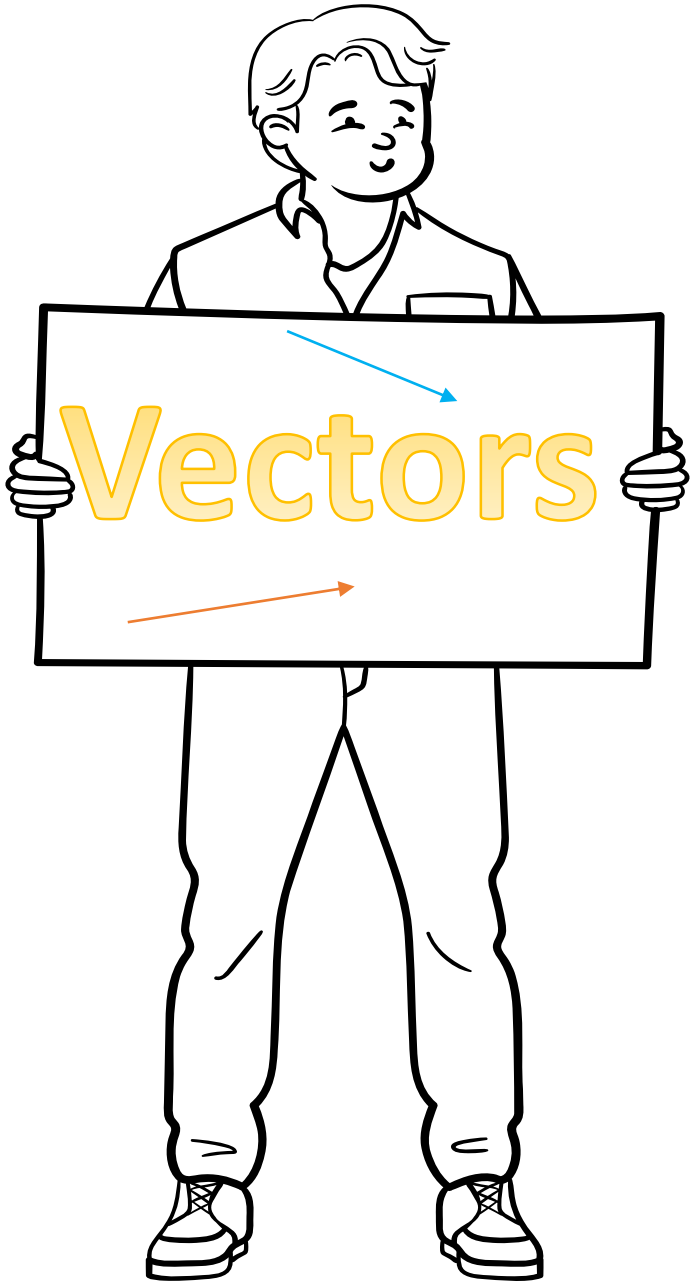
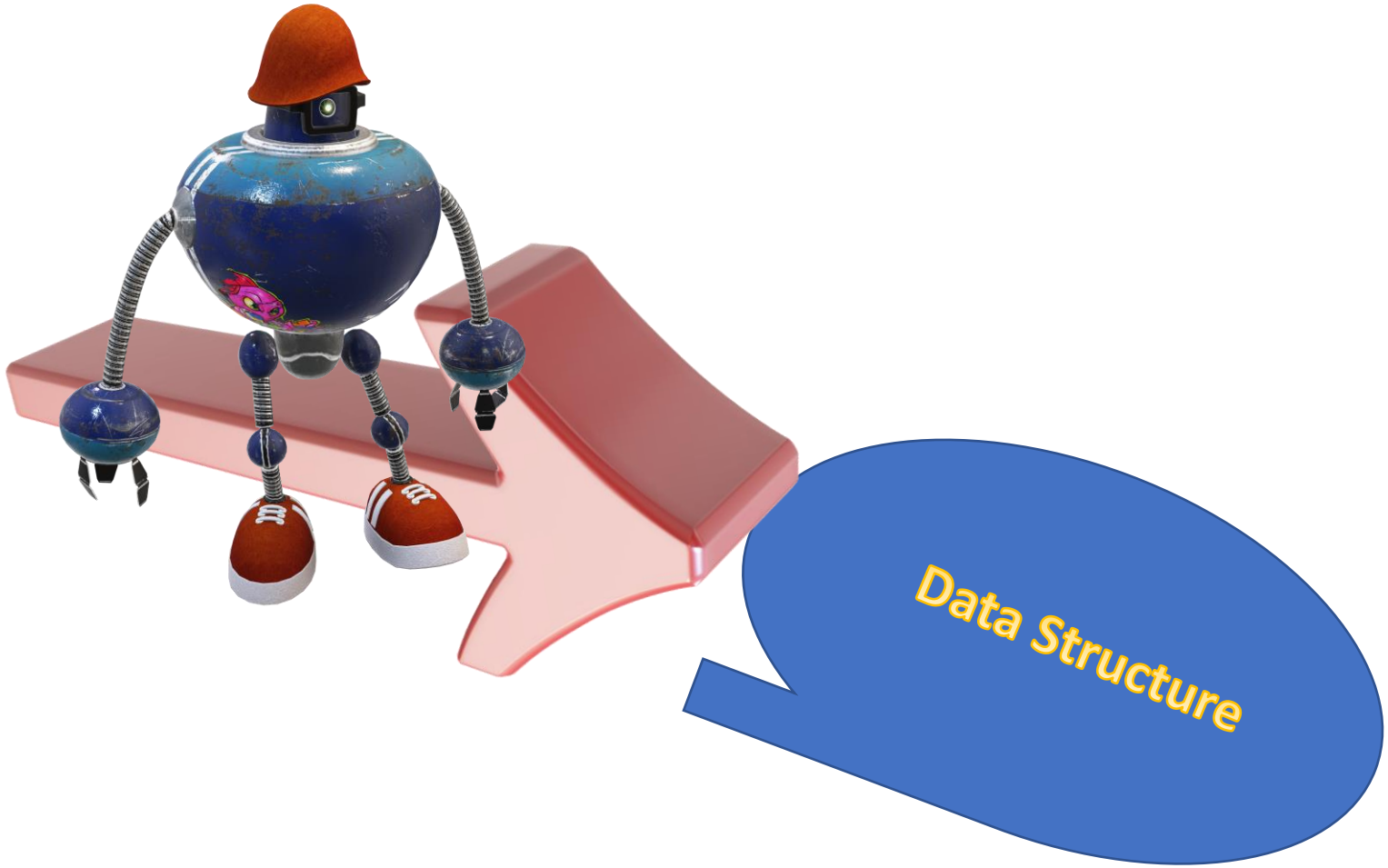
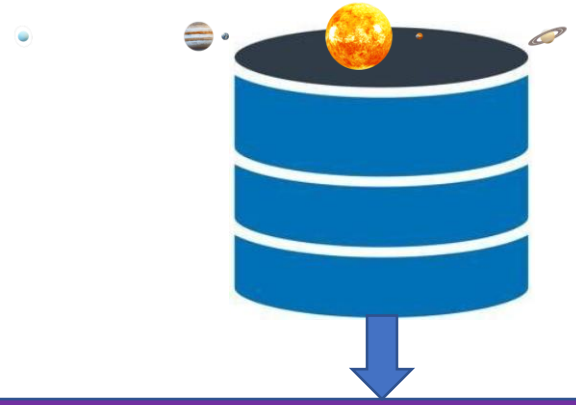


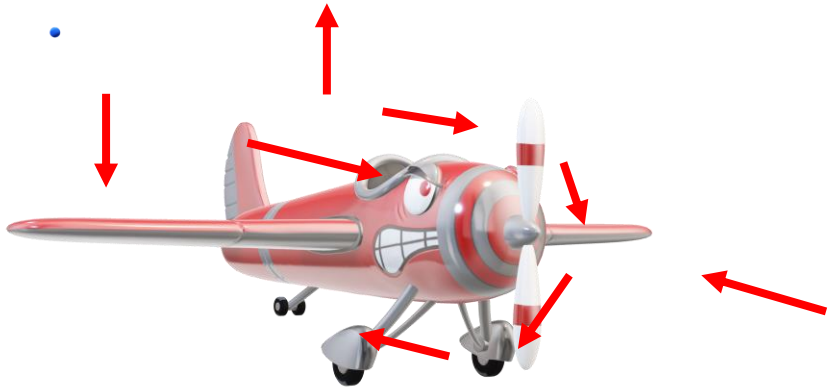


Dr Milan Joshi

What is vector

A vector is a numerical representation of data in a structured format, where each element in the vector represents a specific attribute or feature of the data.





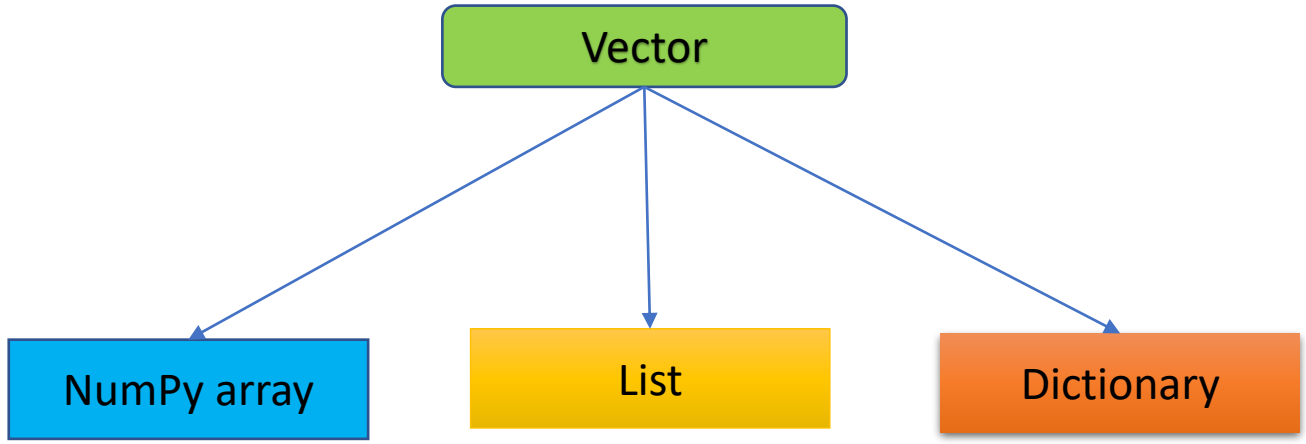
Vector is a fundamental data structure

How to represent and manipulate data in a structured, numerical format.

What is vector

A vector is a numerical representation of an **instance** in a data set in the form of a NumPy array (default is column format) in a structured format, where each element in the vector represents a specific attribute or feature of the data.

Patient Number	Age	BP
patient1	58	122
patient2	71	110
patient3	48	110
patient4	34	123
patient5	62	152
patient6	27	135
patient7	40	139
patient8	58	123



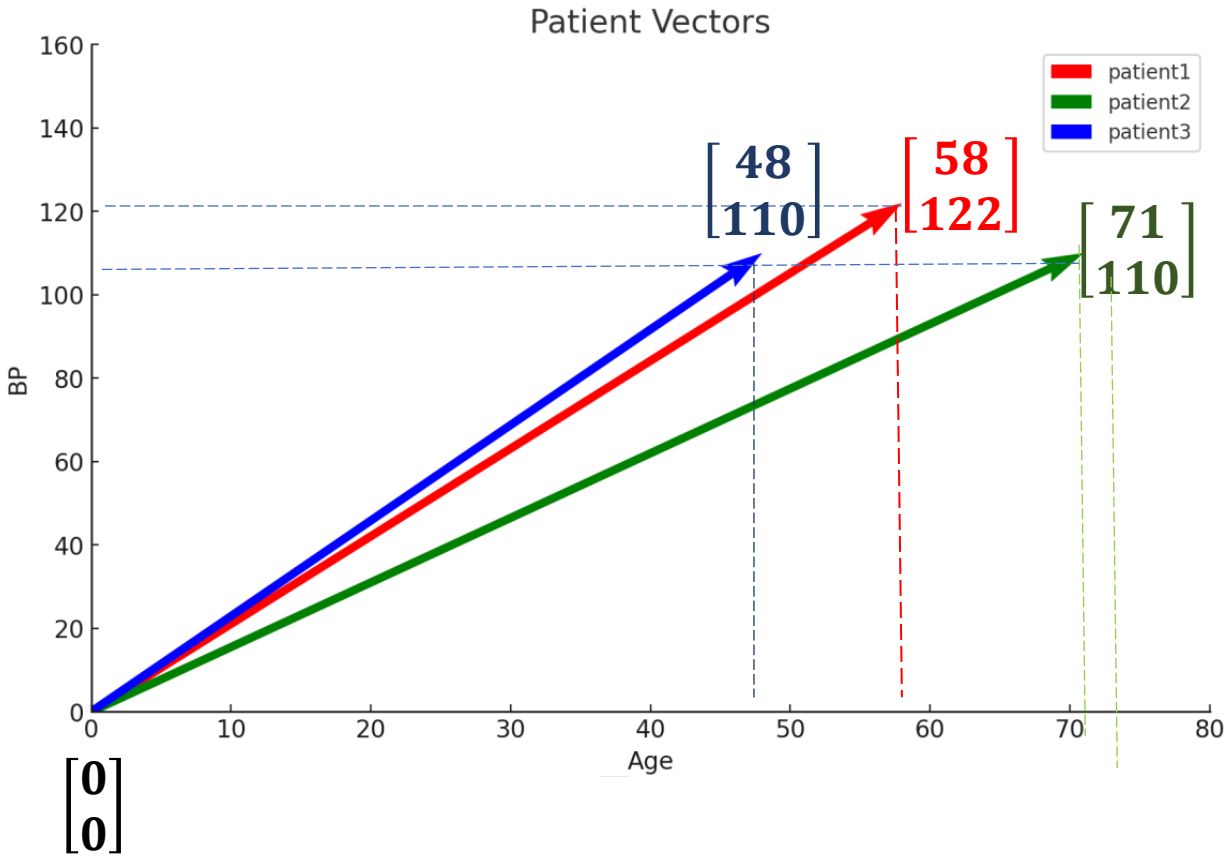
Patient1 is a vector $\rightarrow \begin{bmatrix} Age \\ BP \end{bmatrix} \rightarrow \begin{bmatrix} 58 \\ 122 \end{bmatrix}$

Patient1 is a vector $\rightarrow \begin{bmatrix} Age \\ BP \end{bmatrix} \rightarrow \begin{bmatrix} 71 \\ 110 \end{bmatrix}$

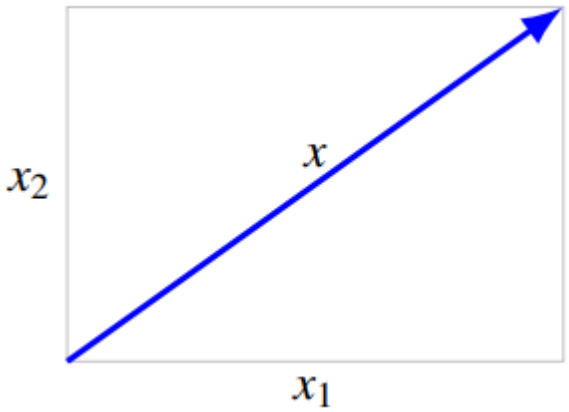
```
import numpy as np

# Define the vectors u and v
u = np.array([58, 122])
v = np.array([71,110])
print(u)
print(v)
```

```python
import numpy as np

# Define patient data as a NumPy array
patient1_array = np.array([58, 122])

# Define patient data as a list
patient1_list = [58, 122]

# Define patient data as a dictionary
patient1_dict = {
    'Age': 58,
    'BP': 122
}

# Print the representations
print("NumPy Array:", patient1_array)
print("List:", patient1_list)
print("Dictionary:", patient1_dict)
```

```
NumPy Array: [ 58 122]
List: [58, 122]
Dictionary: {'Age': 58, 'BP': 122}
```



Components and Dimension of vector



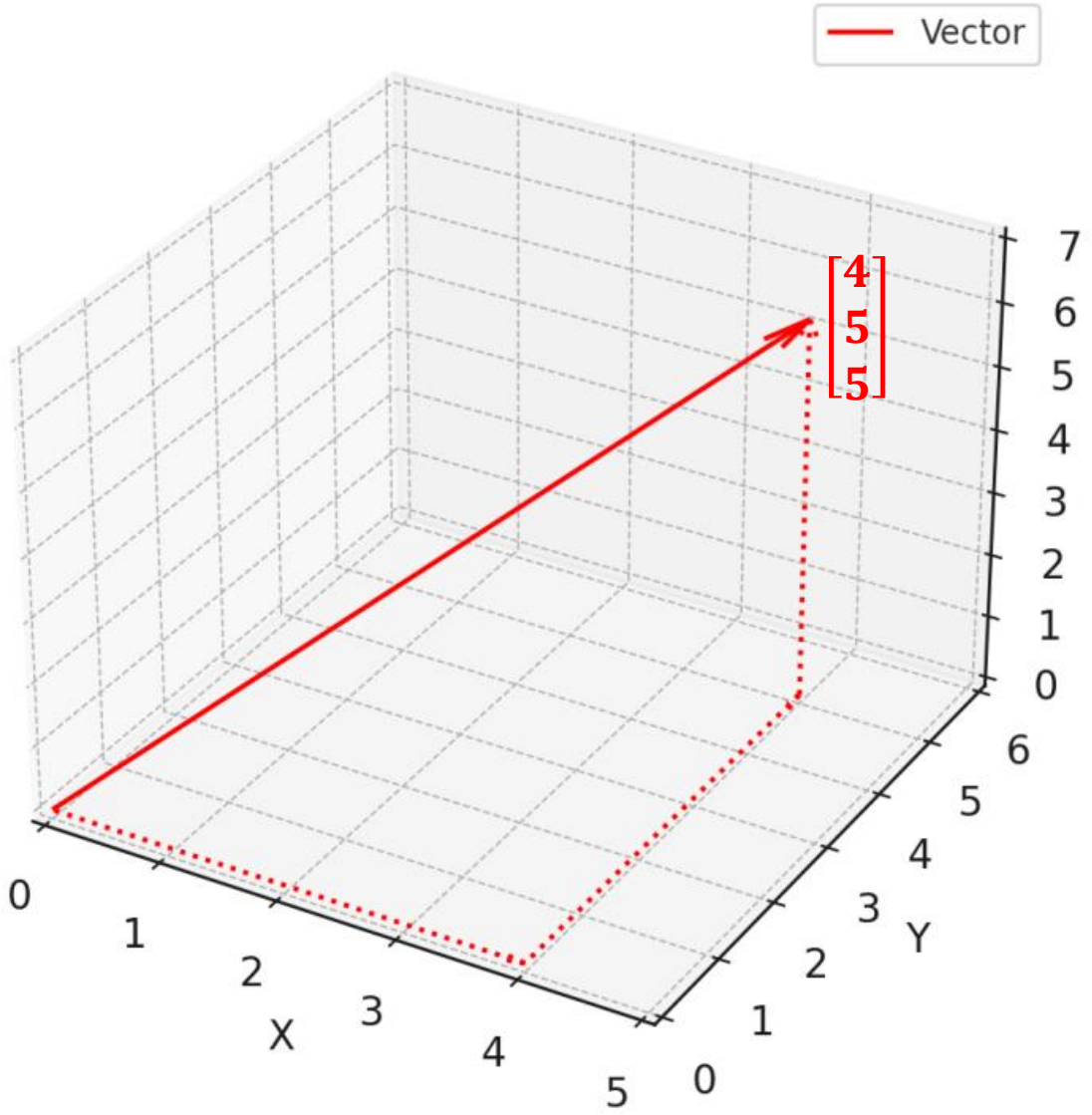
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

x_1 is component along $x - axis$
x_2 is component along y $- axis$

```python
x = [1, 2, 3]
print(f"First component: {x[0]}")
print(f"Second component: {x[1]}")
print(f"Third component: {x[2]}")
```

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2$$

2 Dimensional

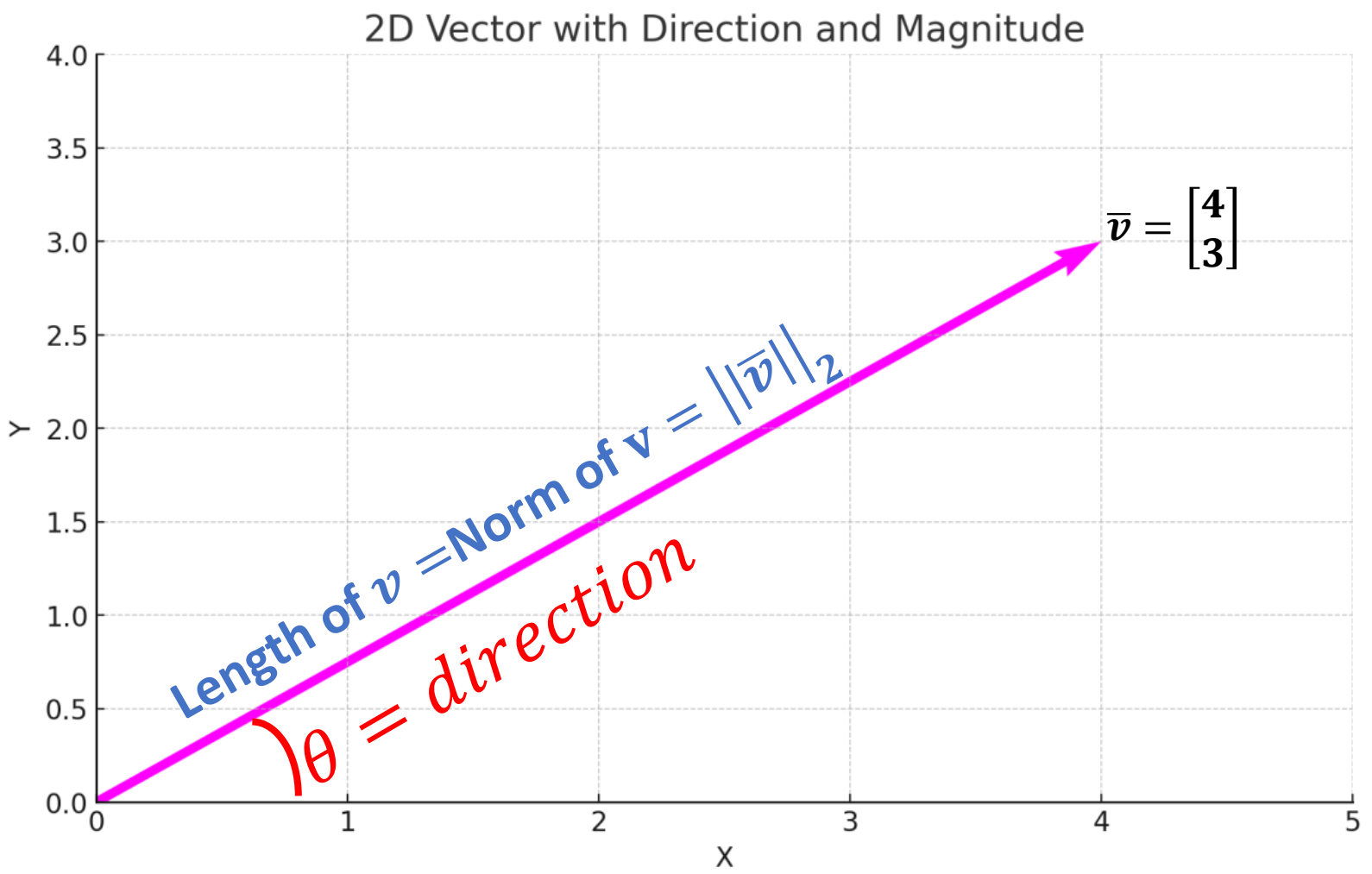


$x - component\ 4$
$y - component\ 5$
$z - component\ 5$

$$Vector = \begin{bmatrix} 4 \\ 5 \\ 5 \end{bmatrix} \in \mathbb{R}^3$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n$$

Vector Direction and Length



Examples of vectors with direction and magnitude

1. Physics: Force Vector

In physics, a force vector represents both the magnitude of the force applied to an object and the direction in which the force is applied.

Example:

- **Magnitude:** 10 N (Newtons)
- **Direction:** 30 degrees from the horizontal axis

2. Engineering: Displacement Vector

In engineering, a displacement vector indicates the change in position of an object from its initial position to its final position.

Example:

- **Magnitude:** 5 meters
- **Direction:** North-East

3. Computer Graphics: Movement Vector

In computer graphics, a movement vector can describe the direction and speed of an object moving across the screen.

Example:

- **Magnitude:** 3 units per second
- **Direction:** Towards the top-right corner of the screen

4. Navigation: Velocity Vector

In navigation, a velocity vector shows the speed and direction of a moving object, such as a boat or an airplane.

Example:

- **Magnitude:** 50 km/h
- **Direction:** 45 degrees East of North

5. Robotics: Joint Movement

In robotics, vectors can describe the movement of robotic joints in a specific direction with a given speed.

Example:

- **Magnitude:** 2 radians per second
- **Direction:** Along the robotic arm's axis

Vectors Example

1. Color: (R, G, B)

Explanation:

- In digital images, colors are often represented as vectors with three components: Red (R), Green (G), and Blue (B).

Example:

- A pure red color can be represented as the vector $(255, 0, 0)$, where 255 is the maximum intensity for red, and 0 for green and blue.
- A shade of purple might be $(128, 0, 128)$.

2. Quantities of n Different Commodities

Explanation:

- This vector represents quantities of different commodities or resources. It could be used in scenarios like inventory management or bill of materials in manufacturing.

Example:

- Suppose a bill of materials for a product includes 3 units of component A, 5 units of component B, and 2 units of component C. This can be represented as $[3, 5, 2]$.

Vectors Example

3. Portfolio

Explanation:

- A portfolio vector represents the shares (or dollar value or fraction) held in each of n assets, with negative values indicating short positions.

Example:

- If an investor holds 100 shares of stock A, -50 shares of stock B (short position), and 200 shares of stock C, the portfolio vector can be $[100, -50, 200]$.

4. Cash Flow

Explanation:

- This vector represents cash flows over different periods.

Example:

- A business might have cash flows of $1000 in the first period, $1500 in the second, and $2000 in the third. This can be represented as $[1000, 1500, 2000]$.

Vectors Example

5. Audio

Explanation:

- In audio processing, vectors can represent acoustic pressure at sample times, where samples are spaced by a constant interval (e.g., 1/44100 seconds apart for CD-quality audio).

Example:

- An audio signal sampled at 44100 Hz might have values like $[0.1, 0.3, -0.2, 0.0, ...]$, where each value represents the acoustic pressure at a given sample time.

6. Features

Explanation:

- In machine learning, feature vectors represent the values of different features or attributes of an entity.

Example:

- For a house, features could include size, number of rooms, and age. A feature vector might be $[2000, 3, 10]$ for a 2000 sq ft house with 3 rooms and 10 years old.

Vectors Example

7. Customer Purchase

Explanation:

- This vector represents the total dollar purchase of different products by a customer over some period.

Example:

- If a customer spends $100 on product A, $50 on product B, and $75 on product C, the vector can be $[100, 50, 75]$.

Word count vector

Example Sentence
Let's use the sentence: "Data science is fun and data science is powerful“

Steps to Create the Word Count Vector Manually
1.Tokenize the sentence: Break the sentence into individual words.
2.Count the occurrences of each word.
3.Create the word count vector.
Tokenize the Sentence

The sentence: "Data science is fun and data science is powerful"
Tokens (words): ["data", "science", "is", "fun", "and“, "data", "science", "is", "powerful"]

Count the Occurrences
•"data": 2
•"science": 2
•"is": 2
•"fun": 1
•"and": 1
•"powerful": 1

Sentence: "Data science is fun and data science is powerful“
Unique Words: ["data", "science", "is", "fun", "and", "powerful"]
Word Count Vector: [2, 2, 2, 1, 1, 1]

Customer Vectors

A customer vector can represent attributes like age, total spend, number of purchases, number of returns, number of reviews written, average rating given, days since last purchase, and loyalty status.

- **Customer 1**: Age 30, Total Spend $500, Number of Purchases 10, Number of Returns 2, Number of Reviews 5, Average Rating 4.2, Days Since Last Purchase 15, Loyalty Status 1 (1 for loyal, 0 for not loyal)
- **Customer 2**: Age 45, Total Spend $1500, Number of Purchases 25, Number of Returns 1, Number of Reviews 10, Average Rating 4.5, Days Since Last Purchase 5, Loyalty Status 1

Vectors:

- **Customer 1 Vector**: [30, 500, 10, 2, 5, 4.2, 15, 1]
- **Customer 2 Vector**: [45, 1500, 25, 1, 10, 4.5, 5, 1]

Product Vectors

A product vector can represent attributes like price, average rating, number of reviews, number of returns, category, stock level, days since release, and warranty period.

- **Product 1**: Price $20, Average Rating 4.5, Number of Reviews 100, Number of Returns 5, Category 1 (1 for electronics, 0 for others), Stock Level 500, Days Since Release 365, Warranty Period 12 months
- **Product 2**: Price $50, Average Rating 4.0, Number of Reviews 200, Number of Returns 10, Category 0, Stock Level 300, Days Since Release 180, Warranty Period 6 months

Vectors:

- **Product 1 Vector**: [20, 4.5, 100, 5, 1, 500, 365, 12]
- **Product 2 Vector**: [50, 4.0, 200, 10, 0, 300, 180, 6]

Review Vectors

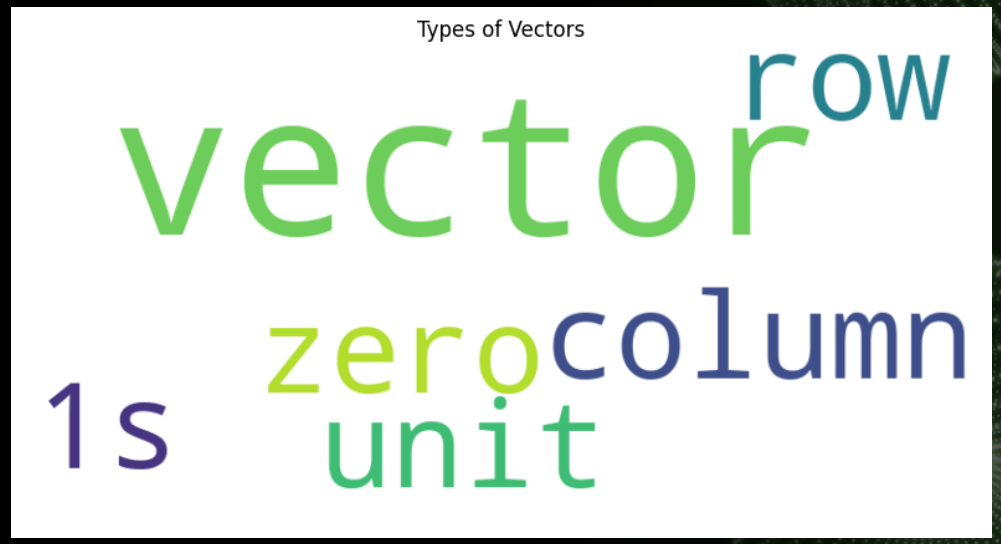
A review vector can represent attributes like helpfulness score, length of review in words, sentiment score, days since review, rating given, verified purchase status, number of upvotes, and number of comments.

- **Review 1**: Helpfulness Score 10, Length of Review 150 words, Sentiment Score 0.8, Days Since Review 30, Rating Given 5, Verified Purchase 1 (1 for verified, 0 for not), Number of Upvotes 20, Number of Comments 2
- **Review 2**: Helpfulness Score 5, Length of Review 50 words, Sentiment Score 0.3, Days Since Review 60, Rating Given 3, Verified Purchase 0, Number of Upvotes 5, Number of Comments 1

Vectors:

- **Review 1 Vector**: [10, 150, 0.8, 30, 5, 1, 20, 2]
- **Review 2 Vector**: [5, 50, 0.3, 60, 3, 0, 5, 1]

Types of vector



1. Row Vector

A row vector is a 1-dimensional array of numbers arranged in a single row. It is typically represented as a horizontal vector. In mathematical notation, it is written as:

$$\mathbf{v} = [v_1, v_2, v_3, \ldots, v_n]$$

Example:

$$\mathbf{v} = [1, 2, 3, 4]$$

2. Column Vector

A column vector is a 1-dimensional array of numbers arranged in a single column. It is typically represented as a vertical vector. In mathematical notation, it is written as:

$$\mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{bmatrix}$$

Example:

$$\mathbf{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

- **Row Vector**: A horizontal vector with elements in a single row.
- **Column Vector**: A vertical vector with elements in a single column.
- **Zero Vector**: A vector with all elements equal to zero.
- **Unit Vector**: A vector with a magnitude of 1, often used to indicate direction.
- **Vector of 1s**: A vector where all elements are 1, used in various computational tasks.

3. Zero Vector

A zero vector is a vector in which all elements are zero. It is denoted as $\mathbf{0}$ and can exist in any dimension. The zero vector is important in vector spaces because it acts as the additive identity.

Example (in 4 dimensions):

$\mathbf{0} = [0, 0, 0, 0]$

4. Unit Vector

A unit vector is a vector with a magnitude (or length) of 1. Unit vectors are often used to indicate direction. In a Cartesian coordinate system, the unit vectors in the directions of the x, y, and z axes are denoted by $\hat{i}$, $\hat{j}$, and $\hat{k}$ respectively.

Example (in 2 dimensions):

$\mathbf{u} = \left[\frac{1}{\sqrt{2}}, \frac{1}{\sqrt{2}}\right]$

Here, the magnitude is:

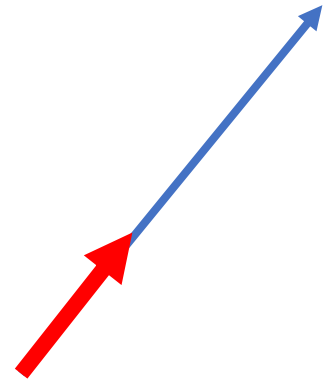
$$\sqrt{\left(\frac{1}{\sqrt{2}}\right)^2 + \left(\frac{1}{\sqrt{2}}\right)^2} = \sqrt{\frac{1}{2} + \frac{1}{2}} = \sqrt{1} = 1$$

5. Vector of 1s

A vector of 1s is a vector in which all elements are 1. This type of vector is used in various mathematical and computational applications, such as initializing weights in machine learning algorithms or creating matrices of ones.

Example (in 4 dimensions):

$\mathbf{1} = [1, 1, 1, 1]$

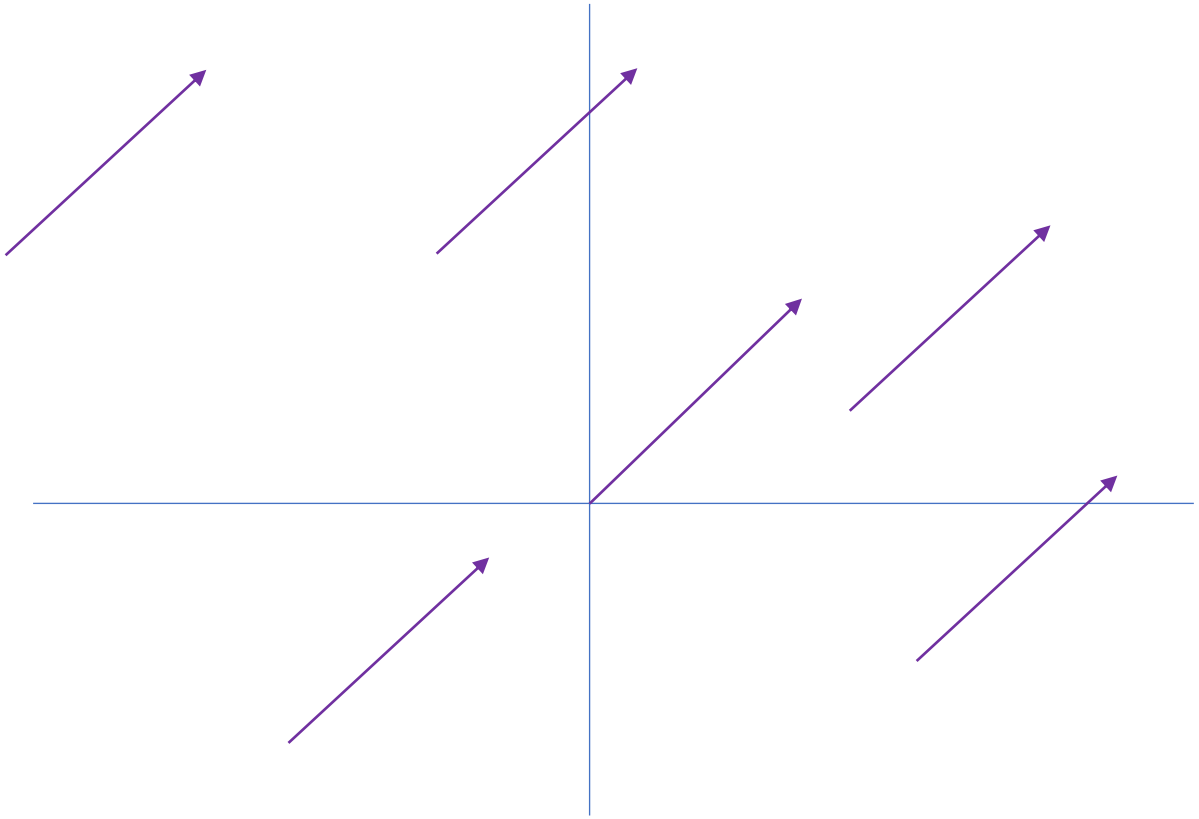


$$u = \hat{u}||u||$$

```python
import numpy as np

# Row Vector
row_vector = np.array([1, 2, 3, 4])

# Column Vector
column_vector = np.array([[1], [2], [3], [4]])

# Zero Vector (4-dimensional)
zero_vector = np.zeros(4)

# Unit Vector (2-dimensional, example: unit vector in the direction of [1, 1])
unit_vector = np.array([1/np.sqrt(2), 1/np.sqrt(2)])

# Vector of 1s (4-dimensional)
vector_of_1s = np.ones(4)

# Print the vectors
print("Row Vector:", row_vector)
print("Column Vector:\n", column_vector)
print("Zero Vector:", zero_vector)
print("Unit Vector:", unit_vector)
print("Vector of 1s:", vector_of_1s)
```

```
Row Vector: [1 2 3 4]
Column Vector:
 [[1]
 [2]
 [3]
 [4]]
Zero Vector: [0. 0. 0. 0.]
Unit Vector: [0.70710678 0.70710678]
Vector of 1s: [1. 1. 1. 1.]
```

Equal Vectors



Two vectors are considered equal if they have the same magnitude (length) and direction. In other words, they must have the same components in the same order.

Operations on vectors

1. Vector Addition

   Vector addition involves adding corresponding components of two vectors to create a new vector. This o
   vectors of the same dimension.

Example:

Let $\mathbf{a} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 4 \\ 1 \end{bmatrix}$.

$$\mathbf{a} + \mathbf{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix} + \begin{bmatrix} 4 \\ 1 \end{bmatrix} = \begin{bmatrix} 2+4 \\ 3+1 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

2. Vector Subtraction

   Vector subtraction involves subtracting corresponding components of one vector from another vector to
   operation is only defined for vectors of the same dimension.

Example:

Let $\mathbf{a} = \begin{bmatrix} 5 \\ 7 \end{bmatrix}$ and $\mathbf{b} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$.

$$\mathbf{a} - \mathbf{b} = \begin{bmatrix} 5 \\ 7 \end{bmatrix} - \begin{bmatrix} 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 5-2 \\ 7-3 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

3. Scalar Multiplication

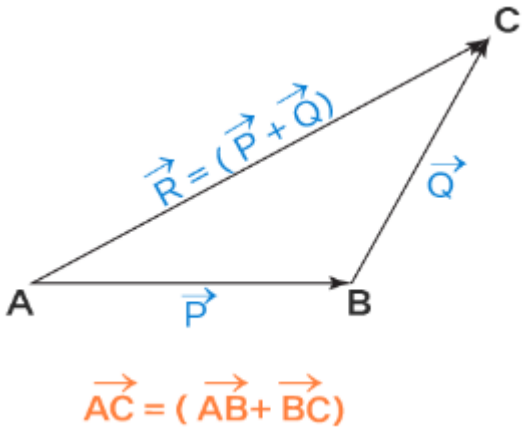
   Scalar multiplication involves multiplying each component of a vector by a scalar (a single number) to create a new vector.

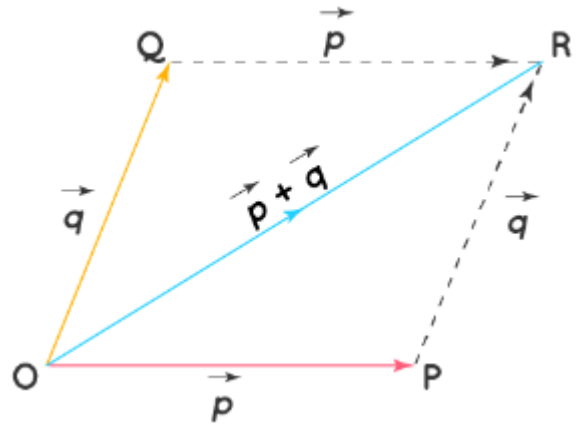
Example:

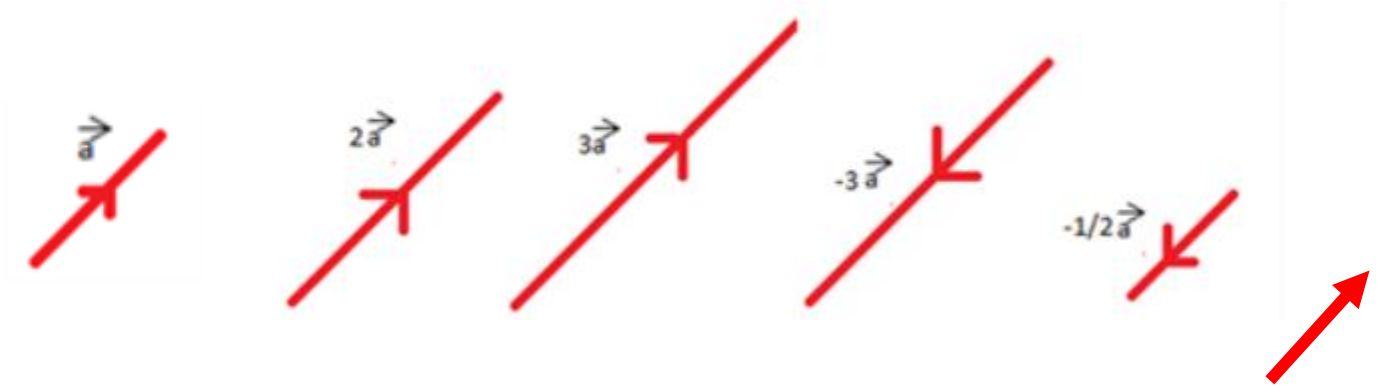
Let $\mathbf{a} = \begin{bmatrix} 1 \\ -3 \end{bmatrix}$ and scalar $k = 4$.

$$k\mathbf{a} = 4 \times \begin{bmatrix} 1 \\ -3 \end{bmatrix} = \begin{bmatrix} 4 \times 1 \\ 4 \times -3 \end{bmatrix} = \begin{bmatrix} 4 \\ -12 \end{bmatrix}$$

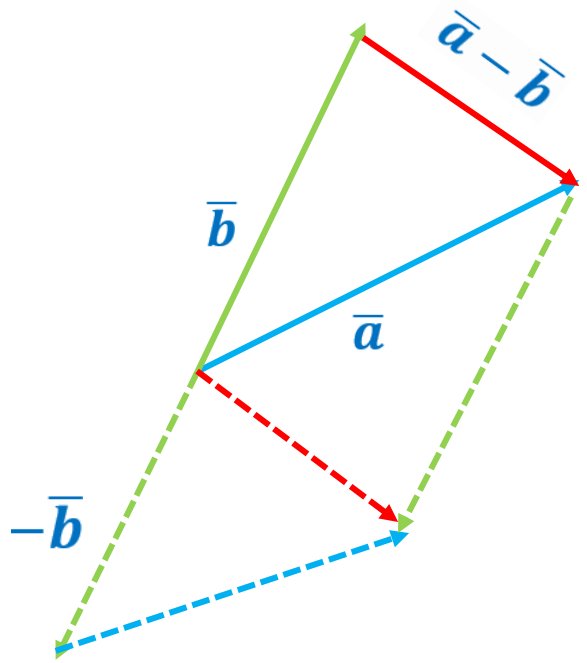
Geometry


C
A
B
$\vec{R} = (\vec{P} + \vec{Q})$
$\vec{Q}$
$\vec{P}$
$\vec{AC} = (\vec{AB} + \vec{BC})$

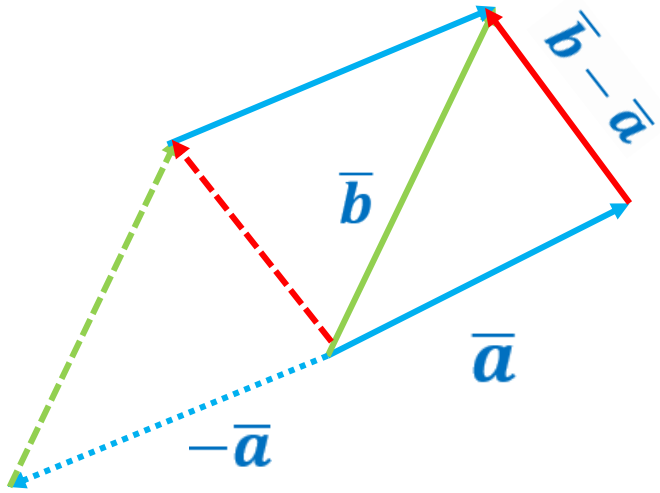

Q
$\vec{p}$
R
$\vec{q}$
$\vec{p} + \vec{q}$
$\vec{q}$
O
P
$\vec{p}$


$\vec{a}$
$2\vec{a}$
$3\vec{a}$
$-3\vec{a}$
$-1/2\vec{a}$

Subtraction



$$\overline{a}-\overline{b}=\overline{a}+(-\overline{b})$$



$$\overline{b}-\overline{a}=\overline{b}+(-\overline{a})$$

Properties

1. **Commutative Property of Addition**:

   $$\mathbf{a} + \mathbf{b} = \mathbf{b} + \mathbf{a}$$

   This means that the order in which vectors are added does not affect the result.

2. **Associative Property of Addition**:

   $$(\mathbf{a} + \mathbf{b}) + \mathbf{c} = \mathbf{a} + (\mathbf{b} + \mathbf{c})$$

   This means that the way in which vectors are grouped when adding does not affect the result.

3. **Additive Identity**:

   $$\mathbf{a} + \mathbf{0} = \mathbf{a}$$

   The zero vector $\mathbf{0}$ acts as an additive identity; adding it to any vector $\mathbf{a}$ results in the same vector $\mathbf{a}$.

4. **Additive Inverse**:

   $$\mathbf{a} + (-\mathbf{a}) = \mathbf{0}$$

   Each vector $\mathbf{a}$ has an additive inverse $-\mathbf{a}$ such that their sum is the zero vector.

5. **Distributive Property of Scalar Multiplication over Vector Addition**:

   $$k(\mathbf{a} + \mathbf{b}) = k\mathbf{a} + k\mathbf{b}$$

   Scalar multiplication distributes over vector addition.

6. **Distributive Property of Scalar Multiplication over Scalar Addition**:

   $$(k + m)\mathbf{a} = k\mathbf{a} + m\mathbf{a}$$

   Scalar addition distributes over scalar multiplication.

7. **Associative Property of Scalar Multiplication**:

   $$k(m\mathbf{a}) = (km)\mathbf{a}$$

   The way in which scalars are grouped when multiplying a vector does not affect the result.

8. **Multiplicative Identity**:

   $$1\mathbf{a} = \mathbf{a}$$

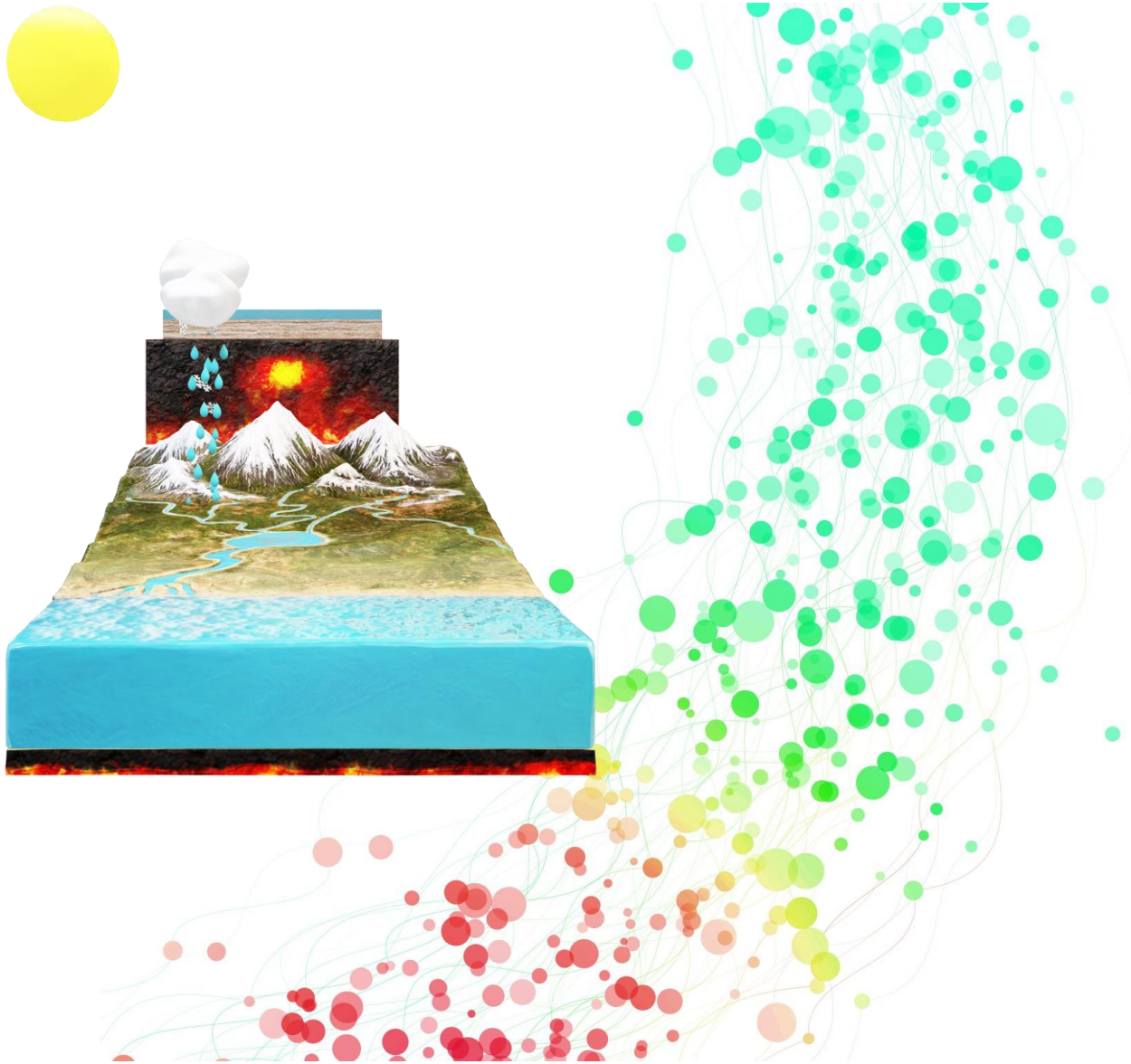
   Multiplying a vector by 1 leaves the vector unchanged.

```python
import numpy as np

# Define vectors a, b, and c
a = np.array([1, 2, 3])
b = np.array([4, 5, 6])
c = np.array([7, 8, 9])

# Define scalars k and m
k = 2
m = 3

# Verify Commutative Property of Addition: a + b = b + a
print("Commutative Property of Addition:")
print("a + b =", a + b)
print("b + a =", b + a)
print("a + b == b + a:", np.array_equal(a + b, b + a))

# Verify Associative Property of Addition: (a + b) + c = a + (b + c)
print("\nAssociative Property of Addition:")
print("(a + b) + c =", (a + b) + c)
print("a + (b + c) =", a + (b + c))
print("(a + b) + c == a + (b + c):", np.array_equal((a + b) + c, a + (b + c)))
```

```python
# Verify Additive Identity: a + 0 = a
zero_vector = np.zeros_like(a)
print("\nAdditive Identity:")
print("a + 0 =", a + zero_vector)
print("a + 0 == a:", np.array_equal(a + zero_vector, a))

# Verify Additive Inverse: a + (-a) = 0
print("\nAdditive Inverse:")
print("a + (-a) =", a + (-a))
print("a + (-a) == 0:", np.array_equal(a + (-a), zero_vector))

# Verify Distributive Property of Scalar Multiplication over Vector Addition: k(a + b) = ka + kb
print("\nDistributive Property of Scalar Multiplication over Vector Addition:")
print("k(a + b) =", k * (a + b))
print("ka + kb =", k * a + k * b)
print("k(a + b) == ka + kb:", np.array_equal(k * (a + b), k * a + k * b))

# Verify Distributive Property of Scalar Multiplication over Scalar Addition: (k + m)a = ka + ma
print("\nDistributive Property of Scalar Multiplication over Scalar Addition:")
print("(k + m)a =", (k + m) * a)
print("ka + ma =", k * a + m * a)
print("(k + m)a == ka + ma:", np.array_equal((k + m) * a, k * a + m * a))
```

```python
# Verify Associative Property of Scalar Multiplication: k(ma) = (km)a
print("\nAssociative Property of Scalar Multiplication:")
print("k(ma) =", k * (m * a))
print("(km)a =", (k * m) * a)
print("k(ma) == (km)a:", np.array_equal(k * (m * a), (k * m) * a))

# Verify Multiplicative Identity: 1a = a
print("\nMultiplicative Identity:")
print("1a =", 1 * a)
print("1a == a:", np.array_equal(1 * a, a))
```

Norm of Vector



Norm

The norm of a vector is a measure of its length or magnitude. It is a function that assigns a strictly positive length or size to all vectors in a vector space, except the zero vector.

Euclidean Norm (L_2 Norm)

The Euclidean norm, also known as the L2 norm, is the most common norm. It is the square root of the sum of the squares of the vector components.

```
l2_norm = np.sqrt(np.sum(vector**2))
```

$$sqrt(sum(square(v_i)))$$

Formula: $\|\mathbf{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$

Example: For the vector $\mathbf{v} = [3, 4]$: $\|\mathbf{v}\|_2 = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$

Manhattan Norm (L_1 Norm)

The Manhattan norm, also known as the L1 norm, is the sum of the absolute values of the vector components.

Formula: $\|\mathbf{v}\|_1 = |v_1| + |v_2| + \cdots + |v_n|$ $\boldsymbol{sum(abs(v_i))}$

```
l1_norm = np.sum(np.abs(vector))
```

Example: For the vector $\mathbf{v} = [3, -4]$: $\|\mathbf{v}\|_1 = |3| + |-4| = 3 + 4 = 7$

L-infinity Norm (Chebyshev Norm)

The L-infinity norm, also known as the Chebyshev norm, is the maximum absolute value of the vector components.

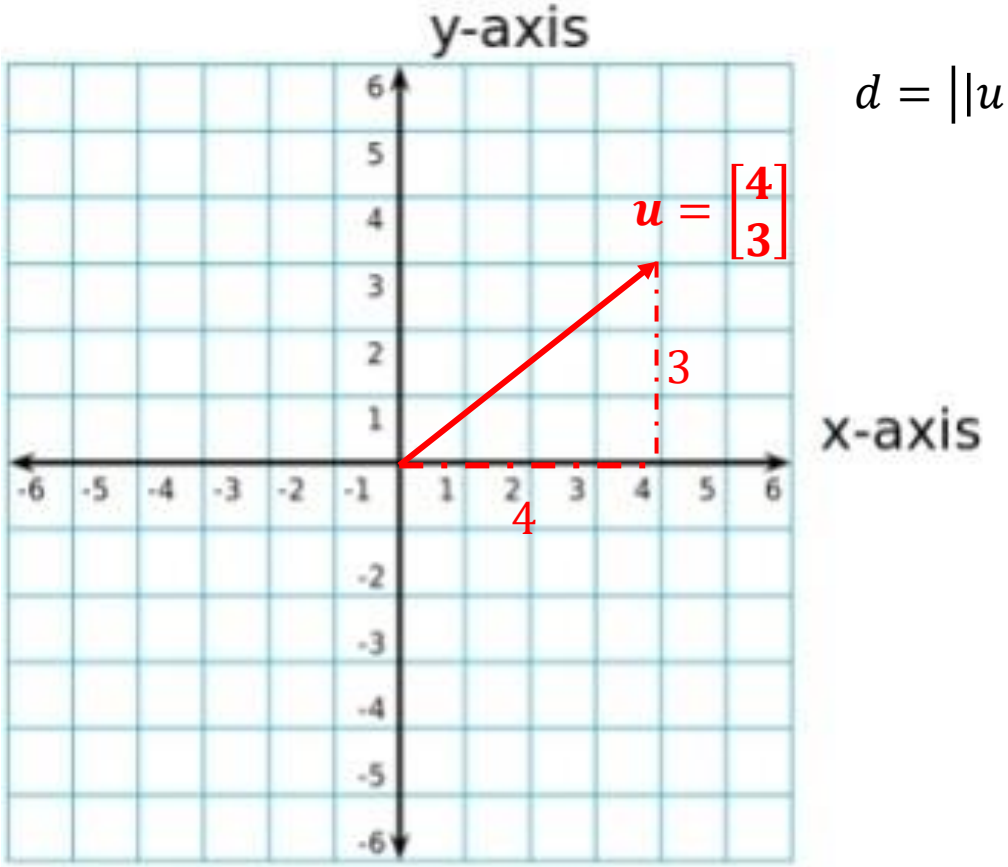
Formula: $\|\mathbf{v}\|_\infty = \max(|v_1|, |v_2|, \ldots, |v_n|)$ $\boldsymbol{max(abs(v_i))}$

```
chebyshev_norm = np.max(np.abs(vector))
```

Example: For the vector $\mathbf{v} = [3, -4]$: $\|\mathbf{v}\|_\infty = \max(|3|, |-4|) = \max(3, 4) = 4$

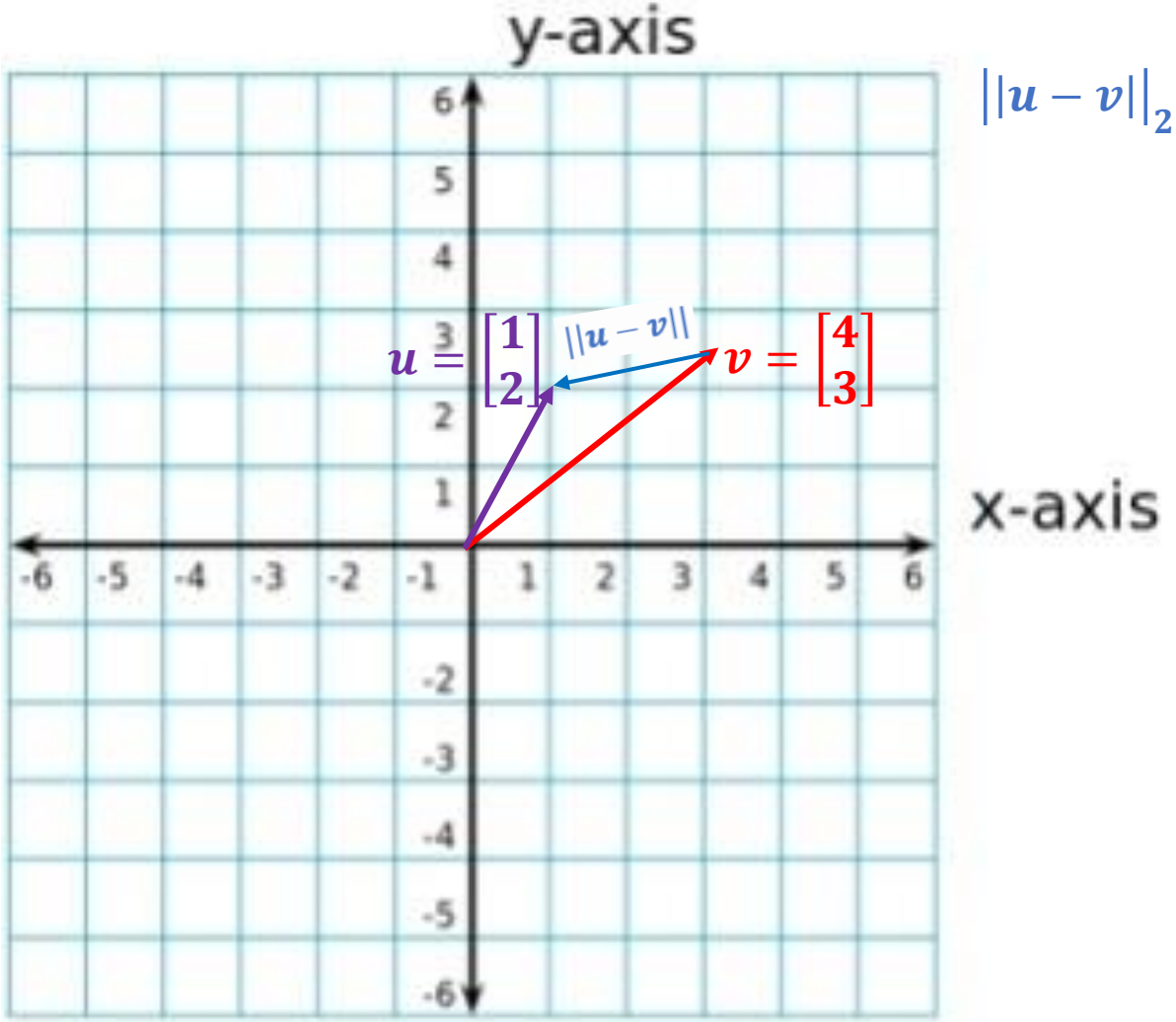
Norm

Norms-L_2



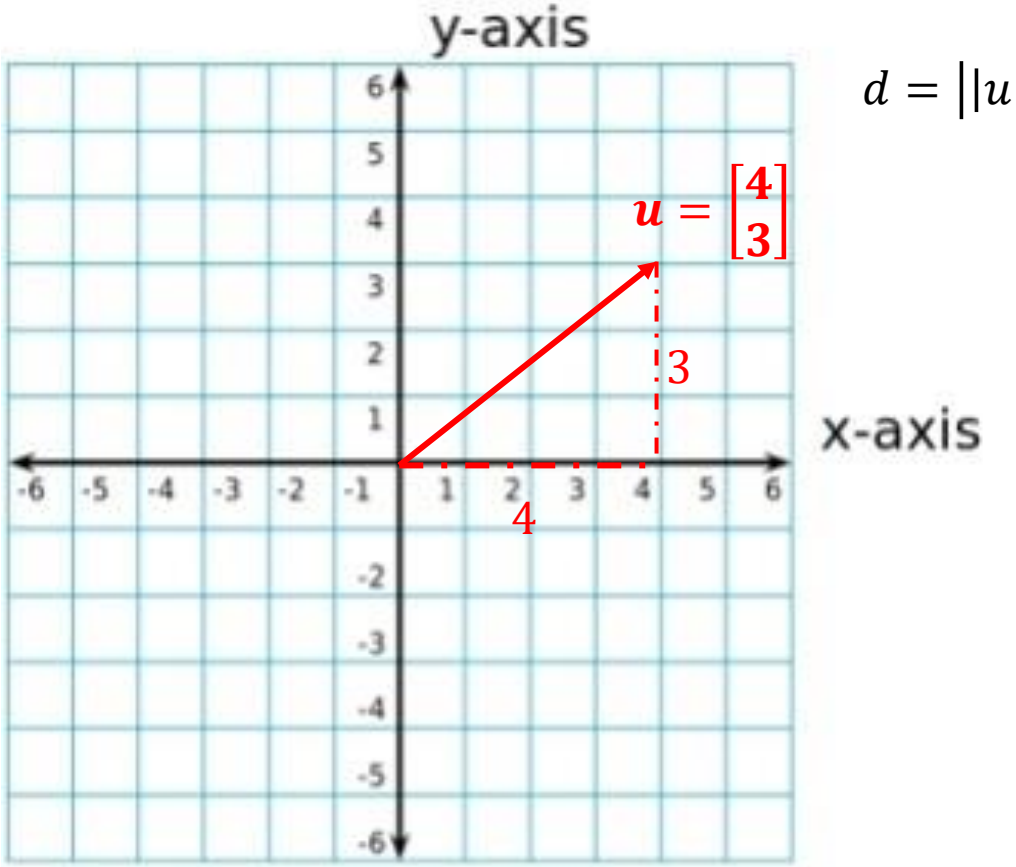
$d = \left|\left|u\right|\right|_2 = \sqrt{4^2 + 3^2} = \sqrt{25} = 5$ by Pythagoras theorem

Norms-L_2



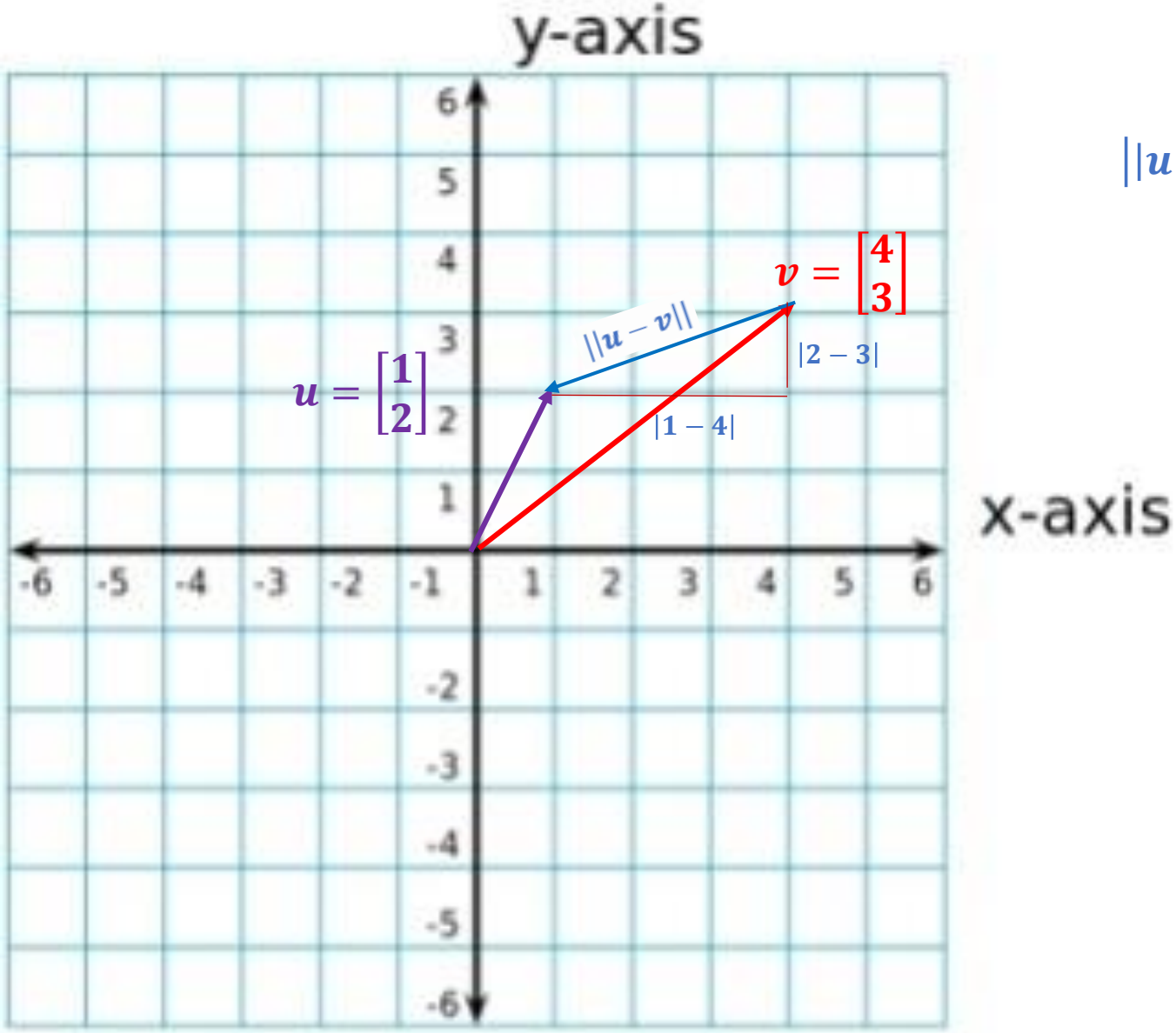
$$||u - v||_2 = \sqrt{(1-4)^2 + (2-3)^2} = \sqrt{9+1} = \sqrt{10}$$

Norms-L_1



$$d = \left|\left|u\right|\right|_1 = |4| + |3| = 7$$

Norms-L_1



$$\left|\left|u - v\right|\right|_1 = |1 - 4| + |2 - 3| = 3 + 1 = 4$$

Norm-L_∞

A king in chess moves one square in any direction: horizontally, vertically, or diagonally. Therefore, the king's movement capabilities align perfectly with the calculation of the L_∞ norm.

Minimum Number of Moves

Given two points (x_1, y_1) and (x_2, y_2) on a chessboard, the minimum number of moves required for a king to move from (x_1, y_1) to (x_2, y_2) is determined by the maximum of the absolute differences in the x and y coordinates, which is exactly the chessboard distance:

$$d_\infty((x_1, y_1), (x_2, y_2)) = \max(|x_2 - x_1|, |y_2 - y_1|)$$

Norm-L_∞

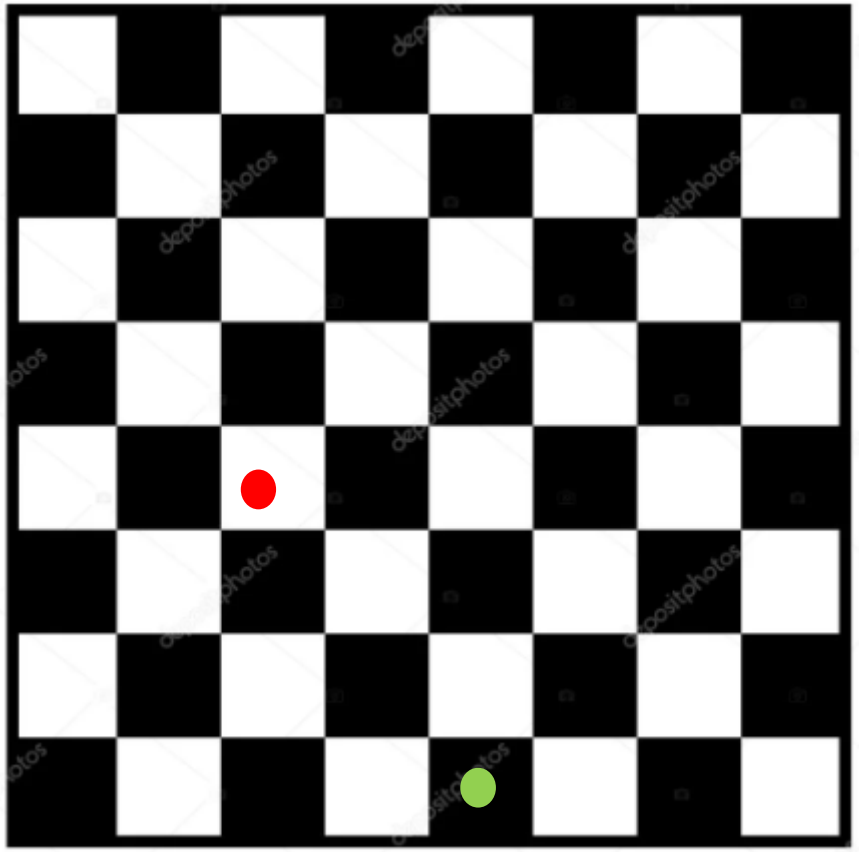
Example

Let's take the same example as before, where we want to move a king from $(2, 3)$ to $(5, 1)$:

1. Compute the differences in the x and y coordinates:
   $|5 - 2| = 3$
   $|1 - 3| = 2$

2. Take the maximum of these differences:
   $d_\infty((2, 3), (5, 1)) = \max(3, 2) = 3$

Thus, the minimum number of moves required for a king to move from $(2, 3)$ to $(5, 1)$ is 3.



Norm-L_∞

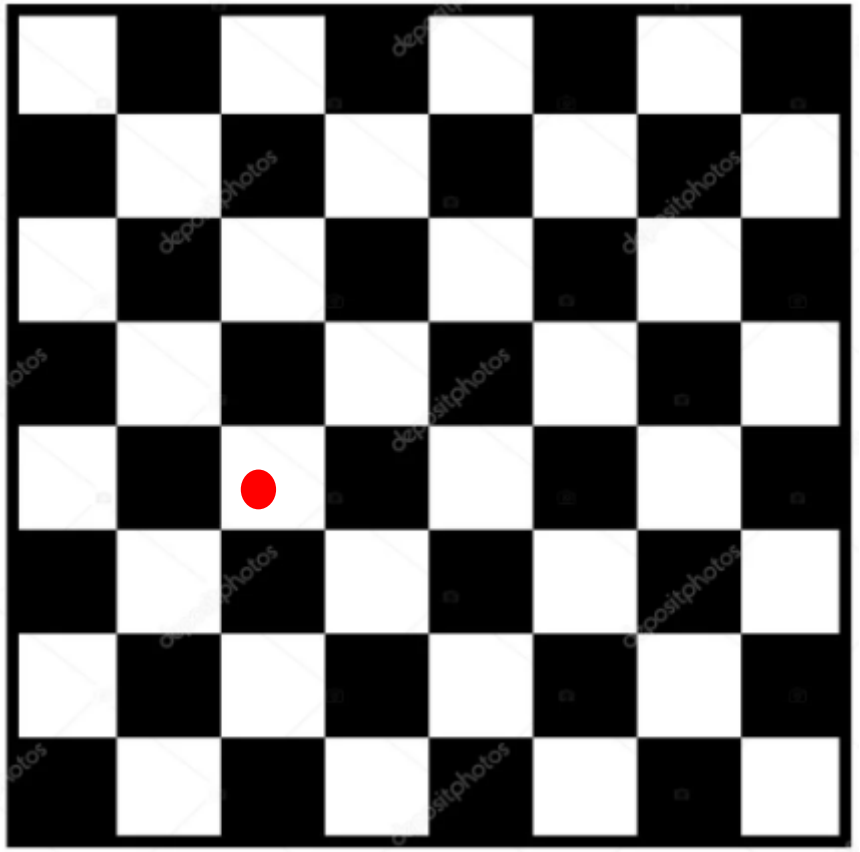
Example

Let's take the same example as before, where we want to move a king from $(2, 3)$ to $(5, 1)$:

1. Compute the differences in the x and y coordinates:
   $|5 - 2| = 3$
   $|1 - 3| = 2$

2. Take the maximum of these differences:
   $d_\infty((2, 3), (5, 1)) = \max(3, 2) = 3$

Thus, the minimum number of moves required for a king to move from $(2, 3)$ to $(5, 1)$ is 3.



Norm-L_∞

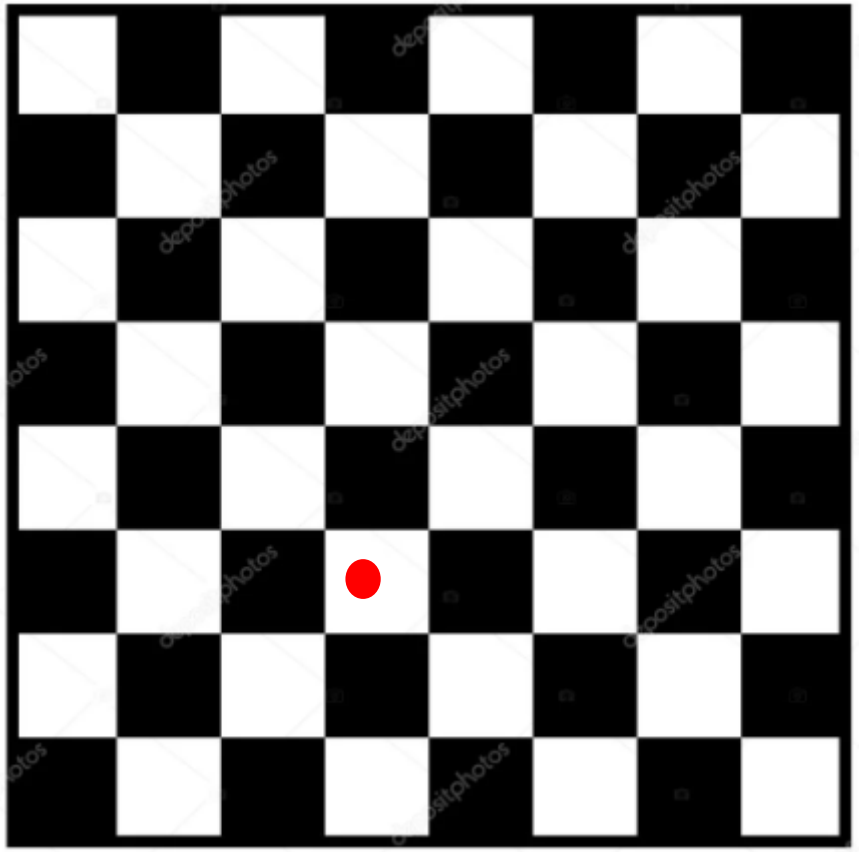
Example

Let's take the same example as before, where we want to move a king from $(2, 3)$ to $(5, 1)$:

1. Compute the differences in the x and y coordinates:
   $|5 - 2| = 3$
   $|1 - 3| = 2$

2. Take the maximum of these differences:
   $d_\infty((2, 3), (5, 1)) = \max(3, 2) = 3$

Thus, the minimum number of moves required for a king to move from $(2, 3)$ to $(5, 1)$ is 3.



Norm-L_∞

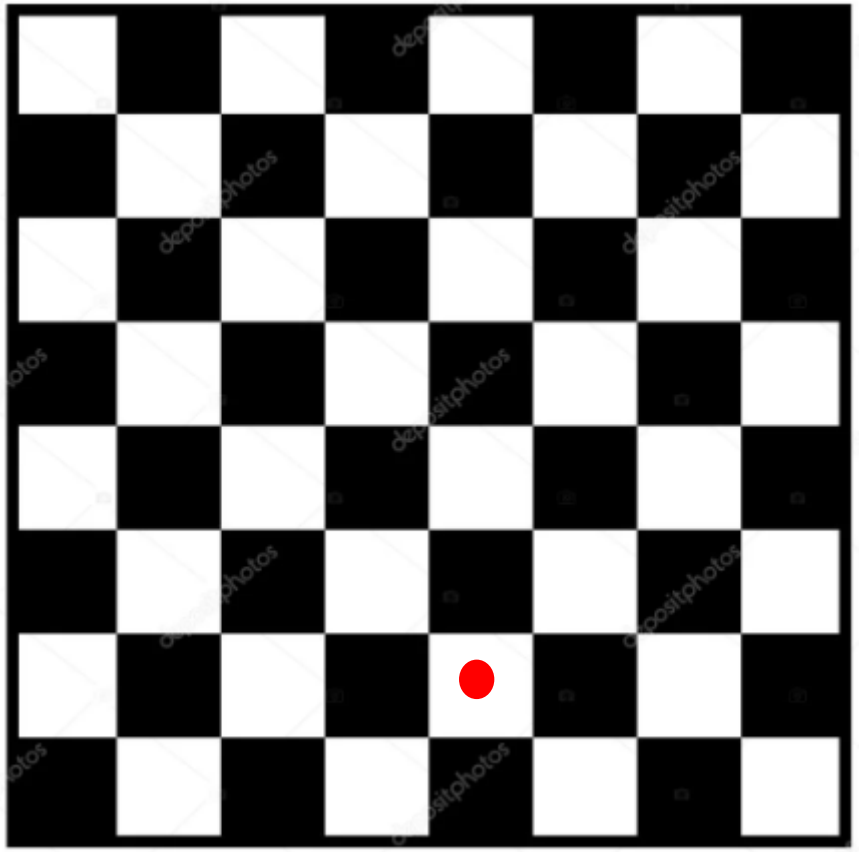
Example

Let's take the same example as before, where we want to move a king from $(2, 3)$ to $(5, 1)$:

1. Compute the differences in the x and y coordinates:
   $|5 - 2| = 3$
   $|1 - 3| = 2$

2. Take the maximum of these differences:
   $d_\infty((2, 3), (5, 1)) = \max(3, 2) = 3$

Thus, the minimum number of moves required for a king to move from $(2, 3)$ to $(5, 1)$ is 3.



Norm-L_∞

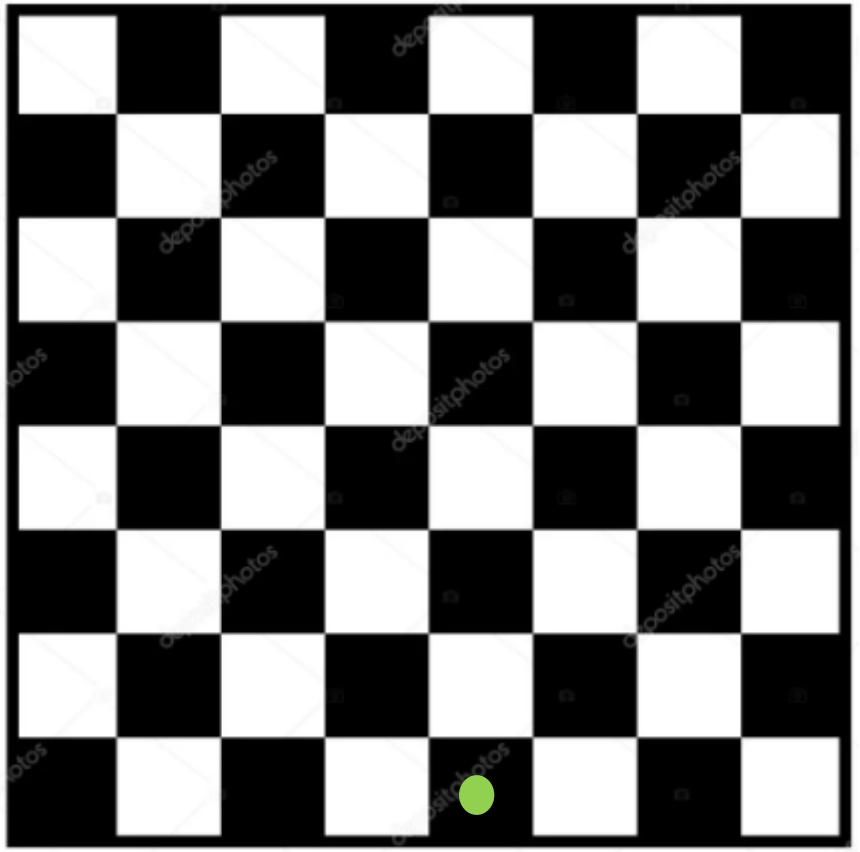
Example

Let's take the same example as before, where we want to move a king from $(2, 3)$ to $(5, 1)$:

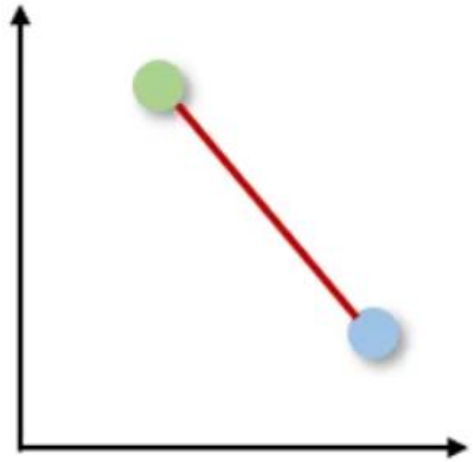
1. Compute the differences in the x and y coordinates:
   $|5 - 2| = 3$
   $|1 - 3| = 2$

2. Take the maximum of these differences:
   $d_\infty((2, 3), (5, 1)) = \max(3, 2) = 3$

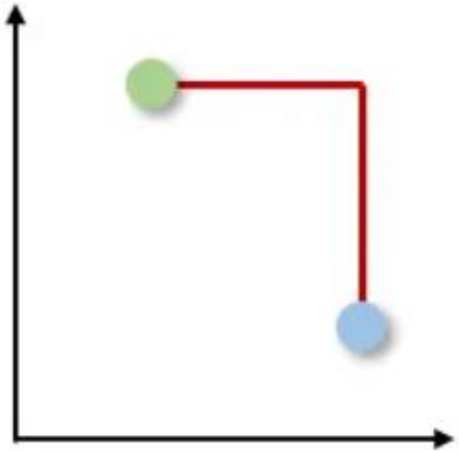
Thus, the minimum number of moves required for a king to move from $(2, 3)$ to $(5, 1)$ is 3.



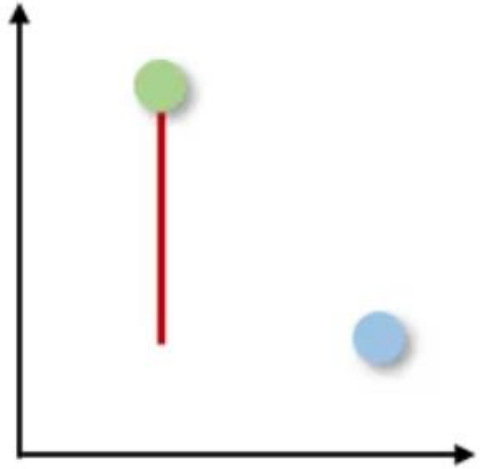
Euclidean



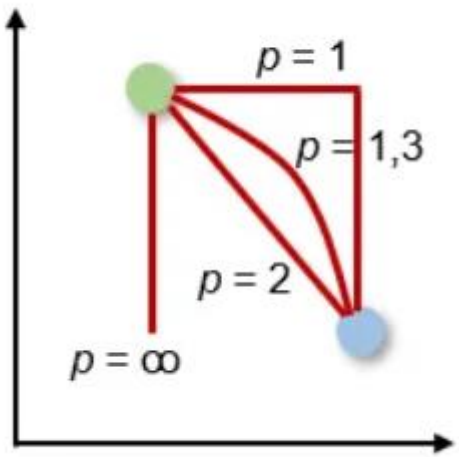
Manhattan



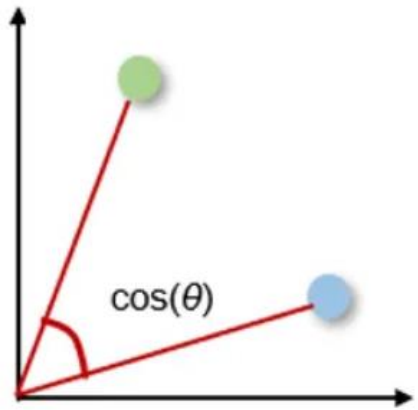
Chebyshev



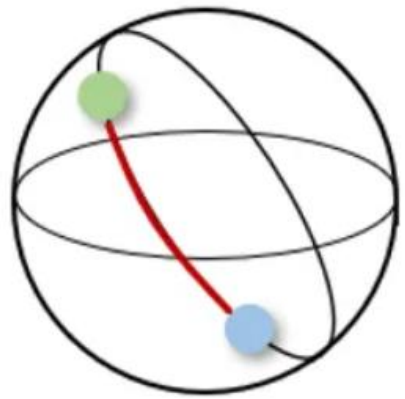
Minkowski



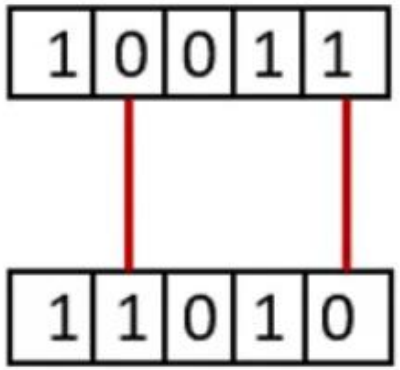
Cosine



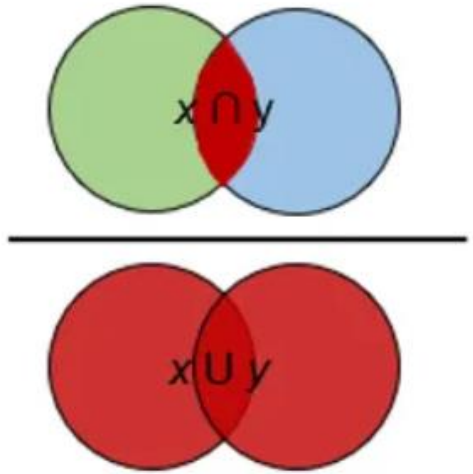
Haversine



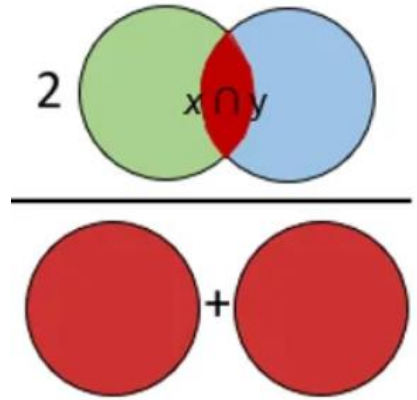
Hamming

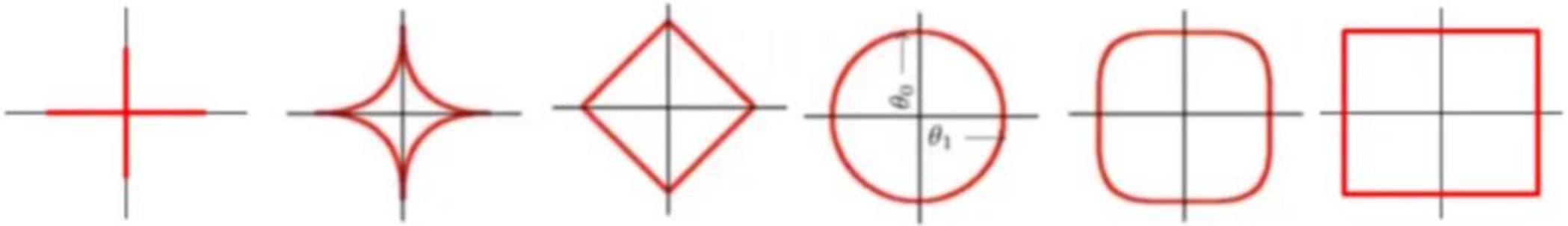


Jaccard



Sörensen-Dice





$p \to 0$

$\|x\|_0 = (\sum_i |x_i|^0)^\infty$

The number of non-zero parameters

$p = 0.5$

$\|x\|_{0.5} = (\sum_i |x_i|^{0.5})^2$

$p = 1$

$\|x\|_1 = (\sum_i |x_i|^1)^1$

$p = 2$

$\|x\|_2 = (\sum_i |x_i|^2)^{1/2}$

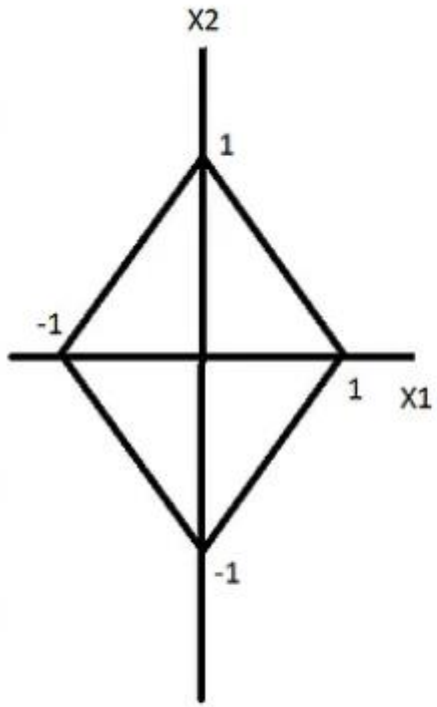
$p = 4$

$\|x\|_4 = (\sum_i |x_i|^4)^{1/4}$

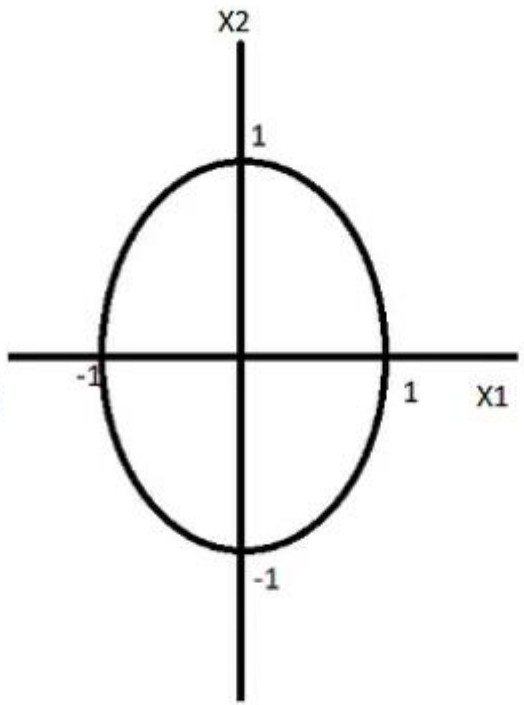
$p \to \infty$

$\|x\|_\infty = (\sum_i |x_i|^\infty)^0$

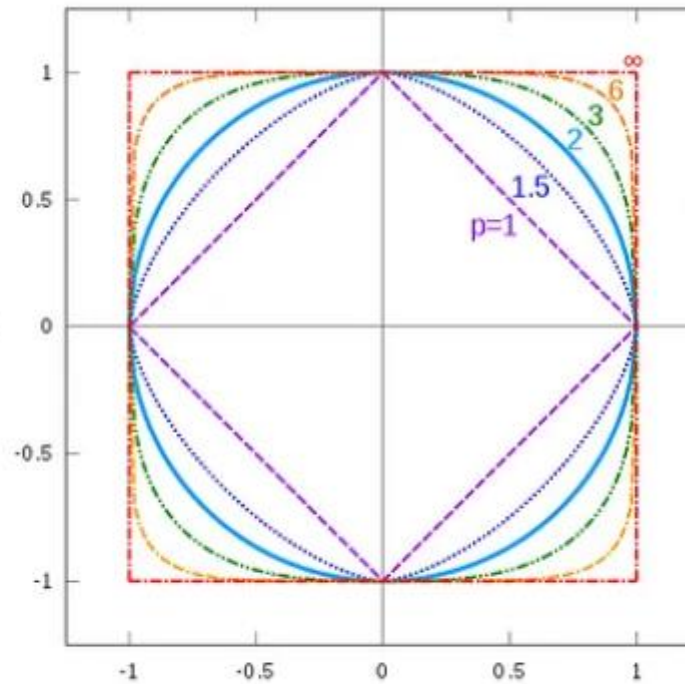
The size of the largest parameter



L1 Norm



L2 Norm



L infinity Norm

Properties of Norm

1. **Non-negativity**:
   - The norm of a vector is always non-negative.
   - $\|\mathbf{v}\| \geq 0$ for all vectors $\mathbf{v}$.
   - The norm is zero if and only if the vector is the zero vector.
   - $\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = \mathbf{0}$.
2. **Homogeneity (Positive Scalability)**:
   - The norm of a scalar multiple of a vector is equal to the absolute value of the scalar times the norm of the vector.
   - $\|c\mathbf{v}\| = |c|\|\mathbf{v}\|$ for any scalar c and vector $\mathbf{v}$.

Properties of Norm

3. **Triangle Inequality**:
    - The norm of the sum of two vectors is less than or equal to the sum of the norms of the two vectors.
    - $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$ for all vectors $\mathbf{u}$ and $\mathbf{v}$.
4. **Subadditivity**:
    - This property is a restatement of the triangle inequality, emphasizing that the norm function is subadditive.
    - $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$.
5. **Definiteness**:
    - The norm is zero if and only if the vector itself is the zero vector.
    - $\|\mathbf{v}\| = 0$ if and only if $\mathbf{v} = \mathbf{0}$.

```python
import numpy as np

# Define a function to calculate all three norms of a vector
def calculate_norms(vector):
    l1_norm = np.sum(np.abs(vector))
    l2_norm = np.sqrt(np.sum(vector**2))
    chebyshev_norm = np.max(np.abs(vector))
    return l1_norm, l2_norm, chebyshev_norm

# Define a function to calculate distances between two vectors
def calculate_distances(vector1, vector2):
    diff_vector = vector1 - vector2
    l1_distance = np.sum(np.abs(diff_vector))
    l2_distance = np.sqrt(np.sum(diff_vector**2))
    chebyshev_distance = np.max(np.abs(diff_vector))
    return l1_distance, l2_distance, chebyshev_distance
```

```python
# Define vectors
vector = np.array([1, 2, -3])
vector1 = np.array([1, 2, 3])
vector2 = np.array([4, -1, 3])

# Calculate norms of a vector
l1_norm, l2_norm, chebyshev_norm = calculate_norms(vector)
print(f"L1 Norm of {vector}: {l1_norm}")
print(f"L2 Norm of {vector}: {l2_norm}")
print(f"Chebyshev Norm of {vector}: {chebyshev_norm}")

# Calculate distances between two vectors
l1_distance, l2_distance, chebyshev_distance = calculate_distances(vector1, vector2)
print(f"L1 Distance between {vector1} and {vector2}: {l1_distance}")
print(f"L2 Distance between {vector1} and {vector2}: {l2_distance}")
print(f"Chebyshev Distance between {vector1} and {vector2}: {chebyshev_distance}")
```

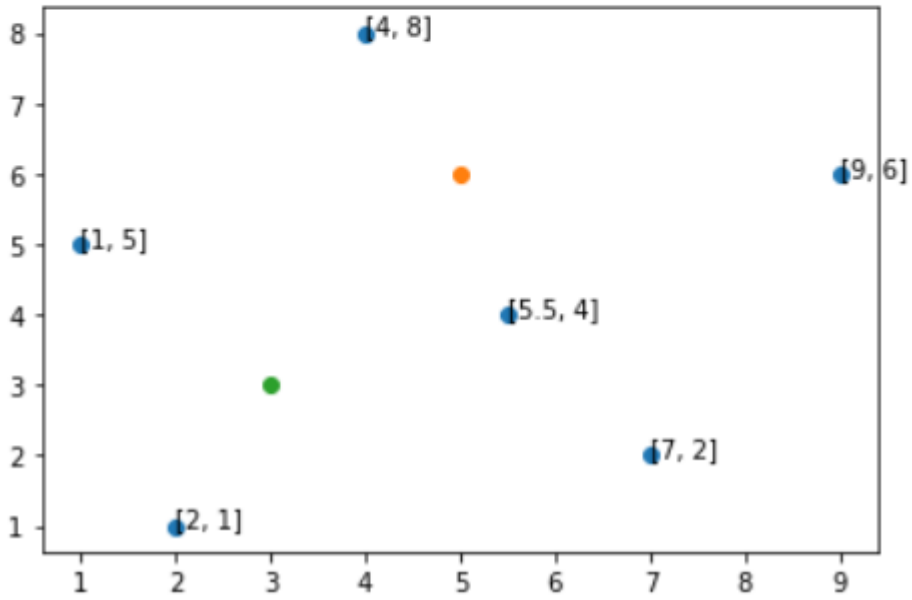
```
L1 Norm of [ 1  2 -3]: 6
L2 Norm of [ 1  2 -3]: 3.7416573867739413
Chebyshev Norm of [ 1  2 -3]: 3
L1 Distance between [1 2 3] and [ 4 -1  3]: 6
L2 Distance between [1 2 3] and [ 4 -1  3]: 4.242640687119285
Chebyshev Distance between [1 2 3] and [ 4 -1  3]: 3
```

Nearest Neighbours

```
# np.argmin: Returns the indices of the minimum values along an axis.
near_neigh = lambda x, z: z[np.argmin([np.linalg.norm(x-y) for y in z])]

x = np.array([5, 6])
y = np.array([3, 3])
z = ([2, 1], [7, 2], [5.5, 4], [4, 8], [1, 5], [9, 6])

print(near_neigh(x, z))
print(near_neigh(y, z))
```

```
[5.5, 4]
[2, 1]
```

```
plt.scatter(*zip(*z))
n = [str(i) for i in z]
for i, txt in enumerate(n):
    plt.annotate(txt, (z[i][0], z[i][1]))
plt.scatter(x[0], x[1])
plt.scatter(y[0], y[1])
plt.show()
```



```python
import numpy as np
import pandas as pd
from scipy.spatial.distance import cdist
from sklearn.datasets import load_iris

# Load the Iris dataset
iris = load_iris()
iris_data = iris.data
iris_feature_names = iris.feature_names
iris_df = pd.DataFrame(iris_data, columns=iris_feature_names)

# Print the Iris dataset
print("Iris Dataset:")
print(iris_df.head())

# Extract the first four columns
data = iris_df.iloc[:, :4].values

# Compute distance matrices
euclidean_distances = cdist(data, data, metric='euclidean')
manhattan_distances = cdist(data, data, metric='cityblock')
chebyshev_distances = cdist(data, data, metric='chebyshev')

# Convert distance matrices to DataFrames for better readability
euclidean_df = pd.DataFrame(euclidean_distances, index=iris_df.index, columns=iris_df.index)
manhattan_df = pd.DataFrame(manhattan_distances, index=iris_df.index, columns=iris_df.index)
chebyshev_df = pd.DataFrame(chebyshev_distances, index=iris_df.index, columns=iris_df.index)

# Print distance matrices
print("\nEuclidean Distance Matrix:")
print(euclidean_df.head())

print("\nManhattan Distance Matrix:")
print(manhattan_df.head())

print("\nChebyshev Distance Matrix:")
print(chebyshev_df.head())
```

Iris Dataset:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

	p1	p2	p3	.	.	.	p150
p1	d(p1,p1)						
p2		d(p2,p2)					
p3			d(p3,p3)				
.				.			
.					.		
.						.	
p150							d(p150,p150)

$$d(p_i, p_i) = 0$$

$$d(p_i, p_j) = d(p_j, p_i)$$

Euclidean Distance Matrix:

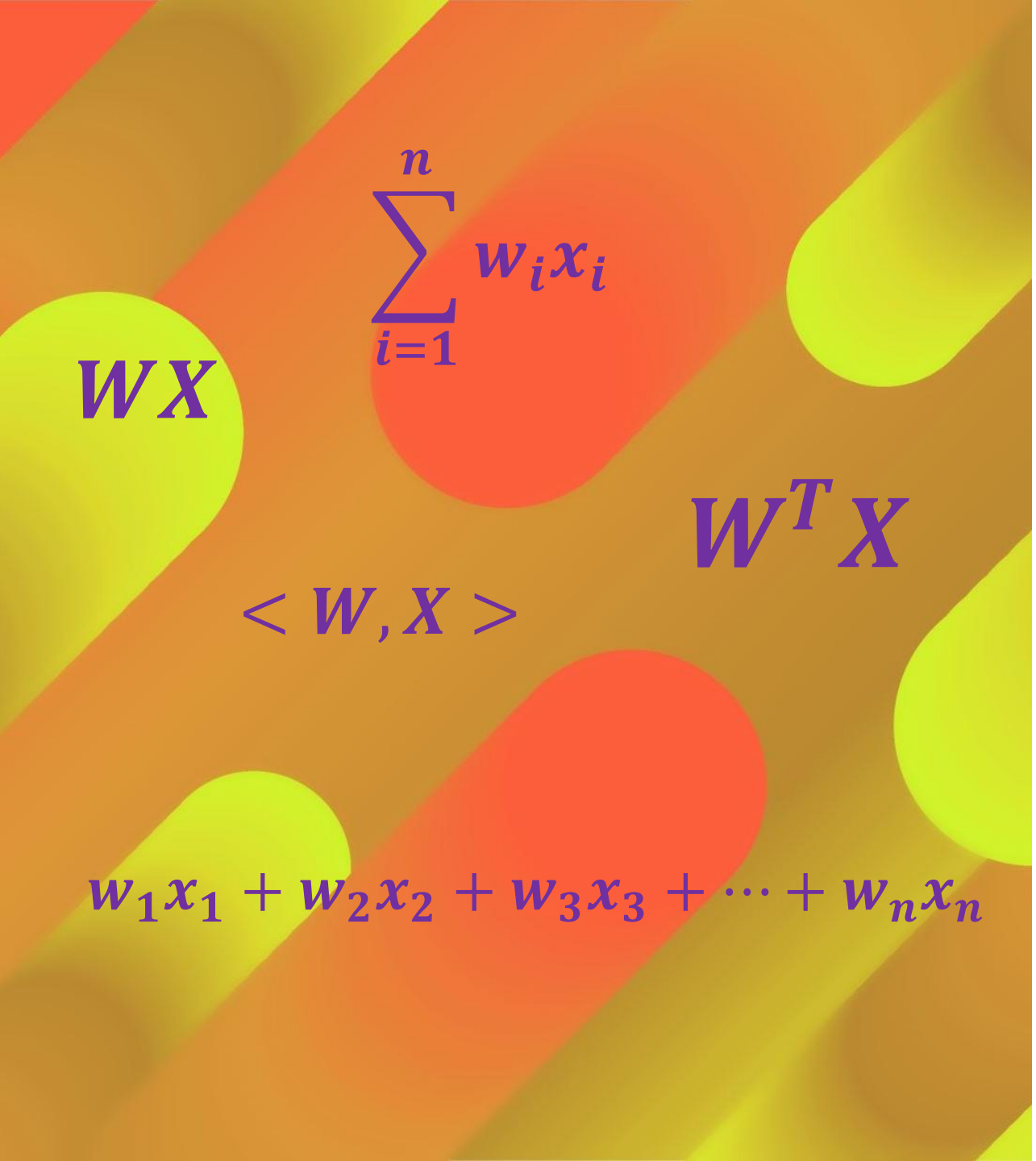
	0	1	2	3	4	5	6	\
0	0.000000	0.538516	0.509902	0.648074	0.141421	0.616441	0.519615	
1	0.538516	0.000000	0.300000	0.331662	0.608276	1.090871	0.509902	
2	0.509902	0.300000	0.000000	0.244949	0.509902	1.086278	0.264575	
3	0.648074	0.331662	0.244949	0.000000	0.648074	1.166190	0.331662	
4	0.141421	0.608276	0.509902	0.648074	0.000000	0.616441	0.458258	

Manhattan Distance Matrix:

	0	1	2	3	4	5	6	7	8	9	...	140	141	142	143	\
0	0.0	0.7	0.8	1.0	0.2	1.2	0.7	0.3	1.3	0.8	...	8.4	8.0	6.9	8.6	
1	0.7	0.0	0.5	0.5	0.7	1.9	0.8	0.6	0.6	0.3	...	8.3	7.9	6.6	8.7	
2	0.8	0.5	0.0	0.4	0.8	2.0	0.5	0.7	0.7	0.6	...	8.6	8.2	7.1	8.8	
3	1.0	0.5	0.4	0.0	1.0	2.0	0.5	0.7	0.5	0.4	...	8.4	8.0	6.9	8.8	
4	0.2	0.7	0.8	1.0	0.0	1.2	0.7	0.3	1.3	0.8	...	8.6	8.2	7.1	8.8	

Chebyshev Distance Matrix:

	0	1	2	3	4	5	6	7	8	9	...	140	141	142	143	\
0	0.0	0.5	0.4	0.5	0.1	0.4	0.5	0.1	0.7	0.4	...	4.2	3.7	3.7	4.5	
1	0.5	0.0	0.2	0.3	0.6	0.9	0.4	0.4	0.5	0.1	...	4.2	3.7	3.7	4.5	
2	0.4	0.2	0.0	0.2	0.4	0.7	0.2	0.3	0.3	0.2	...	4.3	3.8	3.8	4.6	
3	0.5	0.3	0.2	0.0	0.5	0.8	0.3	0.4	0.2	0.3	...	4.1	3.6	3.6	4.4	
4	0.1	0.6	0.4	0.5	0.0	0.4	0.4	0.2	0.7	0.5	...	4.2	3.7	3.7	4.5	



Dot Product

Inner Product

Dot Product

Definition

For two vectors **X** and **Y** in an n-dimensional space, the dot product is defined as:

$$\mathbf{X} \cdot \mathbf{Y} = \sum_{i=1}^{n} X_i Y_i$$

where X_i and Y_i are the components of vectors **X** and **Y**, respectively.

Dot Product

Dot Product of Vectors in $\mathbb{R}^2$

For two vectors X = $[X_1, X_2]$ and Y = $[Y_1, Y_2]$:

$$\mathbf{X} \cdot \mathbf{Y} = X_1 Y_1 + X_2 Y_2$$

Dot Product of Vectors in $\mathbb{R}^3$

For two vectors X = $[X_1, X_2, X_3]$ and Y = $[Y_1, Y_2, Y_3]$:

$$\mathbf{X} \cdot \mathbf{Y} = X_1 Y_1 + X_2 Y_2 + X_3 Y_3$$

General Formula in n-dimensional Space

For two vectors X and Y in $\mathbb{R}^n$:

$$\mathbf{X} \cdot \mathbf{Y} = X_1 Y_1 + X_2 Y_2 + \cdots + X_n Y_n$$

Dot Product

Using Magnitudes and Angle Between Vectors

The dot product can also be expressed in terms of the magnitudes of the vectors and the cosine of the angle θ between them:

$$\mathbf{X} \cdot \mathbf{Y} = \|\mathbf{X}\| \|\mathbf{Y}\| \cos\theta$$

where $\|\mathbf{X}\|$ and $\|\mathbf{Y}\|$ are the magnitudes (or lengths) of vectors **X** and **Y**, respectively.

Properties

1. **Commutative Property:**
   $$\mathbf{X} \cdot \mathbf{Y} = \mathbf{Y} \cdot \mathbf{X}$$

2. **Distributive Property:**
   $$\mathbf{X} \cdot (\mathbf{Y} + \mathbf{Z}) = \mathbf{X} \cdot \mathbf{Y} + \mathbf{X} \cdot \mathbf{Z}$$

3. **Associative Property with Scalars:**
   $$(c\mathbf{X}) \cdot \mathbf{Y} = c(\mathbf{X} \cdot \mathbf{Y})$$
   where c is a scalar.

4. **Zero Vector:**
   $$\mathbf{X} \cdot \mathbf{0} = 0$$
   where $\mathbf{0}$ is the zero vector.

5. **Orthogonal Vectors:**
   If **X** and **Y** are orthogonal (perpendicular), then:
   $$\mathbf{X} \cdot \mathbf{Y} = 0$$

6. **Self Dot Product (Magnitude):**
   $$\mathbf{X} \cdot \mathbf{X} = \|\mathbf{X}\|^2$$
   This property shows that the dot product of a vector with itself gives the square of its magnitude.

More..

Geometric Interpretation

The dot product of two vectors can be interpreted geometrically as a measure of how much one vector extends in the direction of another. It is positive if the angle between the vectors is acute, zero if the vectors are orthogonal, and negative if the angle is obtuse.

Applications

The dot product has various applications in physics, computer graphics, and geometry, such as:

- Calculating the angle between two vectors.
- Projecting one vector onto another.
- Determining whether two vectors are orthogonal.
- Calculating work done by a force in the direction of displacement.

Applications

Weight Vector and Feature Vector: Used in machine learning for scoring.

Vector of Prices and Quantities: Used in economics for calculating total cost.

Cash Flow and Discount Vector: Used in finance for calculating the net present value (NPV).

Portfolio Holdings and Asset Prices: Used in finance to calculate the total value of a portfolio.

1. Weight Vector and Feature Vector

Explanation:

- $\mathbf{w}$ is a weight vector.
- $\mathbf{f}$ is a feature vector.
- $\mathbf{w}^T\mathbf{f}$ is the score, which is the dot product of $\mathbf{w}$ and $\mathbf{f}$.

Example:

Suppose we have a weight vector and a feature vector as follows:

$$\mathbf{w} = \begin{bmatrix} 0.2 \\ 0.3 \\ 0.5 \end{bmatrix}, \quad \mathbf{f} = \begin{bmatrix} 5 \\ 2 \\ 3 \end{bmatrix}$$

The score (dot product) is:

$$\mathbf{w}^T\mathbf{f} = 0.2 \cdot 5 + 0.3 \cdot 2 + 0.5 \cdot 3 = 1 + 0.6 + 1.5 = 3.1$$

2. Vector of Prices and Quantities

Explanation:

- $\mathbf{p}$ is a vector of prices.
- $\mathbf{q}$ is a vector of quantities.
- $\mathbf{p}^T\mathbf{q}$ is the total cost, which is the dot product of $\mathbf{p}$ and $\mathbf{q}$.

Example:

Suppose we have prices and quantities as follows:

$$\mathbf{p} = \begin{bmatrix} 10 \\ 20 \\ 15 \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} 2 \\ 1 \\ 4 \end{bmatrix}$$

The total cost is:

$$\mathbf{p}^T\mathbf{q} = 10 \cdot 2 + 20 \cdot 1 + 15 \cdot 4 = 20 + 20 + 60 = 100$$

Applications

3. Cash Flow and Discount Vector

Explanation:

- $\mathbf{c}$ is a cash flow vector.
- $\mathbf{d}$ is a discount vector with interest rate r.
- The discount vector is:
  $\mathbf{d} = \left[1, \frac{1}{1+r}, \frac{1}{(1+r)^2}, \ldots, \frac{1}{(1+r)^{n-1}}\right]$
- $\mathbf{d}^T\mathbf{c}$ is the net present value (NPV) of cash flow, which is the dot product of $\mathbf{d}$ and $\mathbf{c}$.

Example:

Suppose we have a cash flow vector and an interest rate $r = 0.05$ (5%):

$$\mathbf{c} = \begin{bmatrix} 100 \\ 150 \\ 200 \end{bmatrix}, \quad \mathbf{d} = \left[1, \frac{1}{1.05}, \frac{1}{1.05^2}\right] = [1, 0.9524, 0.9070]$$

The NPV is:

$$\mathbf{d}^T\mathbf{c} = 1 \cdot 100 + 0.9524 \cdot 150 + 0.9070 \cdot 200 = 100 + 142.86 + 181.4 = 424.26$$

4. Portfolio Holdings and Asset Prices

Explanation:

- $\mathbf{s}$ gives portfolio holdings (in shares).
- $\mathbf{p}$ gives asset prices.
- $\mathbf{p}^T\mathbf{s}$ is the total portfolio value, which is the dot product of $\mathbf{p}$ and $\mathbf{s}$.

Example:

Suppose we have a portfolio and asset prices as follows:

$$\mathbf{s} = \begin{bmatrix} 10 \\ 20 \\ 30 \end{bmatrix}, \quad \mathbf{p} = \begin{bmatrix} 50 \\ 60 \\ 70 \end{bmatrix}$$

The total portfolio value is:

$$\mathbf{p}^T\mathbf{s} = 50 \cdot 10 + 60 \cdot 20 + 70 \cdot 30 = 500 + 1200 + 2100 = 3800$$

Applications

Let's consider a vector $\mathbf{x}$:

$$\mathbf{x} = \begin{bmatrix} 4 \\ 8 \\ 6 \\ 10 \end{bmatrix}$$

Step-by-Step Process to De-mean the Vector

1. **Calculate the Mean of the Vector:**

$$\text{avg}(\mathbf{x}) = \frac{4+8+6+10}{4} = \frac{28}{4} = 7$$

2. **Create a Vector of the Mean Values:**

$$\text{avg}(\mathbf{x})\mathbf{1} = 7 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 7 \\ 7 \end{bmatrix}$$

3. **Subtract the Mean Vector from the Original Vector:**

$$\mathbf{x} - \text{avg}(\mathbf{x})\mathbf{1} = \begin{bmatrix} 4 \\ 8 \\ 6 \\ 10 \end{bmatrix} - \begin{bmatrix} 7 \\ 7 \\ 7 \\ 7 \end{bmatrix} = \begin{bmatrix} 4-7 \\ 8-7 \\ 6-7 \\ 10-7 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ -1 \\ 3 \end{bmatrix}$$

So, the de-meaned version of $\mathbf{x}$ is:

$$\mathbf{x}_{\text{de-meaned}} = \begin{bmatrix} -3 \\ 1 \\ -1 \\ 3 \end{bmatrix}$$

Applications

Formula

$$\mathrm{std}(\mathbf{x}) = \frac{\|\mathbf{x} - \mathrm{avg}(\mathbf{x})\mathbf{1}\|}{\sqrt{n}}$$

where $\mathbf{x}$ is the vector, $\mathrm{avg}(\mathbf{x})$ is the mean of the vector, and n is the length of the vector.

$$\text{std}(\mathbf{x}) = \frac{\|\mathbf{x} - \text{avg}(\mathbf{x})\mathbf{1}\|}{\sqrt{n}}$$

Applications

Consider the vector:

$$\mathbf{x} = \begin{bmatrix} 4 \\ 8 \\ 6 \\ 10 \end{bmatrix}$$

1. **Calculate the Mean of the Vector:**
   $\text{avg}(\mathbf{x}) = \frac{4+8+6+10}{4} = \frac{28}{4} = 7$

2. **Create the Mean Vector:**
   $$\text{avg}(\mathbf{x})\mathbf{1} = 7 \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 7 \\ 7 \\ 7 \end{bmatrix}$$

3. **Subtract the Mean Vector from the Original Vector:**
   $$\mathbf{x} - \text{avg}(\mathbf{x})\mathbf{1} = \begin{bmatrix} 4 \\ 8 \\ 6 \\ 10 \end{bmatrix} - \begin{bmatrix} 7 \\ 7 \\ 7 \\ 7 \end{bmatrix} = \begin{bmatrix} -3 \\ 1 \\ -1 \\ 3 \end{bmatrix}$$

4. **Calculate the Norm of the De-meaned Vector:**
   $\|\mathbf{x} - \text{avg}(\mathbf{x})\mathbf{1}\| = \sqrt{(-3)^2 + 1^2 + (-1)^2 + 3^2} = \sqrt{9 + 1 + 1 + 9} = \sqrt{20}$

5. **Divide by $\sqrt{n}$:**
   Since the length of the vector $n = 4$:
   $\text{std}(\mathbf{x}) = \frac{\|\mathbf{x} - \text{avg}(\mathbf{x})\mathbf{1}\|}{\sqrt{n}} = \frac{\sqrt{20}}{\sqrt{4}} = \frac{\sqrt{20}}{2} = \frac{2\sqrt{5}}{2} = \sqrt{5}$

Applications

Angle

In Euclidean space, a Euclidean vector is a geometric object that possesses both a magnitude and a direction. A vector can be pictured as an arrow. Its magnitude is its length, and its direction is the direction to which the arrow points. The magnitude of a vector a is denoted by $\|\mathbf{a}\|$. The dot product of two Euclidean vectors a and b is defined by

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta,$$

where θ is the angle between a and b.

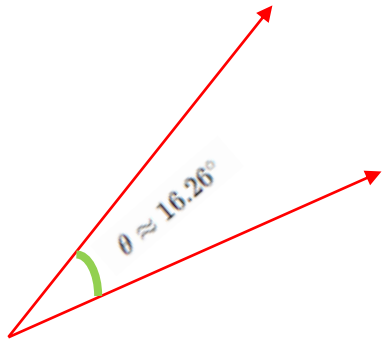
Applications

Example

Consider the vectors:

$$\mathbf{a} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\mathbf{b} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$



Step-by-Step Calculation

1. Calculate the Dot Product:
   $\mathbf{a} \cdot \mathbf{b} = 3 \cdot 4 + 4 \cdot 3 = 12 + 12 = 24$

2. Calculate the Magnitudes:
   $\|\mathbf{a}\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$
   $\|\mathbf{b}\| = \sqrt{4^2 + 3^2} = \sqrt{16 + 9} = \sqrt{25} = 5$

3. Use the Dot Product and Magnitudes to Find $\cos\theta$:
   $\cos\theta = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\|\|\mathbf{b}\|} = \frac{24}{5 \cdot 5} = \frac{24}{25} = 0.96$

4. Calculate θ Using the Inverse Cosine Function:
   $\theta = \cos^{-1}(0.96)$

Using a calculator to find the inverse cosine:

$\theta \approx 16.26^\circ$

Projection of vectors

Scalar Projection

The scalar projection of one vector onto another gives the length of the projection of one vector onto the other. It is a measure of how much of one vector lies in the direction of the other.

Scalar Projection of $\mathbf{u}$ on $\mathbf{v}$

The scalar projection of vector $\mathbf{u}$ onto vector $\mathbf{v}$ is given by:

$$\text{scal}_{\mathbf{v}}(\mathbf{u}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|}$$

Scalar Projection of $\mathbf{v}$ on $\mathbf{u}$

The scalar projection of vector $\mathbf{v}$ onto vector $\mathbf{u}$ is given by:

$$\text{scal}_{\mathbf{u}}(\mathbf{v}) = \frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|}$$

Projection of vectors

Example

Consider the vectors:

$$\mathbf{u} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

1. **Scalar Projection of $\mathbf{u}$ on $\mathbf{v}$:**

$$\text{scal}_{\mathbf{v}}(\mathbf{u}) = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{v}\|}$$

- Calculate the dot product $\mathbf{u} \cdot \mathbf{v}$:

$$\mathbf{u} \cdot \mathbf{v} = 3 \cdot 4 + 4 \cdot 3 = 12 + 12 = 24$$

- Calculate the magnitude $\|\mathbf{v}\|$:

$$\|\mathbf{v}\| = \sqrt{4^2 + 3^2} = \sqrt{16 + 9} = \sqrt{25} = 5$$

- Compute the scalar projection:

$$\text{scal}_{\mathbf{v}}(\mathbf{u}) = \frac{24}{5} = 4.8$$

Projection of vectors

2. **Scalar Projection of $\mathbf{v}$ on $\mathbf{u}$:**

$$\text{scal}_{\mathbf{u}}(\mathbf{v}) = \frac{\mathbf{v} \cdot \mathbf{u}}{\|\mathbf{u}\|}$$

- The dot product is the same as before (since dot product is commutative):

$$\mathbf{v} \cdot \mathbf{u} = 24$$

- The magnitude $\|\mathbf{u}\|$ is:

$$\|\mathbf{u}\| = \sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

- Compute the scalar projection:

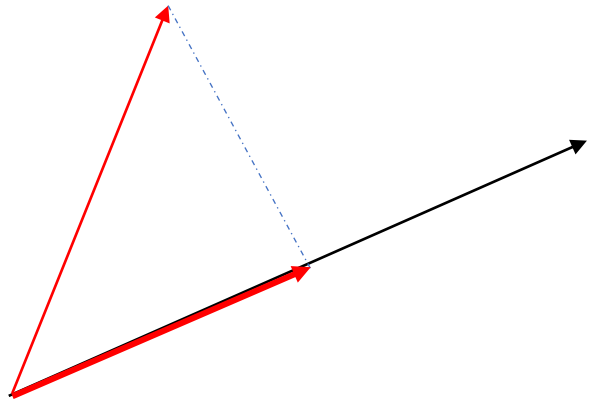
$$\text{scal}_{\mathbf{u}}(\mathbf{v}) = \frac{24}{5} = 4.8$$

- **Scalar Projection of $\mathbf{u}$ on $\mathbf{v}$:** 4.8
- **Scalar Projection of $\mathbf{v}$ on $\mathbf{u}$:** 4.8

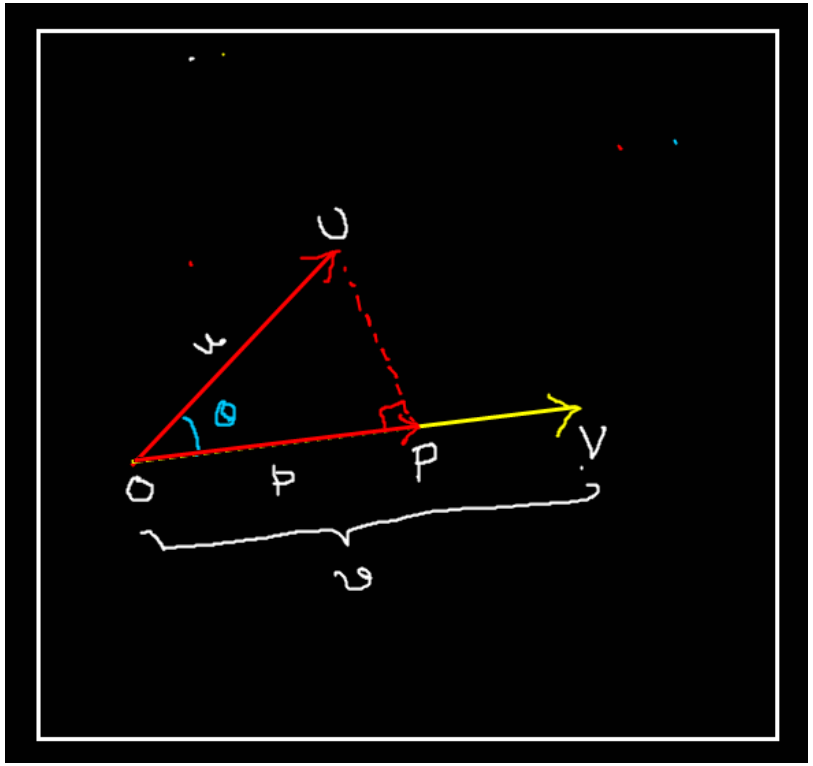
Projection of vectors

Geometric Interpretation

The scalar projection of $\mathbf{u}$ on $\mathbf{v}$ (and vice versa) gives the length of the component of $\mathbf{u}$ that lies along $\mathbf{v}$. If you imagine $\mathbf{v}$ as a line, the scalar projection is the length of the "shadow" that $\mathbf{u}$ casts onto this line. Similarly, for the scalar projection of $\mathbf{v}$ on $\mathbf{u}$, it is the length of the "shadow" that $\mathbf{v}$ casts onto the line defined by $\mathbf{u}$.



Scalar Projection



length of OP is scalar projection of u on v

$$\cos\theta = \frac{OP}{OU}$$

$$\therefore \quad OP = OU\cos\theta$$

$$\downarrow \quad = \|u\|\cos\theta$$

projection of u on v

but $u \cdot v = \|u\|\,\|v\|\cos\theta$

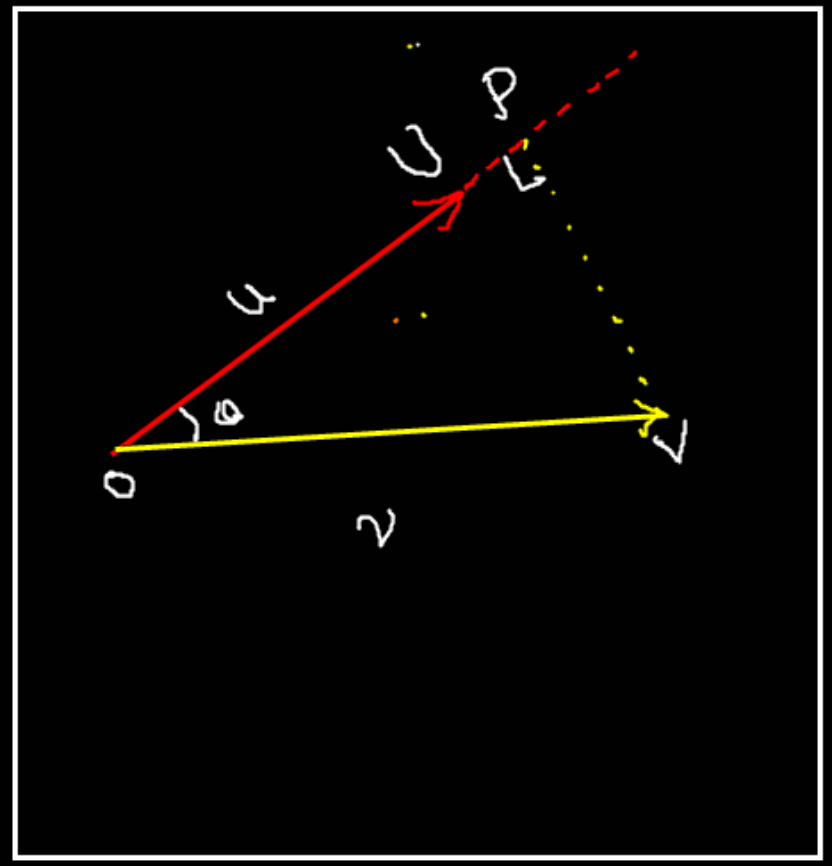
$$u \cdot v = \underbrace{\|u\|\cos\theta}\,\|v\|$$

$$u \cdot v = OP\,\|v\|$$

$$OP = \frac{u \cdot v}{\|v\|}$$

scalar projection of u on v $= \dfrac{u \cdot v}{\|v\|_2}$

Scalar Projection



length of OP is projection of v on u

$$\cos\theta = \frac{OP}{OV}$$

$$OP = OV\cos\theta$$

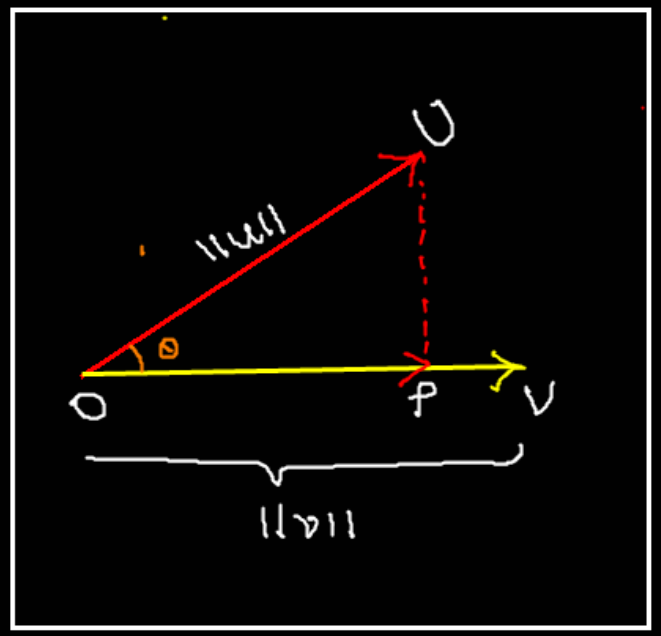
$$= \|v\|\cos\theta$$

but $u \cdot v = \|u\|\,\|v\|\cos\theta$

$$u \cdot v = \|u\|\,OP$$

$$\therefore \quad OP = \frac{u \cdot v}{\|u\|}$$

Vector Projection



Vector OP is vector projection of u on v

length of vector OP = Scalar projection of u on v

$$= \frac{u \cdot v}{\|v\|} \quad \text{— (1)}$$

Direction of vector OP = unit vector along direction of v

$$= \frac{v}{\|v\|} \quad \text{— (2)}$$

$\therefore$ Vector projection of u on v = (1) × (2)

$$= \frac{u \cdot v}{\|v\|} \cdot \frac{v}{\|v\|}$$

$$= \frac{u \cdot v}{\|v\|^2} \cdot v$$

Example of Vector projection

$$\mathbf{u} = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

Vector Projection of $\mathbf{u}$ on $\mathbf{v}$

The vector projection of $\mathbf{u}$ on $\mathbf{v}$ is given by:

$$\text{proj}_{\mathbf{v}}\mathbf{u} = \left(\frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}}\right)\mathbf{v}$$

Step-by-Step Calculation:

1. Calculate the dot product $\mathbf{u} \cdot \mathbf{v}$:
   $\mathbf{u} \cdot \mathbf{v} = 2 \cdot 1 + 3 \cdot 0 + 4 \cdot 1 = 2 + 0 + 4 = 6$
2. Calculate the dot product $\mathbf{v} \cdot \mathbf{v}$:
   $\mathbf{v} \cdot \mathbf{v} = 1 \cdot 1 + 0 \cdot 0 + 1 \cdot 1 = 1 + 0 + 1 = 2$
3. Compute the scalar factor:
   $\frac{\mathbf{u}\cdot\mathbf{v}}{\mathbf{v}\cdot\mathbf{v}} = \frac{6}{2} = 3$
4. Multiply the scalar factor by $\mathbf{v}$:
   $$\text{proj}_{\mathbf{v}}\mathbf{u} = 3 \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \\ 3 \end{bmatrix}$$

Vector Projection of $\mathbf{v}$ on $\mathbf{u}$

The vector projection of $\mathbf{v}$ on $\mathbf{u}$ is given by:

$$\text{proj}_{\mathbf{u}}\mathbf{v} = \left(\frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}}\right)\mathbf{u}$$

Step-by-Step Calculation:

1. Calculate the dot product $\mathbf{v} \cdot \mathbf{u}$:
   $\mathbf{v} \cdot \mathbf{u} = 1 \cdot 2 + 0 \cdot 3 + 1 \cdot 4 = 2 + 0 + 4 = 6$
2. Calculate the dot product $\mathbf{u} \cdot \mathbf{u}$:
   $\mathbf{u} \cdot \mathbf{u} = 2 \cdot 2 + 3 \cdot 3 + 4 \cdot 4 = 4 + 9 + 16 = 29$
3. Compute the scalar factor:
   $\frac{\mathbf{v}\cdot\mathbf{u}}{\mathbf{u}\cdot\mathbf{u}} = \frac{6}{29} \approx 0.2069$
4. Multiply the scalar factor by $\mathbf{u}$:
   $$\text{proj}_{\mathbf{u}}\mathbf{v} = 0.2069 \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \approx \begin{bmatrix} 0.4138 \\ 0.6207 \\ 0.8276 \end{bmatrix}$$

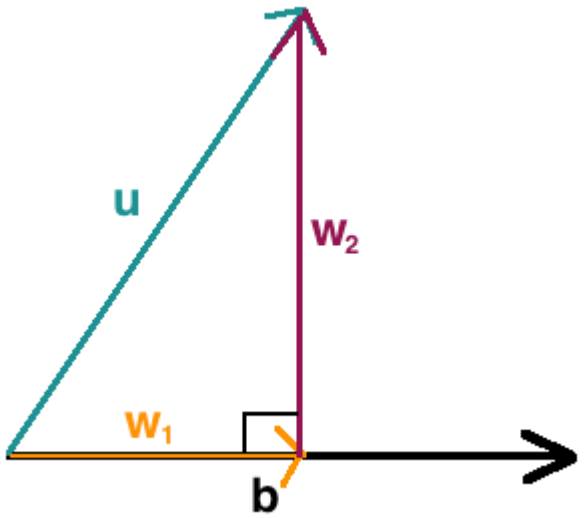
Orthogonal Projection

Orthogonal projection is a process of projecting a vector onto another vector (or a subspace) such that the resulting vector (projection) is the closest point in the subspace to the original vector. This projection is perpendicular (orthogonal) to the subspace, minimizing the distance between the original vector and its projection.

Geometric Interpretation

Imagine a vector $\mathbf{a}$ and a subspace (typically represented by another vector $\mathbf{b}$). The orthogonal projection of $\mathbf{a}$ onto $\mathbf{b}$ is the vector $\mathbf{p}$ in the direction of $\mathbf{b}$ such that the vector $\mathbf{a} - \mathbf{p}$ is orthogonal to $\mathbf{b}$.

Orthogonal Projection



Definition: If $\vec{u} = \overrightarrow{w_1} + \overrightarrow{w_2}$, $\overrightarrow{w_1} \| \vec{b}$ and $\overrightarrow{w_1} \perp \overrightarrow{w_2}$, then $\overrightarrow{w_1}$ the **Orthogonal Projection of $\vec{u}$ Along $\vec{b}$** denoted $w_1 = \text{proj}_{\vec{b}}\vec{u} = \frac{(\vec{u}\cdot\vec{b})}{\|\vec{b}\|^2}\vec{b}$, and $\overrightarrow{w_2}$ is the **Vector Component of $\vec{u}$ Orthogonal to $\vec{b}$** and $\overrightarrow{w_2} = \vec{u} - \text{proj}_{\vec{b}}\vec{u}$.

```python
import numpy as np

# Define the vectors u and v
u = np.array([3, 4])
v = np.array([4, 3])

# Compute the dot product of u and v
dot_product_uv = np.dot(u, v)

# Compute the magnitudes of u and v
magnitude_u = np.linalg.norm(u)
magnitude_v = np.linalg.norm(v)

# Compute the scalar projection of u on v
scalar_projection_u_on_v = dot_product_uv / magnitude_v

# Compute the scalar projection of v on u
scalar_projection_v_on_u = dot_product_uv / magnitude_u

# Compute the vector projection of u on v
vector_projection_u_on_v = (dot_product_uv / (magnitude_v**2)) * v

# Compute the vector projection of v on u
vector_projection_v_on_u = (dot_product_uv / (magnitude_u**2)) * u

# Print the results
print(f"Scalar projection of u on v: {scalar_projection_u_on_v}")
print(f"Scalar projection of v on u: {scalar_projection_v_on_u}")
print(f"Vector projection of u on v: {vector_projection_u_on_v}")
print(f"Vector projection of v on u: {vector_projection_v_on_u}")
```

```
Scalar projection of u on v: 4.8
Scalar projection of v on u: 4.8
Vector projection of u on v: [3.84 2.88]
Vector projection of v on u: [2.88 3.84]
```

Linear Combination

Linear Combinations of Vectors

A linear combination of vectors involves multiplying each vector by a scalar and then adding the results. Formally, given vectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ and scalars $c_1, c_2, \ldots, c_n$, the linear combination of these vectors is given by:

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 + \cdots + c_n\mathbf{v}_n$$

Example

Consider the vectors:

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$$

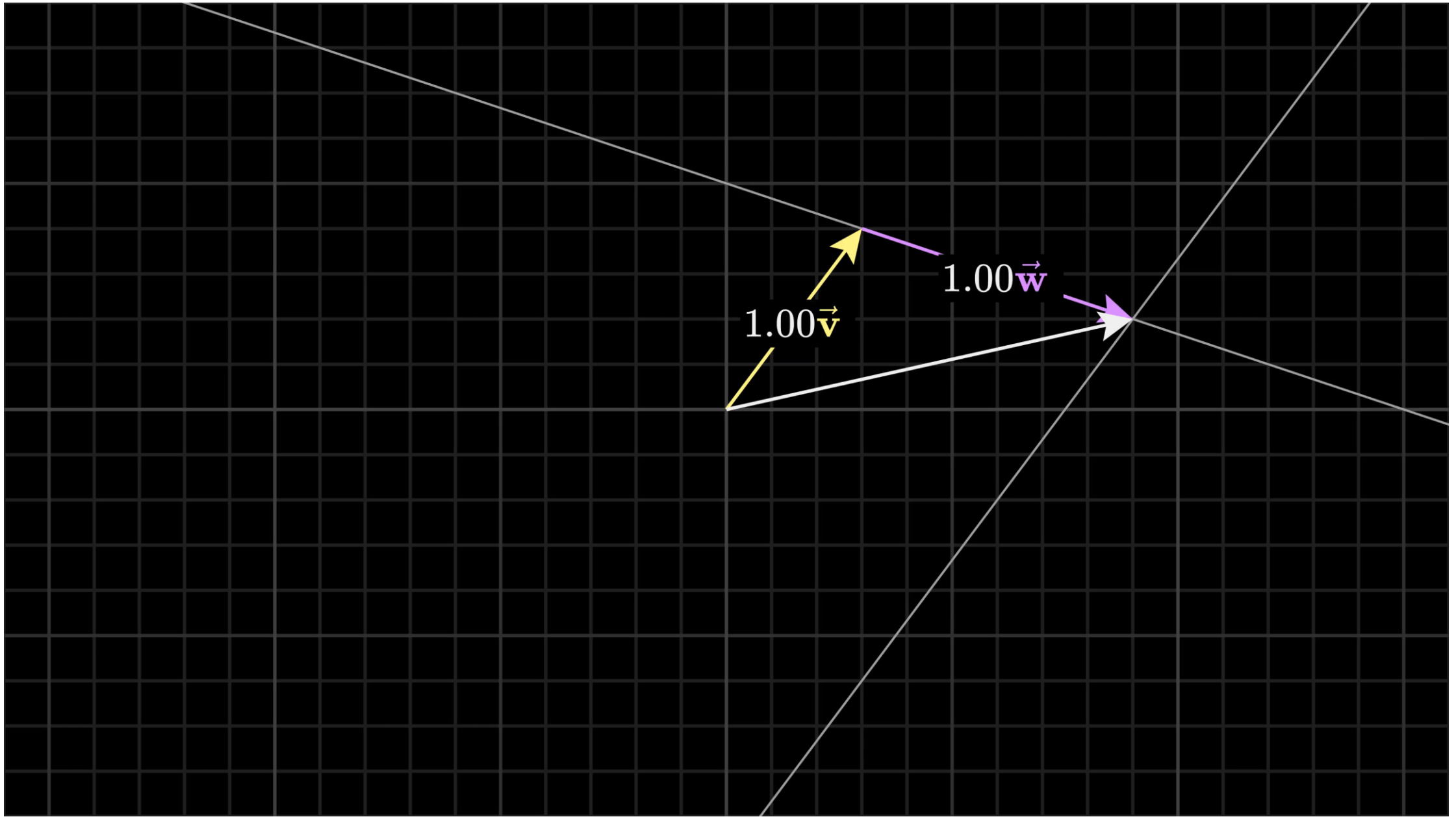
And the scalars:

$$c_1 = 2, \quad c_2 = 3$$

The linear combination of $\mathbf{v}_1$ and $\mathbf{v}_2$ is:

$$\mathbf{v} = c_1\mathbf{v}_1 + c_2\mathbf{v}_2 = 2\begin{bmatrix} 1 \\ 2 \end{bmatrix} + 3\begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 2 \\ 4 \end{bmatrix} + \begin{bmatrix} 9 \\ 12 \end{bmatrix} = \begin{bmatrix} 2+9 \\ 4+12 \end{bmatrix} = \begin{bmatrix} 11 \\ 16 \end{bmatrix}$$

Linear combination



Linear Combination-Applications

1. **Feature Engineering**:

   - **Description**: In machine learning, linear combinations of features can create new features that capture important relationships in the data. For instance, combining different sensor readings in IoT data can create a composite feature that better represents an underlying physical phenomenon.
   - **Example**: In a dataset with features x_1 and x_2, a new feature $x_3 = 0.5x_1 + 0.5x_2$ might be created to capture a combined effect.

2. **Principal Component Analysis (PCA)**:

   - **Description**: PCA is a dimensionality reduction technique that transforms the original features into a new set of features (principal components) that are linear combinations of the original features. These new features capture the most variance in the data with fewer dimensions.
   - **Example**: In a high-dimensional dataset, PCA can be used to reduce the number of features while retaining most of the variability in the data. The principal components are linear combinations of the original features.

Linear Combination-Applications

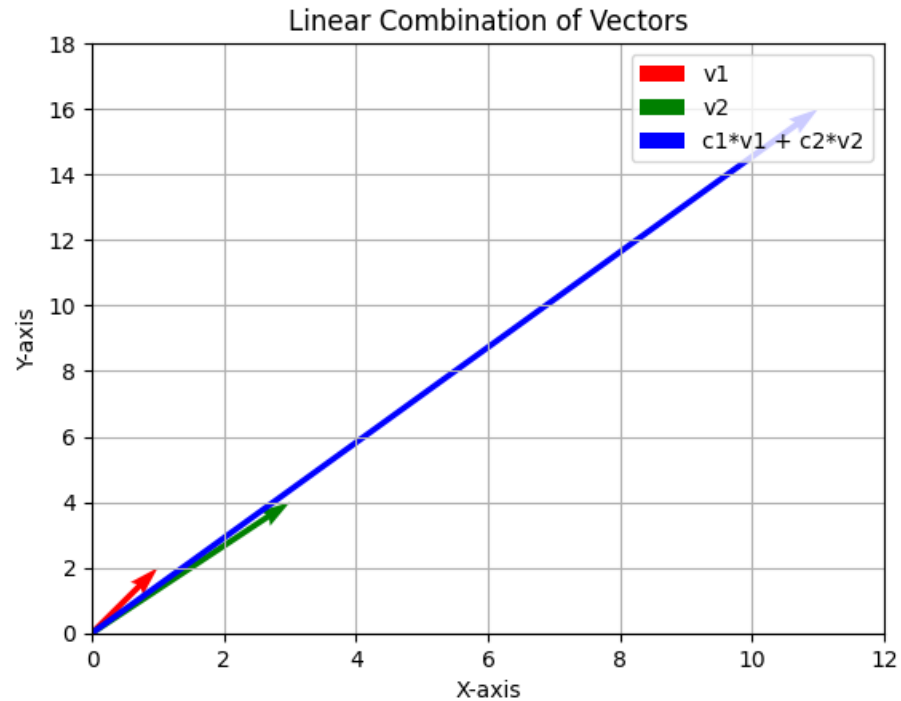
3. **Linear Regression:**

   - **Description:** Linear regression models predict a target variable as a linear combination of input features. The model parameters (weights) are learned from the data to minimize the prediction error.
   - **Example:** Given features x_1, x_2, and x_3, a linear regression model might predict the target variable y using the equation $y = w_1x_1 + w_2x_2 + w_3x_3 + b$, where w_1, w_2, w_3 are the weights and b is the bias term.

```python
import numpy as np
import matplotlib.pyplot as plt

# Define the vectors
v1 = np.array([1, 2])
v2 = np.array([3, 4])

# Define the scalars
c1 = 2
c2 = 3

# Compute the linear combination
v = c1 * v1 + c2 * v2

# Print the result
print("Vector v1:", v1)
print("Vector v2:", v2)
print("Scalar c1:", c1)
print("Scalar c2:", c2)
print("Linear combination:", v)
```

```python
# Plotting the vectors
plt.figure()
plt.quiver(0, 0, v1[0], v1[1], angles='xy', scale_units='xy', scale=1, color='r', label='v1')
plt.quiver(0, 0, v2[0], v2[1], angles='xy', scale_units='xy', scale=1, color='g', label='v2')
plt.quiver(0, 0, v[0], v[1], angles='xy', scale_units='xy', scale=1, color='b', label='c1*v1 + c2*v2')

# Set the limits of the plot
plt.xlim(0, 12)
plt.ylim(0, 18)

# Add grid, legend, and labels
plt.grid()
plt.legend()
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Linear Combination of Vectors')

# Show the plot
plt.show()
```

```
Vector v1: [1 2]
Vector v2: [3 4]
Scalar c1: 2
Scalar c2: 3
Linear combination: [11 16]
```



a = 0.85

b = 0.17

c = 0.56

red: $\vec{u} = (255, 0, 0)$

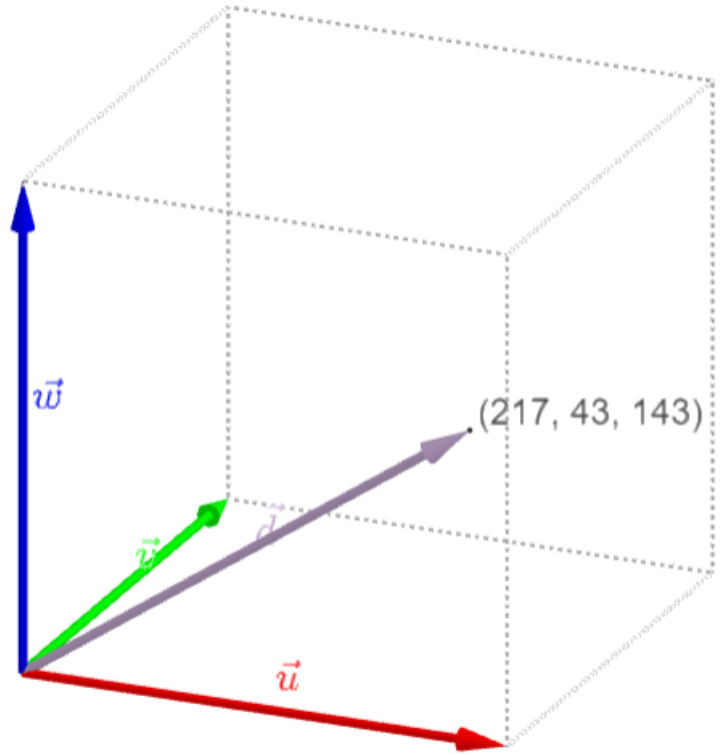
green: $\vec{v} = (0, 255, 0)$

blue: $\vec{w} = (0, 0, 255)$

Their linear combination is vector $\vec{d}$.

$\vec{d} = a\vec{u} + b\vec{v} + c\vec{w}$

The integer components of $\vec{d}$ are:
(217, 43, 143)



https://www.geogebra.org/m/Dq2A7aRv

https://colab.research.google.com/drive/1IRr_MxGrijuoSwfK0I597LA5RI0nIOjZ#scrollTo=QMK4SYJr0r5T

Libraries

1. NumPy

Description:
NumPy is one of the most fundamental packages for scientific computing in Python. It provides support for arrays, matrices, and many mathematical functions to operate on these data structures.

Key Features:

- Support for multi-dimensional arrays and matrices.
- Mathematical functions for linear algebra, Fourier transforms, and random number generation.
- Efficient numerical computation with a syntax similar to MATLAB.

Example Functions:

- `` `numpy.dot()` ``: Computes the dot product of two arrays.
- `` `numpy.linalg.norm()` ``: Computes the norm of a vector.
- `` `numpy.linalg.inv()` ``: Computes the inverse of a matrix.

Libraries

2. SciPy

Description:

SciPy builds on NumPy and provides additional functionalities for scientific and technical computing. It includes modules for optimization, integration, interpolation, eigenvalue problems, algebraic equations, and more.

Key Features:

- Advanced linear algebra operations.
- Integration and ordinary differential equation (ODE) solvers.
- Optimization and root-finding algorithms.

Example Functions:

- `` `scipy.linalg.solve()` ``: Solves a system of linear equations.
- `` `scipy.integrate.quad()` ``: Computes the definite integral of a function.
- `` `scipy.optimize.minimize()` ``: Minimizes a function using various algorithms.

Libraries

3. SymPy

Description:

SymPy is a Python library for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS) while keeping the code as simple as possible to understand and extend.

Key Features:

- Symbolic computation capabilities.
- Simplification of algebraic expressions.
- Solving equations and inequalities.
- Differentiation and integration of symbolic expressions.

Example Functions:

- `` `sympy.diff()` ``: Differentiates an expression.
- `` `sympy.integrate()` ``: Integrates an expression.
- `` `sympy.solve()` ``: Solves an equation.