AI VIETNAM
All-in-One Course
(TA Session)

Working with matrix in Numpy

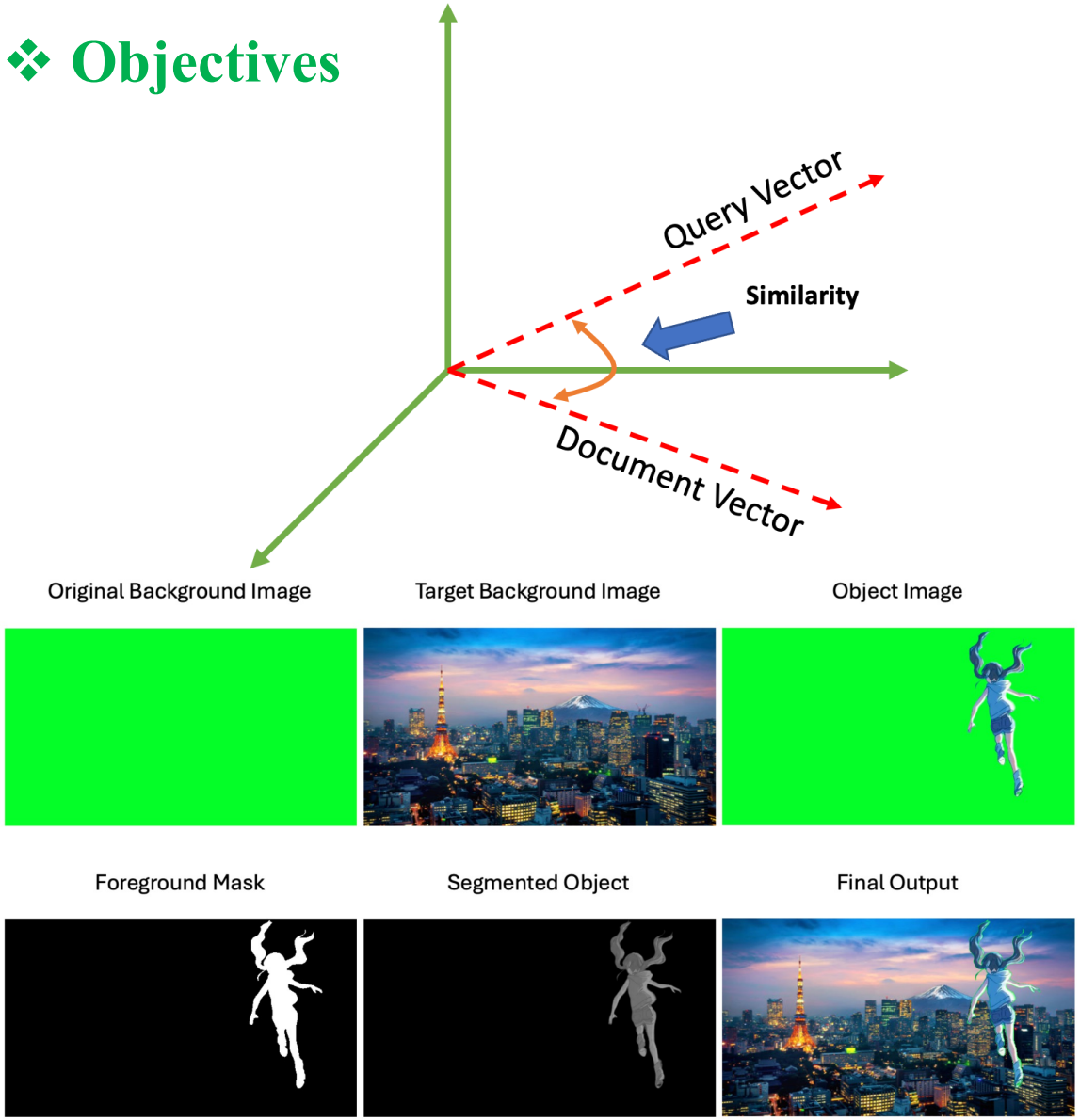
Exercise

Dinh-Thang Duong – TA

Year 2024

Getting Started

❖ Objectives



In this session, we will discuss about:

- Vector and matrix.
- Matrix property.
- Practice coding with some matrix operations: matrix multiplications, inverse matrix…
- Finding eigenvalue and eigenvector.
- Cosine similarity.
- Background Subtraction.

Outline

- Exercise 01
- Exercise 02
- Exercise 03
- Exercise 04
- Question

Exercise 01

Exercise 01

❖ Introduction

Problem statement: Using python, re-implement functions related to basic matrix operations:

1. Length of a vector.
2. Dot product.
3. Multiply matrix by matrix.
4. Multiply matrix by vector.
5. Matrix inverse.

Exercise 01

❖ Revise

x ~ (1)

5

Scalar

x ~ (4)

5
9
2
1

Vector

x ~ (4, 4)

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

Matrix

Exercise 01

❖ Revise

x ~ (4, 5)

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

Matrix

x ~ (5, 4)

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

Transpose of a matrix

x ~ (4, 4)

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

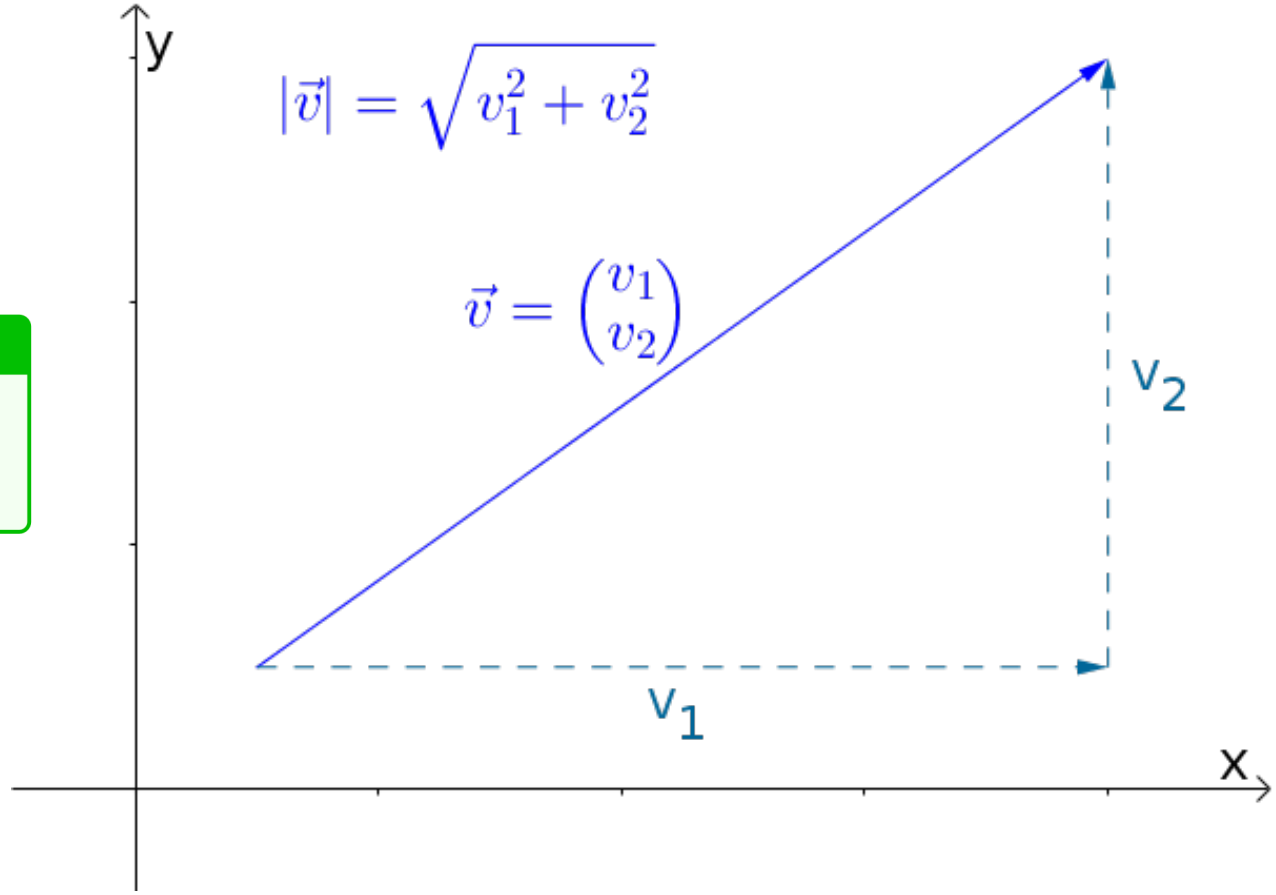
Square Matrix

Exercise 01

❖ Length of a vector

1.1. Length of a vector

- Vector: $\mathbf{v} = [v_1, v_2, ...v_n]^T$
- Length of a vector: $\| \mathbf{v} \| = \sqrt{v_1^2 + v_2^2 + ... + v_n^2}$



https://lakschool.com/en/math/vectors/vector-length-or-magnitude

Exercise 01

❖ Length of a vector: Ver 1

```python
import numpy as np

def compute_vector_length(vector):
    norm = np.sqrt(np.sum([v ** 2 for v in vector]))

    return norm

vector = np.array([-2, 4, 9, 21])
result = compute_vector_length([vector])
print(round(result, 2))
```

```
23.28
```

- $x = \begin{bmatrix} x_1 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{bmatrix}$
- $\|x\| = \sqrt{\sum_{i=1} x_i^2} = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$

Exercise 01

❖ Length of a vector: Ver 2

```python
import numpy as np

def compute_vector_length(vector):
    len_of_vector = np.linalg.norm(vector)

    return len_of_vector

vector = np.array([5, 9, -2, -1])
result = compute_vector_length([vector])
print(round(result, 2))
```

```
10.54
```

numpy.linalg.norm

linalg.norm(*x*, *ord=None*, *axis=None*, *keepdims=False*) [source]

Matrix or vector norm.

This function is able to return one of eight different matrix norms, or one of an infinite number of vector norms (described below), depending on the value of the `ord` parameter.

Parameters:

x : *array_like*

Input array. If *axis* is None, *x* must be 1-D or 2-D, unless *ord* is None. If both *axis* and *ord* are None, the 2-norm of `x.ravel` will be returned.

ord : *{non-zero int, inf, -inf, 'fro', 'nuc'}, optional*

Order of the norm (see table under `Notes`). inf means numpy's `inf` object. The default is None.

axis : *{None, int, 2-tuple of ints}, optional.*

If *axis* is an integer, it specifies the axis of *x* along which to compute the vector norms. If *axis* is a 2-tuple, it specifies the axes that hold 2-D matrices, and the matrix norms of these matrices are computed. If *axis* is None then either a vector norm (when *x* is 1-D) or a matrix norm (when *x* is 2-D) is returned. The default is None.

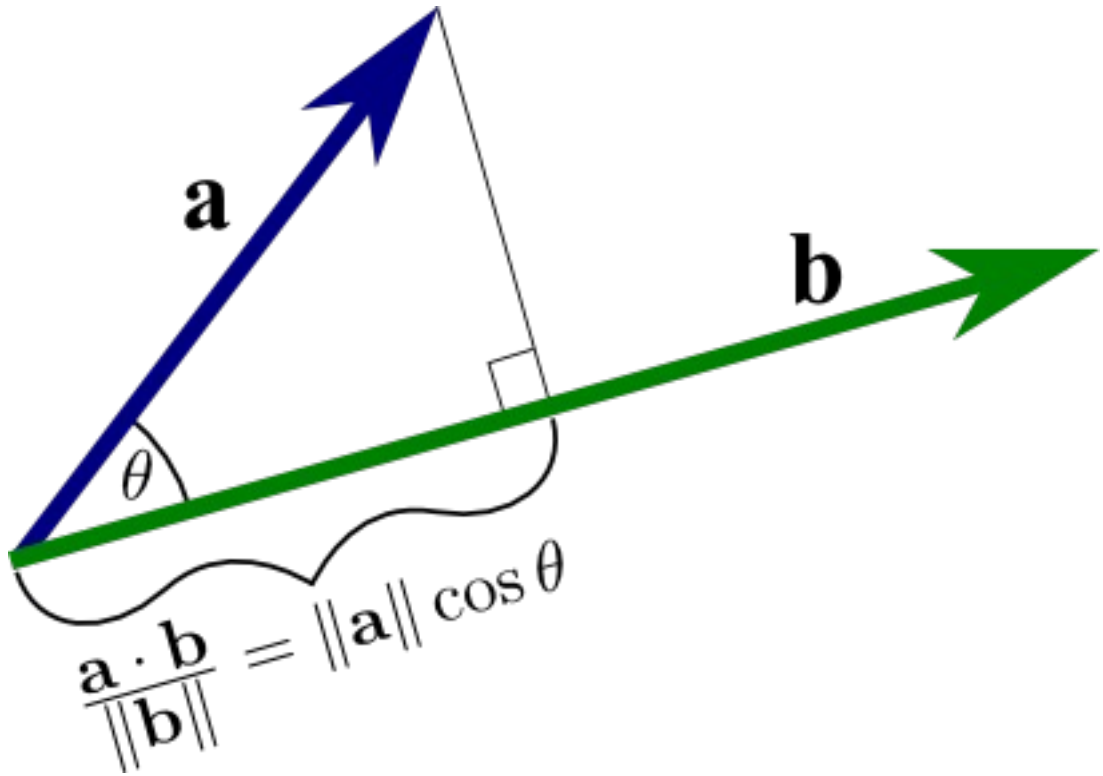
https://numpy.org/doc/stable/reference/generated/numpy.linalg.norm.html

Exercise 01

❖ Dot product

1.2. Dot product

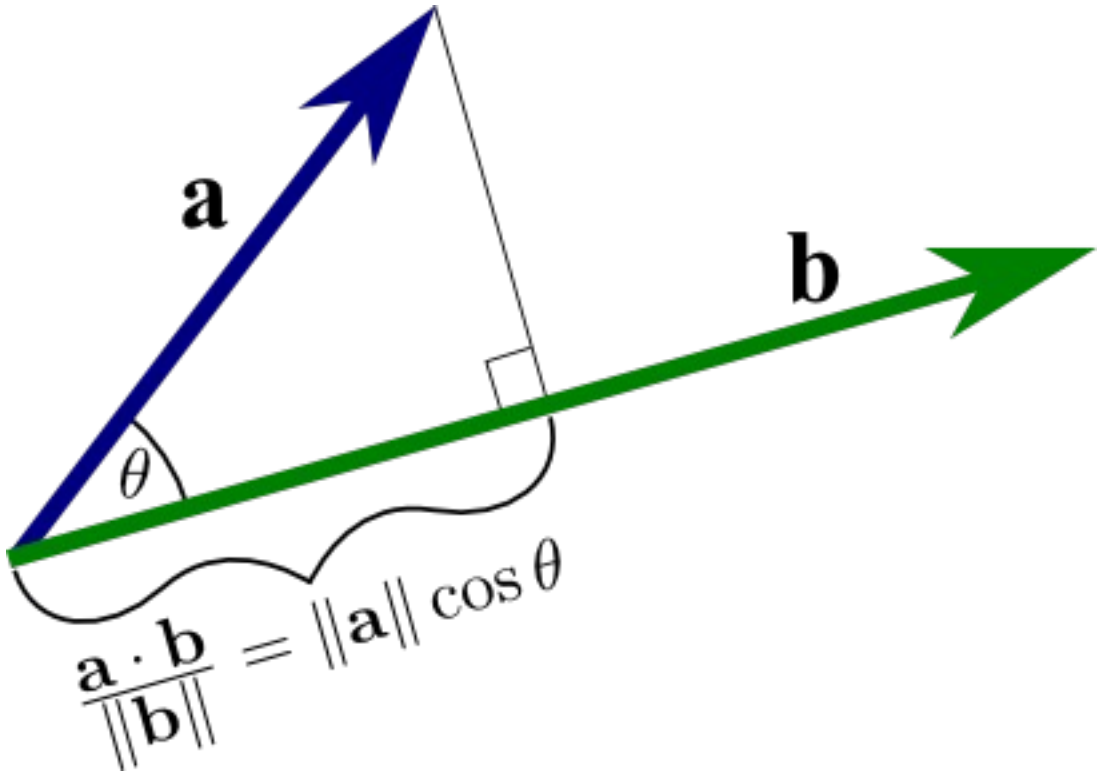
- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ ... \\ v_n \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \\ ... \\ u_n \end{bmatrix}$
- Dot Product: $\mathbf{v} \cdot \mathbf{u} = v_1 * u_1 + v_2 * u_2 + ... + v_n * u_n$



https://mathinsight.org/dot_product

Exercise 01

❖ Dot product between two vectors



- $a \cdot b = \|a\|\|b\| \cos\theta$
- $a \cdot b = \sum_{i=1} a_i b_i$

- $a = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \end{bmatrix} b = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$
- $a \cdot b = a_1 b_1 + a_2 b_2 + a_3 b_3$

https://mathinsight.org/dot_product

AI VIETNAM
All-in-One Course
(TA Session)

Exercise 01

❖ Dot product between two vectors

```python
def compute_dot_product(vector1, vector2):
    result = np.dot(vector1, vector2)

    return result
```

```python
v1 = np.array([0, 1, -1, 2])
v2 = np.array([2, 5, 1, 0])
result = compute_dot_product(v1, v2)

print(round(result,2))
```

```
4
```

numpy.dot

numpy.dot(*a*, *b*, *out*=*None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using **matmul** or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to **multiply** and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

It uses an optimized BLAS library when possible (see **numpy.linalg**).

https://numpy.org/doc/stable/reference/generated/numpy.dot.html

Exercise 01

❖ Dot product between two vectors

```
1 x = np.array([[1, 2],
2               [3, 4]])
3 k = np.array([1, 2])
4 print('result  \n', compute_dot_product(x, k))
5 print('result 1 \n', x.dot(k))
```

```
result
 [ 5 11]
result 1
 [ 5 11]
```

numpy.ndarray.dot

method

ndarray.dot(*b*, *out=None*)

Dot product of two arrays.

Refer to **numpy.dot** for full documentation.

See also

numpy.dot
equivalent function

https://numpy.org/doc/1.21/reference/generated/numpy.ndarray.dot.html

Exercise 01

❖ Dot product between two vectors

```
1 x = np.array([[-1, 2],
2                [3, -4]])
3 k = np.array([1, 2])
4 print('result \n', x@k)
```

```
result
 [ 3 -5]
```

Abstract

This PEP proposes a new binary operator to be used for matrix multiplication, called `@`. (Mnemonic: `@` is `*` for mATrices.)

Specification

A new binary operator is added to the Python language, together with the corresponding in-place version:

Op	Precedence/associativity	Methods
`@`	Same as `*`	`__matmul__`, `__rmatmul__`
`@=`	n/a	`__imatmul__`

No implementations of these methods are added to the builtin or standard library types. However, a number of projects have reached consensus on the recommended semantics for these operations; see Intended usage details below for details.

For details on how this operator will be implemented in CPython, see Implementation details.

https://numpy.org/doc/stable/reference/generated/numpy.dot.html

Exercise 01

❖ Multiply matrix with a vector

1.3. Multiplying a vector by a matrix

- Matrix: $\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$, $\mathbf{A} \in R^{m*n}$
- Vector: $\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ \dots \\ v_n \end{bmatrix}$, $\mathbf{v} \in R^n$
- $\mathbf{c} = \mathbf{A}\mathbf{v} = \begin{bmatrix} a_{11} * v_1 + \dots + a_{1n} * v_n \\ \dots \\ a_{m1} * v_1 + \dots + a_{mn} * v_n \end{bmatrix}$,

  $\mathbf{c} \in R^n$

Exercise 01

❖ Multiply matrix with a vector

```python
def matrix_multi_vector(matrix, vector):
    result = np.dot(matrix, vector)
    return result
```

```python
m = np.array([[-1, 1, 1], [0, -4, 9]])
v = np.array([0, 2, 1])
result = matrix_multi_vector(m, v)
print(result)
```

```
[3 1]
```

$$\begin{bmatrix} 1 & 0 & 2 & 0 \\ 0 & 3 & 0 & 4 \\ 0 & 0 & 5 & 0 \\ 6 & 0 & 0 & 7 \end{bmatrix} \cdot \begin{bmatrix} 2 \\ 5 \\ 1 \\ 8 \end{bmatrix} = \begin{bmatrix} 4 \\ 47 \\ 5 \\ 68 \end{bmatrix}$$

Exercise 01

❖ Multiply matrix with a matrix

1.4. Multiplying a matrix by a matrix

- Matrix A: $\mathbf{A} = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \dots & \dots & \dots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}, \mathbf{A} \in R^{m*n}$
- Matrix B: $\mathbf{B} = \begin{bmatrix} b_{11} & \dots & b_{1k} \\ \dots & \dots & \dots \\ b_{n1} & \dots & b_{nk} \end{bmatrix}, \mathbf{B} \in R^{n*k}$
- $\mathbf{C} = \mathbf{AB} = \begin{bmatrix} a_{11} * b_{11} + \dots + a_{1n} * b_{n1} & \dots & a_{11} * b_{1k} + a_{1n} * b_{nk} \\ \dots & \dots & \dots \\ a_{m1} * b_{11} + \dots + a_{mn} * b_{n1} & \dots & a_{m1} * b_{1k} + a_{mn} * b_{nk} \end{bmatrix},$

  $\mathbf{C} \in R^{m*k}$

Exercise 01

❖ Multiply matrix with a matrix

$$\begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix} \begin{bmatrix} b_1 & b_2 & b_3 \\ b_4 & b_5 & b_6 \\ b_7 & b_8 & b_9 \end{bmatrix} = \begin{bmatrix} c_1 & c_2 & c_3 \\ c_4 & c_5 & c_6 \\ c_7 & c_8 & c_9 \end{bmatrix}$$

https://charchithowitzer.medium.com/matrix-multiplication-why-is-it-a-big-deal-cc8ef7490008

Exercise 01

❖ Multiply matrix with a matrix

```python
def matrix_multi_matrix(matrix1, matrix2):
    result = np.dot(matrix1, matrix2)
    return result
```

```python
m1 = np.array([[0, 1, 2], [2, -3, 1]])
m2 = np.array([[1, -3],[6, 1], [0, -1]])
result = matrix_multi_matrix(m1, m2)
print(result)
```

```
[[  6  -1]
 [-16 -10]]
```

numpy.dot

numpy.dot(*a*, *b*, *out=None*)

Dot product of two arrays. Specifically,

- If both *a* and *b* are 1-D arrays, it is inner product of vectors (without complex conjugation).
- If both *a* and *b* are 2-D arrays, it is matrix multiplication, but using `matmul` or `a @ b` is preferred.
- If either *a* or *b* is 0-D (scalar), it is equivalent to `multiply` and using `numpy.multiply(a, b)` or `a * b` is preferred.
- If *a* is an N-D array and *b* is a 1-D array, it is a sum product over the last axis of *a* and *b*.
- If *a* is an N-D array and *b* is an M-D array (where `M>=2`), it is a sum product over the last axis of *a* and the second-to-last axis of *b*:

```
dot(a, b)[i,j,k,m] = sum(a[i,j,:] * b[k,:,m])
```

It uses an optimized BLAS library when possible (see `numpy.linalg`).

Exercise 01

❖ Inverse matrix

1.5 Matrix inverse

- Matrix A: $\mathbf{A} = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, $\mathbf{A} \in R^{2*2}$
- Determinant of $\mathbf{A} \in R^{2*2}$: $det(\mathbf{A}) = ad - bc$
- if $det(\mathbf{A}) \neq 0$ $\mathbf{A}$ is invertible
- Inverse Matrix: $\mathbf{A^{-1}} = \frac{1}{det(\mathbf{A})} \begin{bmatrix} d & -b \\ -c & a \end{bmatrix}$

Exercise 01

❖ Inverse matrix

$$AA^{-1} = \begin{bmatrix} 1 & 0 \\ -1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} 1 \cdot 1 + 0 \cdot 1 & 1 \cdot 0 + 0 \cdot 1 \\ -1 \cdot 1 + 1 \cdot 1 & -1 \cdot 0 + 1 \cdot 1 \end{bmatrix}$$

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

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

Identity Matrix

https://www.statlect.com/matrix-algebra/inverse-matrix

Exercise 01

❖ Inverse matrix

```python
def inverse_matrix(matrix):
    result = np.linalg.inv(matrix)

    return result
```

```python
m1 = np.array([[-2, 6], [8, -4]])
result = inverse_matrix(m1)
print(result)
```

```
[[0.1  0.15]
 [0.2  0.05]]
```

```python
i = m1 @ result
i = np.round(i, 2)
print(i)
```

```
[[1. 0.]
 [0. 1.]]
```

numpy.linalg.inv

linalg.inv(*a*) [source]

Compute the inverse of a matrix.

Given a square matrix *a*, return the matrix *ainv* satisfying `a @ ainv = ainv @ a = eye(a.shape[0])`.

Parameters:

a : *(..., M, M) array_like*

Matrix to be inverted.

Returns:

ainv : *(..., M, M) ndarray or matrix*

Inverse of the matrix *a*.

Raises:

LinAlgError

If *a* is not square or inversion fails.

Exercise 01

❖ Revise

Some matrix properties:

- $AB \neq BA$ $(A \in \mathbb{R}^{m \times m}, B \in \mathbb{R}^{m \times m})$
- $(rA)^T = rA^T$ $(A \in \mathbb{R}^{m \times N})$
- $(A + B)^T = A^T + B^T$ $(A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n})$
- $(AB)^T = B^T A^T$ $(A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k})$

$$A + B = B + A$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix}$$

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n}$$

$$A + B = \begin{bmatrix} (a_{11} + b_{11}) & \cdots & (a_{1n} + b_{1n}) \\ \vdots & \ddots & \vdots \\ (a_{m1} + b_{m1}) & \cdots & (a_{mn} + b_{mn}) \end{bmatrix} = \begin{bmatrix} (b_{11} + a_{11}) & \cdots & (b_{1n} + a_{1n}) \\ \vdots & \ddots & \vdots \\ (b_{m1} + a_{m1}) & \cdots & (b_{mn} + a_{mn}) \end{bmatrix} = B + A$$

Exercise 01

❖ Revise

$$AB \neq BA$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix}, \quad B = \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mm} \end{bmatrix}$$

$$A \in \mathbb{R}^{m \times m}, B \in \mathbb{R}^{m \times m}$$

$$AB = \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mm} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11} + \cdots + a_{1m}b_{m1}) & \cdots & (a_{11}b_{1m} + \cdots + a_{1m}b_{mm}) \\ \vdots & \ddots & \vdots \\ (a_{m1}b_{11} + \cdots + a_{mm}b_{m1}) & \cdots & (a_{m1}b_{1m} + \cdots + a_{mm}b_{mm}) \end{bmatrix}$$

$$BA = \begin{bmatrix} b_{11} & \cdots & b_{1m} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mm} \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{1m} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mm} \end{bmatrix} = \begin{bmatrix} (b_{11}a_{11} + \cdots + b_{1m}a_{m1}) & \cdots & (b_{11}a_{1m} + \cdots + b_{1m}a_{mm}) \\ \vdots & \ddots & \vdots \\ (b_{m1}a_{11} + \cdots + b_{mm}a_{m1}) & \cdots & (b_{m1}a_{1m} + \cdots + b_{mm}a_{mm}) \end{bmatrix}$$

Exercise 01

❖ Revise

$$(rA)^T = rA^T$$

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, \qquad r$$

$$A \in \mathbb{R}^{m \times n}$$

$$(rA)^T = \left(r \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \right)^T = \left(\begin{bmatrix} r \times a_{11} & \cdots & r \times a_{1n} \\ \vdots & \ddots & \vdots \\ r \times a_{m1} & \cdots & r \times a_{mn} \end{bmatrix} \right)^T = \begin{bmatrix} r \times a_{11} & \cdots & r \times a_{m1} \\ \vdots & \ddots & \vdots \\ r \times a_{1n} & \cdots & r \times a_{mn} \end{bmatrix} = r \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}$$

$$= r \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix} = r \left(\begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}^T \right)^T = r \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \right)^T = rA^T$$

Exercise 01

❖ Revise

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

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix}$$

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n}$$

$$(A+B)^T = \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \right)^T = \left(\begin{bmatrix} (a_{11}+b_{11}) & \cdots & (a_{1n}+b_{1n}) \\ \vdots & \ddots & \vdots \\ (a_{m1}+b_{m1}) & \cdots & (a_{mn}+b_{mn}) \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} (a_{11}+b_{11}) & \cdots & (a_{m1}+b_{m1}) \\ \vdots & \ddots & \vdots \\ (a_{1n}+b_{1n}) & \cdots & (a_{mn}+b_{mn}) \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{m1} \\ \vdots & \ddots & \vdots \\ b_{1n} & \cdots & b_{mn} \end{bmatrix}$$

$$= \left(\begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}^T \right)^T + \left(\begin{bmatrix} b_{11} & \cdots & b_{m1} \\ \vdots & \ddots & \vdots \\ b_{1n} & \cdots & b_{mn} \end{bmatrix}^T \right)^T = \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \right)^T + \left(\begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \right)^T$$

$$= A^T + B^T$$

Exercise 01

❖ Revise

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

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix}$$

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{m \times n}$$

$$(A+B)^T = \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \right)^T = \left(\begin{bmatrix} (a_{11}+b_{11}) & \cdots & (a_{1n}+b_{1n}) \\ \vdots & \ddots & \vdots \\ (a_{m1}+b_{m1}) & \cdots & (a_{mn}+b_{mn}) \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} (a_{11}+b_{11}) & \cdots & (a_{m1}+b_{m1}) \\ \vdots & \ddots & \vdots \\ (a_{1n}+b_{1n}) & \cdots & (a_{mn}+b_{mn}) \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix} + \begin{bmatrix} b_{11} & \cdots & b_{m1} \\ \vdots & \ddots & \vdots \\ b_{1n} & \cdots & b_{mn} \end{bmatrix}$$

$$= \left(\begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}^T \right)^T + \left(\begin{bmatrix} b_{11} & \cdots & b_{m1} \\ \vdots & \ddots & \vdots \\ b_{1n} & \cdots & b_{mn} \end{bmatrix}^T \right)^T = \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \right)^T + \left(\begin{bmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{bmatrix} \right)^T$$

$$= A^T + B^T$$

Exercise 01

❖ Revise

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

$$A = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix}, B = \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix}$$

$$A \in \mathbb{R}^{m \times n}, B \in \mathbb{R}^{n \times k}$$

$$(AB)^T = \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} \right)^T = \left(\begin{bmatrix} (a_{11}b_{11} + \cdots + a_{1n}b_{n1}) & \cdots & (a_{11}b_{1k} + a_{1n}b_{nk}) \\ \vdots & \ddots & \vdots \\ (a_{m1}b_{11} + \cdots + a_{mn}b_{n1}) & \cdots & (a_{m1}b_{1k} + \cdots + a_{mm}b_{nk}) \end{bmatrix} \right)^T$$

$$= \begin{bmatrix} (a_{11}b_{11} + \cdots + a_{1n}b_{n1}) & \cdots & (a_{m1}b_{11} + \cdots + a_{mn}b_{n1}) \\ \vdots & \ddots & \vdots \\ (a_{11}b_{1k} + a_{1n}b_{nk}) & \cdots & (a_{m1}b_{1k} + \cdots + a_{mm}b_{nk}) \end{bmatrix} = \begin{bmatrix} b_{11} & \cdots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1k} & \cdots & b_{nk} \end{bmatrix} \begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}$$

$$= \left(\begin{bmatrix} b_{11} & \cdots & b_{n1} \\ \vdots & \ddots & \vdots \\ b_{1k} & \cdots & b_{nk} \end{bmatrix}^T \right)^T \left(\begin{bmatrix} a_{11} & \cdots & a_{m1} \\ \vdots & \ddots & \vdots \\ a_{1n} & \cdots & a_{mn} \end{bmatrix}^T \right)^T = \left(\begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} \right)^T \left(\begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \right)^T$$

$$= B^T A^T$$

Exercise 02

Exercise 02

❖ Introduction

Problem statement: Using python, re-implement a function find normalized eigenvector and eigenvalue.

2.1 Eigenvector and eigenvalue

- $\mathbf{A} \in R^{n*n}, \mathbf{I}(\text{identity matrix}) \in R^{n*n}, \mathbf{v} \in R^n$
- Eigenvalue (λ): $det(\mathbf{A} - \lambda\mathbf{I}) = 0$
- Eigenvector $(\mathbf{v})$: $\mathbf{A}\mathbf{v} = \lambda\mathbf{v} \Longleftrightarrow (\mathbf{A} - \lambda\mathbf{I})\mathbf{v} = 0$
- Normalize vector: $\frac{\mathbf{v}}{\| \mathbf{v} \|}, v_i = \frac{v_i}{\sqrt{\Sigma_1^n v_i^2}}$

Exercise 02

❖ Eigenvalue and eigenvector

$$Av = \lambda v$$

Square matrix A

Eigenvector

Eigenvalue

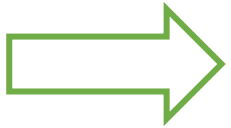
https://www.mathsisfun.com/algebra/eigenvalue.html

Exercise 02

❖ Eigenvalue and eigenvector

$$Av = \lambda v$$

- $A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$
- $v = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$
- $\lambda = 6$



- $Av = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} (-6\times1) + (3\times4) \\ (4\times1) + (5\times4) \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$
- $\lambda v = 6 \begin{bmatrix} 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 6 \\ 24 \end{bmatrix}$

$=> Av = \lambda v$

https://www.mathsisfun.com/algebra/eigenvalue.html

Exercise 02

❖ Find eigenvalue

$$Av = \lambda v \Rightarrow Av = \lambda Iv \Rightarrow Av - \lambda Iv = 0$$

$$|A - \lambda I| = 0$$ (if v is non-zero)

https://www.mathsisfun.com/algebra/eigenvalue.html

Exercise 02

❖ Find eigenvalue

$$Av = \lambda v$$

- $A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$
- $I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$|A - \lambda I| = \left|\begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} - \lambda \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}\right| = \begin{vmatrix} -6-\lambda & 3 \\ 4 & 5-\lambda \end{vmatrix}$$

$$\Rightarrow (-6-\lambda)(5-\lambda) - 3\times 4 = 0 \Leftrightarrow \lambda^2 + \lambda - 42 = 0$$

$$\Rightarrow \lambda = \begin{cases} -7 \\ 6 \end{cases}$$

https://www.mathsisfun.com/algebra/eigenvalue.html

Exercise 02

❖ Find eigenvector

$$Av = \lambda v$$

- $A = \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix}$
- $\lambda = \begin{cases} -7 \\ 6 \end{cases}$

$$\Rightarrow Av = \lambda v$$

$$\Rightarrow \begin{bmatrix} -6 & 3 \\ 4 & 5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = 6 \begin{bmatrix} x \\ y \end{bmatrix}$$

$$\Rightarrow \begin{cases} -6x + 3y = 6x \\ 4x + 5y = 6y \end{cases} \Rightarrow \begin{cases} -12x + 3y = 0 \\ 4x - y = 0 \end{cases} \Rightarrow \begin{cases} x = 1 \\ y = 4 \end{cases}$$

$$\Rightarrow v = \begin{bmatrix} 1 \\ 4 \end{bmatrix}$$

https://www.mathsisfun.com/algebra/eigenvalue.html

AI VIETNAM
All-in-One Course
(TA Session)

Exercise 02

❖ In Numpy

```python
def compute_eigenvalues_eigenvectors(matrix):
    eigenvalues, eigenvectors = np.linalg.eig(matrix)

    return eigenvalues,eigenvectors
```

```python
matrix = np.array([[0.9, 0.2], [0.1, 0.8]])
eigenvalues, eigenvectors = compute_eigenvalues_eigenvectors(matrix)
print(eigenvectors)
```

```
[[ 0.89442719 -0.70710678]
 [ 0.4472136   0.70710678]]
```

numpy.linalg.eig

linalg.eig(*a*) [source]

Compute the eigenvalues and right eigenvectors of a square array.

Parameters:

a : *(..., M, M) array*

Matrices for which the eigenvalues and right eigenvectors will be computed

Returns:

A namedtuple with the following attributes:

eigenvalues : *(..., M) array*

The eigenvalues, each repeated according to its multiplicity. The eigenvalues are not necessarily ordered. The resulting array will be of complex type, unless the imaginary part is zero in which case it will be cast to a real type. When *a* is real the resulting eigenvalues will be real (0 imaginary part) or occur in conjugate pairs

eigenvectors : *(..., M, M) array*

The normalized (unit "length") eigenvectors, such that the column `eigenvectors[:,i]` is the eigenvector corresponding to the eigenvalue `eigenvalues[i]`.

Raises:

LinAlgError

If the eigenvalue computation does not converge.

Exercise 03

Exercise 03

❖ Introduction

Problem statement: Using python, re-implement a function to calculate cosine similarity between two vectors.

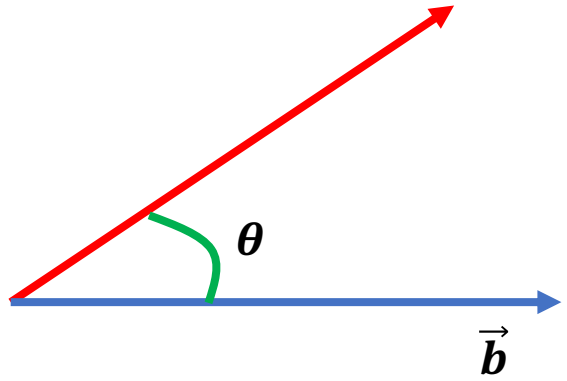
3.1. Cosine Similarity

- Data (vector $\mathbf{x}$, $\mathbf{y}$): $\mathbf{x} = \{x_1, ..., x_N\}$ $\mathbf{y} = \{y_1, ..., y_N\}$
- Cosine Similarity: $cs(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\| \mathbf{x} \| \| \mathbf{y} \|} = \frac{\Sigma_1^n x_i y_i}{\sqrt{\Sigma_1^n x_i^2} \sqrt{\Sigma_1^n y_i^2}}$

Exercise 03

❖ Cosine Similarity

$$a \cdot b = |a||b|\cos(\theta) = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$



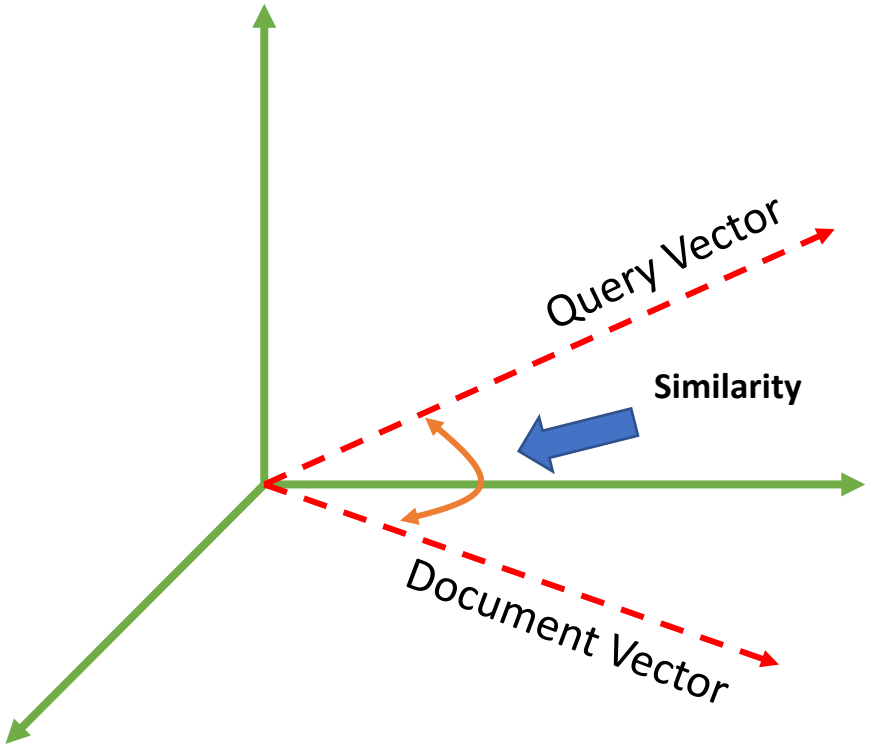
⇒ Dot product favours long vectors (higher value in dimensions)

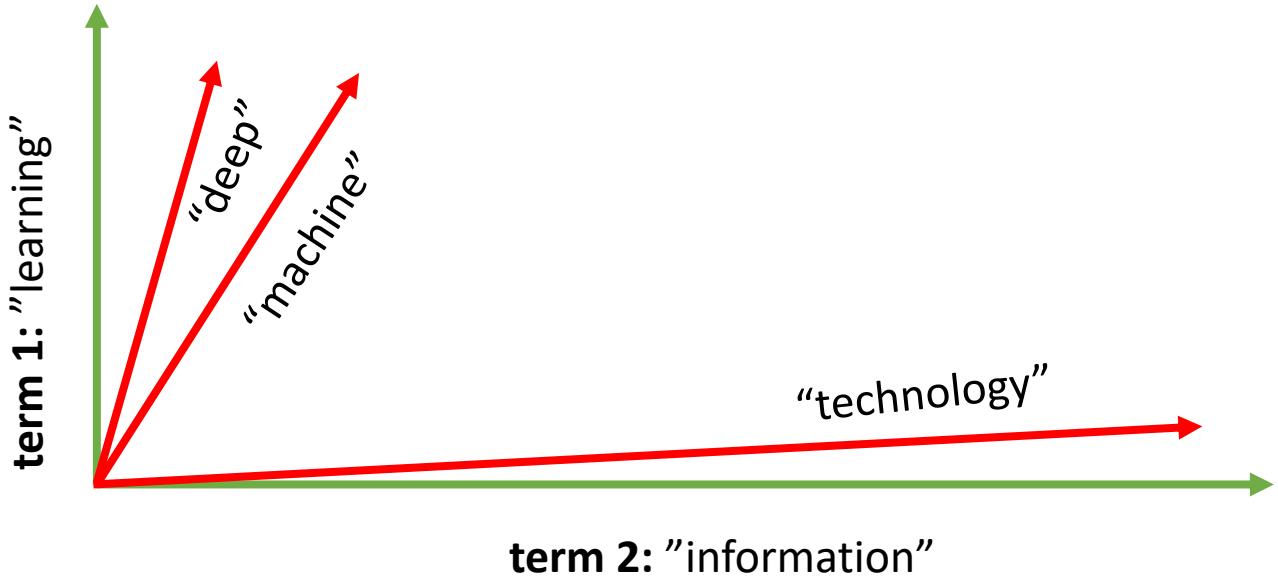
$$|v| = \sqrt{\sum_{i=1}^{n} v_i^2}$$

$$cosine\_similarity(\vec{a}, \vec{b}) = \cos(\theta) = \frac{\vec{a} \cdot \vec{b}}{|\vec{a}||\vec{b}|} = \frac{\sum_{i=1}^{n} a_i b_i}{\sqrt{\sum_{i=1}^{n} a_i^2}\sqrt{\sum_{i=1}^{n} b_i^2}}$$

Exercise 03

❖ Cosine Similarity

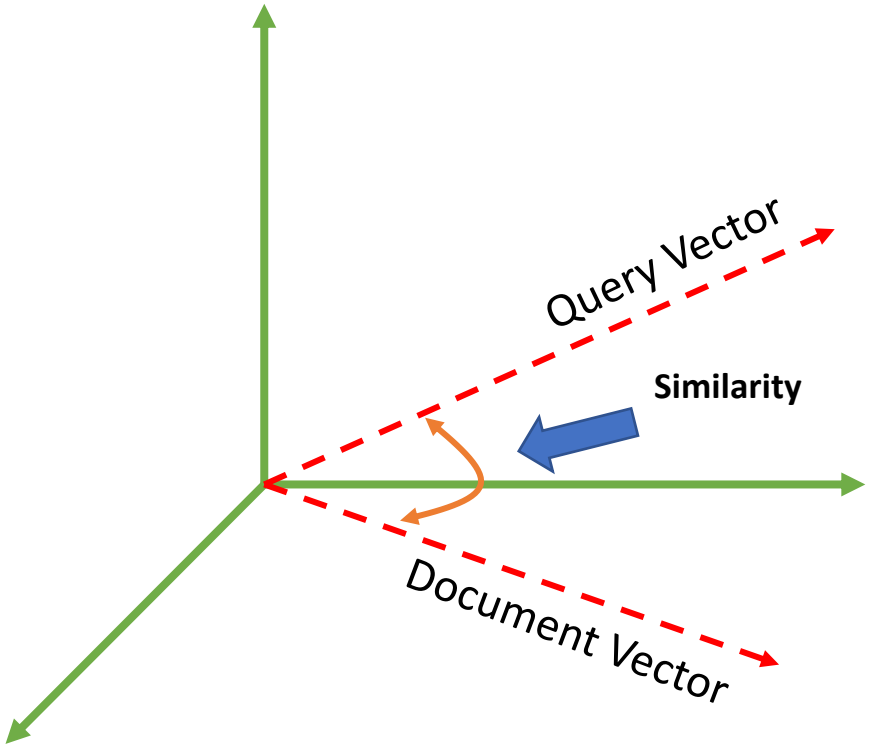





```python
from scipy import spatial

def similarity(a, b):
    return 1 - spatial.distance.cosine(a, b)
```

Exercise 03

❖ Cosine Similarity




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

def compute_cosine(v1, v2):
  cos_sim = compute_dot_product(v1,v2) / (compute_vector_length(v1)*compute_vector_length(v2))

  return cos_sim
```

```python
x = np.array([1, 2, 3, 4])
y = np.array([1, 0, 3, 0])
result = compute_cosine(x,y)
print(round(result, 3))
```

```
0.577
```

QUIZ

Exercise 04

Exercise 04

❖ Introduction

Problem statement: Given a target background image and an object image (in greenscreen), write a python program to do background subtraction.

Original Background Image

Target Background Image

Object Image

Foreground Mask

Segmented Object

Final Output

Exercise 04

❖ Introduction: Background Replacement

Input

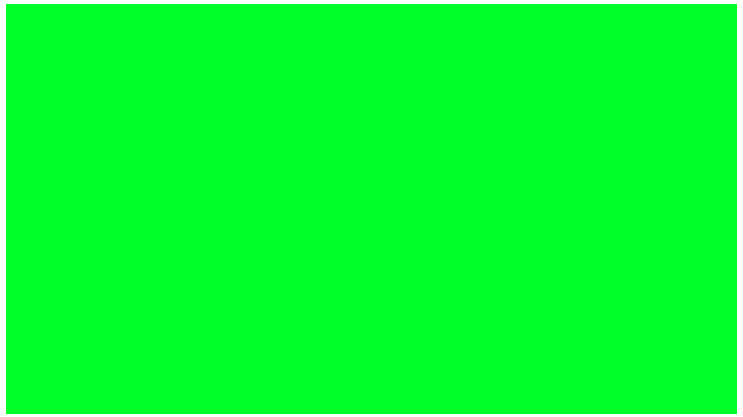
Background Replacement

Output

Exercise 04

❖ Introduction: Input for Background Subtraction

Original Background Image



Object Image

Target Background Image

Exercise 04

❖ Coding: Read image

```
1 import numpy as np
2 from google.colab.patches import cv2_imshow
3 import cv2
4
5 bg1_image = cv2.imread('GreenBackground.png', 1)
6 ob_image = cv2.imread('Object.png', 1)
7 bg2_image = cv2.imread('NewBackground.jpg', 1)
```

Syntax: *cv2.imread(filename, flag)*

Parameters:

1. *filename: The path to the image file.*
2. *flag: The flag specifies the way how the image should be read.*

- ***cv2.IMREAD_COLOR*** *– It specifies to load a color image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can pass integer value **1** for this flag.*
- ***cv2.IMREAD_GRAYSCALE*** *– It specifies to load an image in grayscale mode. Alternatively, we can pass integer value **0** for this flag.*
- ***cv2.IMREAD_UNCHANGED*** *– It specifies to load an image as such including alpha channel. Alternatively, we can pass integer value **-1** for this flag.*

Return Value:

The cv2.imread() function return a NumPy array if the image is loaded successfully.

Exercise 04

❖ Coding: Resize image

```
IMAGE_SIZE = (678, 381)

bg1_image = cv2.resize(bg1_image, IMAGE_SIZE)
ob_image = cv2.resize(ob_image, IMAGE_SIZE)
bg2_image = cv2.resize(bg2_image, IMAGE_SIZE)
```

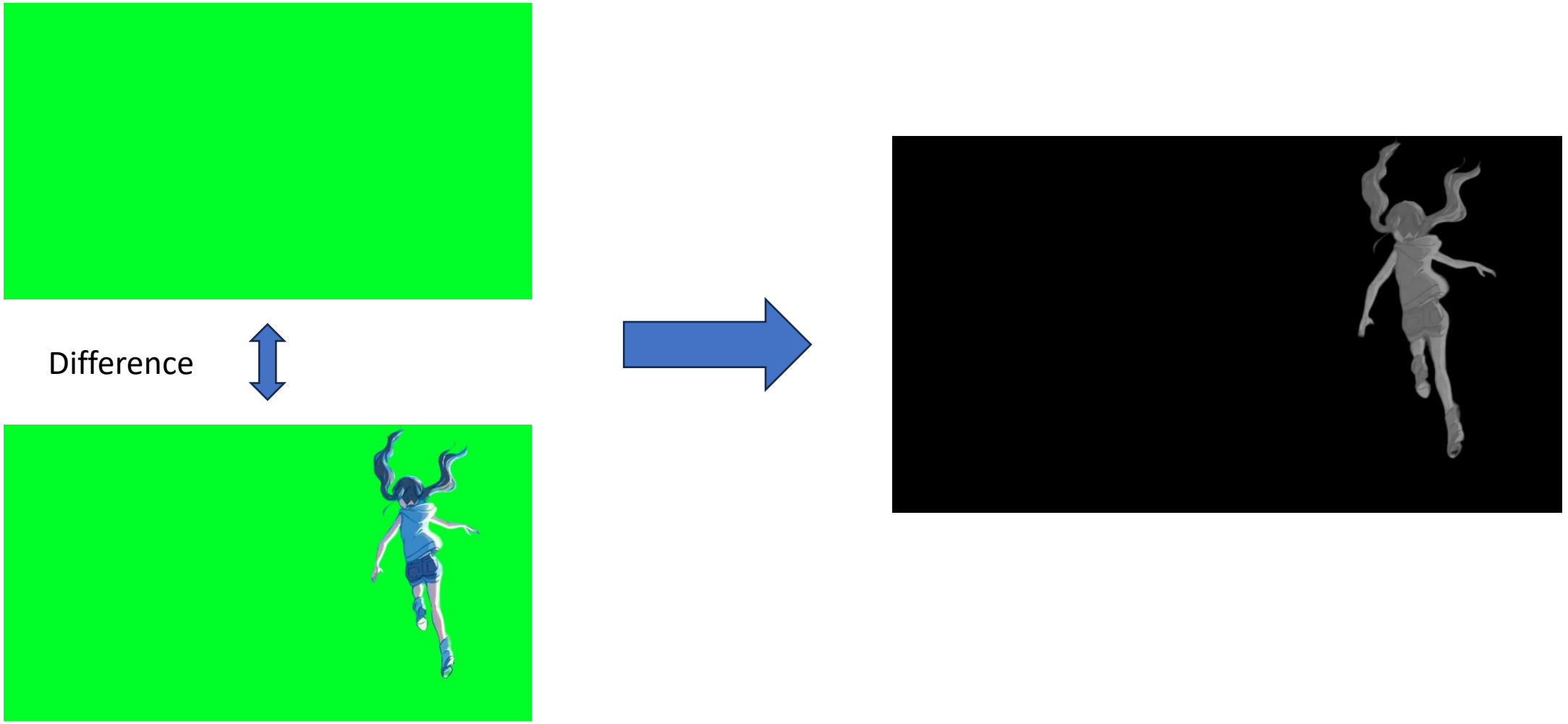
Syntax: cv2.resize(source, dsize, dest, fx, fy, interpolation)

Parameters:

- **source:** Input Image array (Single-channel, 8-bit or floating-point)
- **dsize:** Size of the output array
- **dest:** Output array (Similar to the dimensions and type of Input image array) [optional]
- **fx:** Scale factor along the horizontal axis [optional]
- **fy:** Scale factor along the vertical axis [optional]
- **interpolation:** One of the above interpolation methods [optional]

Exercise 04

❖ Coding



Exercise 04

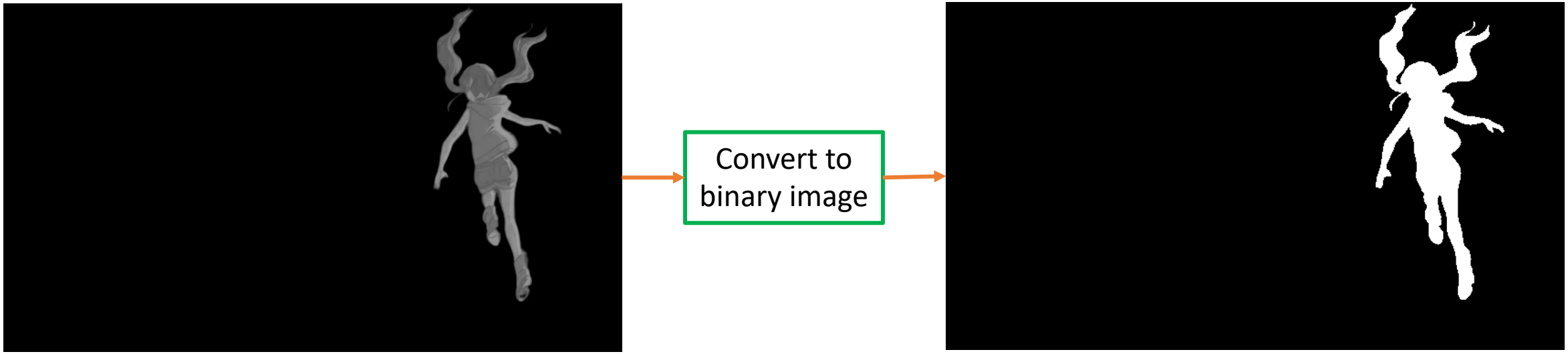
❖ Coding

```python
def compute_difference(bg_img, input_img):
    difference_three_channel = cv2.absdiff(bg_img, input_img)
    difference_single_channel = np.sum(difference_three_channel, axis=2) / 3.0
    difference_single_channel = difference_single_channel.astype('uint8')

    return difference_single_channel
```

```python
difference_single_channel = compute_difference(bg1_image, ob_image)
cv2_imshow(difference_single_channel)
```

Exercise 04

❖ Coding

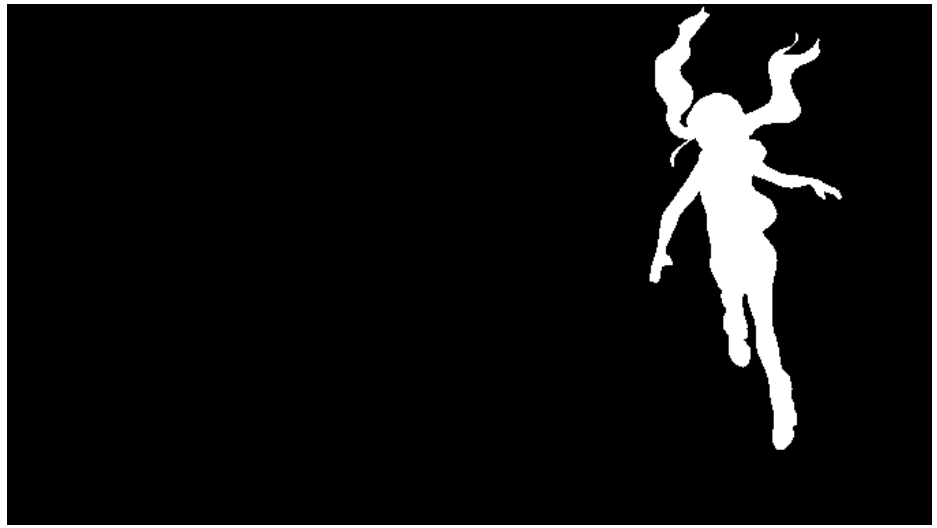


Exercise 04

❖ Coding

```python
def compute_binary_mask(difference_single_channel):
    difference_binary = np.where(difference_single_channel >= 15, 255, 0)
    difference_binary = np.stack((difference_binary,)*3, axis=-1)
    return difference_binary
```

```python
binary_mask = compute_binary_mask(difference_single_channel)
cv2_imshow(binary_mask)
```



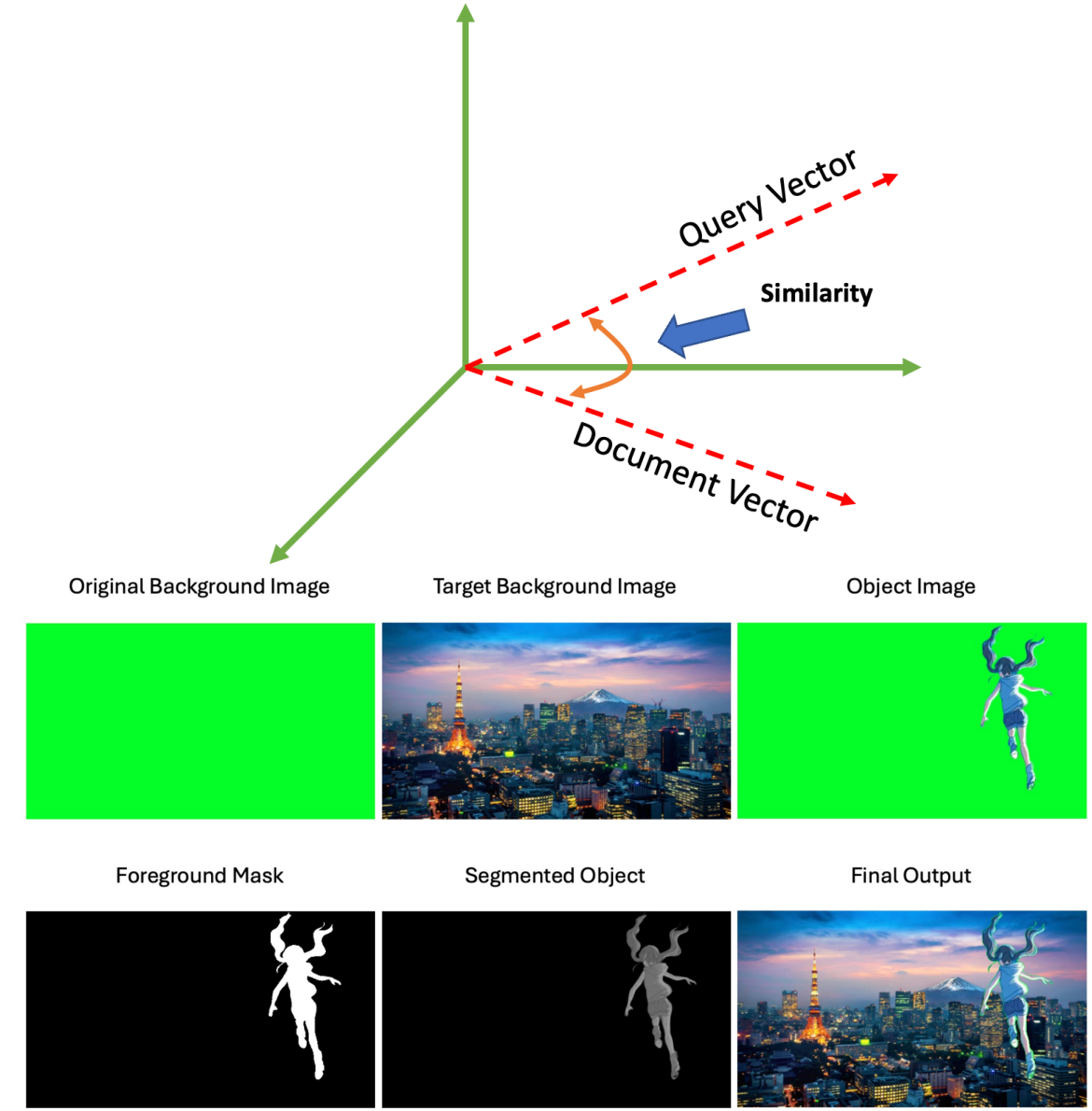
Exercise 04

❖ Coding

```python
def replace_background(bg1_image, bg2_image, ob_image):
    difference_single_channel = compute_difference(bg1_image,ob_image)
    binary_mask = compute_binary_mask(difference_single_channel)

    output = np.where(binary_mask==255, ob_image, bg2_image)

    return output
```

```python
output = replace_background(bg1_image, bg2_image, ob_image)

cv2_imshow(output)
```

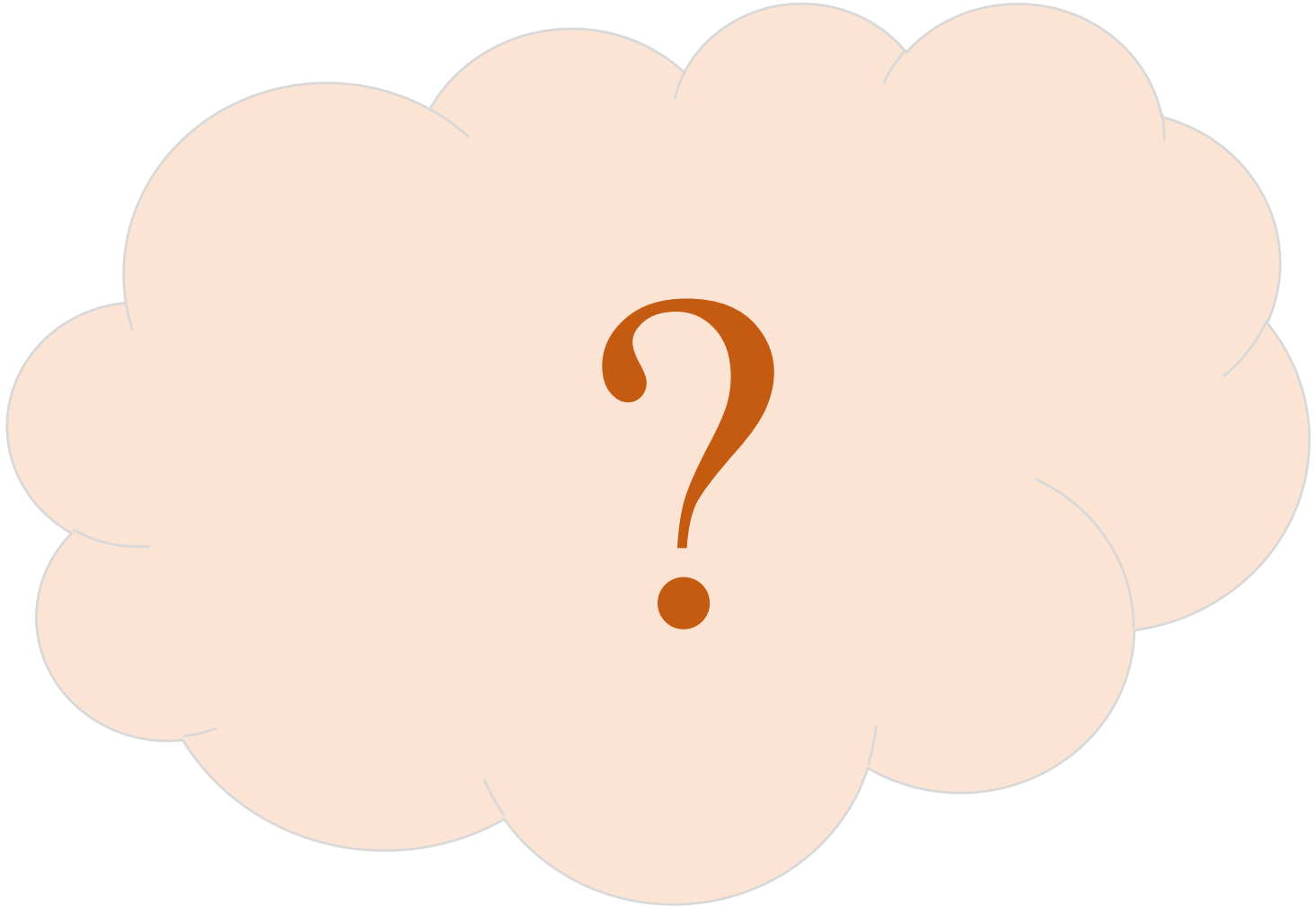
Summarization



In this session, we have discussed:

- Vector and matrix.
- Matrix property.
- Practice coding with some matrix operations: matrix multiplications, inverse matrix…
- Finding eigenvalue and eigenvector.
- Cosine similarity.
- Background Subtraction.

Question




Thank you!