

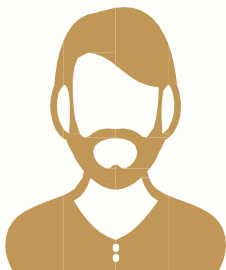
第1章 坐标与变换

第02讲 描述空间的工具 — 向量

传媒与信息工程学院
欧 新 宇

第2讲 描述空间的工具：向量

- ✓ 向量的基本知识回顾
- ✓ 列向量
- ✓ 使用Python语言描述向量
- ✓ 向量的加法和数乘
- ✓ 向量间的乘法
- ✓ 向量的线性组合



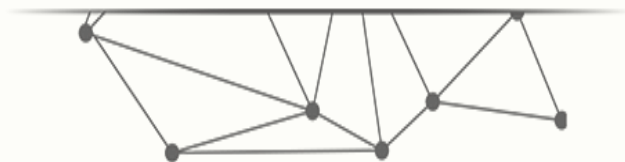
第2讲 描述空间的工具：向量

总体说明

空间是贯穿**线性代数**整个领域的**主干**和**核心**概念，我们所有的**概念**和**应用**都会构架在**空间**这个逻辑实体上。而**向量**和**矩阵**就是我们用来填充这个实体的工具，包括运算、映射、降维、投影、近似求解、特征提取等，都将建立在基于**矩阵**和**向量**的**空间**中实现。



向量的基本知识回顾



向量的基本知识回顾

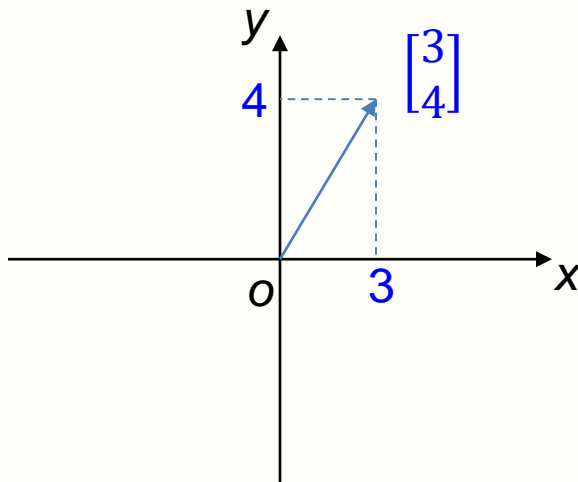
向量的定义

- **向量**：也称欧几里得向量、几何向量、矢量，它指具有**大小**和**方向**的量。它可以形象化地表示为**带箭头的线段**。直观地说，一组排列成行或列的有序数字，就是向量。
- **箭头所指**：代表向量的方向；
- **线段长度**：代表向量的大小；
- **向量的记法**：
 - 印刷体，记作**小写粗斜体字母**，如 $\mathbf{a}, \mathbf{b}, \mathbf{u}, \mathbf{v}$;
 - 手写体，在**字母**上加一**小箭头** “ \rightarrow ”，如 \vec{u}
 - 给定向量的**起点A**和**终点B**，可记作 \overrightarrow{AB} ;
 - 在**空间直角坐标系**中，以**数对**形式表示，如 $(2, 3)$

向量的基本知识回顾

二维向量的空间表示

给定二维向量 $\mathbf{u} = \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ ，它有两个分量，其中 x 分量值为3， y 分量值为4，以原点(0,0)为起点，可以在直角坐标系中构建一条有向线段。

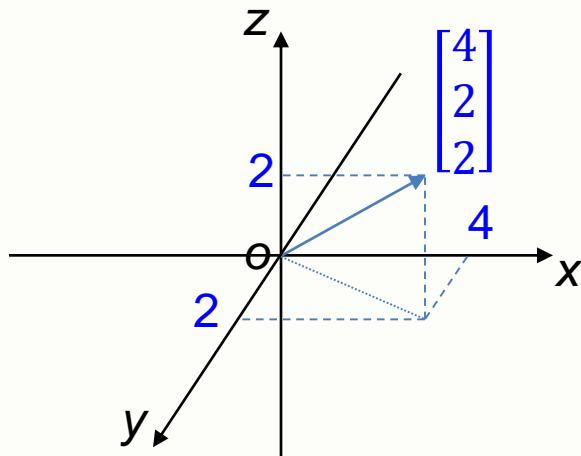


向量的基本知识回顾

三维向量的空间表示

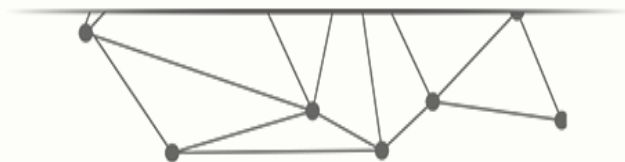
给定三维向量 $\mathbf{u} = \begin{bmatrix} 4 \\ 2 \\ 2 \end{bmatrix}$ ，它有三个分量，其中 x 分量值为4，

y 分量值为2， z 分量值为2，以原点(0,0,0)为起点，可以在三阶笛卡尔坐标系中构建一条有向线段。





列向量



列向量

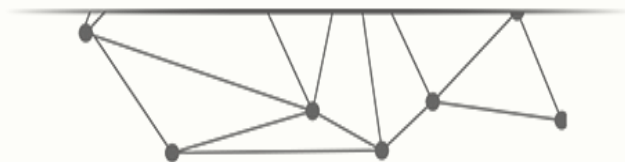
计算机领域主要使用列向量

根据数字的排列方式，向量可以被分为行向量和列向量。在计算机领域中，我们常使用列向量来表示和处理向量。例如，将矩阵 A 映射到向量 x 上时，可以用 Ax 来表示，最常见的应用是求解方程组。列向量通常由两种表示方法。

- 直观表示： $\alpha = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$, $\beta = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$
- 单行表示 (更常用) : $\alpha = [2, 3]^T$, $\beta = [2, 3, 4]^T$



基于Python语言的向量表示



基于Python语言的向量表示

在Python中，最重要，也是最常用的一个库就是**数学计算库 Numpy**，它也是本门课中最主要的python工具包。下面我们将使用numpy库来实现**数组(向量、矩阵)**的创建。

● 创建numpy数组（向量）

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

```
[[1 2 3 4]]
```

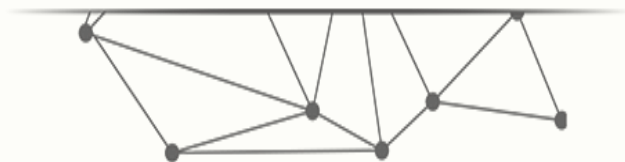
● 获取变量数据类型

```
type(A)
```

```
numpy.ndarray
```



列向量的生成



列向量的生成

在机器学习及大多数计算机任务中，都会以**列向量**的方式对数据进行处理，而**numpy**默认生成的是**行向量**。所以，需要事先进行转换。

最容易也是最直接的方法：**矩阵转置 (Transpose)**。

值得注意的是，在计算机的存储意识中，**向量**是一维的量，它只在一个维度上具有值。因此，无法进行转置。

```
import numpy as np
A = np.array([1, 2, 3, 4])
B = A.transpose()
C = A.T

print('a={}'.format(A))
print('A={}'.format(B))
print('B={}'.format(C))

a=[1 2 3 4]
A=[1 2 3 4]
B=[1 2 3 4]
```

列向量的生成

如何处理呢?

一维向量



二维矩阵

- 当我们使用**向量**来表示一个数据时，可以表示为: $\mathbf{a} = [a_1, a_2, \dots, a_n]$, 此时 \mathbf{a} 是一个维度为 1, 长度为 n 的数据 (向量) ;
- 当我们使用**矩阵**来表示这个向量时，则可以表示为: $A_2 = [a_{11}, a_{12}, \dots, a_{1n}]$, 此时 A_2 是一个维度为 2 的数据 (矩阵), 第一个维度长度为 1, 第二个维度长度为 n , 我们也可以将这样的矩阵理解为一个**行向量**, 一行 n 列, 形态为: $1 \times n$ (1, 4)。
- 在转换为二维矩阵后, 就可以通过矩阵的转置实现**行向量**向**列向量**的转换, 此时的数据 A_2 将转变为一个列向量 B_2 , n 行一列, 形态为: $n \times 1$ (4, 1), 表示为: $A_3 = [a_{11}; a_{21}; \dots; a_{n1}]$ 。

列向量的生成

使用二维矩阵进行转换

```
import numpy as np
A2 = np.array([[1, 2, 3, 4]])
B2 = A2.transpose()
C2 = A2.T
```

```
print('A2={}'.format(A2))
print('B2={}'.format(B2))
print('C2={}'.format(C2))
```

```
A2=[[1 2 3 4]]
B2=[[1]
    [2]
    [3]
    [4]]
C2=[[1]
    [2]
    [3]
    [4]]
```

```
print('A2的形态: {}'.format(A2.shape))
print('B2的形态: {}; C2的形态: {}'.format(B2.shape, C2.shape))
```

A2的形态: (1, 4)

B2的形态: (4, 1); C2的形态: (4, 1)

列向量的生成

【结果分析】

- 原始的一维向量 a 和经过`.transpose()`或`.T`转换后的向量 b 和 c ，都呈现为相同的形态 $(4, 1)$ ，并且值也完全相同。这说明在Python中，转置在向量上是无效的。
- 当我们使用二维矩阵进行转换时，新生成的矩阵 A_2 是一个 $1 \times n$ 的二维矩阵，当经过`.transpose()`和`.T`转换后，两个矩阵都变成了 $n \times 1$ 的矩阵。这说明，原来以二维矩阵显示的行向量，形态为 $(1, 4)$ ；已经转换为以二维矩阵显示的列向量了，形态为 $(4, 1)$ 。

列向量的生成

特别注意

在Python中一维向量和二维矩阵的表示非常容易转换，只需要增加一层中括号"`[]`"就可以实现从一维到二维的转换。

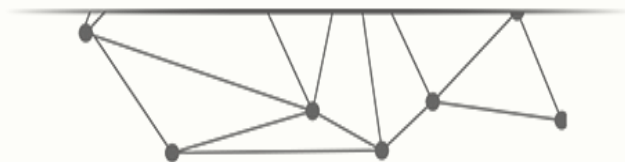
相似地，三维矩阵使用三层中括号表示， n 维矩阵使用 n 层中括号表示。

下面给出一维向量和二维向量的表示。

```
A = np.array([1, 2, 3, 4])  
A2 = np.array([[1, 2, 3, 4]])
```



向量的加法



向量的加法

要进行**矩阵相加**，前提是两个**矩阵**具有**相同的形态**（即 $A.shape = B.shape$ ）。矩阵的加法可以理解为两个矩阵**对应元素**的**相加**（按位相加），生成的结果矩阵维度保持不变（即 $(A+B).shape = A.shape = B.shape$ ）。

给定两个 n 维向量 u 和 v ，它们之间的**加法运算规则**可以表示为：

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} + \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ v_n \end{bmatrix} = \begin{bmatrix} u_1 + v_1 \\ u_2 + v_2 \\ u_3 + v_3 \\ \dots \\ u_n + v_n \end{bmatrix}$$

向量的加法

一个例子

求解矩阵 $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ 和 $\vec{v} = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$ 的**和**的运算结果。

按照运算规则可以表示为：

$$\vec{u} + \vec{v} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} + \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix} = \begin{bmatrix} 1 + 5 \\ 2 + 6 \\ 3 + 7 \\ 4 + 8 \end{bmatrix} = \begin{bmatrix} 6 \\ 8 \\ 10 \\ 12 \end{bmatrix}$$

向量的加法

使用Python语言进行描述

```
import numpy as np
u = np.array([[1,2,3,4]]).T
v = np.array([[5,6,7,8]]).T
w = u + v
```

注意已经转换为
二维数组形式

```
print('u={}\n\n v={}\n\n u+v={}'.format(u,v,w))
```

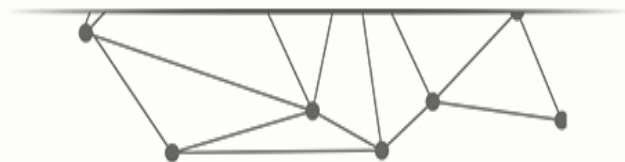
$$u = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$
$$v = \begin{bmatrix} 5 \\ 6 \\ 7 \\ 8 \end{bmatrix}$$

$$u+v = \begin{bmatrix} 6 \\ 8 \\ 10 \\ 12 \end{bmatrix}$$

对于形态为 $1 \times n$ 的**单行矩阵**和**行向量**的**相加**，也遵循**按位相加**的原则。



向量的数乘



向量的数乘

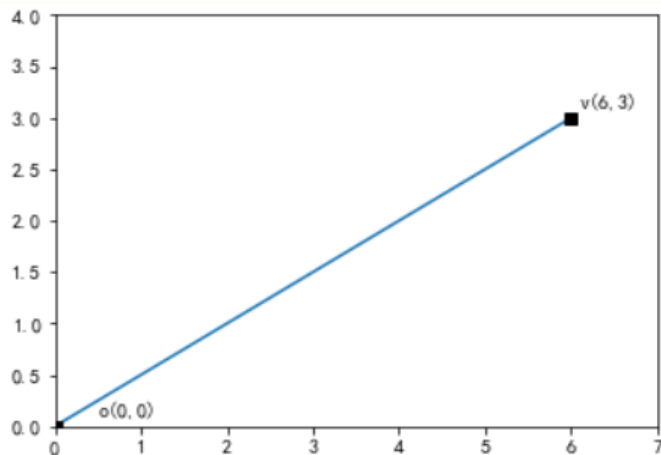
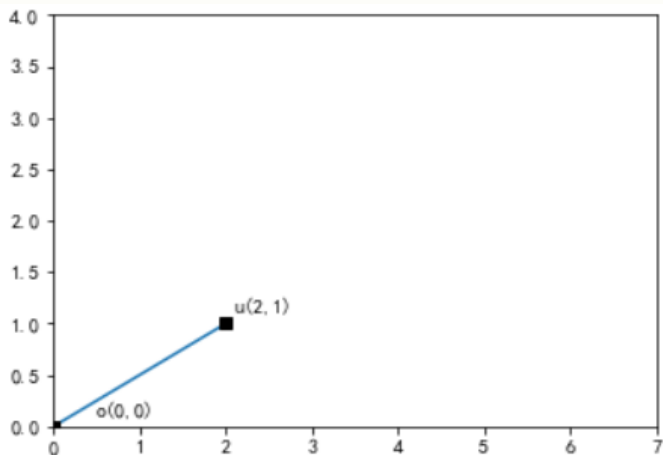
数乘的概念和特性

向量的数乘，又称为向量的**数量乘法**，它表示的是一个**标量**和一个**向量**之间的**乘积**关系。与向量的加法类似，向量的数乘是由标量和向量中的**每个元素依次相乘**，生成的新向量与原来的向量具有**相同的形态**。向量的数乘从**几何**意义上来说，可以理解为向量沿着所在直线的方向拉升相应的倍数，

- 拉升的**倍数**由**标量**决定，
- 拉升的**方向**与**原向量方向**保持**不变**。

向量的数乘

数乘的几何示意图



【结果分析】

从上图可以看到向量 \vec{u} 和 向量 $\vec{v} = 3 * \vec{u}$ 的几何示意图，两个向量具有相同的方向，但具有不同的长度。向量 $\vec{v} = 3 * \vec{u}$ 的长度刚好是向量 \vec{u} 的3倍。

向量的数乘

数乘的运算规则

给定一个 n 维向量 \mathbf{u} 和一个标量 c ，他们的数乘变换运算规则可以表示为：

$$c * \vec{u} = c * \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} = \begin{bmatrix} c * u_1 \\ c * u_2 \\ c * u_3 \\ \dots \\ c * u_n \end{bmatrix}$$

向量的数乘

一个例子

给定标量5和向量 $\vec{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$ ，可以得到它们的数乘结果为：

$$5 * \vec{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = 5 * \begin{bmatrix} 5 * 1 \\ 5 * 2 \\ 5 * 3 \\ 5 * 4 \end{bmatrix} = \begin{bmatrix} 5 \\ 10 \\ 15 \\ 20 \end{bmatrix}。$$

向量的数乘

使用Python语言进行描述

```
import numpy as np
u = np.array([[1,2,3,4]]).T
res = 5*u
print(res)
```

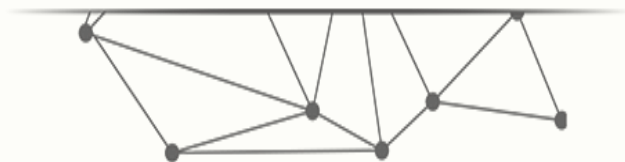
```
[[ 5]
 [10]
 [15]
 [20]]
```

● 结果分析:

向量的数乘是没有方向的，无论左乘还是右乘都具有相同的效果，这意味着 $u * a = a * u$ 。这个结论，可以轻松推广到矩阵的数乘。



向量的乘法：内积和外积



向量的乘法：内积和外积

向量的内积

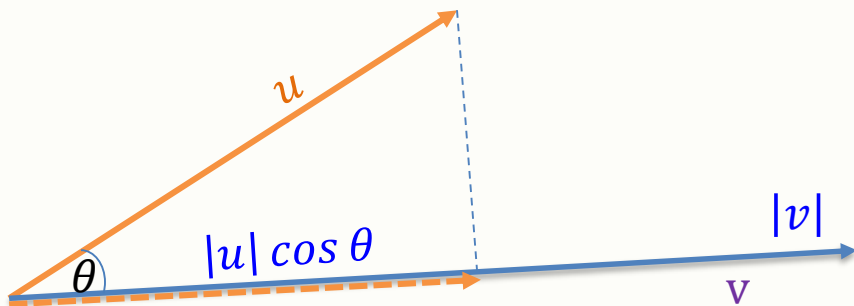
- 内积的**前提**：两个向量维数相同，长度相同
- 向量内积的**结果**：标量
- 内积的**别称**：点乘
- **运算规则**：对应位置上的元素相乘，然后合并相加

$$u \cdot v = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_n \end{bmatrix} \cdot \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ \dots \\ v_n \end{bmatrix} = v_1 v_1 + v_2 v_2 + v_3 v_3 + \dots v_n v_n$$

向量的乘法：内积和外积

向量的内积

- 内积的**几何形式**: $u \cdot v = |u||v| \cos \theta$
- 内积的**几何意义**: 向量 u 在向量 v 方向上的投影长度乘以向量 v 的模长。
- 内积的**几何表示**:



向量的乘法：内积和外积

向量的内积：一个例子

试计算，向量 $u = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix}$ 与向量 $v = \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix}$ 的内积。

解：

$$u \cdot v = \begin{bmatrix} 2 \\ 4 \\ 6 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 3 \\ 5 \end{bmatrix} = 2*1 + 4*3 + 6*5 = 2 + 12 + 30 = 44$$

向量的乘法：内积和外积

向量的内积：Python描述

```
[178]: import numpy as np  
u = np.array([2,4,6])  
v = np.array([1,3,5])  
print(np.dot(u,v))
```

44

'dot' : 点，点乘

● 结果分析：

向量间的**内积**要求两个元素必须是**向量形式**，同时具有**相同的形态**。这意味，以**矩阵形式**表示的“向量”无法进行**内积**运算。

向量的乘法：内积和外积

向量的内积：Python描述

```
import numpy as np
u = np.array([[2,4,6]])
v = np.array([[1,3,5]])
print(np.dot(u,v))
```

行矩阵

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-181-54e4fb57e3e4> in <module>
      2 u = np.array([[2,4,6]])
      3 v = np.array([[1,3,5]])
----> 4 print(np.dot(u,v))
```

ValueError: shapes (1,3) and (1,3) not aligned: 3 (dim 1) != 1 (dim 0)

```
import numpy as np
u = np.array([[2,4,6]]).T
v = np.array([[1,3,5]]).T
print(np.dot(u,v))
```

列矩阵

```
-----
ValueError                                Traceback (most recent call last)
<ipython-input-180-bac849b462b9> in <module>
      2 u = np.array([[2,4,6]]).T
      3 v = np.array([[1,3,5]]).T
----> 4 print(np.dot(u,v))
```

ValueError: shapes (3,1) and (3,1) not aligned: 1 (dim 1) != 3 (dim 0)

结果分析：

可以看到相同形态的二维矩阵无法进行内积运算，哪怕是行数或列数为1的二维数组。这似乎和前面的运算规则相违背。

向量的乘法：内积和外积

向量的内积：Python描述

若需要使用二维数组表示的“向量”进行内积运算，则要求两个数组具有相同的长度，同时两个数组互为转置。

```
import numpy as np
u = np.array([[2,4,6]])
v = np.array([[1,3,5]]).T
print(np.dot(u,v))
```

```
[[44]]
```

具体的运算规则将在后面的矩阵乘法中进行解释。

向量的乘法：内积和外积

向量的外积

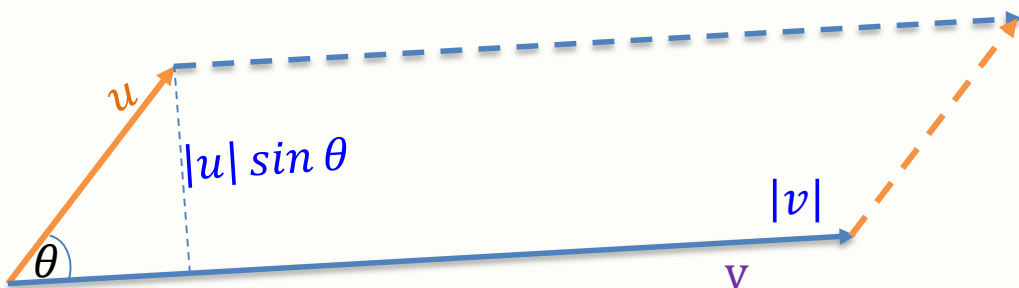
- 向量外积的**结果**：标量（二维）、向量（三维以上）
- 内积的**别称**：叉乘、向量积
- 二维平面的**运算规则**：

$$u \times v = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = u_1 v_2 - u_2 v_1$$

向量的乘法：内积和外积

向量的外积

- 外积的几何形式： $u \times v = |u||v| \sin \theta$
- 几何意义（二维）：向量 u 和向量 v 张成的平行四边形的面积。
- 几何表示（二维）：



向量的乘法：内积和外积

向量的外积

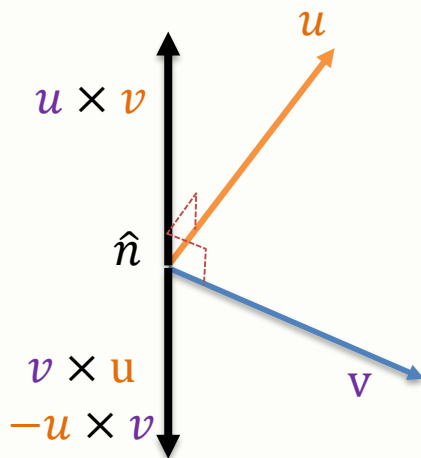
- 三维平面的运算规则：

$$u \times v = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \times \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

向量的乘法：内积和外积

向量的外积

- **几何意义（三维）**：向量 u 和向量 v 张成的平面的法向量，该向量垂直于 u 和 v 向量构成的平面。
- **几何表示（三维）**：



向量的乘法：内积和外积

向量的外积：二个例子（二维）

1. 试计算，向量 $u = \begin{bmatrix} 2 \\ 4 \end{bmatrix}$ 与向量 $v = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$ 的外积。

解：

$$u \times v = \begin{bmatrix} 2 \\ 4 \end{bmatrix} \times \begin{bmatrix} 3 \\ 5 \end{bmatrix} = 2*5 - 4*3 = 10 - 12 = -2$$

向量的乘法：内积和外积

向量的外积：二个例子（三维）

2. 试计算，向量 $u = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix}$ 与向量 $v = \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix}$ 的外积。

解：

$$u \times v = \begin{bmatrix} 2 \\ 3 \\ 4 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix} = \begin{bmatrix} 3 * 6 - 4 * 3 \\ 4 * 1 - 2 * 6 \\ 2 * 3 - 3 * 1 \end{bmatrix} = \begin{bmatrix} 6 \\ -8 \\ 3 \end{bmatrix}$$

向量的乘法：内积和外积

向量的外积：Python描述

```
import numpy as np
u = np.array([2,4])
v = np.array([3,5])
print(np.cross(u,v))
```

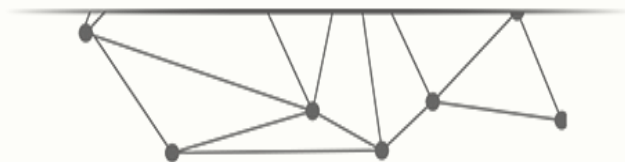
-2

```
import numpy as np
u = np.array([2,3,4])
v = np.array([1,3,6])
print(np.cross(u,v))
```

[6 -8 3]



向量的线性组合



向量的线性组合

概念和运算规则

向量的**线性组合**：基于向量加法和数乘构建的基本运算。

基本**运算规则**：假设存在标量 a, b, c 和向量 u, v, w ，则有：

$$au + bv + cw = a \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} + b \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} + c \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = \begin{bmatrix} au_1 + bv_1 + cw_1 \\ au_2 + bv_2 + cw_2 \\ au_3 + bv_3 + cw_3 \end{bmatrix}$$

向量的线性组合

一个例子

给定标量 $a = 2, b = 4, c = 6$ 和向量 $\mathbf{u} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix}, \mathbf{w} = \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}$,

试求线性组合 $a\mathbf{u} + b\mathbf{v} + c\mathbf{w}$ 。

$$\text{解: } a\mathbf{u} + b\mathbf{v} + c\mathbf{w} = 2 \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 4 \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} + 6 \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix}$$

$$= \begin{bmatrix} 2 * 1 + 4 * 4 + 6 * 7 \\ 2 * 2 + 4 * 5 + 6 * 8 \\ 2 * 3 + 4 * 6 + 6 * 9 \end{bmatrix} = \begin{bmatrix} 60 \\ 72 \\ 84 \end{bmatrix}$$

向量的线性组合

Python描述

```
import numpy as np
u = np.array([[1,2,3]]).T
v = np.array([[4,5,6]]).T
w = np.array([[7,8,9]]).T
print(2*u + 4*v + 6*w)
```

```
[[60]
 [72]
 [84]]
```

```
import numpy as np
u = np.array([1,2,3]).T
v = np.array([4,5,6]).T
w = np.array([7,8,9]).T
print(2*u + 4*v + 6*w)
```

```
[60 72 84]
```

● 结果分析:

向量的线性组合需要将向量转换为**列向量**，因此需要使用**二维数组**来表示列向量。直接进行线性变换，可以运算，但无法获得最终的列向量。

读万卷书 行万里路 只为最好的修炼

QQ: 14777591 (宇宙骑士)

Email: ouxinyu@alumni.hust.edu.cn

Tel: 18687840023