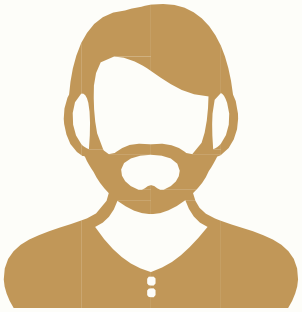
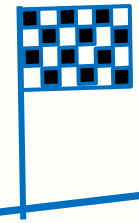
第1章 坐标与变换

第04讲 矩阵

传媒与信息工程学院
欧 新 宇



- 矩阵的定义和基本描述
- 特殊形态的矩阵
- 基于矩阵的向量
- 矩阵的四则运算
- 矩阵与向量乘法的新视角
- 矩阵的应用案例



矩阵的定义和基本描述

矩阵

历史与目标

在数学中，**矩阵（Matrix）**是一个按照长方阵列排列的复数或实数集合，最早来自于方程组的系数及常数所构成的方阵。这一概念由19世纪英国数学家凯利首先提出。作为解决线性方程的工具，矩阵也有不短的历史。成书最早在东汉前期的《九章算术》中，用分离系数法表示线性方程组。

学习线性代数的主要目标就是：**学会利用矩阵来描述系统，并用矩阵软件工具去解决各种问题。**

1. 矩阵的定义和基本描述

矩阵的定义

定义1：由 m×n 个数(i=1,2,...,m; j=1,2,...,n) 排成的 m 行 n 列的矩形数表就称为矩阵。如下所示，可以试用***加粗斜体大写英文字母***来表示一个矩阵。

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{pmatrix}$$

矩阵A 称为 m 行 n 列矩阵，简称 m×n (阶)矩阵。为表示它是一个整体，总是加一个括弧或者方括号来表示它。矩阵中的 m×n 个数称为矩阵A的元素，其中 a_{ij} 表示矩阵A的第 i 行第 j 列元素。m×n 矩阵也可以被记作 $\boldsymbol{A}_{m\times n}$。

1. 矩阵的定义和基本描述

矩阵的定义

在python中，一般使用numpy数组来表示矩阵，实际上对于包括向量、矩阵及张量在内，都习惯使用numpy数组来表示，我们可以使用numpy.array()来实现对数组的定义。

```
import numpy as np
A = np.array([
    [1, 2],
    [3, 4],
    [5, 6],
    [7, 8]])
print("矩阵A = \n{}".format(A))
print("矩阵A的形态为: {}".format(A.shape))
```

```
矩阵A =
[[1 2]
 [3 4]
 [5 6]
 [7 8]]
矩阵A的形态为: (4, 2)
```

1. 矩阵的定义和基本描述

同型矩阵及矩阵相等

如果两个矩阵的行数相等、列数也相等，则称它们为**同型矩阵**。若矩阵 $\boldsymbol{A}=a_{ij}$ 与矩阵 $\boldsymbol{B}=b_{ij}$ 是同型矩阵，并且所有所有对应位置的元素均相等，即：

$$a_{ij} = b_{ij}(i=1,2,..,m;j=1,2,...,n)\ ,$$

那么就称矩阵$\boldsymbol{A}$与矩阵$\boldsymbol{B}$相等，记作：$\boldsymbol{A}=\boldsymbol{B}$。

```
import numpy as np
A1 = np.array([[1,2,3],[4,5,6]])
A2 = np.array([[1,2,3],[4,5,6]])
A3 = np.array([[1,1,1],[2,2,2]])
print("A1与A2的形态相同：{}，A1与A2相等：{}。".format(A1.shape==A2.shape, (A1==A2).all()))
print("A1与A3的形态相同：{}，A1与A3相等：{}。".format(A1.shape==A3.shape, (A1==A3).all()))
```

```
A1与A2的形态相同：True，A1与A2相等：True。
A1与A3的形态相同：True，A1与A3相等：False。
```

特殊形态的矩阵

2. 特殊形态的矩阵

方阵

行数和列数相等的矩阵称为**方阵**，其行数或列数称为它的**阶数**。

例如，一个 n 阶矩阵可记为 A_n。

```
import numpy as np
A = np.array([[1,2,3],[4,5,6],[7,8,9]])
B = np.array([[1,1,1,1,1],[2,2,2,2,2],[3,3,3,3,3],[4,4,4,4,4],[5,5,5,5,5]])

print("矩阵A=\n{}, \n 矩阵B=\n{}".format(A,B))
print("矩阵A的形态为：{}，矩阵B的形态为：{}".format(A.shape, B.shape))
```

```
矩阵A=
[[1 2 3]
 [4 5 6]
 [7 8 9]],
矩阵B=
[[1 1 1 1 1]
 [2 2 2 2 2]
 [3 3 3 3 3]
 [4 4 4 4 4]
 [5 5 5 5 5]]
矩阵A的形态为：(3, 3)，矩阵B的形态为：(5, 5)
```

2. 特殊形态的矩阵

对称矩阵

给定矩阵 $A_{m\times n}$，若其转置矩阵A^T与原矩阵相等，则矩阵A称为**对称矩阵**。不难发现，矩阵对称的前提条件有两点：

1. 矩阵A是一个方阵
2. 矩阵A的每一个元素都满足 $A_{ij}=A_{ji}$

给定矩阵A，我们可以得到A的对称矩阵B。

$$\boldsymbol{A}=\begin{bmatrix}1 & 5 & 6 & 7\\ 5 & 2 & 8 & 9\\ 6 & 8 & 3 & 0\\ 7 & 9 & 0 & 4\end{bmatrix},\quad \mathrm{B}=\begin{bmatrix}1 & 5 & 6 & 7\\ 5 & 2 & 8 & 9\\ 6 & 8 & 3 & 0\\ 7 & 9 & 0 & 4\end{bmatrix}$$

2. 特殊形态的矩阵

对称矩阵(Python描述)

```
import numpy as np
A = np.array([[1,5,6,7],[5,2,8,9],[6,8,3,0],[7,9,0,4]])

print("矩阵A=\n{}\n".format(A))
print("矩阵A的对称矩阵B=\n{}".format(A.T))
```

```
矩阵A=
[[1 5 6 7]
 [5 2 8 9]
 [6 8 3 0]
 [7 9 0 4]]
```

```
矩阵A的对称矩阵B=
[[1 5 6 7]
 [5 2 8 9]
 [6 8 3 0]
 [7 9 0 4]]
```

2. 特殊形态的矩阵

零矩阵

所有元素都为0的矩阵称为**零矩阵**，记作$\boldsymbol{O}$。此外还可以通过下标法标识出零矩阵的形态，例如一个4×5的零矩阵，可以表示为$O_{4\times5}$。值得注意的是，不同型的零矩阵是不同(不相等)的，例如：$O_{4\times5}\neq O_{2\times3}$。

零矩阵最重要的作用就是用来初始化矩阵，

- 一方面可以使用零矩阵来表示实际存储数据矩阵的规模，达到初始化矩阵和申请内存空间的功能；
- 另一方面零矩阵也是占用存储空间最小的矩阵。

2. 特殊形态的矩阵

零矩阵

- ✓ **任意匹配：**

  (A==B).any()

- ✓ **所有匹配**：

  (A==B).all()

- ✓ **按位匹配：**(A==B)

- ✓ **形态匹配：**

  A.shape==B.shape

```
import numpy as np
A = np.zeros([4,5])
print(A)
```

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

```
import numpy as np
A1 = np.zeros([4,5])
A2 = np.zeros([2,3])
A3 = np.zeros([4,5])
print("A1 = A2 is {}。".format(A1==A2))
print("A1 = A3 is {}。".format((A1==A3).all()))
```

```
A1 = A2 is False。
A1 = A3 is True。
```

2. 特殊形态的矩阵

对角矩阵

有一类矩阵，除了对角线外所有的元素都为0，这种矩阵就称为对角矩阵。例如：

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

在对角矩阵中，为0的元素位置可以省去不写。

```
import numpy as np
A = np.diag([1,2,3,4,5])
print(A)
```

```
[[1 0 0 0 0]
 [0 2 0 0 0]
 [0 0 3 0 0]
 [0 0 0 4 0]
 [0 0 0 0 5]]
```

2. 特殊形态的矩阵

对角矩阵

在对角矩阵中，如果对角线上的元素都为1，则该矩阵称为单位矩阵。n阶单位矩阵记作 I_n （在有些文献中也被记作 E_n ）。

例如：

$$\boldsymbol{I} = \begin{bmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & 1 \end{bmatrix}$$

```
import numpy as np
I = np.eye(4)
print(I)
```

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

基于矩阵的向量

基于矩阵的向量

为了规范和便于计算，所有的量（向量、矩阵、张量）都规范成张量，并同时使用矩阵（张量)来进行表示，而在程序中，我们统一使用numpy数组来表示这种量。此时，

- 一个 1×n 的行向量 a^T 就表示成一个只有一行的矩阵；
- 一个 n×1 的列向量 b 则表示成一个只有一列的矩阵。

```
import numpy as np
a = np.array([[1,2,3,4]])
print("a={}".format(a))
print("b=\n{}".format(a.T))
```

```
a=[[1 2 3 4]]
b=
[[1]
 [2]
 [3]
 [4]]
```

【结果分析】我们使用一个二维数组来显示向量（两层中括号），这种方法基本上贯穿于整个计算机领域。其中 ***a*** 用来表示一个四维行向量，***b*** 表示一个四维列向量。

矩阵的四则运算

矩阵的四则运算

矩阵的加法运算

定义2 设存在两个 m×n 的矩阵 $A=a_{ij},B=b_{ij}$，那么矩阵A与B的和记作 A+B，规定为：

$$\boldsymbol{A}+\boldsymbol{B}=\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}+\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mn} \end{bmatrix}$$

$$=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} & \cdots & a_{1n}+b_{1n} \\ a_{21}+b_{21} & a_{22}+b_{21} & \cdots & a_{2n}+b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1}+b_{m1} & a_{m2}+b_{m2} & \cdots & a_{mn}+b_{mn} \end{bmatrix}$$

只有当两个矩阵是同型矩阵时，这两个矩阵才能进行加法运算。

矩阵的四则运算

矩阵加法的运算规律

矩阵加法满足下列运算规律（设A, B, C都是 m×n 的矩阵）：

1. ***A + B = B + A***
2. ***(A + B) + C = A + (B + C)***

设矩阵 $A=a_{ij}$，记 $-A=-a_{ij}$，$-A$ 称为矩阵A的负矩阵，显然有：

$$A + (-A) = O$$

由此规定矩阵的减法为：***A - B = A + (-B)***

矩阵的四则运算

矩阵的加法运算(Python代码)

```
import numpy as np
A = np.array([[1,2],[3,4],[5,6]])
B = np.array([[10,20],[30,40],[50,60]])
print("A=\n{},\n B=\n{},\n A+B=\n{}, \n B-A=\n{}".format(A,B,A+B,B-A))
```

```
A=
[[1 2]
 [3 4]
 [5 6]],
 B=
[[10 20]
 [30 40]
 [50 60]],
```

```
 A+B=
[[11 22]
 [33 44]
 [55 66]],
```

```
 B-A=
[[ 9 18]
 [27 36]
 [45 54]]
```

矩阵的四则运算

【例1】超市年度销售总额

假设有四个超市，它们上半年和下半年的销售清单如下所示，试求这四个超市全年的销售清单。

上半年销售表	大米	面粉	食油
超市一	150	250	50
超市二	250	500	100
超市三	300	700	120
超市四	450	850	80

下半年销售表	大米	面粉	食油
超市一	180	350	60
超市二	300	550	120
超市三	350	850	150
超市四	500	850	100

将表单写成矩阵形式：

$$A=\begin{bmatrix}150 & 250 & 50\\ 250 & 500 & 100\\ 300 & 700 & 120\\ 450 & 850 & 80\end{bmatrix},\quad B=\begin{bmatrix}180 & 350 & 60\\ 300 & 550 & 120\\ 350 & 850 & 150\\ 500 & 850 & 100\end{bmatrix}$$

则全年的销售情况可以用矩阵的加法来表示：

$$C=A+B=\begin{bmatrix}150+180 & 250+350 & 50+60\\ 250+300 & 500+550 & 100+120\\ 300+350 & 700+850 & 120+150\\ 450+500 & 850+850 & 80+100\end{bmatrix}$$

矩阵的四则运算

【例1】超市年度销售总额

下面给出Python代码的实现方法：

```
import numpy as np

A = np.array([[150,250,50],[250,500,100],[300,700,120],[450,850,80]])
B = np.array([[180,350,60],[300,550,120],[350,850,150],[500,850,100]])
C = A + B
print("C=\n{}".format(C))
```

```
C=
[[ 330  600  110]
 [ 550 1050  220]
 [ 650 1550  270]
 [ 950 1700  180]]
```

基于numpy的矩阵加法也是按位相加的规则，运算过程直观、简单。但需要注意的是numpy加法运算的两个元素也必须具有相同的形态。

矩阵的四则运算

矩阵的数量乘法运算

定义3 数 λ 与矩阵A的乘积记作 λA 或 $A\lambda$，规定为：

$$\lambda A = A\lambda = \lambda \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \lambda a_{11} & \lambda a_{12} & \cdots & \lambda a_{1n} \\ \lambda a_{21} & \lambda a_{22} & \cdots & \lambda a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda a_{m1} & \lambda a_{m2} & \cdots & \lambda a_{mn} \end{bmatrix}$$

数乘矩阵满足下列运算规律（设 ***A,B*** 为 m×n 矩阵，λ, μ为标量）：

- $(\lambda\mu)A = \lambda(\mu A)$
- $(\lambda+\mu)A = \lambda A + \mu A$
- $\lambda(A+B) = \lambda A + \lambda B$

矩阵相加与数乘矩阵合起来，统称为矩阵的**线性运算**。

矩阵的四则运算

【例2】计算期末总成绩

甲、乙、丙三位同学在期末考试中，4门课程的成绩分别由矩阵A给出，而他们的平时成绩则由矩阵B给出，若期末考试成绩占总成绩的90%，而平时成绩占10%，请计算这三名同学的总成绩。

$$A=\begin{bmatrix}85 & 85 & 65 & 98\\ 75 & 95 & 70 & 95\\ 80 & 70 & 76 & 92\end{bmatrix},\quad B=\begin{bmatrix}90 & 70 & 80 & 92\\ 80 & 90 & 82 & 92\\ 85 & 75 & 90 & 90\end{bmatrix}$$

解：矩阵C表示总成绩，显然有：

$$C=0.9A+0.1B=\begin{bmatrix}85.5 & 83.5 & 66.5 & 97.4\\ 75.5 & 94.5 & 71.2 & 94.7\\ 80.5 & 70.5 & 77.4 & 91.8\end{bmatrix}$$

矩阵的四则运算

【例2】计算期末总成绩

甲、乙、丙三位同学在期末考试中，4门课程的成绩分别由矩阵A给出，而他们的平时成绩则由矩阵B给出，若期末考试成绩占总成绩的90%，而平时成绩占10%，请计算这三名同学的总成绩。

```
import numpy as np

A = np.array([[85,85,65,98],[75,95,70,95],[80,70,76,92]])
B = np.array([[90,70,80,92],[80,90,82,92],[82,75,90,90]])
C = 0.9*A + 0.1*B
print("C=\n{}".format(C))
```

```
C=
[[85.5 83.5 66.5 97.4]
 [75.5 94.5 71.2 94.7]
 [80.2 70.5 77.4 91.8]]
```

矩阵的四则运算

矩阵与矩阵的乘法

定义4 设$A=(a_{ij})$是一个m×s矩阵，$B=(b_{ij})$是一个s×n矩阵，那么矩阵A与矩阵B的乘积是一个m×n矩阵$C=(c_{ij})$，其中

$c_{ij}=a_{i1}b_{1j}+a_{i2}b_{2j}+\ldots+a_{is}b_{sj}=\sum_{k=1}^{s}a_{ik}b_{kj}$ $(i=1,2,\ldots,m;j=1,2,\ldots,n)$，

并把此乘积记作 $C=AB$。

$$\mathrm{C}=A\times\mathrm{B}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\\ a_{31} & a_{32}\end{bmatrix}\begin{bmatrix}b_{11} & b_{12} & b_{13}\\ b_{21} & b_{22} & b_{23}\end{bmatrix}$$

$$=\begin{bmatrix}a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{12}+a_{12}b_{22} & a_{11}b_{13}+a_{12}b_{21}\\ a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21}\\ a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21} & a_{11}b_{11}+a_{12}b_{21}\end{bmatrix}$$

矩阵的四则运算

矩阵乘法的要点

通过观察上面的计算公式，可以总结出矩阵乘法 C=AB 的三个要点：

- **AB可乘的条件**：A的列数 = B的行数
- **AB乘积C的形状**：A的行 × B的列
- **AB乘积C的元素构成**：A的行与B的列的内积

矩阵的四则运算

【例3】矩阵乘法

求矩阵 $A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 1 & 1 & 1\end{bmatrix}$，$B=\begin{bmatrix}1 & 7\\ 2 & 8\\ 3 & 9\end{bmatrix}$ 的乘积AB。

解：因为A是3×2矩阵，B是2×3矩阵，A的列数等于B的行数，所以矩阵A与B可以相乘，其乘积AB=C是一个2×2矩阵。按定理可得：

$$C=AB=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\\ 1 & 1 & 1\end{bmatrix}\begin{bmatrix}1 & 7\\ 2 & 8\\ 3 & 9\end{bmatrix}=\begin{bmatrix}1*1+2*2+3*3 & 1*7+2*8+3*9\\ 4*1+5*2+6*3 & 4*7+5*8+6*9\\ 1*1+1*2+1*3 & 1*7+1*8+1*9\end{bmatrix}$$

$$=\begin{bmatrix}14 & 50\\ 32 & 122\\ 6 & 24\end{bmatrix}$$

矩阵的四则运算

【例3】矩阵乘法

下面给出该乘积的Python代码。注意我们使用np.dot(A,B)实现矩阵乘法。

```
import numpy as np

A = np.array([[1,2,3],[4,5,6],[1,1,1]])
B = np.array([[1,7],[2,8],[3,9]])
C = np.dot(A,B)
print("C=\n{}".format(C))
```

```
C=
[[ 14  50]
 [ 32 122]
 [  6  24]]
```

值得注意的是矩阵乘法是不符合交换律的，换句话说，上例中的BA是没有意义的。

矩阵的四则运算

【例3】矩阵乘法

```
import numpy as np

A = np.array([[1,2,3],[4,5,6],[1,1,1]])
B = np.array([[1,7],[2,8],[3,9]])
C = np.dot(B,A)
print("C=\n{}".format(C))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-19-4de9c015f434> in <module>
      3 A = np.array([[1,2,3],[4,5,6],[1,1,1]])
      4 B = np.array([[1,7],[2,8],[3,9]])
----> 5 C = np.dot(B,A)
      6 print("C=\n{}".format(C))

ValueError: shapes (3,2) and (3,3) not aligned: 2 (dim 1) != 3 (dim 0)
```

矩阵的四则运算

【例4】矩阵乘法 – 交换律

【例4】 下面再给出一个例子：求矩阵 $A=\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}$，$B=\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}$ 的乘积AB及BA。

解：按定义有

$$AB=\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}=\begin{bmatrix}-16 & -32\\ 8 & 16\end{bmatrix}$$

$$BA=\begin{bmatrix}2 & 4\\ -3 & -6\end{bmatrix}\begin{bmatrix}-2 & 4\\ 1 & -2\end{bmatrix}=\begin{bmatrix}0 & 0\\ 0 & 0\end{bmatrix}$$

```python
import numpy as np

A = np.array([[-2,4],[1,-2]])
B = np.array([[2,4],[-3,-6]])
AB = np.dot(A,B)
BA = np.dot(B,A)
print("AB=\n{}".format(AB))
print("BA=\n{}".format(BA))
```

```
AB=
[[-16 -32]
 [  8  16]]
BA=
[[0 0]
 [0 0]]
```

矩阵的四则运算

【例4】矩阵乘法 – 交换律【结果分析】

在【例3】中A是3×3矩阵，B是2×3矩阵，乘积AB有意义，而BA却没有意义。由此可知，在矩阵的乘法中必须注意**矩阵相乘的顺序**。AB是A左乘B的乘积，BA是A右乘B的乘积，AB有意义时，BA可以没有意义。

又若A是 m×n 矩阵，B是 n×m 矩阵，则AB与BA都有意义，但AB是m阶方阵，BA是n阶方阵。

- 当m≠n时AB≠BA。
- 当m=n，即A、B是同阶方阵。如【例4】，A与B都是2阶方阵，从而AB与BA也都是2阶方阵，但AB与BA仍然可以不相等。

总之，矩阵乘法不满足交换律，即在一般情况下，AB≠BA。

矩阵的四则运算

【例4】矩阵乘法 – 交换律【结果分析】

【例4】 还表明，矩阵 A≠O，B≠O，但却有 BA≠O。因此要特别注意：

- 若有两个矩阵A、B满足AB=O，不能得出A=O或B=O的结论；
- 若A≠O而 A(X−Y)=O，也不能得出X=Y的结论。

矩阵的四则运算

矩阵乘法的运算规律

矩阵乘法虽然不满足交换律，但仍满足下列结合律和分配率（假设运算时可行的）

- **结合律**：$(AB)C=A(BC)$
- $\lambda(AB)=(\lambda A)B=A(\lambda B)$，其中 λ 是标量
- **分配律**：$A(B+C)=AB+AC,(B+C)A=BA+CA$

对于单位矩阵 $\boldsymbol{I}$，不难得出结论：$\boldsymbol{I}_m\boldsymbol{A}_{m\times n}=\boldsymbol{A}_{m\times n}$，$\boldsymbol{A}_{m\times n}\boldsymbol{I}_n=\boldsymbol{A}_{m\times n}$，或简写成：$\boldsymbol{IA}=\boldsymbol{AI}=\boldsymbol{A}$。

可见单位矩阵 $\boldsymbol{I}$ 在矩阵乘法中的作用类似于标量 1。

矩阵的四则运算

【例5】服装厂的生产量

有甲、乙、丙、丁4个服装厂，一个月的产量情况由下表给出，若甲厂生产8个月，乙厂生产10个月，丙厂生产5个月，而丁厂生产9个月，则共生产帽子、衣服、裤子各多少？用矩阵来描述。

品种	甲	乙	丙	丁
帽	20	4	2	7
衣	10	18	5	6
裤	5	7	16	3

解：上表可以矩阵化为：$A=\begin{bmatrix}20 & 4 & 2 & 7\\ 10 & 18 & 5 & 6\\ 5 & 7 & 16 & 3\end{bmatrix}$，$b=\begin{bmatrix}8\\ 10\\ 5\\ 9\end{bmatrix}$

则，总产量可以表示为：$C=A\times b=\begin{bmatrix}8\times 20 & 10\times 4 & 5\times 2 & 9\times 7\\ 8\times 10 & 10\times 18 & 5\times 5 & 9\times 6\\ 8\times 5 & 10\times 7 & 5\times 16 & 9\times 3\end{bmatrix}=\begin{bmatrix}273\\ 339\\ 217\end{bmatrix}$

矩阵的四则运算

【例5】服装厂的生产量

有甲、乙、丙、丁4个服装厂，一个月的产量情况由下表给出，若甲厂生产8个月，乙厂生产10个月，丙厂生产5个月，而丁厂生产9个月，则共生产帽子、衣服、裤子各多少？用矩阵来描述。

品种	甲	乙	丙	丁
帽	20	4	2	7
衣	10	18	5	6
裤	5	7	16	3

```
import numpy as np

A = np.array([[20,4,2,7],[10,18,5,6],[5,7,16,3]])
b = np.array([[8],[10],[5],[9]])
C = np.dot(A, b)
print("C=\n{}".format(C))
```

```
C=
[[273]
 [339]
 [217]]
```

矩阵的四则运算

矩阵的幂

根据矩阵的乘法法则，定义矩阵的幂。设A是n阶方阵，定义：

$$A^1=A,\ A^2=A^1A^1,\ \ldots,\ A^{k+1}=A^kA^1$$

其中 k 为正整数。也就是说，A^k 就是 k 个 A 连乘。显然只有方阵满足条件，幂运算才有意义。

由于矩阵乘法适合结合律，所以矩阵的幂满足以下运算规律：

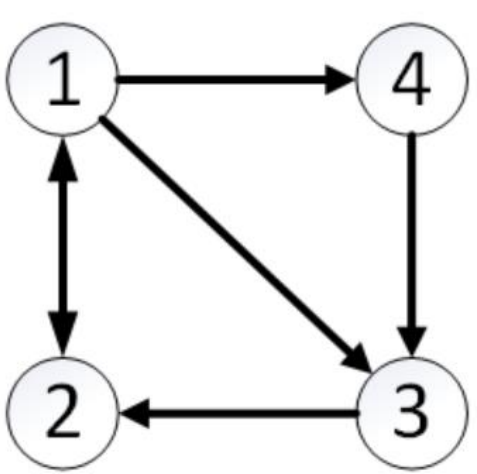
$$A^kA^l=A^{k+l},\ (A^k)^l=A^{kl}$$

其中 k、l 为正整数。又因矩阵乘法一般不满足交换律，所以对于两个 n 阶矩阵 A 与 B，一般来说 $(AB)^k\neq A^kB^k$。

矩阵的四则运算

【例6】四个城市间的单向航线

若四个城市的航向图如下所示：



$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix},$$

$$a_{ij}=\begin{cases}1 & \text{从}i\text{市到}j\text{市有1条单向航线}\\ 0 & \text{从}i\text{市到}j\text{市没有单向航线}\end{cases}$$

若记$A^2=b_{ij}$，则b_{ij}为从 i 市经过一次中转到 j 市的单向航线条数。

$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix},\ \text{有}A^2=\begin{bmatrix}2&1&1&0\\0&1&1&1\\1&0&0&0\\0&2&1&1\end{bmatrix}.$$

矩阵的四则运算

【例6】四个城市间的单向航线

$$A=(a_{ij})=\begin{bmatrix}0&1&1&1\\1&0&0&0\\0&1&0&0\\1&0&1&0\end{bmatrix}，有A^2=\begin{bmatrix}2&1&1&0\\0&1&1&1\\1&0&0&0\\0&2&1&1\end{bmatrix}.$$

- ✓ b_{23}=1，显示从(2)经过一次中转后到达(3)市的单向航线有一条：(2)->(1)->(3);
- ✓ b_{42}=2，显示从(4)经过一次中转后到达(2)市的单向航线有2条：(4)->(1)->(2)，(4)->(3)->(2);
- ✓ b_{11}=2，显示过(1)市的双向航线有2条：(1)->(2)->(1)，(1)->(4)->(1);
- ✓ b_{33}=0，显示(3)市没有双向航线。

矩阵的四则运算

【例7】四个城市间的单向航线(Python实现)

假设给定上面的航线矩阵A，试求从(4)市到(2)市，最多经过4次转机，有几种到达方法。

解：基本的思路是直飞(A)，一次转机（A^2），二次转机（A^3），三次转机（A^4），四次转机（A^5）之和。使用矩阵乘法显然很复杂，下面我们使用Python代码来完成这个例子。

```
import numpy as np
A = np.array([[0,1,1,1],[1,0,0,0],[0,1,0,0],[1,0,1,0]])
A2 = np.dot(A,A)
A3 = np.dot(A2,A)
A4 = np.dot(A3,A)
A5 = np.dot(A4,A)
print(A[3,1] + A2[3,1] + A3[3,1] + A4[3,1] + A5[3,1])
```

```
11
```

需要注意的是: 矩阵乘法的代码语法，不能使用 A*A(元素乘)，而要使用点乘 .dot(A,B)。

矩阵的转置

矩阵的转置

转置的定义

【定义5】 给定矩阵$A_{m\times n}$，若将其行和列的元素进行位置互换，可以得到一个新的矩阵$B_{n\times m}$。那么矩阵B就称为矩阵A的**转置矩阵**，并记作 $B=A^T$。同时，矩阵A也称为矩阵B的**转置矩阵**。行和列的互换操作就称为矩阵的转置。

下面给出矩阵转置的Python代码：

```
import numpy as np
A = np.array([[1,2,3,4],[5,6,7,8]])

print("矩阵A=\n{}\n".format(A))
print("矩阵A的转置=\n{}".format(A.T))
```

```
矩阵A=
[[1 2 3 4]
 [5 6 7 8]]

矩阵A的转置=
[[1 5]
 [2 6]
 [3 7]
 [4 8]]
```

注意向量（一维数组）无法执行转置运算。

矩阵的转置

转置的运算规律

矩阵的转置也是一种运算，满足下述运算规律（假设运算都是可行的）：

- $(A^T)^T=A$
- $(A+B)^T=A^T+B^T$
- $(\lambda A)^T=\lambda A^T$
- $(AB)^T=B^T A^T$

矩阵的转置

【例8】矩阵转置的例子

已知 $A=\begin{bmatrix}2 & 0 & -1\\ 1 & 3 & 2\end{bmatrix}$，$B=\begin{bmatrix}1 & 7 & -1\\ 4 & 2 & 3\\ 2 & 0 & 1\end{bmatrix}$，求$(AB)^T$。

此处只给出Python代码的实现方法：

```
import numpy as np
A = np.array([[2,0,-1],[1,3,2]])
B = np.array([[1,7,-1],[4,2,3],[2,0,1]])
print(np.dot(A,B).T)
```

```
[[ 0 17]
 [14 13]
 [-3 10]]
```

矩阵与向量的乘法的新视角

矩阵与向量的乘法的新视角

矩阵与向量的乘法可以理解成: 向量 x 到向量 y 的线性变换。

【定义 6】对于向量 $y=[y_1,y_2,\ldots,y_m]^T$，若它能由向量 $x=[x_1,x_2,\ldots,x_n]^T$ 线性表示，即有：

$$\begin{cases} y_1 = a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ y_2 = a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ \cdots \\ y_m = a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n \end{cases}$$

则称此关系式为向量 x 到向量 y 的线性变换，可以写成输出向量 y 等于系数矩阵 A 左乘输入向量 x：

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_m \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \ldots & a_{1n} \\ a_{21} & a_{22} & \ldots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & \ldots & a_{mn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_m \end{bmatrix} = \begin{bmatrix} a_{11}x1 + a_{12}x2 + \ldots + a_{1n}xn \\ a_{21}x1 + a_{22}x2 + \ldots + a_{2n}xn \\ \cdots \\ a_{m1}x1 + a_{m2}x2 + \ldots + a_{mn}xn \end{bmatrix} = AX$$

矩阵与向量的乘法的新视角

在进行向量与矩阵乘法的时候，矩阵A在左，列向量x在右，即Ax的顺序不能变。对照矩阵与矩阵的乘法规则，我们可以总结矩阵与向量的乘法规则：当把列向量看作是一个列数为1的特殊矩阵时，那么运算过程就变得比较简单了。

- 矩阵在左，列向量在右，矩阵的列数和列向量的维数必须相等
- 矩阵和列向量相乘的结果也是一个列向量；
- 矩阵的行数就是结果向量的维数；
- 乘法运算的实施过程就是：矩阵的每行和列向量的对应元素分别相乘后，再相加。

矩阵与向量的乘法的新视角

【例8】 给出一个矩阵与向量相乘的例子：

$$\begin{bmatrix} 1 & 2 \\ 4 & 5 \\ 7 & 8 \end{bmatrix} \begin{bmatrix} 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1*3+2*4 \\ 4*3+5*4 \\ 7*3+8*4 \end{bmatrix} = \begin{bmatrix} 11 \\ 32 \\ 53 \end{bmatrix}.$$

```
import numpy as np
A = np.array([[1,2],
              [4,5],
              [7,8]])
x = np.array([[3,4]]).T
print(np.dot(A,x))
```

```
[[11]
 [32]
 [53]]
```

【结果分析】

从程序运行的结果来看，原始向量 u 表示二维平面的一个点，其在二维平面中的坐标为 A(3, 4)，经过与矩阵 A的乘法运算后，最终将原始点 A 转换为新的目标点B，其空间坐标为B(11,32,53)。

从以上的例子可以总结出矩阵所发挥的重要作用：在指定矩阵的作用下，原始空间中的向量被**映射**转换到了目标空间的新坐标，向量的空间位置由此发生了变化，甚至在映射后，目标空间的维数想较于原始空间都有可能发生改变。

矩阵的应用案例

矩阵的应用案例

【例9】 生产成本核算问题

某厂生产三种产品，它的成本分为三类。每一类成本中，给出生产单个产品时估计需要的量。同时给出每季度生产每种产品数量的估计。该公司希望在股东会上用一个表格展示出每一季度三类成本的数量：原料费、工资和管理费。

成本（元）	产品A	产品B	产品C
原材料	0.1	0.3	0.15
劳动	0.3	0.4	0.25
企业管理费	0.1	0.2	0.15

产品	夏	秋	冬	春
A	4000	4500	4500	4000
B	2000	2600	2400	2200
C	5800	6200	6000	6000

解：我们用矩阵的方法来考虑此问题，设产品分类成本矩阵为M，季度产量矩阵为P，则有：

$$M=\begin{bmatrix}0.1 & 0.3 & 0.15\\ 0.3 & 0.4 & 0.25\\ 0.1 & 0.2 & 0.15\end{bmatrix},\quad P=\begin{bmatrix}4000 & 4500 & 4500 & 4000\\ 2000 & 2600 & 2400 & 2200\\ 5800 & 6200 & 6000 & 6000\end{bmatrix}.$$

矩阵的应用案例

【例9】 生产成本核算问题

成本（元）	产品A	产品B	产品C	产品	夏	秋	冬	春
原材料	0.1	0.3	0.15	A	4000	4500	4500	4000
劳动	0.3	0.4	0.25	B	2000	2600	2400	2200
企业管理费	0.1	0.2	0.15	C	5800	6200	6000	6000

- 如果按 $Q=M*P$ 构造乘积，则MP的第一列表示夏季的成本。
  - 原料费： 0.1∗4000+0.3∗2000+0.15∗5800=1870
  - 工资： 0.3∗4000+0.4∗2000+0.25∗5800=3450
  - 管理费和其他： 0.1∗4000+0.2∗2000+0.15∗5800=1670
- MP的第二列表示秋季的成本。
  - 原料费： 0.1∗4500+0.3∗2600+0.15∗6200=2100
  - 工资: 0.3∗4500+0.4∗2600+0.25∗6200
  - 管理费和其他: 0.1∗4500+0.2∗2600+0.15∗6200
- MP的第三列和第四列表示冬季和秋季的成本。

矩阵的应用案例

【例9】 生产成本核算问题

成本（元）	产品A	产品B	产品C
原材料	0.1	0.3	0.15
劳动	0.3	0.4	0.25
企业管理费	0.1	0.2	0.15

产品	夏	秋	冬	春
A	4000	4500	4500	4000
B	2000	2600	2400	2200
C	5800	6200	6000	6000

MP第一行的元素表示四个季度原料的总成本，第二和第三行的元素分别表示四个季度中每一季度工资和管理的成本。每一类成本的年度总成本可由矩阵的每一行元素相加得到。每一列元素相加，即可得到每一季度的总成本。下表汇总了总成本。

成本（元）	夏	秋	冬	春	全年
原材料	1870	2160	2070	1960	8060
劳动	3450	3940	3810	3580	14780
企业管理费	1670	1900	1830	1740	7140
总成本	6990	8000	7710	7280	29980

矩阵的应用案例

【例9】 生产成本核算问题

```
import numpy as np
import pandas as pd

M = np.array([[0.1,0.3,0.15],[0.3,0.4,0.25],[0.1,0.2,0.15]])
P = np.array([[4000,4500,4500,4000],[2000,2600,2400,2200],[5800,6200,6000,6000]])
Q = np.dot(M,P)

# 将numpy数组转换为DataFrame
df = pd.DataFrame(Q)
# 添加行列的标题
df.columns = ['夏','秋','东','春']
df.index = ['原材料','劳动','企业管理费']
# 计算各列数据总和并作为新列添加到末尾
df['全年'] = df.apply(lambda x:x.sum(), axis=1)
# 计算各列数据总和并作为新行添加到末尾
df.loc['总成本'] = df.apply(lambda x:x.sum())

# 输出pandas数据
display(df)
```

	夏	秋	东	春	全年
原材料	1870.0	2160.0	2070.0	1960.0	8060.0
劳动	3450.0	3940.0	3810.0	3580.0	14780.0
企业管理费	1670.0	1900.0	1830.0	1740.0	7140.0
总成本	6990.0	8000.0	7710.0	7280.0	29980.0

矩阵的应用案例

【例10】 婚姻状况计算模型

某城镇中，有8000位已婚女性和2000位单身女性，假设每年有30%的已婚女性离婚，20%的单身女性结婚。假设所有女性的总数为一常数，试求1年后有多少已婚女性和单身女性？2年后呢？10年后呢?

解：我们可以用如下的思路构建矩阵A。

1).矩阵A的第一行元素分别为1年后仍处于婚姻状态的已婚女性和已婚的单身女性百分比;

2).第二行元素分别为1年后离婚的已婚女性和未婚的单身女性的百分比。则：

$$A = \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix}$$

矩阵的应用案例

【例10】 婚姻状况计算模型

若令 $x=\begin{bmatrix}8000\\2000\end{bmatrix}$，则1年后已婚女性和单身女性人数可以用 A 乘以x计算.

- 8000*0.7表示仍然在婚姻状态的已婚女性，2000*0.2为转变为已婚的单身女性，两者相加就是1年后已婚女性的总人数；
- 8000*0.3表示1年后离婚的已婚女性，2000*0.8表示仍然未婚的未婚女性，两者相加就是1年后未婚女性的总人数。

$$Ax=\begin{bmatrix}0.7 & 0.2\\0.3 & 0.8\end{bmatrix}\begin{bmatrix}8000\\2000\end{bmatrix}=\begin{bmatrix}6000\\4000\end{bmatrix}$$

1年后将有6000位已婚女性，4000位单身女性。

矩阵的应用案例

【例10】 婚姻状况计算模型

要求2年后已婚女性和单身女性的数量，计算

$$A^2x = A(Ax)$$

$$= \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix} \begin{bmatrix} 6000 \\ 4000 \end{bmatrix}$$

$$= \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix} \begin{bmatrix} 0.7 & 0.2 \\ 0.3 & 0.8 \end{bmatrix} \begin{bmatrix} 6000 \\ 4000 \end{bmatrix} = \begin{bmatrix} 5000 \\ 5000 \end{bmatrix}$$

2年后，一半的女性将为未婚，一般的女性将为单身。

一般地，n年后已婚女性和单身女性的数量可由$A^n x$求得。

问题是，当n很大的时候，计算会变得很复杂！

矩阵的应用案例

【例10】 婚姻状况计算模型（Python）

按照以上的分析，我们将使用Python代码实现后续的计算，求解10年后的已婚女性和单身女性的数量。

```
import numpy as np
A = np.array([[0.7,0.2],[0.3,0.8]])
x = np.array([[8000],[2000]])

n = 8 # n=1:A^2; n=2:A^3
product = np.dot(A,x)

for i in range(n-1):
    product = np.dot(A, product)

print(product)
```

```
[[4015.625]
 [5984.375]]
```

【结果分析】

由程序的输出结果可知，10年后已婚女性和诞生女性的数量分别为：4016人和5984人。

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