# Artificial Intelligence

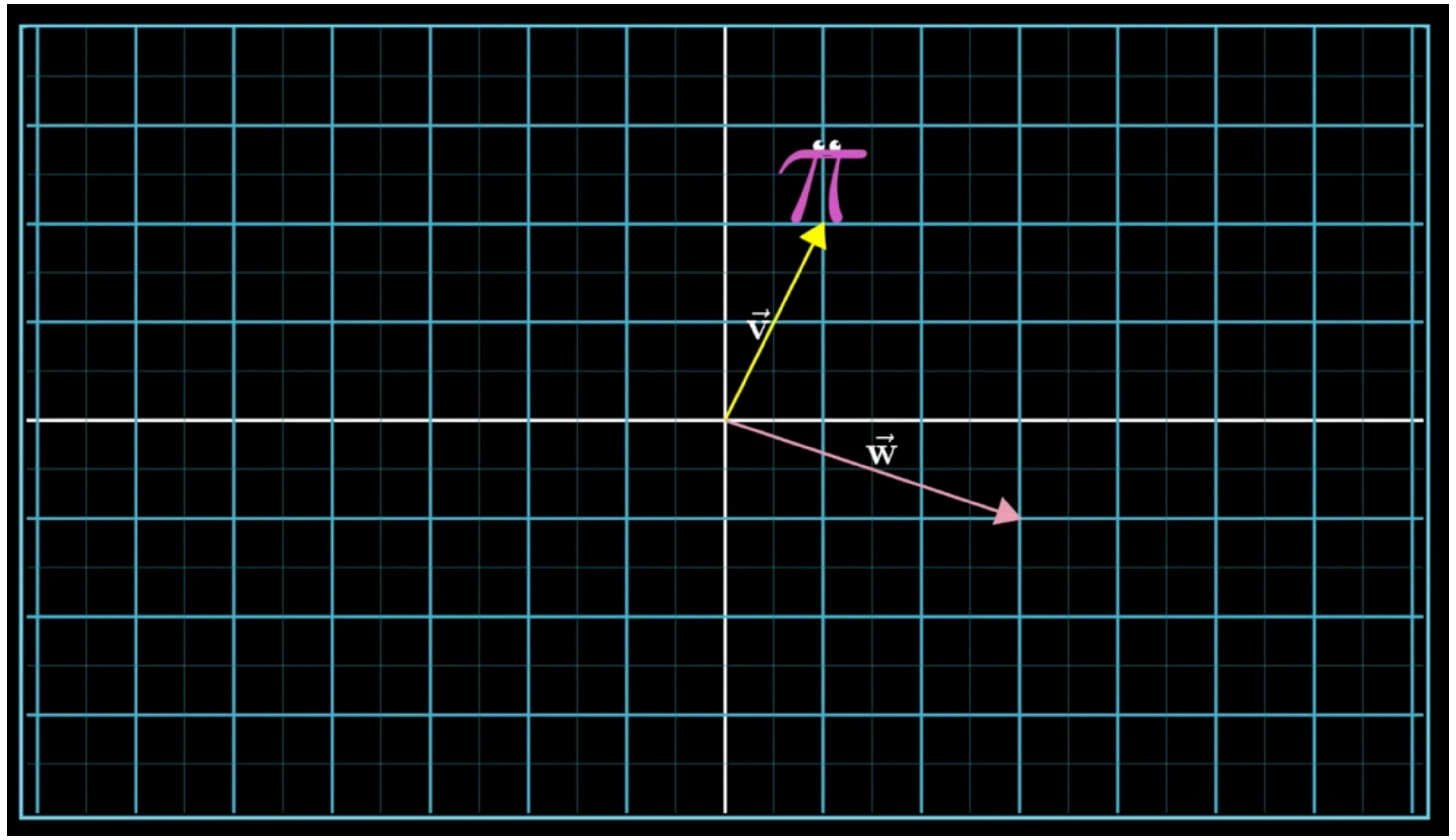
## 向量，矩阵与Numpy

**Zhipeng Ye and 2021.04.08**

# 大纲

## 向量，矩阵 与 Numpy

- 向量
  - 定义
  - 向量运算
    - 加法，减法，数乘，内积，叉积，外积（张量积）
- 矩阵
  - 定义
  - 矩阵运算
    - 加法，减法，与数相乘，矩阵相乘
- Numpy
  - Numpy 实现基本线性代数运算



# 向量

## 向量的定义

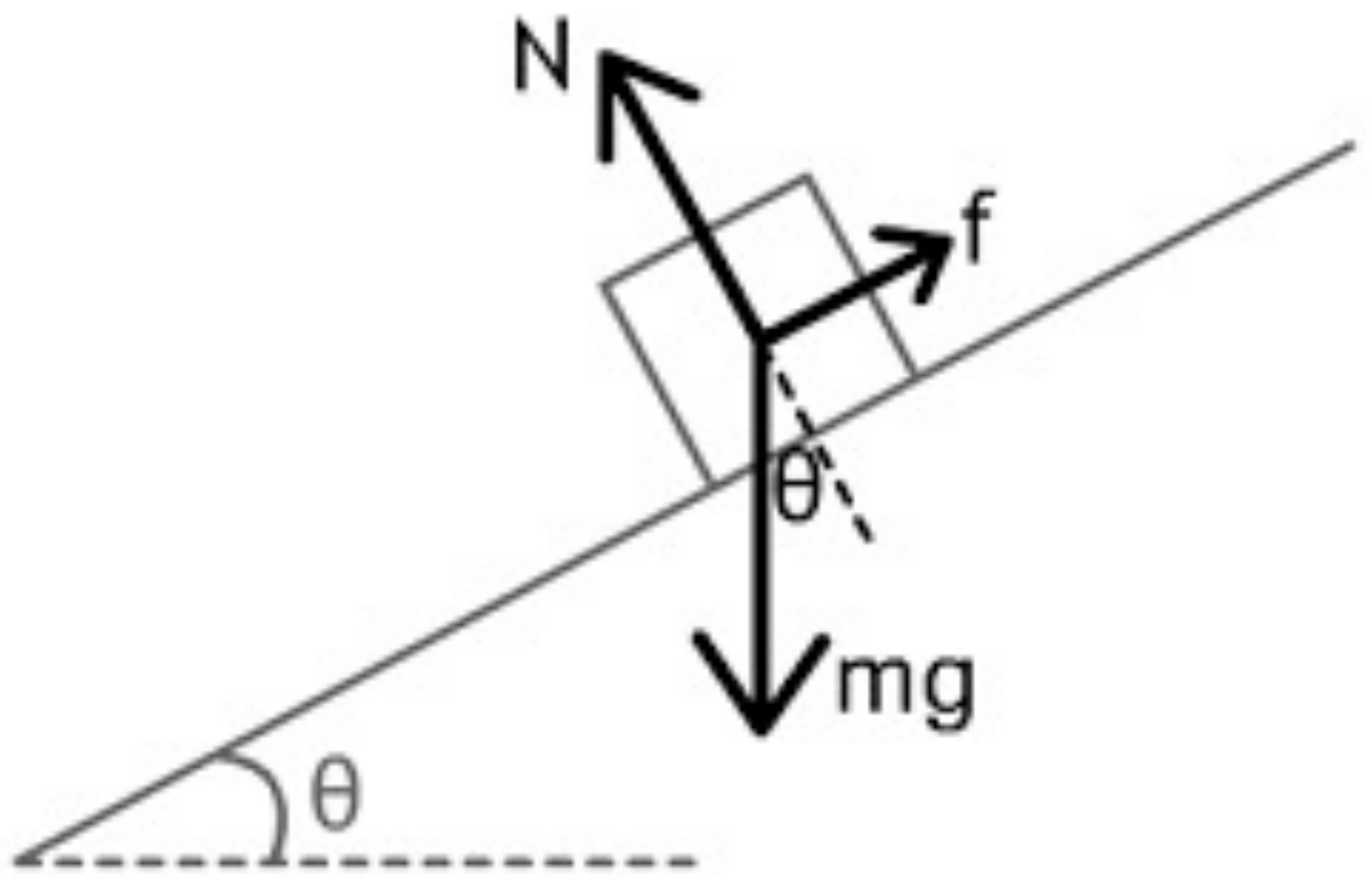
- 向量是具有大小和方向的量，又称矢量

，只有大小的量，称为标量

- 物理里面最常见的矢量就是力

$$F = m\,\vec{a}$$

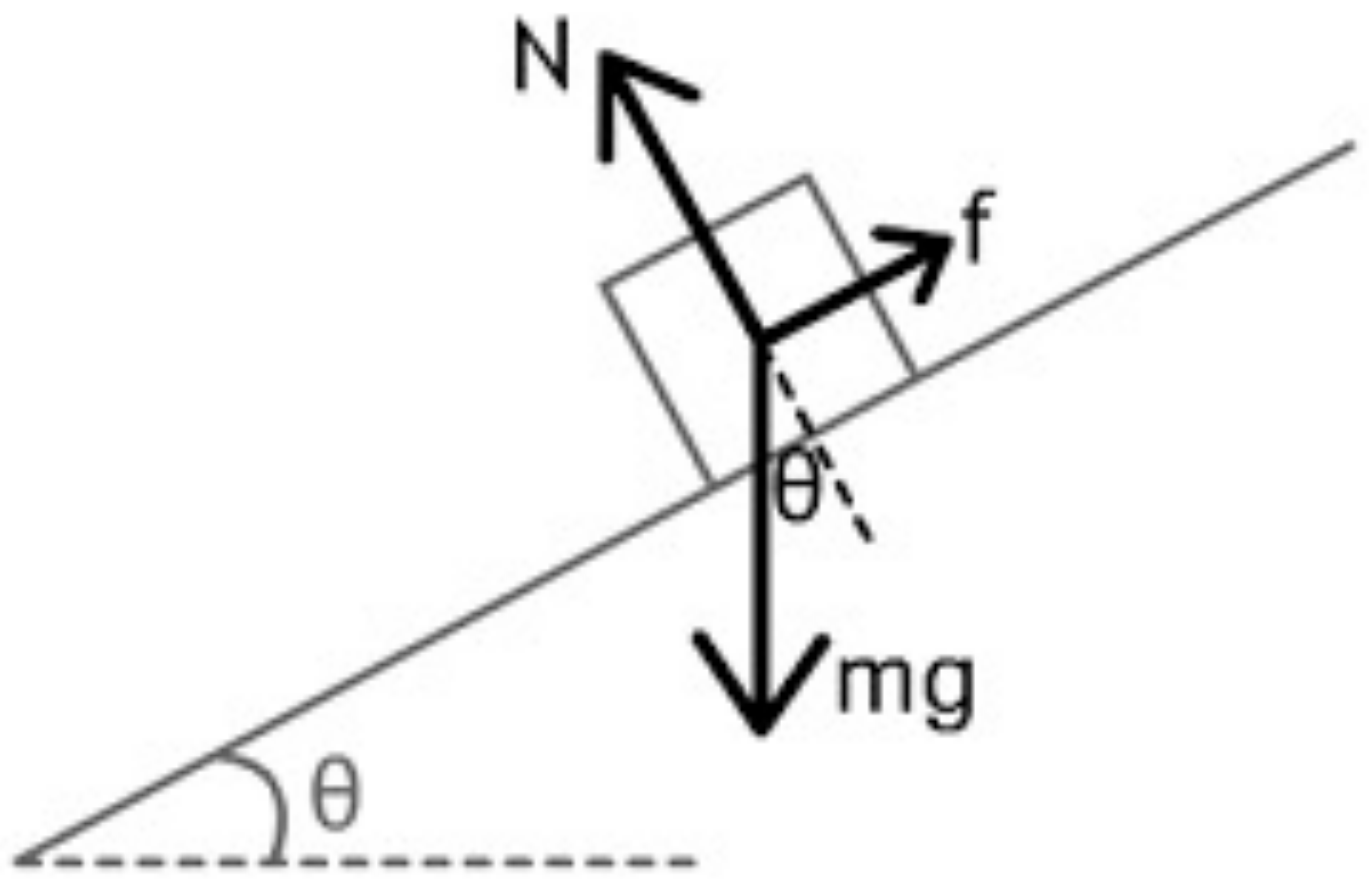
- 想一下物理里面，还有哪些量是矢量？



# 向量

## 向量的定义

- 想一下物理里面，还有哪些量是矢量？
- 力矩 $M = r * F$
- 位移 r
- 速度 $v$
- 加速度 a
- 动量 p $= mv$
- ……



# 向量

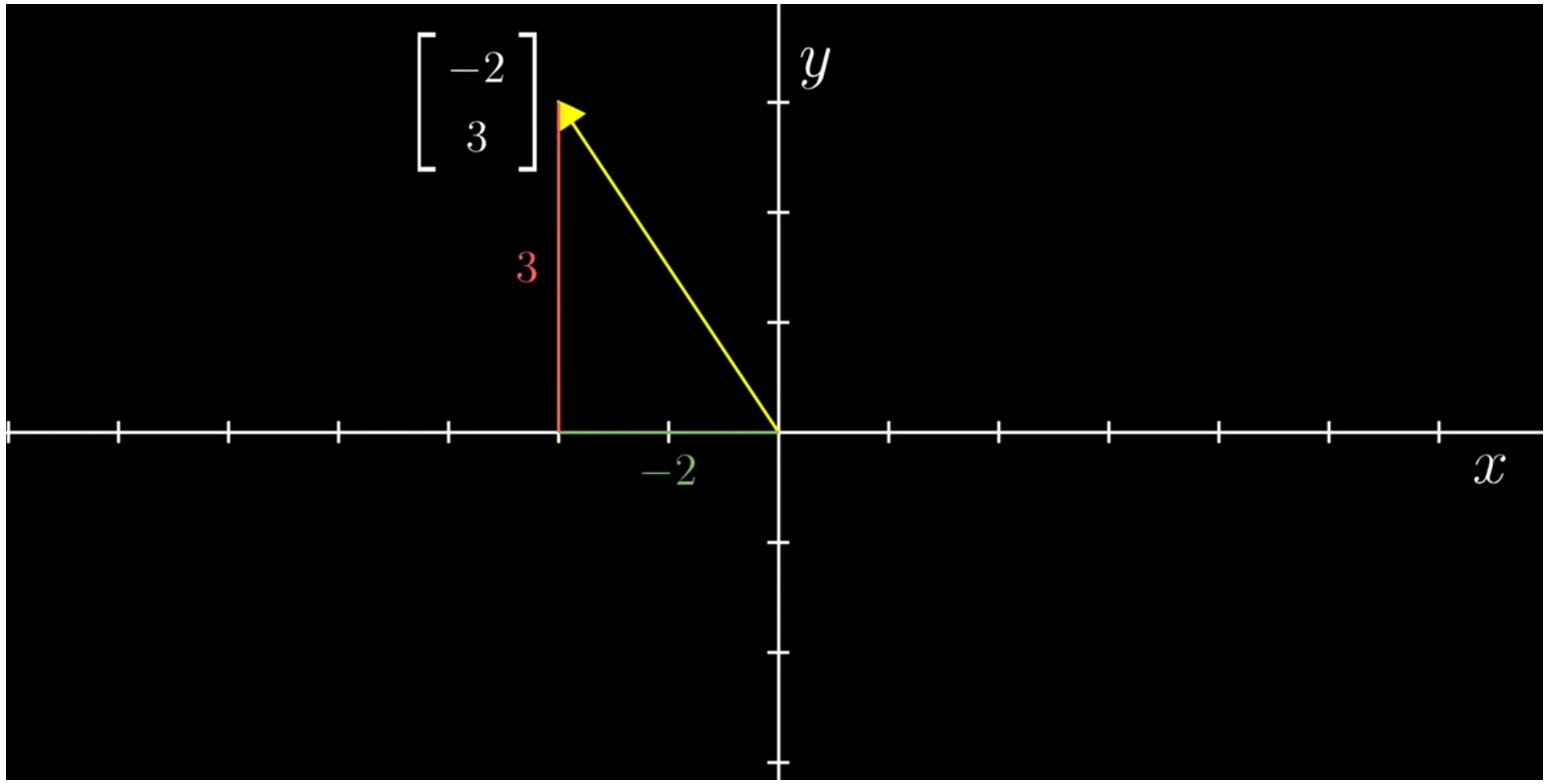
## 向量的定义

- 如何定量的用数学方式表达向量的大小与方向呢？
- 将向量进行正交分解！
- 分解成不同标准正交向量上的投影。

$$\vec{x} = \lambda_1 \overrightarrow{e_1} + \lambda_2 \overrightarrow{e_2}$$

- 因为都是在固定的坐标系下定量表示，$\vec{x}$ 就可以用数组的方式来表示：

$$\vec{x} = \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix}$$

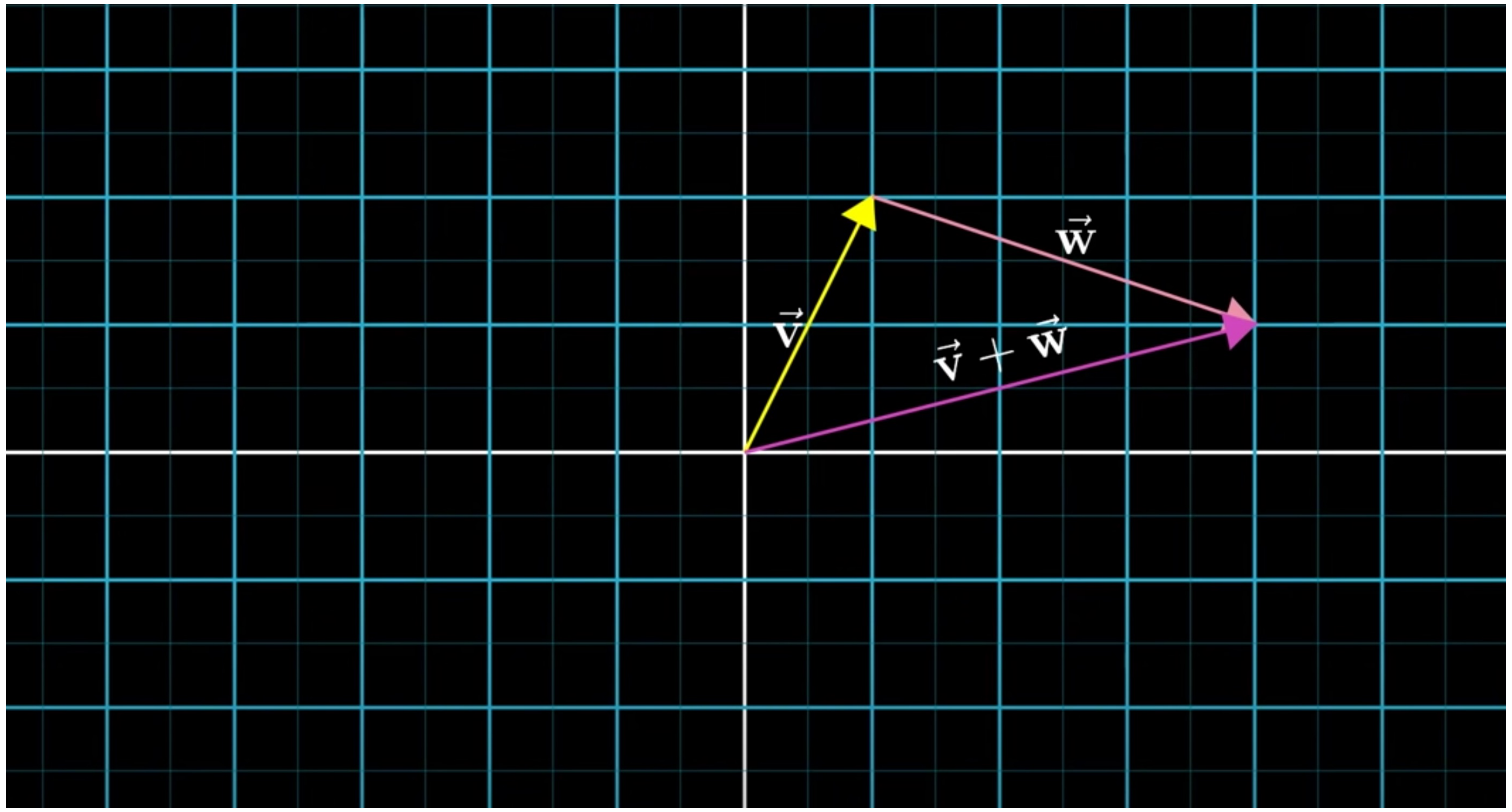


# 向量

## 向量的运算-加法

- $\vec{v} = \binom{1}{2}, \vec{w} = \binom{3}{-1}$
- 几何方式：
  - 三角形法则
- 代数方式：

$$\vec{v} + \vec{w} = \begin{pmatrix} 1+3 \\ 2-1 \end{pmatrix} = \begin{pmatrix} 4 \\ 1 \end{pmatrix}$$

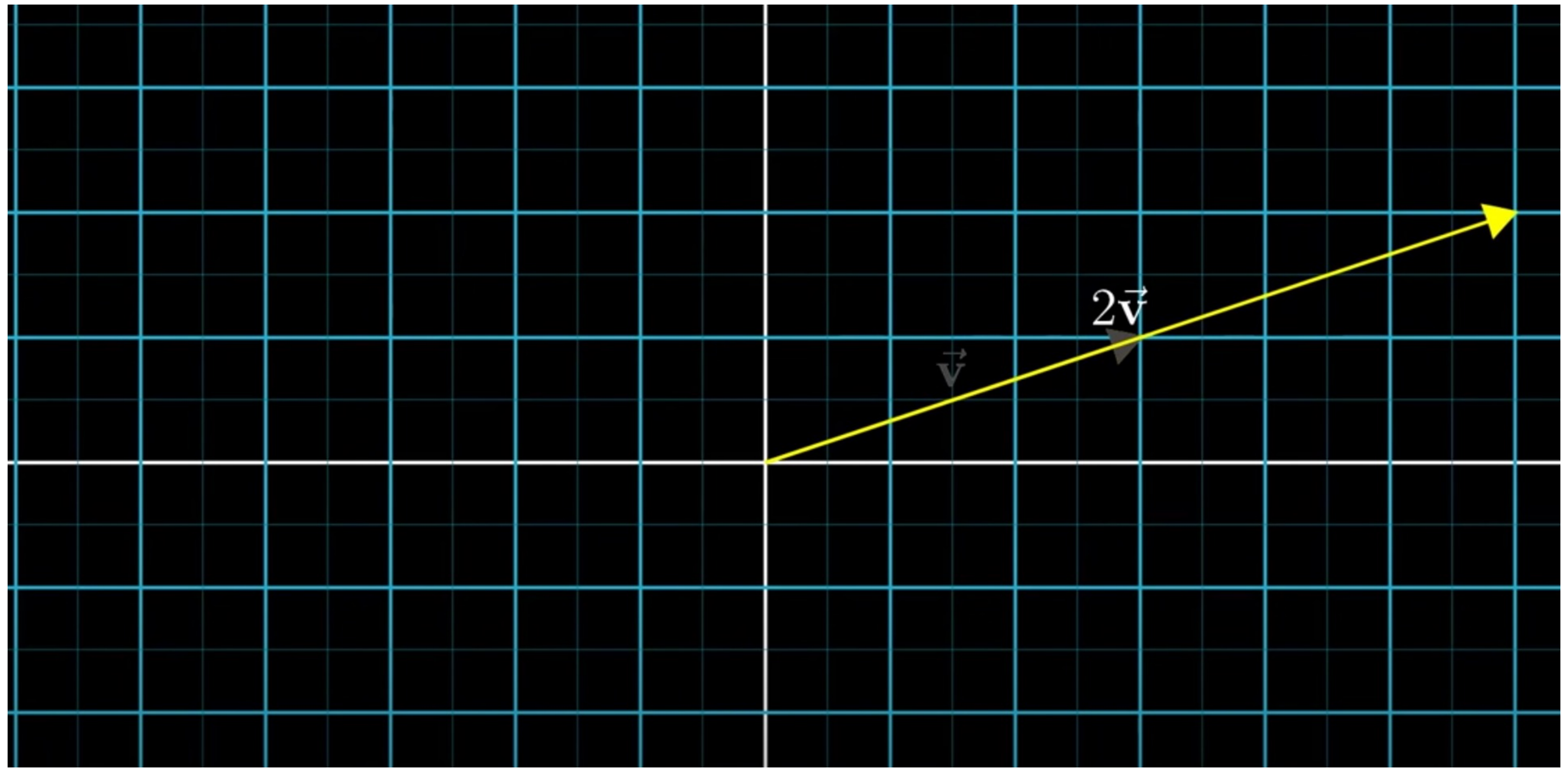


# 向量

## 向量的运算-数乘

- $\vec{\mathrm{v}} * \lambda = ?$
- 几何方式：
  - 拉伸
- 代数方式：

$$\vec{\mathrm{v}} * \lambda = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} * \lambda = \begin{pmatrix} \lambda x_1 \\ \lambda x_2 \end{pmatrix}$$



# 向量

## 向量的运算-减法

- $\vec{\mathrm{v}} = \binom{1}{2}$ , $\overrightarrow{W} = \binom{3}{-1}$.
- 求 $\vec{\mathrm{v}} - \vec{w}$

$$\vec{v} - \vec{w} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} + \begin{pmatrix} -3 \\ 1 \end{pmatrix} = \begin{pmatrix} -2 \\ 3 \end{pmatrix}$$

- 几何上？

# 向量

## 向量的运算-内积的定义

- $\vec{\mathrm{v}} = \binom{1}{2}$ , $\overrightarrow{W} = \binom{2}{1}$.
- $< \vec{\mathrm{v}}, \vec{w} >= \vec{v} \cdot \vec{w} = 1 * 2 + 2 * 1 = 4$
- 几何上：
- $\vec{\mathrm{v}} \cdot \vec{w} = |\vec{v}| * |\vec{w}| * cos\theta$
- Why?

# 向量

## 向量的运算-模长（大小）

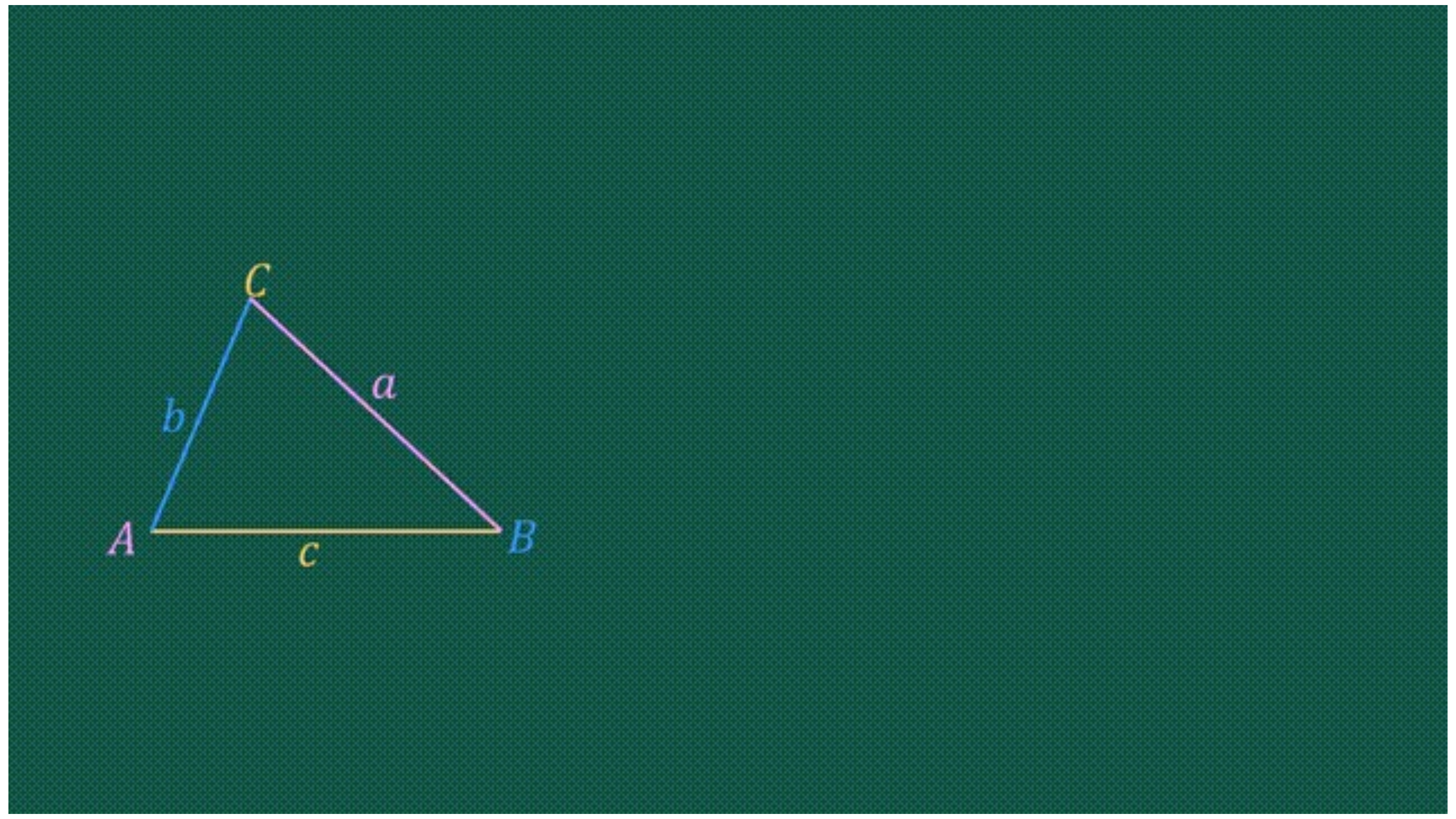
- $\vec{\mathrm{v}} = \binom{1}{2}$
- $|\vec{v}| = \sqrt{1^2 + 2^2} = \sqrt{5}$
- 对于一般向量 $\vec{\mathrm{a}} = \binom{x_1}{x_2}$,大小为
- $|\vec{a}| = \sqrt{x_1^2 + x_2^2}$
- 所以向量的模长就是指向量对应的标量

  大小
- [范数阅读]

# 向量

## 向量的运算-内积的定义

- $\vec{v} \cdot \vec{w} = |\vec{v}| * |\vec{w}| * cos\theta$
- 证明余弦定理：$a^2 = b^2 + c^2 - 2bc \ * cosA$




$$a^2 = b^2 + c^2 - 2bc \ * cosA$$

$$|\vec{a}|^2 = \left|\vec{b}\right|^2 + |\vec{c}|^2 - 2\left|\vec{b}\right||\vec{c}|cosA$$

$$2\left|\vec{b}\right||\vec{c}|cosA = \left|\vec{b}\right|^2 + |\vec{c}|^2 - |\vec{a}|^2$$

$$2\left|\vec{b}\right||\vec{c}|cosA = \left|\vec{b}\right|^2 + |\vec{c}|^2 - \left|\vec{b} - \vec{c}\right|^2$$

$$\vec{b} \ \cdot \vec{c} = \left|\vec{b}\right||\vec{c}|cosA$$

*即证：*

$$\vec{v} \cdot \vec{w} = |\vec{v}| * |\vec{w}| * cos\theta$$

# 向量

## 向量的运算-内积的几何意义与应用

- 几何意义：一个向量在另一个向量投影的积
- 应用：

1. 计算有用功
2. 计算向量夹角
3. 计算数据相关性
4. 人脸识别等
5. 。。。

$$a^2 = b^2 + c^2 - 2bc \ * cosA$$

$$|\vec{a}|^2 = \left|\vec{b}\right|^2 + |\vec{c}|^2 - 2\left|\vec{b}\right||\vec{c}|cosA$$

$$2\left|\vec{b}\right||\vec{c}|cosA = \left|\vec{b}\right|^2 + |\vec{c}|^2 - |\vec{a}|^2$$

$$2\left|\vec{b}\right||\vec{c}|cosA = \left|\vec{b}\right|^2 + |\vec{c}|^2 - \left|\vec{b} - \vec{c}\right|^2$$

$$\vec{b} \ \cdot \vec{c} = \left|\vec{b}\right||\vec{c}|cosA$$

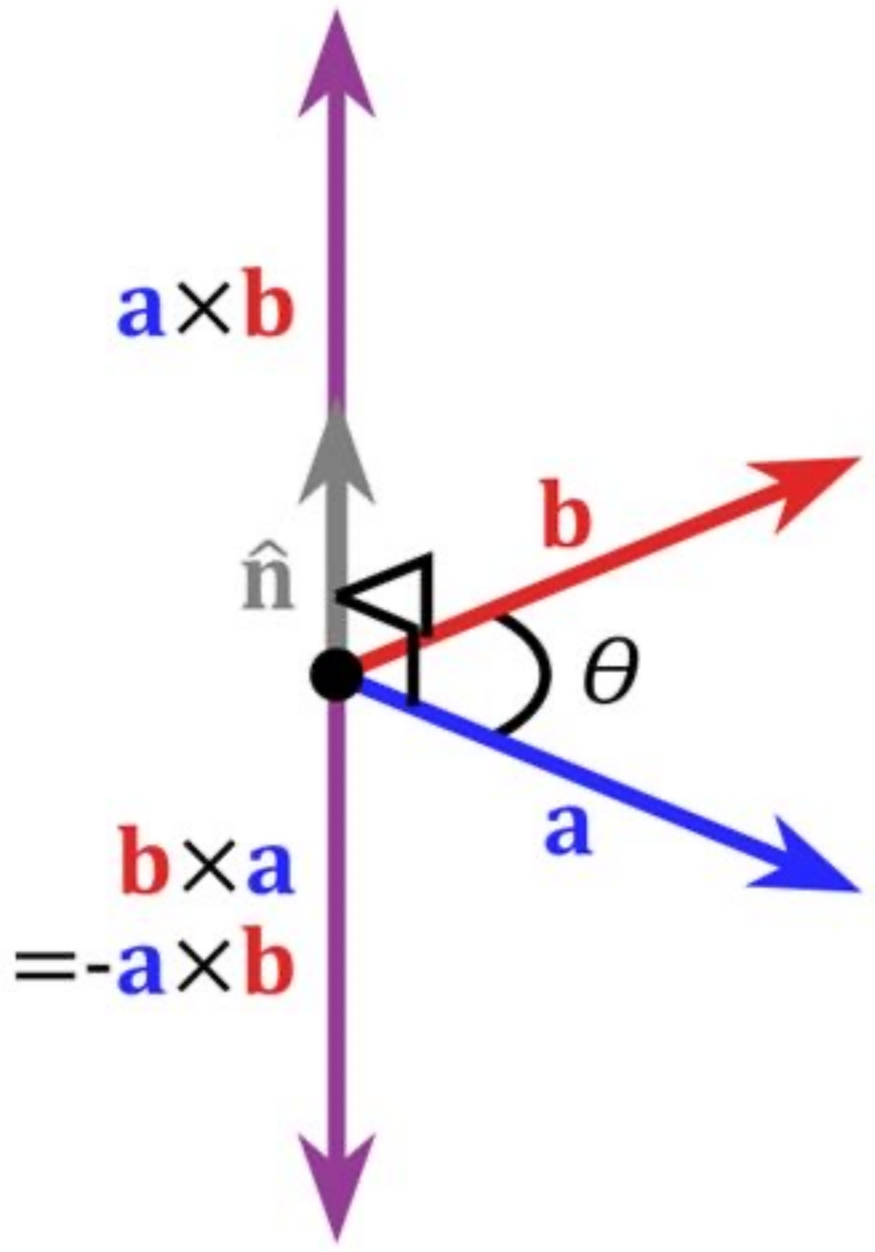
*即证：*

$$\vec{\mathrm{v}} \cdot \vec{w} = |\vec{v}| * |\vec{w}| * cos\theta$$

# 向量

## 向量的运算-叉积的定义

- 两个向量叉积的结果是垂直于两个向量组成平面的法向量
- 大小 $|a \times b| = |a||b|sin\theta$ 等于 a,b向量围成平行四边形的面积
- 方向 根据右手螺旋定则



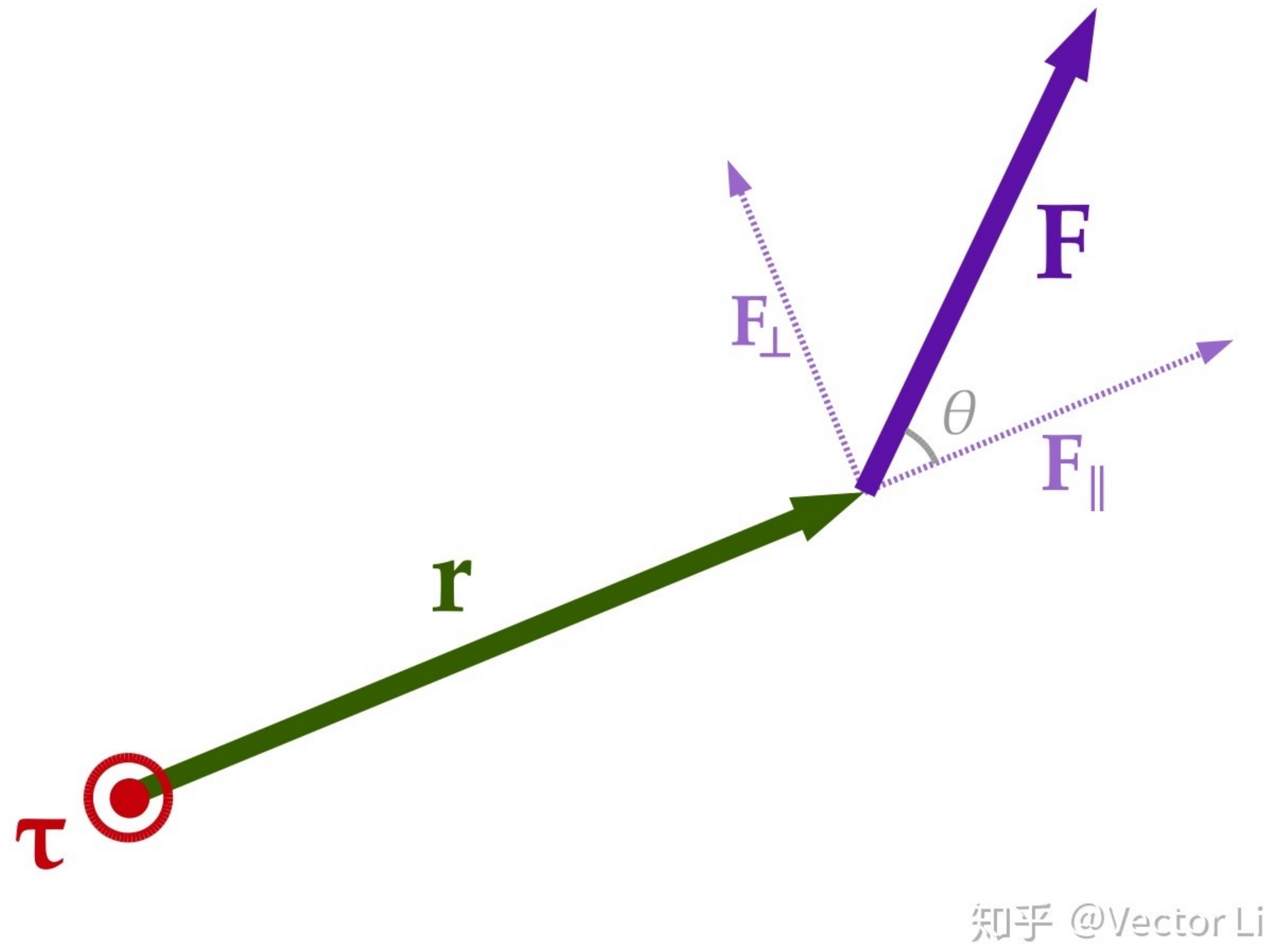
# 向量

## 向量的运算-叉积的坐标表示

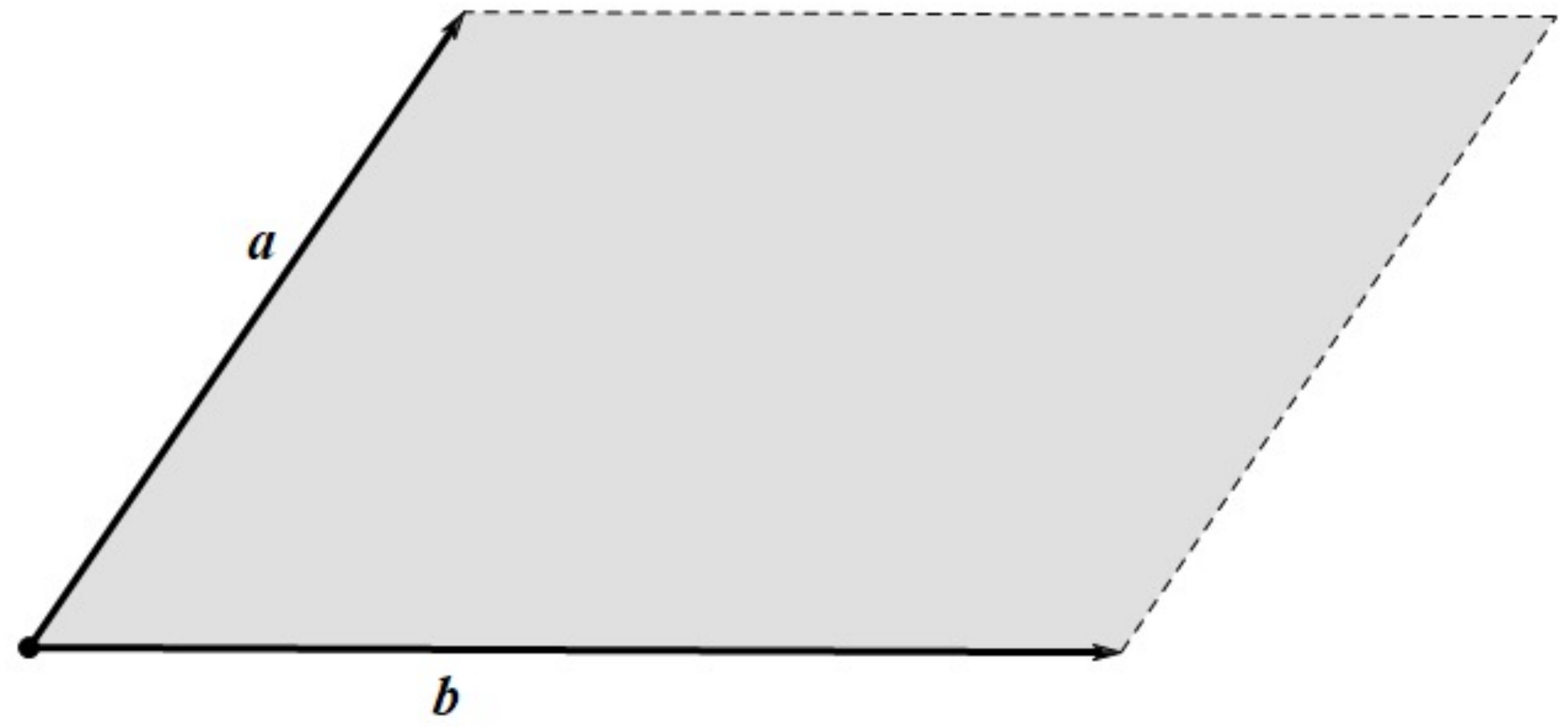
- $a = \{a_x, a_y, a_z\}, b = \{b_x, b_y, b_z\}$
- $a \times b = (a_x i + a_y j + a_z k) \times (b_x i + b_y j + b_z k) = a_x b_x (i \times i) + a_x b_y (i \times j) + a_x b_z (i \times k) + a_y b_x (j \times i) + a_y b_y (j \times j) + a_y b_z (j \times k) + a_z b_x (k \times i) + a_z b_y (k \times j) + a_z b_z (k \times k)$
- 因为$i \times i = j \times j = k \times k = 0$，$i \times j = k, j \times k = i, k \times i = j, j \times i = -k, k \times j = -i, i \times k = -j$
- $a \times b = (a_y b_z - a_z b_y) i + (a_z b_x - a_x b_z) j + (a_x b_y - a_y b_x) k$
- $\mathrm{a} \times b = \begin{vmatrix} i & j & k \\ a_x & a_y & a_z \\ b_x & b_y & b_z \end{vmatrix}$

# 向量

## 向量的运算-叉积的应用




算力矩




求向量围成的图形面积

# 向量

## 向量的运算-外积（张量积）的定义

- $a = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}, b = \begin{bmatrix} b_x & b_y & b_z \end{bmatrix}.$

- $a \otimes b = \begin{vmatrix} a_x b_x & a_x b_y & a_x b_z \\ a_y b_x & a_y b_y & a_y b_z \\ a_z b_x & a_z b_y & a_z b_z \end{vmatrix}$

# 向量

## 向量的运算-总结

- 向量的内积是降维操作，向量的叉积是向量

，向量的外积是升维操作。

# 矩阵

## 矩阵的定义

- 矩阵是 $m*n$ 的二维数组，记作：
- $A=\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$

- 矩阵的应用场景：图像视频，数据处理（表格数据），图的矩阵表达，数据压缩，物理和工程都大量应用。

# 矩阵

## 矩阵间的运算-加法与减法

已知 $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$

- 矩阵加法：
- $\mathrm{A} + B = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$
- 矩阵减法：
- $A - B = \begin{bmatrix} a_{11} - b_{11} & a_{12} - b_{12} \\ a_{21} - b_{21} & a_{22} - b_{22} \end{bmatrix}$

# 矩阵

## 矩阵的乘法

已知 $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, b, $\vec{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$

- 矩阵与数相乘：
- $\mathrm{A} * \mathrm{b} = \begin{bmatrix} b * a_{11} & b * a_{12} \\ b * a_{21} & b * a_{22} \end{bmatrix}$
- 矩阵与向量相乘：
- $A * \vec{c} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} =$

$$\begin{bmatrix} a_{11} * c_1 + a_{12} * c_2 \\ a_{21} * c_1 + a_{22} * c_2 \end{bmatrix}$$

已知 A 是 $\mathrm{m} * n$ 的矩阵，$\vec{c}$ 是 $\mathrm{n} * 1$的向量。则

$\mathrm{A} * \vec{c}$ 是 $m * 1$的向量

物理意义：

矩阵对向量的乘法，是对向量进行的线性变换。

在线性空间中，可以分解成旋转与拉伸操作。

# 矩阵

## 矩阵的乘法

已知 $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$, $B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$.

则

$$\mathrm{A} * B = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} * \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

$$= \begin{bmatrix} \mathrm{a}_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

已知 A 是 $\mathrm{m} * n$ 的矩阵，$B$是 $n * \mathrm{k}$ 的矩阵，则结果是 $m * k$ 的 C 矩阵。

- 矩阵 A 是一种线性变换，矩阵 B 也是一种线性变换。则 $\mathrm{A} * B$ 得到的 C 矩阵是合成前两个线性变换的结果
- 注意：矩阵乘法并不满足交换律。

$$\mathrm{A} * B \neq B * A$$

# Numpy

## Numpy 是什么

- **NumPy** is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

# Numpy

## Numpy 是什么

- 如何用 Python 计算向量与矩阵的运算
- 任务：
  1. 实现向量的加分，减法，模长，内积，叉积，外积
  2. 实现矩阵的加法，减法，乘法（数乘，向量，矩阵）
  3. 大量的循环，执行速度较慢
  4. Numpy 是底层用C语言运算的，向量运算库，做了非常多的优化。

```
def dot_product(a,b):
    result = 0
    for i in range(len(a)):
        result +=a[i]*b[i]
    return result

def matrix_multiply(a,b):
    a_row = len(a)
    a_column = len(a[0])
    b_column = len(b[0])
    matrix = []
    for i in range(a_row):
        row_ = []
        for k in range(b_column):
            dot_product = 0
            for j in range(a_column):
                dot_product += a[i][j]*b[j][k]
            row_.append(dot_product)
        matrix.append(row_)
    return matrix
```

# Numpy

## Numpy 的使用

- 下载安装
  - 命令行安装：
    - 切换到对应的 conda 环境
      - conda activate env_name
    - pip 安装
      - pip install numpy
    - 测试：
      - import numpy as np
      - a = np.array([1,2,3])
      - print(a)

# Numpy

## Numpy 的使用

- 向量的创建：
  - a = np.array([2,3,6])
  - b = np.arrange(5)
  - c = np.linspace(0, 2*np.pi, 5)
  - d = np.array([[11, 12, 13, 14, 15], [16, 17, 18, 19, 20], [21, 22, 23, 24, 25], [26, 27, 28 ,29, 30], [31, 32, 33, 34, 35]])
  - e = np.zeros((2,3))
  - f = np.ones((3,1))
  - g = np.random.rand(3,2)
  - h = np.random.randint(0,10,(3,2))

# Numpy

## Numpy 的使用

- 获取向量的值与属性
  - 直接print()
  - 切片 [:,2],[1:3,2:5],[::2,::2] 等
  - 形状与维度
    - a.shape
    - a.ndim

# Numpy

## Numpy 的使用

- Numpy的数学运算
  - +,-,*,/,//,%,**
  - <<,>>,&,^,|,~
  - ==, >,<,>=,<=,!=
  - a.dot(b), np.cross(a,b),np.linalg.norm(a)，np.outer(a,b)
  - 函数的向量化运算：
    - np.sin(), np.cos(), np.tan() …

# Numpy

## Numpy 的使用

- Numpy的其他操作
  - max, min, sum
  - 布尔屏蔽
    - a = np.arange(0,100,10)
    - print(a[a>=50])
  - Where函数
    - a = np.arange(0, 100, 10)
    - b = np.where(a < 50)
    - c = np.where(a >= 50)[0]
    - print(b) # >>>(array([0, 1, 2, 3, 4]),)
      print(c) # >>>[5 6 7 8 9]

# Numpy

## Numpy 的应用场景

- 向量，矩阵的科学运算
  - 序列运算
  - 解线性方程组
  - 数学建模
  - 机器学习
  - 深度学习
  - ……

- › 数据类型相关
- › 可选的Scipy加速支持(numpy.dual)
- ˅ **具有自动域的数学函数(numpy.emath)**
- › 浮点错误处理
- › 离散傅立叶变换(numpy.fft)
- › 财金相关
- › 实用的功能
- › 特殊的NumPy帮助功能
- › 索引相关
- › 输入和输出
- › 线性代数(numpy.linalg)
- › 逻辑函数
- › 操作掩码数组
- › 数学函数