & p \end{array} \right) - d \cdot \det \left(\begin{array}{ccc} & & \\ e & f & g \\ i & j & k \\ m & n & o \end{array} \right)$$

Minor of element b is $\det(M_{12})$,
 Cofactor(A) is a matrix composed of C_{ij} .
 $C_{ij} = (-1)^{i+j} \det(M_{ij})$

e	g	h	
i	k	l	
m	o	p	

e	g	h
i	k	l
m	o	p

C_{11}	C_{12}	C_{13}	C_{14}
C_{21}	C_{22}	C_{23}	C_{24}
C_{31}	C_{32}	C_{33}	C_{34}
C_{41}	C_{42}	C_{43}	C_{44}

Inverse of A
 $A^{-1} = 1/\det(A) \cdot (\text{Cofactor}(A))^T$

NO INVERSE
 if $\det(A)=0$ or
 matrix is not square

Transpose and Inverse Properties

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

$$(A^{-1})^{-1} = A$$

$$(A^T)^T = A$$

$$(A^T)^{-1} = (A^{-1})^T$$

$$(AB)^T = B^T \cdot A^T$$

$$(A+B)^T = A^T + B^T$$

$$(AB)^{-1} = B^{-1} \cdot A^{-1}$$

$$(k \cdot A)^{-1} = 1/k \cdot A^{-1}$$

$$(k \cdot A)^T = k \cdot A^T$$

$$\det(A^T) = \det(A)$$

$$\det(A^{-1}) = 1/\det(A)$$

$$\det(A \cdot B) = \det(A) \cdot \det(B)$$

Tensors (Multilinear Map)

- Vectors = 1st order Tensor

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad (1\text{D arrays})$$

- Matrices = 2nd order Tensor

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (2\text{D arrays})$$

- 3rd order Tensor , is a stack of matrices (3D arrays)

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad \longrightarrow \quad 2 \times 2 \times 3 \text{ Tensor (} l, j, k \text{ indices)}$$

- Tensor may have n order.

Matrix Operations

- Matrix/Tensor Construction
- Dimensions (Shape) and reshaping, array slicing, masking.
- Matrix Summation
- Transpose of the Matrix
- Matrix Multiplication
- Inverse of the Matrix
- Stacking (Vertical, Horizontal, Depth)
- Splitting (Vertical, Horizontal)
- Distances (Mean Squared Error, Mean Absolute Error, Cosine Similarity)
- **NOT COVERED : Eigenvalues, Eigenvectors, SVD , Tensor Dot**

Matrix/Tensor Construction

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$$



```
import numpy as np
a = np.array( [ [1, 2] , [3, 4] ] )
print(a)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 01.arrays_matrices.py
[[1 2]
 [3 4]]
```

$$\begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix}$$

np.full((2, 2), 10,dtype=int)

2 rows, 2 columns

$$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

number_of_rows=2

np.identity(number_of_rows,dtype=int)

$$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$$

np.zeros(2)

default dtype=float

$$\begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$$

number_of_rows=2

np.identity(number_of_rows,dtype=float)

Matrix/Tensor Construction

```
import numpy as np
np.random.seed(0) # seed for reproducibility
random_tensor = np.random.randint(100, size=(3, 4, 5))
print(random_tensor)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 02.random.py
[[[44 47 64 67 67]
 [ 9 83 21 36 87]
 [70 88 88 12 58]
 [65 39 87 46 88]]

 [[81 37 25 77 72]
 [ 9 20 80 69 79]
 [47 64 82 99 88]
 [49 29 19 19 14]]

 [[39 32 65  9 57]
 [32 31 74 23 35]
 [75 55 28 34  0]
 [ 0 36 53  5 38]]]
```

Matrix Dimensions

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

print("\nRESHAPED TO (2,3) ARRAY")
b=a.reshape((2,3))
print(f"MATRIX DIM : {b.shape}")
print(f"MATRIX : {b}")

print("\nRESHAPED TO (6,1) ARRAY")
c=a.reshape((6,))
print(f"MATRIX DIM : {c.shape}")
print(f"MATRIX : {c}")
```

1	2
3	4
5	6

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 03.matrix.dim.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

RESHAPED TO (2,3) ARRAY
MATRIX DIM : (2, 3)
MATRIX : [[1 2 3]
 [4 5 6]]

RESHAPED TO (6,1) ARRAY
MATRIX DIM : (6,)
MATRIX : [1 2 3 4 5 6]
```

Array Slicing

```
import numpy as np
```

```
print("INITIAL ARRAY")
a = np.array( [ [1, 2, 11] , [3, 4, 12] , [5, 6, 13] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")
```

```
b=np.copy(a[1,:2])
print(f"\nMATRIX DIM : {b.shape}")
print(f"MATRIX : {b}")
```

```
a[1:,1]=8
print(f"\nMATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")
```

```
a[2:,1]=b
print(f"\nMATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")
```

```
mpekmezci@cobalt: $ python3 04.array.slicing.py
INITIAL ARRAY
MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  4 12]
           [ 5  6 13]]

MATRIX DIM : (2, )
MATRIX : [3 4]

MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  8 12]
           [ 5  8 13]]

MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  8 12]
           [ 5  3  4]]
```

Array Masking

```
import numpy as np

print("INITIAL ARRAY")
a = np.arange(15)
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

wanted_indices=[3,8,12]

b=np.copy(a[wanted_indices])
print(f"\nMATRIX DIM : {b.shape}")
print(f"MATRIX : {b}")
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 05.array.masking.py
INITIAL ARRAY
MATRIX DIM : (15,)
MATRIX : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

MATRIX DIM : (3,)
MATRIX : [ 3  8 12]
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$
```

Matrix Summation

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=a+a
print(f"\n MATRIX:{b} \n")
```

1	2
3	4
5	6

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 06.summation.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

MATRIX: [[ 2  4]
 [ 6  8]
 [10 12]]
```

Matrix Transpose

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=a.T
print(f"\n MATRIX:{b} \n")
```

1	2
3	4
5	6

```
ipekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ VI 07.transpose.py
ipekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 07.transpose.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
           [3 4]
           [5 6]]

MATRIX:[[1 3 5]
        [2 4 6]]
```

Matrix Multiplication

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=np.matmul(a.T , a)
print(f"\n MATRIX:{b} \n")
```

1	2
3	4
5	6

```
mpkmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ vi 08.multiplication.py
mpkmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 08.multiplication.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
           [3 4]
           [5 6]]

MATRIX: [[35 44]
          [44 56]]
```

Matrix Inverse

```

import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1,2,3] , [5,7,11] , [13,17,23] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=np.linalg.inv(a)
print(f"\n INVERSE MATRIX:{b} \n")

c=np.matmul(b,a)
print(f"\n A INVERSE * A :{c} \n")

```

1	2	3
5	7	11
13	17	23

Very close to 0

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 09.inverse.py
INITIAL ARRAY
MATRIX DIM : (3, 3)
MATRIX : [[ 1  2  3]
 [ 5  7 11]
 [13 17 23]]

INVERSE MATRIX:[[ -2.16666667  0.41666667  0.08333333]
 [ 2.33333333 -1.33333333  0.33333333]
 [-0.5          0.75         -0.25        ]]

A INVERSE * A :[[ 1.00000000e+00  1.08246745e-15  1.58206781e-15]
 [-4.99600361e-16  1.00000000e+00 -6.10522664e-16]
 [ 2.77555756e-16  9.43689571e-16  1.00000000e+00]]
```

Horizontal Stacking

```
import numpy as np

a = np.array( [ [1, 2, 3] , [4, 5, 6] , [7, 8, 9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12, 13] , [14, 15, 16] , [17, 18, 19] ] )
print(f"MATRIX B : {b}")

c=np.hstack((a,b))
print(f"\n HSTACK:{c} \n")
```

1	2	3	11	12	13
4	5	6	14	15	16
7	8	9	17	18	19

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 10.hstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

HSTACK:[[ 1  2  3 11 12 13]
 [ 4  5  6 14 15 16]
 [ 7  8  9 17 18 19]]
```

Vertical Stacking

```
import numpy as np

a = np.array( [ [1, 2, 3] , [4, 5, 6] , [7, 8, 9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12, 13] , [14, 15, 16] , [17, 18, 19] ] )
print(f"MATRIX B : {b}")

c=np.vstack((a,b))
print(f"\n VSTACK:{c} \n")
```

1	2	3		11	12	13
4	5	6		14	15	16
7	8	9		17	18	19

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 11.vstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

VSTACK:[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [11 12 13]
 [14 15 16]
 [17 18 19]]
```

Depth Stacking

```
import numpy as np

a = np.array( [ [1, 2, 3] , [4, 5, 6] , [7, 8, 9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12, 13] , [14, 15, 16] , [17, 18, 19] ] )
print(f"MATRIX B : {b}")

c=np.dstack((a,b))
print(f"\n DSTACK:{c} \n")
```

1	2	3		11	12	13
4	5	6		14	15	16
7	8	9		17	18	19

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 12.dstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

DSTACK:[[[ 1 11]
 [ 2 12]
 [ 3 13]]

 [[ 4 14]
 [ 5 15]
 [ 6 16]]

 [[ 7 17]
 [ 8 18]
 [ 9 19]]]
```

Horizontal Splitting

```
import numpy as np  
  
a = np.array( [ [1, 2, 3, 4] , [5, 6, 7, 8] ] )  
print(f"MATRIX A : {a}")
```

```
b,c=np.hsplit(a,2)  
print(f"\n First Split:{b}\n")  
print(f"\n Second Split:{c}\n")
```

1	2	3	4
5	6	7	8

```
mekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ vi 13.hsplit.py  
mekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 13.hsplit.py  
MATRIX A : [[1 2 3 4]  
 [5 6 7 8]]  
  
First Split:[[1 2]  
 [5 6]]  
  
Second Split:[[3 4]  
 [7 8]]
```

Vertical Splitting

```
import numpy as np

a = np.array( [ [1, 2] ,[3,4] , [5, 6], [7,8 ] ] )
print(f"MATRIX A : {a}")

b,c=np.vsplit(a,2)
print(f"\n First Split:{b} \n")
print(f"\n Second Split:{c} \n")
```

1	2
3	4
5	6
7	8

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 14.vsplit.py
MATRIX A : [[1 2]
 [3 4]
 [5 6]
 [7 8]]

First Split:[[1 2]
 [3 4]]

Second Split:[[5 6]
 [7 8]]
```

Distance – Root Mean Squared Error – Euclidean Distance

```
import numpy as np

a = np.array( [ [1, 2] ,[3,4] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [5, 6] ,[7,8] ] )
print(f"MATRIX B : {b}")

rmse = np.sqrt((np.square(a - b)).mean())
print(rmse)
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$RMSE(A, B) = \left(\frac{1}{N * M} \sum_{i=0}^N \sum_{j=0}^M (a_{ij} - b_{ij})^2 \right)^{\frac{1}{2}}$$

N=2, M=2
in this case

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 15_rmse.py
MATRIX A : [[1 2]
 [3 4]]
MATRIX B : [[5 6]
 [7 8]]
4.0
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$
```

Distance - Mean Absolute Error

```
import numpy as np  
  
a = np.array( [ [1, 2] ,[3,4] ] )  
print(f"MATRIX A : {a}")  
  
b = np.array( [ [5, 6] ,[7,8] ] )  
print(f"MATRIX B : {b}")  
  
mae=np.absolute(a-b).mean()  
print(mae)
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$MAE(A, B) = \frac{1}{N * M} \sum_{i=0}^N \sum_{j=0}^M |a_{ij} - b_{ij}|$$

N=2, M=2
in this case

```
print(mse)  
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 16.mae.py  
MATRIX A : [[1 2]  
 [3 4]]  
MATRIX B : [[5 6]  
 [7 8]]  
4.0
```

Distance - Cosine Distance

```
import numpy as np
from numpy.linalg import norm

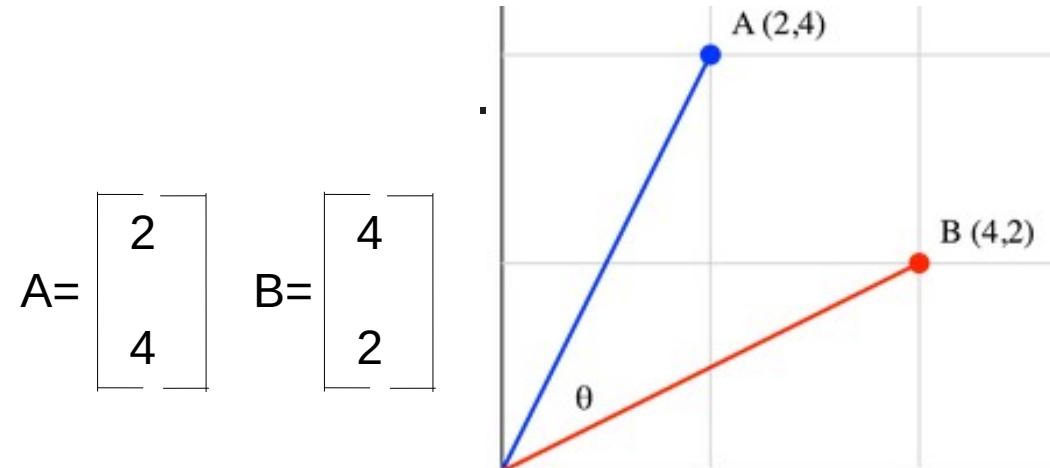
A = np.array([2, 4])
B = np.array([4, 2])

# compute cosine similarity
cosine = np.dot(A, B) / (norm(A) * norm(B))
print("Cosine Distance:", 1-cosine)
```

$$\text{COSINEDISTANCE}(A, B) = 1 - \text{COSINESIMILARITY}(A, B)$$

$$\text{COSINESIMILARITY}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \cos(\hat{A}B) = \cos(\theta)$$

$$\|A\| = \text{Euclidean (L2) Norm of } A = \left(\sum_{i=0}^N \sum_{j=0}^M (a_{ij})^2 \right)^{\frac{1}{2}}$$



```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 17.cosine.py
Cosine Distance: 0.2000000000000018
```

Distance – Cosine Distance - Matrix

```
import numpy as np
from numpy.linalg import norm

A = np.array([[1, 2, 2], [3, 2, 2], [-2, 1, -3]])
B = np.array([[4, 2, 4], [2, -2, 5], [3, 4, -4]])

cosine = np.sum(A * B, axis=1) / (norm(A, axis=1) * norm(B,
axis=1))
print("Cosine Similarity:", cosine)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 17.cosine.matrix.py
Cosine Similarity: [0.88888889 0.5066404  0.41739194]
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$
```

ENG 346

Data Structures and Algorithms for Artificial Intelligence

Matrix Operations and Numpy

Dr. Mehmet PEKMEZCI

mpekmezci@gtu.edu.tr

<https://github.com/mehmetpekmezci/GTU-ENG-346>

ENG-346 Teams code is **0uv7jlm**

ENG 346 – Data Structures and Algorithms for Artificial Intelligence

Codes

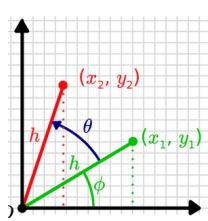
- <https://github.com/mehmetpekmezci/GTU-ENG-346/tree/main/matrix-numpy-codes>

Matrices

- Matrices can be used to store data
 - Image data ($n \times m$) matrix
 - Sound data ($n \times 1$) matrix
 - Any Log data ($n \times m$) matrix,
 - e.g. : log of humidity, temperature, ... values in a room
 - Geometric data
- Matrices can be used for Linear Transformations (Linear Function)
 - Rotation
 - Translation
 - Stretching

Matrices - Geometric Meaning

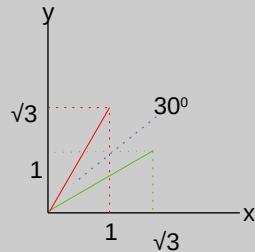
• Rotation



$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix}$$

NOTE: 3D rotation matrix is a 3x3 matrix, not a 2x2x2 tensor

Example :



$$\begin{bmatrix} 1 \\ \sqrt{3} \end{bmatrix} = \begin{bmatrix} \cos(\pi/6) & -\sin(\pi/6) \\ \sin(\pi/6) & \cos(\pi/6) \end{bmatrix} \begin{bmatrix} \sqrt{3} \\ 1 \end{bmatrix}$$

```
import numpy as np
theta = np.radians(30)
c, s = np.cos(theta), np.sin(theta)
R = np.array(((c, -s), (s, c)))
green_vector=np.array([np.sqrt(3),1])
print(green_vector)
red_vector=np.dot(R,green_vector)
print(red_vector)
```

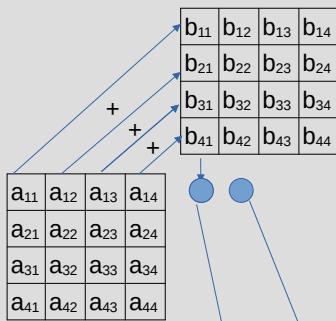
<https://articulatedrobotics.xyz/tutorials/coordinate-transforms/rotation-matrices-2d/>

ENG 346 – Data Structures and Algorithms for Artificial Intelligence

Multiplication of Matrices

Example : 4x4 matrix A=

A.B =



- We can multiply A and B if A(N,M) . B(M,K)
- A(N,M) . B(M,K) = C(N,K)

$$a_{11}.b_{12} + a_{12}.b_{22} + a_{13}.b_{32} + a_{14}.b_{42}$$

$$a_{11}.b_{11} + a_{12}.b_{21} + a_{13}.b_{31} + a_{14}.b_{41}$$

Matrix Example

- Example :

$$x + 2y = 4$$

$$3x - 4y = 5$$

$$\begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 4 \\ 5 \end{bmatrix}$$

$$A \cdot X = B$$

$$\begin{array}{ccc} & \begin{bmatrix} x \\ y \end{bmatrix} & \\ \begin{bmatrix} 1 & 2 \\ 3 & -4 \end{bmatrix} & + & \begin{bmatrix} x+2y \\ 3x-4y \end{bmatrix} \\ & & = \begin{bmatrix} 4 \\ 5 \end{bmatrix} \end{array} \rightarrow \begin{array}{l} x + 2y = 4 \\ 3x - 4y = 5 \end{array}$$

Determinants and Inverse of a Matrix

Example : 4x4 matrix A=

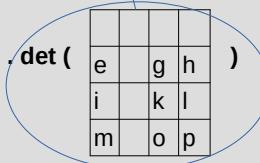
a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

Minor of element b is $\det(M_{12})$,
Cofactor(A) is a matrix composed of C_{ij} .
 $C_{ij} = (-1)^{i+j} \det(M_{ij})$

e	g	h
i	k	l
m	o	p

$$\det(A) = |A| =$$

$$a \cdot \det \left(\begin{array}{ccc} & & \\ f & g & h \\ & j & k & l \\ & n & o & p \end{array} \right) - b \cdot \det \left(\begin{array}{ccc} & & \\ e & g & h \\ i & k & l \\ m & o & p \end{array} \right)$$



C ₁₁	C ₁₂	C ₁₃	C ₁₄
C ₂₁	C ₂₂	C ₂₃	C ₂₄
C ₃₁	C ₃₂	C ₃₃	C ₃₄
C ₄₁	C ₄₂	C ₄₃	C ₄₄

$$+ c \cdot \det \left(\begin{array}{ccc} & & \\ e & f & h \\ i & j & l \\ m & n & p \end{array} \right) - d \cdot \det \left(\begin{array}{ccc} & & \\ e & f & g \\ i & j & k \\ m & n & o \end{array} \right)$$

Inverse of A
 $A^{-1} = 1/\det(A) \cdot (\text{Cofactor}(A))^T$

NO INVERSE
if $\det(A)=0$ or
matrix is not square

Transpose and Inverse Properties

$$A \cdot A^{-1} = A^{-1} \cdot A = I$$

$$(A^{-1})^{-1} = A$$

$$(A^T)^T = A$$

$$(A^T)^{-1} = (A^{-1})^T$$

$$(AB)^T = B^T \cdot A^T$$

$$(A+B)^T = A^T + B^T$$

$$(AB)^{-1} = B^{-1} \cdot A^{-1}$$

$$(k \cdot A)^{-1} = 1/k \cdot A^{-1}$$

$$(k \cdot A)^T = k \cdot A^T$$

$$\det(A^T) = \det(A)$$

$$\det(A^{-1}) = 1/\det(A)$$

$$\det(A \cdot B) = \det(A) \cdot \det(B)$$

Tensors (Multilinear Map)

- Vectors = 1st order Tensor

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix} \quad (1D \text{ arrays})$$

- Matrices = 2nd order Tensor

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad (2D \text{ arrays})$$

- 3rd order Tensor , is a stack of matrices (3D arrays)

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \longrightarrow 2 \times 2 \times 3 \text{ Tensor (} l, j, k \text{ indices)}$$

- Tensor may have n order.

Matrix Operations

- Matrix/Tensor Construction
- Dimensions (Shape) and reshaping, array slicing, masking.
- Matrix Summation
- Transpose of the Matrix
- Matrix Multiplication
- Inverse of the Matrix
- Stacking (Vertical, Horizontal, Depth)
- Splitting (Vertical, Horizontal)
- Distances (Mean Squared Error, Mean Absolute Error, Cosine Similarity)
- **NOT COVERED : Eigenvalues, Eigenvectors, SVD , Tensor Dot**

Matrix/Tensor Construction

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \longleftrightarrow$$

```
import numpy as np
a = np.array( [ [1, 2], [3, 4] ] )
print(a)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 01.arrays_matrices.py
[[1 2]
 [3 4]]
```

$\begin{bmatrix} 10 & 10 \\ 10 & 10 \end{bmatrix}$	np.full((2, 2), 10,dtype=int) # 2 rows, 2 columns
--	---

$\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$	number_of_rows=2 np.identity(number_of_rows,dtype=int)
--	--

$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 0.0 \end{bmatrix}$	np.zeros(2) # default dtype=float
--	---

$\begin{bmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{bmatrix}$	number_of_rows=2 np.identity(number_of_rows,dtype=float)
--	--

Matrix/Tensor Construction

```
import numpy as np
np.random.seed(0) # seed for reproducibility
random_tensor = np.random.randint(100, size=(3, 4, 5))
print(random_tensor)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 02.random.py
[[[44 47 64 67 67]
 [ 9 83 21 36 87]
 [70 88 88 12 58]
 [65 39 87 46 88]]

 [[81 37 25 77 72]
 [ 9 20 80 69 79]
 [47 64 82 99 88]
 [49 29 19 19 14]]

 [[39 32 65  9 57]
 [32 31 74 23 35]
 [75 55 28 34  0]
 [ 0 36 53  5 38]]]
```

Matrix Dimensions

```
import numpy as np
print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

print("\nRESHAPED TO (2,3) ARRAY")
b=a.reshape((2,3))
print(f"MATRIX DIM : {b.shape}")
print(f"MATRIX : {b}")

print("\nRESHAPED TO (6,1) ARRAY")
c=a.reshape((6,))
print(f"MATRIX DIM : {c.shape}")
print(f"MATRIX : {c}")
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 03.matrix.dim.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

RESHAPED TO (2,3) ARRAY
MATRIX DIM : (2, 3)
MATRIX : [[1 2 3]
 [4 5 6]]

RESHAPED TO (6,1) ARRAY
MATRIX DIM : (6,)
MATRIX : [1 2 3 4 5 6]
```

Array Slicing

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1, 2,11] , [3, 4,12] , [5,6,13] ] )
print(f" MATRIX DIM : {a.shape}")
print(f" MATRIX : {a}")

b=np.copy(a[1,:])
print(f"\n MATRIX DIM : {b.shape}")
print(f" MATRIX : {b}")

a[1:,1]=8
print(f"\n MATRIX DIM : {a.shape}")
print(f" MATRIX : {a}")

a[2:,1]=b
print(f"\n MATRIX DIM : {a.shape}")
print(f" MATRIX : {a}")
```

```
mpekmezci@cobalt:~$ python3 04.array.slicing.py
INITIAL ARRAY
MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  4 12]
           [ 5  6 13]]

MATRIX DIM : (2, 4)
MATRIX : [3 4]

MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  8 12]
           [ 5  8 13]]

MATRIX DIM : (3, 3)
MATRIX : [[ 1  2 11]
           [ 3  8 12]
           [ 5  3  4]]
```

Array Masking

```
import numpy as np

print("INITIAL ARRAY")
a = np.arange(15)
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

wanted_indices=[3,8,12]

b=np.copy(a[wanted_indices])
print(f"\nMATRIX DIM : {b.shape}")
print(f"MATRIX : {b}")
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 05.array.masking.py
INITIAL ARRAY
MATRIX DIM : (15,)
MATRIX : [ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14]

MATRIX DIM : (3,)
MATRIX : [ 3  8 12]
```

Matrix Summation

```
import numpy as np
print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f" MATRIX DIM : {a.shape}")
print(f" MATRIX : {a}")

b=a+a
print(f"\n MATRIX:{b} \n")
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 06.summation.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

MATRIX:[[ 2  4]
 [ 6  8]
 [10 12]]
```

Matrix Transpose

```
import numpy as np
print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=a.T
print(f"\n MATRIX:{b} \n")
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ vi 07.transpose.py
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 07.transpose.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

MATRIX:[[1 3 5]
 [2 4 6]]
```

Matrix Multiplication

```
import numpy as np
print("INITIAL ARRAY")
a = np.array( [ [1, 2] , [3, 4] , [5,6] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")
b=np.matmul(a.T , a)
print(f"\n MATRIX:{b} \n")
```

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 08.multiplication.py
INITIAL ARRAY
MATRIX DIM : (3, 2)
MATRIX : [[1 2]
 [3 4]
 [5 6]]

MATRIX: [[35 44]
 [44 56]]
```

Matrix Inverse

```
import numpy as np

print("INITIAL ARRAY")
a = np.array( [ [1,2,3] , [5,7,11] , [13,17,23] ] )
print(f"MATRIX DIM : {a.shape}")
print(f"MATRIX : {a}")

b=np.linalg.inv(a)
print(f"\n INVERSE MATRIX:{b} \n")

c=np.matmul(b,a)
print(f"\n A INVERSE * A :{c} \n")
```

$$\begin{bmatrix} 1 & 2 & 3 \\ 5 & 7 & 11 \\ 13 & 17 & 23 \end{bmatrix}$$

Very close to 0

```
ipekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 09.inverse.
INITIAL ARRAY
MATRIX DIM : (3, 3)
MATRIX : [[ 1  2  3]
 [ 5  7 11]
 [13 17 23]]

INVERSE MATRIX:[[-2.16666667  0.41666667  0.08333333]
 [ 2.33333333 -1.33333333  0.33333333]
 [-0.5          0.75        -0.25       ]]

A INVERSE * A :[[ 1.00000000e+00  1.08246745e-15  0.58206781e-15]
 [-4.99600361e-16  1.00000000e+00 -6.10622664e-16]
 [ 2.77555756e-16  9.43689571e-16  1.00000000e+00]]
```

Horizontal Stacking

```
import numpy as np

a = np.array( [ [1, 2,3] , [4, 5, 6 ] , [7,8,9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12,13] , [14, 15, 16 ] , [17,18,19] ] )
print(f"MATRIX B : {b}")

c=np.hstack((a,b))
print(f"\n HSTACK:{c}\n")
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 10.hstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

HSTACK:[[ 1  2  3 11 12 13]
 [ 4  5  6 14 15 16]
 [ 7  8  9 17 18 19]]
```

1	2	3	11	12	13
4	5	6	14	15	16
7	8	9	17	18	19

Vertical Stacking

```
import numpy as np

a = np.array( [ [1, 2,3] , [4, 5, 6 ] , [7,8,9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12,13] , [14, 15, 16 ] , [17,18,19] ] )
print(f"MATRIX B : {b}")

c=np.vstack((a,b))
print(f"\n VSTACK:{c}\n")
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 11.vstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

VSTACK:[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [11 12 13]
 [14 15 16]
 [17 18 19]]
```

1	2	3	11	12	13
4	5	6	14	15	16
7	8	9	17	18	19

Depth Stacking

```
import numpy as np

a = np.array( [ [1, 2,3] , [4, 5, 6 ] , [7,8,9] ] )
print(f"MATRIX A : {a}")

b = np.array( [ [11, 12,13] , [14, 15, 16 ] , [17,18,19] ] )
print(f"MATRIX B : {b}")

c=np.dstack((a,b))
print(f"\n DSTACK:{c}\n")
```

1	2	3	11	12	13
4	5	6	14	15	16
7	8	9	17	18	19

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 12.dstack.py
MATRIX A : [[1 2 3]
 [4 5 6]
 [7 8 9]]
MATRIX B : [[11 12 13]
 [14 15 16]
 [17 18 19]]

DSTACK:[[[ 1 11]
 [ 2 12]
 [ 3 13]]

 [[ 4 14]
 [ 5 15]
 [ 6 16]]

 [[ 7 17]
 [ 8 18]
 [ 9 19]]]
```

Horizontal Splitting

```
import numpy as np

a = np.array( [ [1, 2, 3, 4] , [5, 6, 7, 8] ] )
print(f"MATRIX A : {a}")

b,c=np.hsplit(a,2)
print(f"\n First Split:{b}\n")
print(f"\n Second Split:{c}\n")
```

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ vi 13.hsplit.py
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 13.hsplit.py
MATRIX A : [[1 2 3 4]
 [5 6 7 8]]

First Split:[[1 2]
 [5 6]]

Second Split:[[3 4]
 [7 8]]
```

Vertical Splitting

```
import numpy as np

a = np.array( [ [1, 2] ,[3,4] , [5, 6], [7,8 ]])
print(f"MATRIX A : {a}")

b,c=np.vsplit(a,2)
print(f"\n First Split:{b} \n")
print(f"\n Second Split:{c} \n")
```

1	2
3	4
5	6
7	8

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 14.vsplit.py
MATRIX A : [[1 2]
 [3 4]
 [5 6]
 [7 8]]

First Split:[[1 2]
 [3 4]]

Second Split:[[5 6]
 [7 8]]
```

Distance – Root Mean Squared Error – Euclidean Distance

```
import numpy as np
a = np.array( [ [1, 2],[3,4] ] )
print(f"MATRIX A : {a}")
b = np.array( [ [5, 6],[7,8] ] )
print(f"MATRIX B : {b}")
rmse = np.sqrt((np.square(a - b)).mean())
print(rmse)
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$RMSE(A, B) = \left(\frac{1}{N * M} \sum_{i=0}^N \sum_{j=0}^M (a_{ij} - b_{ij})^2 \right)^{\frac{1}{2}} \quad N=2, M=2 \text{ in this case}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 15.rmse.py
MATRIX A : [[1 2]
 [3 4]]
MATRIX B : [[5 6]
 [7 8]]
4.0
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$
```

Distance - Mean Absolute Error

```
import numpy as np
a = np.array( [ [1, 2] ,[3,4] ] )
print(f"MATRİX A : {a}")
b = np.array( [ [5, 6],[7,8] ] )
print(f"MATRİX B : {b}")

mae=np.absolute(a-b).mean()
print(mae)
```

$$A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \quad B = \begin{bmatrix} 5 & 7 \\ 6 & 8 \end{bmatrix}$$

$$MAE(A, B) = \frac{1}{N * M} \sum_{i=0}^N \sum_{j=0}^M |a_{ij} - b_{ij}|$$

N=2, M=2
in this case

```
print(mse)
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 16.mae.py
MATRİX A : [[1 2]
 [3 4]]
MATRİX B : [[5 6]
 [7 8]]
4.0
```

Distance - Cosine Distance

```
import numpy as np
from numpy.linalg import norm

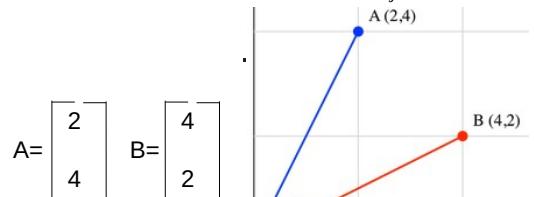
A = np.array([2, 4])
B = np.array([4, 2])

# compute cosine similarity
cosine = np.dot(A, B) / (norm(A) * norm(B))
print("Cosine Distance:", 1-cosine)
```

$$\text{COSINEDISTANCE}(A, B) = 1 - \text{COSINESIMILARITY}(A, B)$$

$$\text{COSINESIMILARITY}(A, B) = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \cos(\hat{A}B) = \cos(\theta)$$

$$\|A\| = \text{Euclidean}(L_2) \text{ Norm of } A = \left(\sum_{i=0}^N \sum_{j=0}^M (a_{ij})^2 \right)^{\frac{1}{2}}$$



$$A = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \quad B = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 17.cosine.py
Cosine Distance: 0.20000000000000018
```

Distance - Cosine Distance - Matrix

```
import numpy as np
from numpy.linalg import norm

A = np.array([[1, 2, 2], [3, 2, 2], [-2, 1, -3]])
B = np.array([[4, 2, 4], [2, -2, 5], [3, 4, -4]])

cosine = np.sum(A * B, axis=1) / (norm(A, axis=1) * norm(B,
axis=1))
print("Cosine Similarity:", cosine)
```

```
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ python3 17.cosine.matrix.py
Cosine Similarity: [0.88888889 0.5066404  0.41739194]
mpekmezci@cobalt:~/workspace/GTU-ENG-346-PRIVATE/matrix-numpy-codes$ █
```