

线性代数学习笔记

线性代数是研究向量空间和线性变换的数学分支，是现代数学、物理、计算机科学和工程学的基石。

目录

1. 向量与向量空间	3
1.1. 向量的定义	3
1.2. 向量空间	3
1.3. 线性组合与张成空间	3
2. 线性相关与线性无关	4
2.1. 核心概念	4
2.2. 基与维数	4
3. 矩阵	5
3.1. 基本定义	5
3.2. 矩阵运算	5
3.3. 关系矩阵乘特征矩阵：加权聚合	5
3.4. 为什么不是对应元素相乘？	7
3.5. 特殊矩阵	8
3.6. 矩阵的秩	8
4. 线性方程组	10
4.1. 矩阵表示	10
4.2. 解的存在性	11
5. 行列式	12
5.1. 行列式到底想衡量什么？	12
5.2. 为什么二阶行列式是 $ad - bc$ ？	13
5.3. 定义	14
5.4. 为什么交换两列会让行列式变号？	14
5.5. 为什么交换两行也会让行列式变号？	17
5.6. 为什么 $\det(AB) = \det(A)\det(B)$ ？	18
5.7. 基本性质	19
5.8. 秩和行列式的严格关系	19
5.9. 为什么 $\det(A) \neq 0$ 能推出 A 可逆？	22
5.10. 齐次方程组与零空间	23
6. 特征值与特征向量	24
6.1. 定义	24
6.2. 几何直觉：只被拉伸、不被旋转的方向	24
6.3. 特征多项式	25
6.4. 从方程角度看特征向量	26
6.5. 代数重数与几何重数	27
6.6. 二阶矩阵的快速判断	28
6.7. 特征值与线性系统的长期行为	29
6.8. 对称矩阵的特殊地位	29
6.9. 常见误区	30
6.10. 为什么要抽象出特征值？	30

6.11.	工程中的应用	30
6.12.	重要性质	33
7.	线性变换	33
7.1.	为什么要研究线性变换?	33
7.2.	定义	33
7.3.	矩阵表示	34
7.4.	常见几何变换	35
7.5.	核与像	35
8.	内积空间	37
8.1.	为什么要引入内积?	37
8.2.	内积的定义	38
8.3.	欧几里得内积	38
8.4.	范数与正交性	39
8.5.	投影	39
8.6.	格拉姆-施密特正交化	40
9.	矩阵对角化	40
9.1.	为什么要对角化?	40
9.2.	可对角化条件	40
9.3.	对称矩阵的对角化	42
9.4.	应用: 矩阵的幂	42
10.	奇异值分解 (Singular Value Decomposition, SVD)	42
10.1.	为什么需要 SVD?	42
10.2.	定义	43
10.3.	与特征值的关系	44
10.4.	SVD 和特征值分解有什么区别?	45
10.5.	低秩近似	45
10.6.	和 PCA 的关系	46
10.7.	常见误区	47
11.	附录: 实对称矩阵为什么给出正交特征向量	47
12.	矩阵乘法在工程中如何优化?	48
12.1.	第一个核心: 数据复用	48
12.2.	分块为什么和原矩阵乘法等价?	49
12.3.	分块如何降低拷贝?	51
12.4.	第二个核心: 存储层级	53
12.5.	第三个核心: 并行粒度	54
12.6.	第四个核心: 访存模式	54
12.7.	Tensor Core 在做什么?	55
12.8.	CUTLASS 中最核心考虑的点	55
12.9.	CPU 上也是同一套思想	56
12.10.	一个总结视角	57
13.	参考资源	57

1. 向量与向量空间

1.1. 向量的定义

定义 1.1.1: 向量（或矢量）是同时具有大小和方向的量。在 n 维实空间中，向量可以表示为：

$$\vec{v} = (v_1, v_2, \dots, v_n)^T \in \mathbb{R}^n \quad (1)$$

向量有两种基本运算：

1. **向量加法**：对应分量相加
2. **数乘**：每个分量乘以标量

例 1.1.1: 设 $\vec{u} = (1, 2, 3)^T$, $\vec{v} = (4, 5, 6)^T$, 则：

$$\vec{u} + \vec{v} = (5, 7, 9)^T \quad (2)$$

$$2\vec{u} = (2, 4, 6)^T \quad (3)$$

1.2. 向量空间

定义 1.2.1: 一个向量空间（或线性空间） V 是在一个域 F （通常为 $\mathbb{R}$ 或 $\mathbb{C}$ ）上的集合，配备加法和数乘运算，满足以下八条公理：

1. 加法交换律： $\vec{u} + \vec{v} = \vec{v} + \vec{u}$
2. 加法结合律： $(\vec{u} + \vec{v}) + \vec{w} = \vec{u} + (\vec{v} + \vec{w})$
3. 存在零向量： $\vec{0} + \vec{v} = \vec{v}$
4. 存在加法逆元： $\vec{v} + (-\vec{v}) = \vec{0}$
5. 数乘结合律： $a(b\vec{v}) = (ab)\vec{v}$
6. 数乘单位元： $1\vec{v} = \vec{v}$
7. 分配律（对向量）： $a(\vec{u} + \vec{v}) = a\vec{u} + a\vec{v}$
8. 分配律（对标量）： $(a + b)\vec{v} = a\vec{v} + b\vec{v}$

1.3. 线性组合与张成空间

定义 1.3.1: 给定向量组 $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_k \in V$ 和标量 $c_1, c_2, \dots, c_k \in F$ ，表达式：

$$c_1\vec{v}_1 + c_2\vec{v}_2 + \dots + c_k\vec{v}_k \quad (4)$$

称为这些向量的**线性组合**。

定义 1.3.2: 向量组 $\vec{v}_1, \dots, \vec{v}_k$ 的**张成空间** (span) 是这些向量所有线性组合能够到达的点的集合:

$$\text{span}\{\vec{v}_1, \dots, \vec{v}_k\} = \{c_1\vec{v}_1 + \dots + c_k\vec{v}_k : c_i \in F\} \quad (5)$$

直观地说, 张成空间就是“这些向量能拼出来的全部空间”。

例 1.3.1: 在 $\mathbb{R}^3$ 中:

- $\vec{e}_1 = (1, 0, 0)^T$ 的张成空间是 x 轴 (一条直线)
- $\vec{e}_1, \vec{e}_2 = (0, 1, 0)^T$ 的张成空间是 xy 平面 (一个二维平面)
- $\vec{e}_1, \vec{e}_2, \vec{e}_3 = (0, 0, 1)^T$ 的张成空间是整个 $\mathbb{R}^3$ (三维空间)

2. 线性相关与线性无关

2.1. 核心概念

定义 2.1.1: 向量组 $\vec{v}_1, \vec{v}_2, \dots, \vec{v}_k$ 称为**线性相关**, 如果存在不全为零的标量 $c_1, c_2, \dots, c_k$, 使得:

$$c_1\vec{v}_1 + c_2\vec{v}_2 + \dots + c_k\vec{v}_k = \vec{0} \quad (6)$$

若只有当 $c_1 = c_2 = \dots = c_k = 0$ 时上式才成立, 则称这些向量**线性无关**。

定理 2.1.1: 向量组线性相关的充要条件是: 其中至少有一个向量可以表示为其余向量的线性组合。

例 2.1.1: 判断 $\vec{v}_1 = (1, 2, 3)^T$, $\vec{v}_2 = (2, 4, 6)^T$ 是否线性相关。

注意到 $\vec{v}_2 = 2\vec{v}_1$, 因此它们是线性相关的。

2.2. 基与维数

定义 2.2.1: 向量空间 V 的一个**基** (basis) 是 V 中一组线性无关的向量, 且 V 中每个向量都可以表示为这组向量的线性组合。

定义 2.2.2: 向量空间 V 的**维数** (dimension)，记作 $\dim(V)$ ，是 V 的任意一个基中向量的个数。

定理 2.2.1: $\mathbb{R}^n$ 的维数为 n 。标准基为：

$$\vec{e}_1 = (1, 0, \dots, 0)^T, \quad \vec{e}_2 = (0, 1, \dots, 0)^T, \quad \dots, \quad \vec{e}_n = (0, 0, \dots, 1)^T \quad (7)$$

3. 矩阵

3.1. 基本定义

定义 3.1.1: 一个 $m \times n$ 的**矩阵** A 是由 $m \times n$ 个数排列成的矩形阵列：

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \quad (8)$$

3.2. 矩阵运算

定义 3.2.1: 矩阵加法： 同型矩阵对应元素相加。

矩阵乘法： 设 A 是 $m \times n$ 矩阵， B 是 $n \times p$ 矩阵，则乘积 AB 是 $m \times p$ 矩阵，其元素为：

$$(AB)_{ij} = \sum_{k=1}^n a_{ik} b_{kj} \quad (9)$$

例 3.2.1:

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \begin{pmatrix} 5 & 6 \\ 7 & 8 \end{pmatrix} = \begin{pmatrix} 1 \times 5 + 2 \times 7 & 1 \times 6 + 2 \times 8 \\ 3 \times 5 + 4 \times 7 & 3 \times 6 + 4 \times 8 \end{pmatrix} = \begin{pmatrix} 19 & 22 \\ 43 & 50 \end{pmatrix} \quad (10)$$

3.3. 关系矩阵乘特征矩阵：加权聚合

在很多数据问题里，矩阵乘法还可以理解为：用“元素之间的关系矩阵”去聚合每个元素的 d 维特征向量。

假设

$$A \in \mathbb{R}^{n \times n} \quad (11)$$

表示元素之间的关系，比如第 i 个元素和第 j 个元素的关系强度是

$$A_{ij} \quad (12)$$

还有一个特征矩阵

$$X \in \mathbb{R}^{n \times d} \quad (13)$$

其中第 j 行

$$X_j \in \mathbb{R}^d \quad (14)$$

表示第 j 个元素的 d 维向量。

那么相乘

$$Y = AX \quad (15)$$

结果是

$$Y \in \mathbb{R}^{n \times d} \quad (16)$$

其中第 i 行是

$$Y_i = \sum_{j=1}^n A_{ij} X_j \quad (17)$$

也就是说：第 i 个元素的新向量，是所有元素向量的加权和，权重来自第 i 行的关系矩阵。

例 3.3.1: 如果 A_{ij} 表示“第 i 个元素应该从第 j 个元素那里吸收多少信息”，那么 Y_i 就是第 i 个元素聚合了其他元素信息之后的新表示。

例如

$$A = \begin{pmatrix} 0.1 & 0.7 & 0.2 \\ 0.5 & 0.3 & 0.2 \\ 0.0 & 0.4 & 0.6 \end{pmatrix} \quad (18)$$

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \quad (19)$$

那么

$$Y_1 = 0.1x_1 + 0.7x_2 + 0.2x_3 \quad (20)$$

这表示第 1 个元素的新表示主要来自第 2 个元素，因为权重 0.7 最大。

这个操作在很多模型里都很常见：

1. 在图神经网络 GNN 里， A 是邻接矩阵或归一化邻接矩阵， AX 表示每个节点聚合邻居节点的特征。
2. 在 Attention 里， A 是 attention score 或 attention probability， X 是 value 向量， AX 表示每个 token 根据注意力权重聚合其他 token 的信息。
3. 在线性代数里，这就是对 X 的行向量做线性组合。 A 的每一行决定输出中对应一行如何由输入的所有行组合得到。

一句话总结：

$$(n \times n) \cdot (n \times d) \quad (21)$$

的意义是：用 n 个元素之间的关系，对 n 个 d 维向量进行加权聚合，得到新的 n 个 d 维向量。

3.4. 为什么不是对应元素相乘？

初学者常问：矩阵乘法为什么不能像加法那样，直接把对应位置的元素相乘？

定义 3.4.1: 对应元素相乘的运算确实存在，称为**阿达马积**（Hadamard product），记作 $A \odot B$ 。但这不是线性代数中的标准“矩阵乘法”。

标准矩阵乘法定义为“第 i 行乘第 j 列”，核心原因是：**矩阵的本质是线性变换，矩阵乘法的本质是线性变换的复合。**

例 3.4.1: 假设矩阵 A 表示“旋转 90 度”，矩阵 B 表示“拉伸 2 倍”。那么 AB 表示“先拉伸 2 倍，再旋转 90 度”。

如果按对应元素相乘，这种“先做 B 再做 A ”的组合意义就丢失了。行乘列的定义保证了：

$$(AB)\vec{v} = A(B\vec{v}) \quad (22)$$

即“对向量先施加 B ，再施加 A ”，与变换的复合顺序完全一致。

例 3.4.2: 用具体矩阵验证上述直觉。在 $\mathbb{R}^2$ 中：

- 旋转 90 度的矩阵： $R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$
- 沿两个坐标轴各拉伸 2 倍的矩阵： $S = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$

按行乘列计算 RS （先拉伸，再旋转）：

$$RS = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 0 \times 2 + (-1) \times 0 & 0 \times 0 + (-1) \times 2 \\ 1 \times 2 + 0 \times 0 & 1 \times 0 + 0 \times 2 \end{pmatrix} = \begin{pmatrix} 0 & -2 \\ 2 & 0 \end{pmatrix} \quad (23)$$

取向量 $\vec{v} = (1, 0)^T$ ，验证变换复合：

1. 先拉伸： $S\vec{v} = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}$
2. 再旋转： $R(S\vec{v}) = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix}$

直接用复合矩阵计算：

$$(RS)\vec{v} = \begin{pmatrix} 0 & -2 \\ 2 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \quad (24)$$

两种方法结果完全相同，说明行乘列的定义确实正确描述了“先拉伸、再旋转”这一复合变换。

例 3.4.3: 作为对比, 如果按对应元素相乘 (阿达马积):

$$R \odot S = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad (25)$$

结果变成了零矩阵! 任何向量经过这个“运算”都会变成零向量, 旋转和拉伸的意义完全消失。这就是对应元素相乘不能作为标准矩阵乘法的原因——它无法描述线性变换的复合。

定理 3.4.1: 矩阵乘法满足结合律: $(AB)C = A(BC)$, 但一般不满足交换律: $AB \neq BA$ 。

结合律的成立也依赖于“行乘列”的定义——它保证了三个变换连续作用时, 顺序可以任意分组。

3.5. 特殊矩阵

定义 3.5.1:

- **单位矩阵** I_n : 对角线为 1, 其余为 0 的 $n \times n$ 矩阵
- **对角矩阵**: 非对角元素全为 0 的矩阵
- **转置矩阵** A^T : 将 A 的行列互换得到的矩阵
- **对称矩阵**: 满足 $A = A^T$ 的矩阵
- **可逆矩阵**: 存在 A^{-1} 使得 $AA^{-1} = A^{-1}A = I$

3.6. 矩阵的秩

定义 3.6.1: 设 A 是一个 $m \times n$ 矩阵, 列向量为

$$\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n \in \mathbb{R}^m \quad (26)$$

。

A 的**列空间** (column space) 定义为所有列向量的张成空间:

$$\text{Col}(A) = \text{span}\{\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n\} \subset \mathbb{R}^m \quad (27)$$

定义 3.6.2: 矩阵 A 的**秩** (rank) 定义为列空间的维数, 记作:

$$\text{rank}(A) = \dim(\text{Col}(A)) \quad (28)$$

也就是说, $\text{rank}(A)$ 是 A 的列向量中最多能取出多少个线性无关向量。

定理 3.6.1: 矩阵的列秩等于行秩。也就是说，如果把 A 的行向量张成的空间记为 $\text{Row}(A)$ ，那么：

$$\dim(\text{Col}(A)) = \dim(\text{Row}(A)) \quad (29)$$

因此也可以把 $\text{rank}(A)$ 理解为行空间的维数。

直观地说， $\text{rank}(A)$ 衡量的是矩阵 A 真正包含多少个独立方向的信息。这个概念不是由行阶梯形矩阵定义出来的；行阶梯形只是计算它的一种方法。

如果把 A 看成线性变换 $\vec{x} \rightarrow A\vec{x}$ ，那么 $\text{rank}(A)$ 就是这个变换的像空间维数：

$$\text{rank}(A) = \dim(\text{Im}(A)) \quad (30)$$

也就是说， $\text{rank}(A)$ 描述 $A\vec{x}$ 最多能扫出几维空间：

- $\text{rank}(A) = 0$ ：所有向量都被压到零向量。
- $\text{rank}(A) = 1$ ：结果只能落在一条直线上。
- $\text{rank}(A) = 2$ ：结果能扫出一个平面。
- $\text{rank}(A) = n$ ：在 n 维空间中没有损失维度。

定理 3.6.2: 计算 $\text{rank}(A)$ 的常用方法是对 A 做初等行变换，把它化成行阶梯形矩阵。行阶梯形矩阵中非零行的个数，就是 $\text{rank}(A)$ 。

定理 3.6.3: 初等行变换不改变矩阵的行空间，因此不改变矩阵的行秩。由列秩等于行秩可知，它也不改变矩阵的秩。

证明：设矩阵 A 的行向量为 $\vec{r}_1, \vec{r}_2, \dots, \vec{r}_m$ ，做一次初等行变换后得到矩阵 B 。只要证明 $\text{Row}(A) = \text{Row}(B)$ ，就有

$$\dim(\text{Row}(A)) = \dim(\text{Row}(B)) \quad (31)$$

分三种初等行变换讨论。

1. 交换两行：只是改变行向量的排列顺序，行向量张成的空间不变。
2. 某一行乘以非零数 c ：若新行向量为 $\vec{r}'_i = c\vec{r}_i$ ，则 $\vec{r}'_i$ 是旧行向量的线性组合；反过来，因为 $c \neq 0$ ，有 $\vec{r}_i = (\frac{1}{c})\vec{r}'_i$ ，所以旧行向量也是新行向量的线性组合。因此行空间不变。
3. 某一行加上另一行的 c 倍：若把第 i 行改为

$$\vec{r}'_i = \vec{r}_i + c\vec{r}_j \quad (i \neq j) \quad (32)$$

其余行不变。显然 $\vec{r}'_i$ 是旧行向量的线性组合，所以 $\text{Row}(B) \subset \text{Row}(A)$ 。反过来，

$$\vec{r}_i = \vec{r}'_i - c\vec{r}_j \quad (33)$$

而 $\vec{r}_j$ 在新矩阵中没有改变，所以旧的第 i 行也是新行向量的线性组合，其余旧行本来就等于对应的新行。因此 $\text{Row}(A) \subset \text{Row}(B)$ 。

三种情况下都有 $\text{Row}(A) = \text{Row}(B)$ 。连续做多次初等行变换时，每一步都保持行空间不变，所以最终行秩不变。又因为前面已经知道列秩等于行秩，所以矩阵的秩也不变。 ■

定理 3.6.4: 对 $n \times n$ 方阵 A ，以下说法等价：

1. A 可逆
2. $\text{rank}(A) = n$
3. 齐次方程 $A\vec{x} = \vec{0}$ 只有零解

证明：把 A 做初等行变换化成行阶梯形矩阵。

若 $\text{rank}(A) = n$ ，说明每一列都有主元，没有自由变量。于是齐次方程

$$A\vec{x} = \vec{0} \quad (34)$$

只能有零解。同时，对任意 $\vec{b}$ ，方程 $A\vec{x} = \vec{b}$ 都能唯一解出所有未知数，所以 A 对应的线性变换没有丢失任何方向，可以逆回来，因此 A 可逆。

若 $\text{rank}(A) < n$ ，说明主元个数少于未知数个数，至少有一个自由变量。于是齐次方程

$$A\vec{x} = \vec{0} \quad (35)$$

有非零解。也就是说，存在 $\vec{v} \neq \vec{0}$ 使得 $A\vec{v} = \vec{0}$ 。这表示 A 把非零方向压成了零方向，信息已经丢失，不可能存在逆变换，因此 A 不可逆。 ■

例 3.6.1: 设

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{pmatrix} \quad (36)$$

第二行是第一行的 2 倍，所以它没有提供新的独立信息。做初等行变换：

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 6 \\ 1 & 1 & 1 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 2 & 3 \\ 0 & 0 & 0 \\ 0 & -1 & -2 \end{pmatrix} \quad (37)$$

行阶梯形中有 2 个非零行，因此：

$$\text{rank}(A) = 2 \quad (38)$$

4. 线性方程组

4.1. 矩阵表示

m 个方程、 n 个未知数的线性方程组可以写成矩阵形式：

$$A\vec{x} = \vec{b} \quad (39)$$

其中 A 是 $m \times n$ 系数矩阵， $\vec{x}$ 是未知数向量， $\vec{b}$ 是常数向量。

4.2. 解的存在性

定理 4.2.1: 线性方程组 $A\vec{x} = \vec{b}$ 有解的充要条件是:

$$\text{rank}(A) = \text{rank}(A | \vec{b}) \quad (40)$$

其中 $(A | \vec{b})$ 是增广矩阵。

定理 4.2.2: 对于 $n \times n$ 线性方程组 $A\vec{x} = \vec{b}$:

1. 若 $\text{rank}(A) = n$, 则 A 可逆, 方程组有唯一解 $\vec{x} = A^{-1}\vec{b}$
2. 若 $\text{rank}(A) < n$ 且 $\text{rank}(A) = \text{rank}(A | \vec{b})$, 则有无穷多解
3. 若 $\text{rank}(A) < \text{rank}(A | \vec{b})$, 则无解

证明: 分三种情况证明。

1. 若 $\text{rank}(A) = n$, 由前面的方阵可逆判别定理可知 A 可逆。于是可在方程

$$A\vec{x} = \vec{b} \quad (41)$$

两边左乘 A^{-1} , 得到:

$$A^{-1}A\vec{x} = A^{-1}\vec{b} \quad (42)$$

因为 $A^{-1}A = I$, 所以:

$$\vec{x} = A^{-1}\vec{b} \quad (43)$$

这说明解存在。并且如果 $\vec{x}_1$ 和 $\vec{x}_2$ 都是解, 则:

$$A\vec{x}_1 = \vec{b}, \quad A\vec{x}_2 = \vec{b} \quad (44)$$

两式相减:

$$A(\vec{x}_1 - \vec{x}_2) = \vec{0} \quad (45)$$

左乘 A^{-1} 得:

$$\vec{x}_1 - \vec{x}_2 = \vec{0} \quad (46)$$

因此 $\vec{x}_1 = \vec{x}_2$, 解唯一。

2. 若 $\text{rank}(A) < n$ 且 $\text{rank}(A) = \text{rank}(A | \vec{b})$, 由上一个定理可知方程组有解。设其中一个解为 $\vec{x}_0$ 。

由于 $\text{rank}(A) < n$, 把 A 行化简后至少有一个非主元变量, 也就是至少有一个自由变量。因此齐次方程

$$A\vec{y} = \vec{0} \quad (47)$$

有非零解。记某个非零解为 $\vec{y}_0$ 。

那么对任意标量 t , 都有:

$$A(\vec{x}_0 + t\vec{y}_0) = A\vec{x}_0 + tA\vec{y}_0 = \vec{b} + t\vec{0} = \vec{b} \quad (48)$$

因此 $\vec{x}_0 + t\vec{y}_0$ 都是原方程组的解。 t 可以取无穷多个值，所以方程组有无穷多解。

3. 若 $\text{rank}(A) < \text{rank}(A \mid \vec{b})$ ，由上一个定理

$$\text{rank}(A) = \text{rank}(A \mid \vec{b}) \quad (49)$$

是方程组有解的充要条件，因此这里条件不满足，方程组无解。

■

5. 行列式

5.1. 行列式到底想衡量什么？

行列式不是为了制造一个复杂公式而发明的。它最核心的目标是：给一个方阵 A ，用一个数描述这个线性变换对空间“体积”的缩放比例。

在二维中，“体积”就是面积。单位正方形由两个基向量围成：

$$\vec{e}_1 = (1, 0)^T, \quad \vec{e}_2 = (0, 1)^T \quad (50)$$

经过矩阵 A 作用后，两个基向量变成 $A\vec{e}_1$ 和 $A\vec{e}_2$ 。单位正方形也随之变成一个平行四边形。这个平行四边形的有向面积，就是 $\det(A)$ 。

在三维中，单位立方体经过 A 作用后会变成一个平行六面体。这个平行六面体的有向体积，就是 $\det(A)$ 。更高维时也一样：行列式衡量的是单位超立方体经过线性变换后的有向体积。

这里的“有向”很重要：

1. 若 $\det(A) > 0$ ，说明体积被放大或缩小，但空间方向没有翻转
2. 若 $\det(A) < 0$ ，说明体积缩放的同时，空间方向发生了翻转
3. 若 $\det(A) = 0$ ，说明体积被压成了 0，空间被压扁到低维，矩阵不可逆

例 5.1.1: 对角矩阵

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \quad (51)$$

它把 x 方向拉伸 2 倍，把 y 方向拉伸 3 倍。单位正方形变成宽 2、高 3 的矩形，面积变成 6，所以：

$$\det(A) = 6 \quad (52)$$

这正好等于两个方向缩放比例的乘积。

例 5.1.2: 矩阵

$$B = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad (53)$$

它把所有向量投影到 x 轴上。单位正方形被压成一条线段，面积变成 0，所以：

$$\det(B) = 0 \quad (54)$$

这说明 B 不可逆：二维信息已经丢失，只剩下一维，无法恢复原来的点。

5.2. 为什么二阶行列式是 $ad - bc$?

设

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (55)$$

矩阵的两列就是两个基向量变换后的结果：

$$A\vec{e}_1 = (a, c)^T, \quad A\vec{e}_2 = (b, d)^T \quad (56)$$

因此 $\det(A)$ 应该是由向量 $(a, c)^T$ 和 $(b, d)^T$ 围成的平行四边形的有向面积。

这个面积可以这样理解：如果两个向量是 $(a, 0)^T$ 和 $(0, d)^T$ ，它们围成的矩形面积是 ad 。但一般情况下，两个向量会倾斜，倾斜带来的“斜切部分”需要扣掉，扣掉的正是 bc 。最后得到：

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc \quad (57)$$

也可以从列变换看得更清楚。对平行四边形来说，把第二列加上第一列的倍数，只是在水平方向做剪切，不改变面积。因此可以把

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (58)$$

通过不改变面积的列操作化成一个更容易看面积的形状；最后留下来的有效高度正好对应 $d - \left(\frac{b}{a}\right)c$ （先假设 $a \neq 0$ ），面积就是：

$$a \cdot \left(d - \left(\frac{b}{a} \right) c \right) = ad - bc \quad (59)$$

当 $a = 0$ 时，也可以交换列或改用另一列做同样分析，最终仍然得到同一个公式。负号记录的是方向，而绝对值 $|ad - bc|$ 才是普通面积。

这个公式也可以从几个基本要求推出。行列式作为面积缩放因子，至少应该满足：

1. 单位矩阵不改变面积，所以 $\det(I) = 1$
2. 如果一列向量被拉伸 k 倍，面积也拉伸 k 倍
3. 如果两列向量相同，平行四边形退化成线段，面积为 0
4. 交换两列会把平行四边形方向反过来，所以行列式变号

对二阶矩阵，这些要求会唯一地推出 $ad - bc$ 。所以 $ad - bc$ 不是随便凑出来的公式，而是“有向面积”这个几何对象在坐标里的表达。

例 5.2.1: 设

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad (60)$$

两列向量是 $(1, 3)^T$ 和 $(2, 4)^T$ 。行列式为：

$$\det(A) = 1 \cdot 4 - 2 \cdot 3 = -2 \quad (61)$$

绝对值 2 表示单位正方形的面积被放大为 2；负号表示方向被翻转。

5.3. 定义

定义 5.3.1: $n \times n$ 矩阵 A 的**行列式** $\det(A)$ 或 $|A|$ 是一个标量，递归定义为：

当 $n = 1$ 时， $\det(A) = a_{11}$ 。

当 $n > 1$ 时：

$$\det(A) = \sum_{j=1}^n (-1)^{1+j} a_{1j} \det(A_{1j}) \quad (62)$$

其中 A_{1j} 是删除第 1 行第 j 列后得到的子矩阵。

这个递归定义叫做**按第一行展开**。它看起来形式复杂，但思想并不神秘：高维体积可以分解成若干个低一维体积的组合。 a_{1j} 给出第一行第 j 个方向上的系数， $\det(A_{1j})$ 给出删掉这一行和这一列后剩余部分的体积缩放，符号 $(-1)^{1+j}$ 用来记录方向翻转。

也就是说，递归公式不是行列式的本质，“有向体积缩放”才是本质。递归公式只是一个方便计算的坐标表达。

定理 5.3.1: 从上面的定义可以推出，行列式作为列向量的函数，具有三条核心性质：

1. **对每一列线性**：固定其他列时，某一列的线性组合可以从行列式中拆出来。
2. **交错性**：如果两列相同，则行列式为 0；交换两列，行列式变号。
3. **归一化**： $\det(I) = 1$ 。

这三条性质比递归公式更接近行列式的本质：行列式是由列向量围成的有向体积。下面讨论 $\det(A)$ 和 $\text{rank}(A)$ 的关系时，主要用的就是这三条性质。

5.4. 为什么交换两列会让行列式变号？

先把递归定义展开成一个更适合证明性质的形式。设 S_n 是 $1, 2, \dots, n$ 的所有排列组成的集合， $\text{sgn}(\sigma)$ 表示排列 σ 的符号：偶排列为 1，奇排列为 -1。

这里的**排列**，就是把 $1, 2, \dots, n$ 重新排一个顺序。例如 $n = 3$ 时， $(2, 3, 1)$ 就是一个排列，它表示：

$$\sigma(1) = 2, \quad \sigma(2) = 3, \quad \sigma(3) = 1 \quad (63)$$

排列中如果有一对位置 $i < j$ ，但前面的数反而更大，也就是 $\sigma(i) > \sigma(j)$ ，就叫做一个**逆序**。逆序个数为偶数的排列叫**偶排列**，逆序个数为奇数的排列叫**奇排列**。

例如 $(2, 3, 1)$ 里有两个逆序：

$$2 > 1, \quad 3 > 1 \quad (64)$$

所以它是偶排列， $\text{sgn}(\sigma) = 1$ 。而 $(2, 1, 3)$ 里只有一个逆序 $2 > 1$ ，所以它是奇排列， $\text{sgn}(\sigma) = -1$ 。

定理 5.4.1: 由按第一行展开的递归定义，可以推出行列式的排列公式：

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n a_{r, \sigma(r)} \quad (65)$$

这个公式的含义是：计算行列式时，要从矩阵的每一行各选一个元素，并且这些元素不能来自同一列。每一种选法都对应一个排列 σ 。

具体地说， $\sigma(r)$ 表示第 r 行选第几列。所以

$$\prod_{r=1}^n a_{r, \sigma(r)} \quad (66)$$

就是把这些选出来的元素相乘：

$$a_{1, \sigma(1)} a_{2, \sigma(2)} \cdots a_{n, \sigma(n)} \quad (67)$$

因为 σ 是排列， $\sigma(1), \sigma(2), \dots, \sigma(n)$ 恰好是 $1, 2, \dots, n$ 的重新排序，所以每一列也恰好被选一次。前面的 $\text{sgn}(\sigma)$ 决定这一项前面带正号还是负号：偶排列取正号，奇排列取负号。最后对所有排列 $\sigma \in S_n$ 求和，就得到整个行列式。

以二阶矩阵为例， S_2 只有两个排列：

1. $\sigma = (1, 2)$ 是偶排列，对应选 a_{11} 和 a_{22} ，得到正项 $a_{11}a_{22}$ 。
2. $\sigma = (2, 1)$ 是奇排列，对应选 a_{12} 和 a_{21} ，得到负项 $-a_{12}a_{21}$ 。

所以：

$$\det \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} = a_{11}a_{22} - a_{12}a_{21} \quad (68)$$

证明：这个公式可以用归纳法从递归定义推出。

当 $n = 1$ 时， S_1 只有一个排列，公式就是 $\det(A) = a_{11}$ ，和定义一致。

假设 $n - 1$ 阶矩阵已经满足排列公式。对 $n \times n$ 矩阵按第一行展开：

$$\det(A) = \sum_{j=1}^n (-1)^{1+j} a_{1j} \det(A_{1j}) \quad (69)$$

对每个子矩阵 A_{1j} 使用归纳假设。子矩阵的每一项，等价于在剩下的 $n - 1$ 行和剩下的 $n - 1$ 列之间选一个排列；再加上第一行已经选了第 j 列，就得到原矩阵的一个完整排列 σ ，并且满足 $\sigma(1) = j$ 。

系数 $(-1)^{1+j}$ 正好记录“把第 j 列移到第一列”带来的符号变化，子矩阵内部的 sgn 记录剩余列的排列符号。两部分相乘，正好得到完整排列 σ 的符号 $\text{sgn}(\sigma)$ 。

当 j 从 1 到 n 变化时，所有排列都被恰好枚举一次。因此递归定义等价于：

$$\det(A) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n a_{r, \sigma(r)} \quad (70)$$

■

现在用这个公式证明“交换两列会变号”。

设 B 是把 A 的第 p 列和第 q 列交换后得到的矩阵。令 τ 表示交换 p 和 q 的对换。则 B 的第 c 列等于 A 的第 $\tau(c)$ 列，所以：

$$b_{rc} = a_{r, \tau(c)} \quad (71)$$

由排列公式：

$$\det(B) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n b_{r, \sigma(r)} \quad (72)$$

代入 $b_{rc} = a_{r, \tau(c)}$ ：

$$\det(B) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n a_{r, \tau(\sigma(r))} \quad (73)$$

令

$$\rho(r) = \tau(\sigma(r)) \quad (74)$$

也就是先按 σ 选列，再交换 p 、 q 两列。因为 τ 是一次对换，所以它是奇排列：

$$\text{sgn}(\tau) = -1 \quad (75)$$

因此：

$$\text{sgn}(\rho) = \text{sgn}(\tau) \text{sgn}(\sigma) = -\text{sgn}(\sigma) \quad (76)$$

也就是：

$$\text{sgn}(\sigma) = -\text{sgn}(\rho) \quad (77)$$

当 σ 跑遍 S_n 时， $\rho = \tau\sigma$ 也刚好跑遍 S_n 。所以上面的求和可以改名为对 ρ 求和：

$$\det(B) = - \sum_{\rho \in S_n} \text{sgn}(\rho) \prod_{r=1}^n a_{r, \rho(r)} = -\det(A) \quad (78)$$

这就从行列式的递归定义出发，严格证明了：交换两列，行列式变号。

二维公式可以看作这个结论的最小例子。设

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (79)$$

交换两列得到：

$$A' = \begin{pmatrix} b & a \\ d & c \end{pmatrix} \quad (80)$$

于是：

$$\det(A') = bc - ad = -(ad - bc) = -\det(A) \quad (81)$$

几何上，这是因为行列式不是普通面积，而是**有向面积**。两个列向量的顺序决定了平行四边形的方向：从第一列转向第二列，如果方向是正的，行列式为正；交换两列后，转向顺序反过来，面积大小不变，但方向相反，所以符号变号。

高维中也是同一个道理。 $n \times n$ 矩阵的列向量按顺序围成一个有向 n 维体积。交换其中两列，不会改变这个平行体的大小，但会把这组有序方向的取向反过来。因此有向体积变号：

$$\det(\dots, \vec{a}_i, \dots, \vec{a}_j, \dots) = -\det(\dots, \vec{a}_j, \dots, \vec{a}_i, \dots) \quad (82)$$

这就是行列式交错性的来源。后面说“有两列相同则行列式为 0”，用的正是这个性质。

5.5. 为什么交换两行也会让行列式变号？

交换两行的证明和交换两列很像，但下标变化的位置不同。交换两列时，是“每一行选到的列号”被换掉；交换两行时，是“同一个选列方案中，哪一行拿哪一个因子”被换掉。

设 B 是把 A 的第 p 行和第 q 行交换后得到的矩阵。仍令 τ 表示交换 p 和 q 的对换。则 B 的第 r 行等于 A 的第 $\tau(r)$ 行，所以：

$$b_{rc} = a_{\tau(r),c} \quad (83)$$

由排列公式：

$$\det(B) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n b_{r,\sigma(r)} \quad (84)$$

代入 $b_{rc} = a_{\tau(r),c}$ ：

$$\det(B) = \sum_{\sigma \in S_n} \text{sgn}(\sigma) \prod_{r=1}^n a_{\tau(r),\sigma(r)} \quad (85)$$

现在把乘积里的行下标重新命名。令

$$s = \tau(r) \quad (86)$$

因为 τ 只是交换两个编号，所以当 r 从 1 到 n 跑一遍时， s 也从 1 到 n 跑一遍。并且 $\tau^{-1} = \tau$ ，所以 $r = \tau(s)$ 。于是

$$\prod_{r=1}^n a_{\tau(r),\sigma(r)} = \prod_{s=1}^n a_{s,\sigma(\tau(s))} \quad (87)$$

令

$$\rho(s) = \sigma(\tau(s)) \quad (88)$$

也就是先交换第 p 、 q 两个行号，再按原来的 σ 去选列。因为 τ 是一次对换，所以

$$\text{sgn}(\tau) = -1 \quad (89)$$

因此：

$$\operatorname{sgn}(\rho) = \operatorname{sgn}(\sigma) \operatorname{sgn}(\tau) = -\operatorname{sgn}(\sigma) \quad (90)$$

也就是：

$$\operatorname{sgn}(\sigma) = -\operatorname{sgn}(\rho) \quad (91)$$

当 σ 跑遍 S_n 时， $\rho = \sigma\tau$ 也刚好跑遍 S_n 。所以：

$$\det(B) = - \sum_{\rho \in S_n} \operatorname{sgn}(\rho) \prod_{s=1}^n a_{s, \rho(s)} = -\det(A) \quad (92)$$

这就严格证明了：交换两行，行列式也会变号。

用二阶矩阵看这个结论最直接。设

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (93)$$

交换两行得到：

$$B = \begin{pmatrix} c & d \\ a & b \end{pmatrix} \quad (94)$$

于是：

$$\det(B) = cb - da = -(ad - bc) = -\det(A) \quad (95)$$

还有一种更短的理解方式。交换 A 的两行，等价于先看转置矩阵 A^T ，再交换 A^T 的两列，最后再转置回来。由于 $\det(A^T) = \det(A)$ ，而交换两列已经证明会变号，所以交换两行也一定会变号。

5.6. 为什么 $\det(AB) = \det(A) \det(B)$ ？

如果把矩阵看成线性变换， AB 表示先做 B ，再做 A 。

假设 B 先把单位体积缩放了 $\det(B)$ 倍，随后 A 又把体积缩放了 $\det(A)$ 倍，那么总缩放比例自然就是：

$$\det(A) \cdot \det(B) \quad (96)$$

因此：

$$\det(AB) = \det(A) \det(B) \quad (97)$$

这个性质是行列式作为“体积缩放因子”最重要的证据之一。它说明行列式和线性变换复合完全兼容。

例 5.6.1: 设

$$S = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix}, \quad R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad (98)$$

S 把面积放大 6 倍, 所以 $\det(S) = 6$ 。 R 是旋转 90 度, 不改变面积, 也不翻转方向, 所以 $\det(R) = 1$ 。

复合变换 RS 先拉伸再旋转, 面积仍然只被放大 6 倍:

$$\det(RS) = \det(R) \det(S) = 1 \cdot 6 = 6 \quad (99)$$

5.7. 基本性质

定理 5.7.1: 行列式具有以下性质:

1. $\det(I) = 1$
2. $\det(A^T) = \det(A)$
3. $\det(AB) = \det(A) \cdot \det(B)$
4. 若 A 可逆, $\det(A^{-1}) = \frac{1}{\det(A)}$
5. 交换两行 (列), 行列式变号
6. 某行 (列) 乘以 k , 行列式乘以 k
7. 某行 (列) 加上另一行 (列) 的 k 倍, 行列式不变

例 5.7.1: 二阶行列式:

$$\det \begin{pmatrix} a & b \\ c & d \end{pmatrix} = ad - bc \quad (100)$$

三阶行列式 (萨吕法则):

$$\det \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} = aei + bfg + cdh - ceg - afh - bdi \quad (101)$$

5.8. 秩和行列式的严格关系

定理 5.8.1: 对 $n \times n$ 方阵 A , 有:

$$\det(A) \neq 0 \Leftrightarrow \text{rank}(A) = n \quad (102)$$

等价地:

$$\det(A) = 0 \Leftrightarrow \text{rank}(A) < n \quad (103)$$

证明: 对 A 做初等行变换, 把它化成行阶梯形矩阵 R 。每一次初等行变换都等价于左乘一个初等矩阵, 所以存在初等矩阵 $E_1, E_2, \dots, E_k$, 使得

$$R = E_k \dots E_2 E_1 A \quad (104)$$

初等行变换不改变矩阵的行秩, 因此也不改变矩阵的秩:

$$\text{rank}(R) = \text{rank}(A) \quad (105)$$

另一方面, 每个初等矩阵都可逆, 并且行列式非零。具体地说, 交换两行对应行列式乘以 -1 , 某行乘以非零数 c 对应行列式乘以 c , 某行加上另一行的 c 倍对应行列式不变。因此

$$\det(E_i) \neq 0 \quad (i = 1, 2, \dots, k) \quad (106)$$

由 $\det(BC) = \det(B)\det(C)$ 得:

$$\det(R) = \det(E_k) \dots \det(E_1) \det(A) \quad (107)$$

因为 $\det(E_k) \dots \det(E_1) \neq 0$, 所以

$$\det(R) = 0 \Leftrightarrow \det(A) = 0 \quad (108)$$

现在只需要分析行阶梯形矩阵 R 。

若 $\text{rank}(A) = n$, 则 $\text{rank}(R) = n$ 。因为 R 是 $n \times n$ 行阶梯形矩阵, 它有 n 个非零行, 也就有 n 个主元。共有 n 列, 所以每一列都是主元列, R 是对角线元素全非零的上三角矩阵。因此

$$\det(R) = r_{11}r_{22} \dots r_{nn} \neq 0 \quad (109)$$

从而 $\det(A) \neq 0$ 。

若 $\text{rank}(A) < n$, 则 $\text{rank}(R) < n$ 。行阶梯形矩阵 R 的非零行少于 n 行, 所以至少有一行是零行。含有零行的方阵行列式为 0, 因此 $\det(R) = 0$, 从而 $\det(A) = 0$ 。

所以

$$\text{rank}(A) = n \Leftrightarrow \det(A) \neq 0 \quad (110)$$

也等价于

$$\text{rank}(A) < n \Leftrightarrow \det(A) = 0 \quad (111)$$

■

也可以直接从行列式的定义性质证明这个结论。

证明: 设

$$A = [\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n] \quad (112)$$

其中 $\vec{a}_1, \dots, \vec{a}_n$ 是 A 的列向量。

先证明

$$\text{rank}(A) < n \Rightarrow \det(A) = 0 \quad (113)$$

由 $\text{rank}(A) < n$ 可知列向量线性相关。于是存在不全为 0 的数 $c_1, \dots, c_n$, 使得

$$c_1 \vec{a}_1 + \dots + c_n \vec{a}_n = \vec{0} \quad (114)$$

取一个 $c_j \neq 0$, 则

$$\vec{a}_j = - \sum_{i \neq j} \left(\frac{c_i}{c_j} \right) \vec{a}_i \quad (115)$$

由于行列式对第 j 列线性,

$$\det(\vec{a}_1, \dots, \vec{a}_j, \dots, \vec{a}_n) = - \sum_{i \neq j} \left(\frac{c_i}{c_j} \right) \det(\vec{a}_1, \dots, \vec{a}_i, \dots, \vec{a}_n) \quad (116)$$

右边每一项里, 第 j 列被替换成了 $\vec{a}_i$, 而原来的第 i 列仍然也是 $\vec{a}_i$ 。所以每一项对应的矩阵都有两列完全相同。

这里用到的**交错性**是行列式的一个基本性质: 如果交换矩阵的两列, 行列式会变号。也就是说, 若 M' 是把 M 的两列交换后得到的矩阵, 则:

$$\det(M') = -\det(M) \quad (117)$$

现在假设 M 有两列相同, 比如第 i 列和第 j 列相同。交换这两列以后, 矩阵看起来没有任何变化, 所以 $M' = M$ 。于是同一个矩阵的行列式一方面等于自己, 另一方面又要变号:

$$\det(M) = \det(M') = -\det(M) \quad (118)$$

因此:

$$2 \det(M) = 0 \quad (119)$$

在实数或复数里, 2 不是 0, 所以只能有:

$$\det(M) = 0 \quad (120)$$

这就是“有两列相同, 由交错性可知行列式为 0”的完整含义。

因此, 右边求和中的每一项都是 0, 整个右边也是 0。于是

$$\det(A) = 0 \quad (121)$$

这证明了

$$\text{rank}(A) < n \Rightarrow \det(A) = 0 \quad (122)$$

其逆否命题就是

$$\det(A) \neq 0 \Rightarrow \text{rank}(A) = n \quad (123)$$

再证明

$$\text{rank}(A) = n \Rightarrow \det(A) \neq 0 \quad (124)$$

$\text{rank}(A) = n$ 表示 $\vec{a}_1, \dots, \vec{a}_n$ 是 $\mathbb{R}^n$ 的一组基。也就是说, 可以通过初等列变换把这组基变成标准基 $\vec{e}_1, \dots, \vec{e}_n$, 矩阵也就被化成单位矩阵 I 。

这几类初等列变换对行列式的影响都来自上面的定义性质:

1. 交换两列: 行列式变号。

2. 某列乘以非零数 c : 行列式乘以 c 。
3. 某列加上另一列的 c 倍: 由列线性和交错性可知, 行列式不变。

因此, 从 A 到 I 的整个过程中, 行列式只会乘上一些非零常数, 或者保持不变。设总的非零倍数为 k , 则

$$\det(I) = k \det(A) \quad (125)$$

但 $\det(I) = 1$, 且 $k \neq 0$, 所以 $\det(A) \neq 0$ 。

两个方向合起来得到

$$\text{rank}(A) = n \Leftrightarrow \det(A) \neq 0 \quad (126)$$

■

这条结论只适用于方阵, 因为 $\det(A)$ 只对方阵定义; 而 $\text{rank}(A)$ 对任意矩阵都有定义。

5.9. 为什么 $\det(A) \neq 0$ 能推出 A 可逆?

前面已经证明了:

$$\det(A) \neq 0 \Rightarrow \text{rank}(A) = n \quad (127)$$

现在还需要严格说明: 对 $n \times n$ 方阵, $\text{rank}(A) = n$ 为什么能推出 A 可逆。

定理 5.9.1: 若 A 是 $n \times n$ 方阵, 且 $\det(A) \neq 0$, 则 A 可逆。

证明: 由 $\det(A) \neq 0$ 可知 $\text{rank}(A) = n$ 。因此对 A 做初等行变换时, 每一列都能找到主元, 最终可以把 A 化成单位矩阵 I 。

每一次初等行变换都等价于左乘一个初等矩阵。设这些初等矩阵为 $E_1, E_2, \dots, E_k$, 则有:

$$E_k \dots E_2 E_1 A = I \quad (128)$$

记

$$B = E_k \dots E_2 E_1 \quad (129)$$

则

$$BA = I \quad (130)$$

还需要说明 B 本身可逆。每个初等矩阵都可逆:

1. 交换两行, 再交换一次就变回来。
2. 某行乘以非零数 c , 再乘以 $\frac{1}{c}$ 就变回来。
3. 某行加上另一行的 c 倍, 再加上另一行的 $-c$ 倍就变回来。

所以 $E_1, E_2, \dots, E_k$ 都可逆, 它们的乘积 B 也可逆。

由 $BA = I$, 两边左乘 B^{-1} :

$$B^{-1}BA = B^{-1}I \quad (131)$$

得到:

$$A = B^{-1} \quad (132)$$

因此 $B = A^{-1}$ ，也就是说 A 可逆。 ■

这个证明也解释了高斯消元求逆矩阵的原理：如果 A 能通过初等行变换化成 I ，那么这些行变换的总效果就是 A^{-1} 。实际计算时把增广矩阵 $(A | I)$ 化成 $(I | A^{-1})$ ，就是在同时记录这个矩阵 B 。

5.10. 齐次方程组与零空间

现在把线性方程组和行列式联系起来。若线性方程组右边是零向量：

$$A\vec{x} = \vec{0} \quad (133)$$

这样的方程组称为**齐次线性方程组**。

齐次方程组一定至少有一个解： $\vec{x} = \vec{0}$ 。因为任何矩阵乘以零向量都会得到零向量。这个解叫做**零解**。

真正有信息量的问题是：除了零向量以外，还有没有别的解？如果存在 $\vec{x} \neq \vec{0}$ 使得 $A\vec{x} = \vec{0}$ ，就称齐次方程组有**非零解**。

定义 5.10.1: 矩阵 A 的**零空间** (null space) 是所有会被 A 映射成零向量的输入向量：

$$\text{Null}(A) = \{\vec{x} : A\vec{x} = \vec{0}\} \quad (134)$$

直观地说，零空间就是矩阵 A 在变换过程中“压扁掉”的方向。比如投影到 x 轴的矩阵

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad (135)$$

会把所有 y 轴方向的向量压成零向量：

$$P \begin{pmatrix} 0 \\ y \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \quad (136)$$

所以 y 轴就在 P 的零空间里。

对方阵来说，齐次方程组有没有非零解，可以用行列式判断：

定理 5.10.1: 对 $n \times n$ 方阵 A ，以下说法等价：

1. $A\vec{x} = \vec{0}$ 有非零解
2. A 把某些非零方向压成零，因而不可逆
3. $\det(A) = 0$

为什么 1 和 3 等价？可以从可逆性看。

如果 $\det(A) \neq 0$ ，则 A 可逆。此时若

$$A\vec{x} = \vec{0} \quad (137)$$

两边左乘 A^{-1} ，得到：

$$\vec{x} = A^{-1}\vec{0} = \vec{0} \quad (138)$$

所以齐次方程组只有零解，不可能有非零解。

反过来，如果 $\det(A) = 0$ ，则 A 不可逆。不可逆意味着 A 的列向量线性相关，也就是存在一组不全为零的系数 $x_1, x_2, \dots, x_n$ ，使得：

$$x_1 \vec{a}_1 + x_2 \vec{a}_2 + \dots + x_n \vec{a}_n = \vec{0} \quad (139)$$

其中 $\vec{a}_1, \vec{a}_2, \dots, \vec{a}_n$ 是 A 的列向量。把这些系数组成向量

$$\vec{x} = (x_1, x_2, \dots, x_n)^T \quad (140)$$

上式正是：

$$A\vec{x} = \vec{0} \quad (141)$$

由于这些系数不全为零，所以 $\vec{x} \neq \vec{0}$ 。因此齐次方程组有非零解。

从秩的角度看， $\det(A) = 0$ 等价于 $\text{rank}(A) < n$ 。根据维数关系，零空间的维数大于 0，所以存在非零解。反过来，若 $A\vec{x} = \vec{0}$ 有非零解，则 A 的列向量线性相关， $\text{rank}(A) < n$ ，因此 $\det(A) = 0$ 。

几何上也可以这样理解： $\det(A) = 0$ 表示 A 把 n 维体积压成 0，也就是至少有一个方向被压没；这个被压没的非零方向，就是齐次方程 $A\vec{x} = \vec{0}$ 的非零解方向。

6. 特征值与特征向量

6.1. 定义

定义 6.1.1: 设 A 是 $n \times n$ 矩阵，若存在非零向量 $\vec{v}$ 和标量 λ ，使得：

$$A\vec{v} = \lambda\vec{v} \quad (142)$$

则称 λ 为 A 的**特征值**， $\vec{v}$ 为对应的**特征向量**。

6.2. 几何直觉：只被拉伸、不被旋转的方向

特征向量的核心几何意义是：在对矩阵 A 描述的线性变换中，存在一些特殊的方向，向量沿着这些方向变换后，只被拉伸或压缩，**方向不发生改变**（或者恰好反向）。

例 6.2.1: 考虑沿 x 轴的拉伸变换，矩阵为 $A = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix}$ 。

- 对于 $\vec{v}_1 = (1, 0)^T$ ： $A\vec{v}_1 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 2 \\ 0 \end{pmatrix} = 2\vec{v}_1$ 方向没变，长度变成 2 倍。所以 $\lambda_1 = 2$ ， $\vec{v}_1 = (1, 0)^T$ 是特征对。
- 对于 $\vec{v}_2 = (0, 1)^T$ ： $A\vec{v}_2 = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 1\vec{v}_2$ 方向没变，长度不变。所以 $\lambda_2 = 1$ ， $\vec{v}_2 = (0, 1)^T$ 是特征对。
- 但对于 $\vec{w} = (1, 1)^T$ ： $A\vec{w} = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$ 结果不再是 $(1, 1)^T$ 的倍数，方向发生了改变，所以 $\vec{w}$ 不是特征向量。

例 6.2.2: 投影矩阵的几何直觉。将二维向量投影到 x 轴上的矩阵为 $P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ 。

- $\vec{v}_1 = (1, 0)^T$: $P\vec{v}_1 = (1, 0)^T = 1 \cdot \vec{v}_1$, 所以 $\lambda_1 = 1$
- $\vec{v}_2 = (0, 1)^T$: $P\vec{v}_2 = (0, 0)^T = 0 \cdot \vec{v}_2$, 所以 $\lambda_2 = 0$

特征值 1 表示“投影后保持不变的那些向量”（在投影方向上的向量）。特征值 0 表示“投影后被完全抹掉的那些向量”（垂直于投影方向的向量）。

例 6.2.3: 旋转矩阵没有实特征值（一般情况）。考虑将平面逆时针旋转 90 度的矩阵：

$$R = \begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix} \quad (143)$$

直接代入定义。设 $\vec{v} = (x, y)^T \neq \vec{0}$ 是特征向量，则：

$$R\vec{v} = \begin{pmatrix} -y \\ x \end{pmatrix} = \lambda \begin{pmatrix} x \\ y \end{pmatrix} \quad (144)$$

所以

$$-y = \lambda x, \quad x = \lambda y \quad (145)$$

把第二个式子代入第一个式子，得到：

$$-y = \lambda^2 y \quad (146)$$

如果 $y = 0$ ，则由 $x = \lambda y$ 得 $x = 0$ ，这会使 $\vec{v}$ 变成零向量，矛盾。因此 $y \neq 0$ ，可以推出：

$$\lambda^2 + 1 = 0 \quad (147)$$

在实数范围内这个方程无解；如果允许复数，则 $\lambda = \pm i$ 。

这说明：在实平面上，**不存在**一个方向在旋转 90 度后保持方向不变——每个向量都被旋转了。只有在复数域上，才能找到特征向量。这也是复数特征值的几何来源。

6.3. 特征多项式

现在把前面的定义整理成一个可计算的方法。特征值的定义是：存在非零向量 $\vec{v}$ ，使得：

$$A\vec{v} = \lambda\vec{v} \quad (148)$$

把右边移到左边：

$$A\vec{v} - \lambda\vec{v} = \vec{0} \quad (149)$$

这里需要把 $\lambda\vec{v}$ 写成矩阵乘法形式。因为单位矩阵 I 不改变向量，所以：

$$\lambda\vec{v} = \lambda I\vec{v} \quad (150)$$

于是：

$$(A - \lambda I)\vec{v} = \vec{0} \quad (151)$$

注意这里的未知量是 $\vec{v}$ ，而 λ 暂时看成一个待定的数。对每一个候选的 λ ， $A - \lambda I$ 都是一个矩阵，上式就是一个齐次线性方程组。

齐次方程组永远有零解，但特征向量要求 $\vec{v} \neq \vec{0}$ 。因此， λ 想成为特征值，必须让下面这个齐次方程组有非零解：

$$(A - \lambda I)\vec{v} = \vec{0} \quad (152)$$

根据前面“齐次方程组与零空间”的结论，方阵 M 满足 $M\vec{x} = \vec{0}$ 有非零解，当且仅当 $\det(M) = 0$ 。这里令 $M = A - \lambda I$ ，就得到：

$$(A - \lambda I)\vec{v} = \vec{0} \text{ 有非零解} \Leftrightarrow \det(A - \lambda I) = 0 \quad (153)$$

所以求特征值的问题，等价于求哪些 λ 会让矩阵 $A - \lambda I$ 的行列式变成 0。直观地说，选对 λ 后， $A - \lambda I$ 会把某些非零方向压成零；这些被压成零的方向，正是 A 中只被缩放、不改变方向的特征向量方向。

定理 6.3.1: 特征值是特征方程的解：

$$\det(A - \lambda I) = 0 \quad (154)$$

左侧 $p(\lambda) = \det(A - \lambda I)$ 称为 A 的**特征多项式**。它之所以是多项式，是因为行列式展开后只含有 λ 的幂次项。对 $n \times n$ 矩阵， $p(\lambda)$ 是一个 n 次多项式。

例 6.3.1: 求矩阵 $A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix}$ 的特征值和特征向量。

特征方程：

$$\det(A - \lambda I) = \det \begin{pmatrix} 4 - \lambda & 2 \\ 1 & 3 - \lambda \end{pmatrix} = (4 - \lambda)(3 - \lambda) - 2 = \lambda^2 - 7\lambda + 10 = 0 \quad (155)$$

解得： $\lambda_1 = 5, \lambda_2 = 2$ 。

对于 $\lambda_1 = 5$ ：

$$A - 5I = \begin{pmatrix} -1 & 2 \\ 1 & -2 \end{pmatrix} \vec{v} = \vec{0} \Rightarrow \vec{v}_1 = (2, 1)^T \quad (156)$$

对于 $\lambda_2 = 2$ ：

$$A - 2I = \begin{pmatrix} 2 & 2 \\ 1 & 1 \end{pmatrix} \vec{v} = \vec{0} \Rightarrow \vec{v}_2 = (1, -1)^T \quad (157)$$

6.4. 从方程角度看特征向量

由定义 $A\vec{v} = \lambda\vec{v}$ 可得：

$$(A - \lambda I)\vec{v} = \vec{0} \quad (158)$$

这说明：对一个已经确定的特征值 λ ，对应的特征向量并不是凭空猜出来的，而是齐次线性方程组 $(A - \lambda I)\vec{v} = \vec{0}$ 的非零解。换句话说，所有属于 λ 的特征向量，再加上零向量，构成了矩阵 $A - \lambda I$ 的零空间。

定义 6.4.1: 对特征值 λ ，集合

$$E_\lambda = \text{Ker}(A - \lambda I) = \{\vec{v} : (A - \lambda I)\vec{v} = \vec{0}\} \quad (159)$$

称为 λ 对应的**特征空间**。

注意定义中特征向量要求 $\vec{v} \neq \vec{0}$ ，但特征空间作为一个子空间必须包含零向量。因此更准确地说：特征空间中的所有非零向量都是特征向量。

这个视角非常有用，因为它把“求特征向量”转化成了已经熟悉的线性方程组问题。实际计算时，通常分三步：

1. 写出特征方程 $\det(A - \lambda I) = 0$
2. 解出所有特征值 λ
3. 对每个 λ ，求零空间 $\text{Ker}(A - \lambda I)$ ，得到对应的特征空间

例 6.4.1: 设

$$A = \begin{pmatrix} 3 & 1 \\ 0 & 3 \end{pmatrix} \quad (160)$$

特征方程：

$$\det(A - \lambda I) = \det \begin{pmatrix} 3 - \lambda & 1 \\ 0 & 3 - \lambda \end{pmatrix} = (3 - \lambda)^2 = 0 \quad (161)$$

唯一的特征值是 $\lambda = 3$ 。求特征向量：

$$(A - 3I)\vec{v} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} y \\ 0 \end{pmatrix} = \vec{0} \quad (162)$$

所以 $y = 0$ ， x 任意非零。对应的特征空间为：

$$E_3 = \text{span}\{(1, 0)^T\} \quad (163)$$

虽然特征值 3 在特征多项式中出现了两次，但它只提供了一个方向的特征向量。这类矩阵不能仅靠特征向量组成整个平面的基，因此不可对角化。

6.5. 代数重数与几何重数

特征值是否“够用”，不能只看特征值出现了几次，还要看它能提供多少个线性无关的特征向量。这里有两个必须区分的概念。

定义 6.5.1: 若 λ 作为特征多项式 $p(\lambda)$ 的根重复出现 k 次, 则称 k 为 λ 的**代数重数**。

特征空间 $E_\lambda = \text{Ker}(A - \lambda I)$ 的维数称为 λ 的**几何重数**。

代数重数来自多项式, 几何重数来自线性方程组。直观地说, 代数重数表示这个特征值在方程中“占了多少份额”, 几何重数表示它实际贡献了多少个独立方向。

定理 6.5.1: 对任意特征值 λ , 都有:

$$1 \leq \dim(E_\lambda) \leq \lambda \text{ 的代数重数} \quad (164)$$

如果每个特征值的几何重数都等于它的代数重数, 那么所有特征空间拼起来就能提供足够多的线性无关特征向量, 矩阵可以对角化。如果某个特征值的几何重数小于代数重数, 就会缺少特征向量, 矩阵通常只能化成更一般的约旦标准形, 而不能化成对角矩阵。

例 6.5.1: 对角矩阵

$$B = \begin{pmatrix} 3 & 0 \\ 0 & 3 \end{pmatrix} = 3I \quad (165)$$

它的特征方程同样是 $(3 - \lambda)^2 = 0$, 特征值 3 的代数重数也是 2。但

$$B - 3I = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \quad (166)$$

因此任何非零向量都是特征向量, $E_3 = \mathbb{R}^2$, 几何重数为 2。和上一个例子相比, 两个矩阵有相同的特征值和相同的代数重数, 但特征空间完全不同: A 只有一个特征方向, B 有整个平面的特征方向。

6.6. 二阶矩阵的快速判断

对 2×2 矩阵

$$A = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \quad (167)$$

特征多项式可以写成:

$$p(\lambda) = \det(A - \lambda I) = \lambda^2 - \text{tr}(A)\lambda + \det(A) \quad (168)$$

也就是说, 二阶矩阵的两个特征值满足:

$$\lambda_1 + \lambda_2 = \text{tr}(A) = a + d \quad (169)$$

$$\lambda_1 \lambda_2 = \det(A) = ad - bc \quad (170)$$

这两个量很适合做快速 sanity check。计算特征值之后, 先检查和是否等于迹、积是否等于行列式, 通常可以立刻发现符号错误。

例 6.6.1: 若

$$A = \begin{pmatrix} 0 & 1 \\ -2 & -3 \end{pmatrix} \quad (171)$$

则 $\text{tr}(A) = -3$, $\det(A) = 2$, 所以特征多项式为:

$$\lambda^2 + 3\lambda + 2 = 0 \quad (172)$$

解得 $\lambda_1 = -1, \lambda_2 = -2$ 。它们的和为 -3 , 积为 2 , 正好对应迹和行列式。

6.7. 特征值与线性系统的长期行为

特征值真正强大的地方, 是它能直接描述反复作用后的趋势。考虑离散系统:

$$\vec{x}_{k+1} = A\vec{x}_k \quad (173)$$

如果初始向量刚好是某个特征向量 $\vec{v}$, 那么:

$$\vec{x}_0 = \vec{v} \Rightarrow \vec{x}_k = A^k \vec{v} = \lambda^k \vec{v} \quad (174)$$

也就是说, 系统不会改变方向, 只会按照 λ^k 改变大小和符号。

1. 若 $|\lambda| < 1$, 沿该方向的分量会逐渐衰减到 0
2. 若 $|\lambda| > 1$, 沿该方向的分量会指数增长
3. 若 $\lambda < 0$, 方向会在每一步翻转
4. 若 λ 是复数, 系统通常表现为旋转加缩放

当矩阵可对角化时, 任意初始向量都可以分解到一组特征向量上:

$$\vec{x}_0 = c_1 \vec{v}_1 + c_2 \vec{v}_2 + \dots + c_n \vec{v}_n \quad (175)$$

反复作用 A 后得到:

$$A^k \vec{x}_0 = c_1 \lambda_1^k \vec{v}_1 + c_2 \lambda_2^k \vec{v}_2 + \dots + c_n \lambda_n^k \vec{v}_n \quad (176)$$

这个公式说明了为什么最大模特征值如此重要: 当 k 很大时, $|\lambda|$ 最大的项通常会主导整体行为。如果最大模特征值唯一, 并且初始向量在对应特征方向上有非零分量, 那么系统最终会越来越接近这个主导特征向量的方向。幂迭代法 (power iteration) 正是利用这个事实来近似计算最大特征值和对应特征向量。

6.8. 对称矩阵的特殊地位

在一般矩阵中, 可能出现复特征值、特征向量不够、特征向量彼此不正交等现象。但实对称矩阵 $A = A^T$ 的行为要好得多。

定理 6.8.1: 实对称矩阵的特征值全为实数, 并且不同特征值对应的特征向量相互正交。

这条性质解释了为什么很多工程问题会自然地落到对称矩阵上。协方差矩阵、惯性矩阵、二次型矩阵、无向图的拉普拉斯矩阵都是对称矩阵。对称性带来的结果是: 它们不仅能给出实数特征值, 还能给出一组正交方向。正交方向之间互不干扰, 因此可以把复杂问题拆成彼此独立的一维问题。

以 PCA 为例，协方差矩阵是对称矩阵，所以它的特征向量可以选成一组正交基。每个特征值表示数据沿对应方向的方差大小。选取最大的几个特征值对应的特征向量，就是选择数据变化最明显的几个互相垂直方向。这比随便选择坐标轴更合理，因为这些方向是由数据本身决定的。

6.9. 常见误区

1. **零向量不是特征向量。**虽然 $A\vec{0} = \lambda\vec{0}$ 对任意 λ 都成立，但如果允许零向量做特征向量，任何数都会变成特征值，定义就失去意义。
2. **特征值可以为 0。**当 $\lambda = 0$ 时，存在非零向量被 A 映射到零向量，说明 A 有非平凡零空间，因此 A 不可逆。
3. **不是所有矩阵在实数域上都有实特征值。**二维旋转矩阵就是典型例子。换到复数域后， $n \times n$ 矩阵总有 n 个特征值（按代数重数计）。
4. **特征值相同不代表矩阵相同。**矩阵的特征值描述了很多核心信息，但不能完整决定矩阵。前面的 $\begin{pmatrix} 3 & 1 \\ 0 & 3 \end{pmatrix}$ 和 $3I$ 就有相同特征值，却有完全不同的特征空间和对角化性质。
5. **求特征向量时不要忘记回代。**解出 λ 只是第一步，必须继续求 $\text{Ker}(A - \lambda I)$ 。特征向量不是一个唯一向量，而是一个方向；同一特征向量的任意非零倍数仍是特征向量。

6.10. 为什么要抽象出特征值？

特征值与特征向量的伟大之处，在于它们提供了一种“化繁为简”的视角：

定理 6.10.1: 若 $n \times n$ 矩阵 A 有 n 个线性无关的特征向量 $\vec{v}_1, \dots, \vec{v}_n$ ，对应的特征值为 $\lambda_1, \dots, \lambda_n$ ，则：

$$A = PDP^{-1} \quad (177)$$

其中 $P = [\vec{v}_1, \dots, \vec{v}_n]$ ， $D = \text{diag}(\lambda_1, \dots, \lambda_n)$ 。

这个等式的含义极其深刻：

1. P^{-1} ：把任意向量“翻译”到特征向量构成的坐标系下
2. D ：在这个坐标系中，变换变得极其简单——每个坐标轴方向只是独立的拉伸
3. P ：做完简单的拉伸后，再“翻译”回原来的坐标系

例 6.10.1: 计算 A^{100} 看似困难，但如果 $A = PDP^{-1}$ ，则：

$$A^{100} = PD^{100}P^{-1} = P \text{diag}(\lambda_1^{100}, \dots, \lambda_n^{100})P^{-1} \quad (178)$$

原本需要 100 次矩阵乘法，现在变成：换基底 $\rightarrow$ 每个方向单独算 100 次幂 $\rightarrow$ 换回来。

6.11. 工程中的应用

特征值与特征向量是现代科学与工程中无处不在的工具：

1. Google PageRank

PageRank 想解决的问题是：网页数量巨大，仅仅统计一个页面被多少网页链接并不够，因为来自重要网页的链接应该比来自普通网页的链接更有分量。于是可以把整个互联网看成

一个有向图：网页是节点，超链接是边。若页面 j 指向页面 i ，就认为用户有一定概率从 j 跳到 i 。

设 M 是转移矩阵，其中 M_{ij} 表示从页面 j 跳转到页面 i 的概率。若当前用户分布是 $\vec{r}_k$ ，下一步分布就是：

$$\vec{r}_{k+1} = M\vec{r}_k \quad (179)$$

如果经过很多次跳转以后，网页访问概率趋于稳定，那么稳定分布 $\vec{r}$ 应该满足：

$$M\vec{r} = \vec{r} \quad (180)$$

这正是特征值问题，而且对应的特征值是 1。PageRank 向量 $\vec{r}$ 的每个分量就是对应网页的重要性分数。这个分数不是孤立计算出来的，而是在整个链接网络中互相递推得到的：一个页面重要，是因为很多重要页面指向它。

实际系统中还会加入“随机跳转”机制：用户不一定只沿着链接走，也可能直接输入一个网址或打开书签。这样可以避免某些封闭网页团体把权重困在内部，也能保证算法更容易收敛。工程上通常使用幂迭代法反复计算 $\vec{r}_{k+1} = M\vec{r}_k$ ，直到前后两次结果变化很小。

这个例子体现了特征向量的一个典型意义：**稳定状态**。当系统反复执行同一个线性变换后，如果最终保持某种比例不变，那么这个不变方向往往就是特征向量。

2. 主成分分析 (PCA)

3. 结构振动分析

桥梁、建筑、汽车车身、飞机机翼都会振动。工程师关心的不是“会不会动”这么简单，而是关心结构在哪些频率下最容易振动，以及振动时每个部位怎样运动。这两个问题分别对应**固有频率**和**振型**。

一个简化的无阻尼结构系统可以写成：

$$M\ddot{\vec{x}} + K\vec{x} = \vec{0} \quad (181)$$

这里 M 是质量矩阵， K 是刚度矩阵， $\vec{x}$ 描述各个自由度的位移。若假设结构按某个固定形状振动：

$$\vec{x}(t) = \vec{v} \cos(\omega t) \quad (182)$$

代入方程可得：

$$K\vec{v} = \omega^2 M\vec{v} \quad (183)$$

这不是普通特征值问题，而是广义特征值问题。 ω^2 是特征值， $\vec{v}$ 是特征向量。 ω 给出固有角频率， $\vec{v}$ 给出振型，也就是结构在这个频率下的相对变形形状。

为什么这在工程上关键？因为外部激励如果接近某个固有频率，结构会发生共振。共振时即使外力不大，振幅也可能逐渐放大，导致疲劳、裂纹甚至失效。桥梁抗风设计、发动机支架设计、建筑抗震分析、机翼颤振分析，本质上都需要先知道系统有哪些固有模态。

特征向量在这里不是抽象方向，而是非常具体的形状。例如一根梁的第一振型可能整体上下弯曲，第二振型可能中间有节点、两侧反向运动。工程师可以据此判断传感器放在哪里、加强筋加在哪里、哪些频率应该避开。

4. 控制系统稳定性

控制系统常常研究一个系统在平衡点附近受到扰动后会怎样。例如无人机姿态偏了一点不会自动回正，电机速度被扰动后会不会恢复，温控系统会不会在目标温度附近稳定下来。很多非线性系统在平衡点附近可以线性化为：

$$\vec{x}' = A\vec{x} \quad (184)$$

这里 $\vec{x}$ 表示偏离平衡点的状态， A 是状态矩阵。如果 A 有一组特征向量，那么初始扰动可以分解到这些特征方向上。沿着某个特征方向，系统满足：

$$\vec{x}(t) = e^{\lambda t} \vec{v} \quad (185)$$

因此 λ 的实部决定这个方向上的扰动会衰减还是增长：

1. 若 $\Re(\lambda) < 0$ ，则 $e^{\lambda t}$ 衰减，扰动逐渐消失
2. 若 $\Re(\lambda) > 0$ ，则 $e^{\lambda t}$ 增长，扰动被放大
3. 若 λ 有非零虚部，系统会出现振荡

对连续时间系统 $\vec{x}' = A\vec{x}$ ，若所有特征值实部都小于 0，平衡点局部渐近稳定。对离散时间系统 $\vec{x}_{k+1} = A\vec{x}_k$ ，对应条件是所有特征值的模都小于 1。

这就是为什么控制工程里经常看“极点”。在线性系统中，极点通常就是状态矩阵或传递函数分母对应的特征值。设计控制器时，本质上是在改变闭环系统的特征值位置，让系统既稳定，又满足速度、超调、振荡等性能要求。

例如一个系统如果有特征值 $\lambda = -0.1$ ，扰动会衰减，但衰减很慢；如果有 $\lambda = -5$ ，衰减更快；如果有 $\lambda = 0.2$ ，系统就不稳定。若特征值是 $-1 \pm 10i$ ，系统稳定但会快速振荡。特征值把这些动态行为压缩成了几个关键数字。

5. 量子力学

量子力学中，系统状态不是普通三维向量，而是希尔伯特空间中的向量，常写作 $\vec{\psi}$ 。可观测量，比如能量、动量、角动量、自旋，对应线性算子。为了保证测量结果是实数，这些算子通常是厄米算子。

若算子 H 表示能量，那么能量本征态满足：

$$H\vec{\psi} = E\vec{\psi} \quad (186)$$

这里 E 是能量特征值， $\vec{\psi}$ 是对应的能量本征态。测量能量时，可能得到的数值不是任意连续值，而是这个算子的特征值。测量后，系统状态会落到对应的特征向量方向上。

这解释了“能级”的数学来源。比如电子在原子中的能量不是任意的，而是由哈密顿算子的特征值决定。求解量子系统的能谱，就是求一个算子的特征值；求系统在某个能级下的状态，就是求对应的特征函数或特征向量。

特征向量在这里表示一种测量后保持确定性的状态。如果系统已经处在 H 的某个特征向量上，那么测量能量一定得到对应的 E 。如果系统是多个特征向量的线性组合，测量结果就带有概率性；每个特征方向上的投影大小决定对应结果出现的概率。

所以在线性代数里看似简单的等式 $A\vec{v} = \lambda\vec{v}$ ，在量子力学中变成了非常具体的物理陈述：某个状态在某个可观测量下具有确定测量值。

6.12. 重要性质

定理 6.12.1: 设 $\lambda_1, \lambda_2, \dots, \lambda_n$ 是 $n \times n$ 矩阵 A 的特征值，则：

1. $\det(A) = \lambda_1 \lambda_2 \dots \lambda_n$
2. $\text{tr}(A) = \lambda_1 + \lambda_2 + \dots + \lambda_n$
3. 若 A 是对称矩阵，则所有特征值都是实数，且不同特征值对应的特征向量正交

7. 线性变换

7.1. 为什么要研究线性变换？

线性代数表面上在研究向量和矩阵，本质上是在研究“空间怎样被变换”。一个变换可以把平面上的点移动到另一个位置，也可以把三维空间压到二维平面，还可以把一组数据从高维特征空间投影到低维空间。

但是任意变换都太复杂。例如 $T(x) = x^2$ 会把直线弯成曲线， $T(x) = \sin(x)$ 会产生周期性弯折。线性代数只研究一类最简单、最可控的变换：它们保持向量加法和数乘。这类变换就是线性变换。

直观地说，线性变换有三个重要特征：

1. 原点保持不动
2. 直线仍然变成直线，不会被弯成曲线
3. 平行关系和比例关系被保留

因此，旋转、缩放、投影、剪切都是线性变换；平移不是线性变换，因为它会把原点移动到别处。

7.2. 定义

定义 7.2.1: 设 V 和 W 是向量空间，映射 $T: V \rightarrow W$ 称为**线性变换**，如果满足：

1. $T(\vec{u} + \vec{v}) = T(\vec{u}) + T(\vec{v})$ （保持加法）
2. $T(c\vec{v}) = cT(\vec{v})$ （保持数乘）

这两个条件看起来抽象，但它们表达的是同一件事：先组合再变换，和先变换再组合，结果必须一样。

如果一个向量可以写成线性组合：

$$\vec{x} = c_1 \vec{v}_1 + c_2 \vec{v}_2 + \dots + c_k \vec{v}_k \quad (187)$$

那么线性变换一定满足：

$$T(\vec{x}) = c_1 T(\vec{v}_1) + c_2 T(\vec{v}_2) + \dots + c_k T(\vec{v}_k) \quad (188)$$

也就是说，只要知道一组基向量被变到哪里，就能知道所有向量被变到哪里。这是线性变换最重要的简化。

例 7.2.1: 旋转是线性变换。设 R 表示把平面向量逆时针旋转 90 度，则：

$$R(\vec{u} + \vec{v}) = R\vec{u} + R\vec{v} \quad (189)$$

$$R(c\vec{u}) = cR\vec{u} \quad (190)$$

几何上看，先把两个向量相加再旋转，和先分别旋转再相加，得到的平行四边形是同一个。

例 7.2.2: 平移不是线性变换。设 $T(\vec{x}) = \vec{x} + \vec{b}$ ，其中 $\vec{b} \neq \vec{0}$ 。

因为

$$T(\vec{0}) = \vec{b} \neq \vec{0} \quad (191)$$

但任何线性变换都必须满足 $T(\vec{0}) = \vec{0}$ 。所以平移不满足线性变换的定义。

7.3. 矩阵表示

定理 7.3.1: 设 $T: \mathbb{R}^n \rightarrow \mathbb{R}^m$ 是线性变换，则存在唯一的 $m \times n$ 矩阵 A ，使得：

$$T(\vec{x}) = A\vec{x} \quad \text{对所有 } \vec{x} \in \mathbb{R}^n \quad (192)$$

为什么线性变换一定可以用矩阵表示？关键在于标准基。

在 $\mathbb{R}^n$ 中，任意向量都可以写成：

$$\vec{x} = x_1\vec{e}_1 + x_2\vec{e}_2 + \dots + x_n\vec{e}_n \quad (193)$$

由于 T 是线性的：

$$T(\vec{x}) = x_1T(\vec{e}_1) + x_2T(\vec{e}_2) + \dots + x_nT(\vec{e}_n) \quad (194)$$

这说明我们只需要知道 $T(\vec{e}_1), T(\vec{e}_2), \dots, T(\vec{e}_n)$ 。把这些向量按列排成矩阵：

$$A = [T(\vec{e}_1), T(\vec{e}_2), \dots, T(\vec{e}_n)] \quad (195)$$

就得到：

$$T(\vec{x}) = A\vec{x} \quad (196)$$

因此矩阵不是凭空出现的符号表格，而是线性变换对基向量作用结果的记录。矩阵的第 j 列，就是第 j 个基向量变换后的样子。

例 7.3.1: 设 $T: \mathbb{R}^2 \rightarrow \mathbb{R}^2$ 表示:

$$T(\vec{e}_1) = (2, 1)^T, \quad T(\vec{e}_2) = (-1, 3)^T \quad (197)$$

那么对应矩阵为:

$$A = \begin{pmatrix} 2 & -1 \\ 1 & 3 \end{pmatrix} \quad (198)$$

对任意 $\vec{x} = (x, y)^T$, 有:

$$T(\vec{x}) = xT(\vec{e}_1) + yT(\vec{e}_2) = \begin{pmatrix} 2x - y \\ x + 3y \end{pmatrix} = A\vec{x} \quad (199)$$

7.4. 常见几何变换

很多熟悉的几何操作都可以写成线性变换:

1. 缩放

$$A = \begin{pmatrix} 2 & 0 \\ 0 & 3 \end{pmatrix} \quad (200)$$

表示 x 方向放大 2 倍, y 方向放大 3 倍。

2. 旋转

$$R = \begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \quad (201)$$

表示逆时针旋转角度 θ 。

3. 投影

$$P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad (202)$$

表示把平面投影到 x 轴上。

4. 剪切

$$S = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix} \quad (203)$$

表示保持 y 坐标不变, 同时让 x 坐标随着 y 发生偏移。

这些例子说明: 线性变换虽然定义抽象, 但它覆盖了大量基础几何操作。矩阵乘法表示线性变换的复合, 行列式描述体积缩放, 特征值描述不变方向, 前面几章的概念都可以统一到“线性变换”这个视角下。

7.5. 核与像

前面在矩阵方程 $A\vec{x} = \vec{0}$ 里已经见过零空间。在线性变换的语言里, 同一个概念通常叫做**核**: 它问的是哪些输入会被变换压成零。

定义 7.5.1: 线性变换 $T : V \rightarrow W$ 的**核** (kernel) 和**像** (image) 定义为:

$$\text{Ker}(T) = \{\vec{v} \in V : T(\vec{v}) = \vec{0}\} \quad (204)$$

$$\text{Im}(T) = \{T(\vec{v}) \in W : \vec{v} \in V\} \quad (205)$$

先看核的公式:

$$\text{Ker}(T) = \{\vec{v} \in V : T(\vec{v}) = \vec{0}\} \quad (206)$$

这个集合记号可以逐字读成:

1. $\vec{v} \in V$: 先从输入空间 V 里拿一个向量 $\vec{v}$
2. $T(\vec{v}) = \vec{0}$: 这个向量经过变换后会变成零向量

所以整句话就是: $\text{Ker}(T)$ 是所有“会被 T 压成零”的输入向量组成的集合。

再看像的公式:

$$\text{Im}(T) = \{T(\vec{v}) \in W : \vec{v} \in V\} \quad (207)$$

它可以读成: 让输入 $\vec{v}$ 在整个 V 里变化, 把所有可能的输出 $T(\vec{v})$ 收集起来, 这个集合就是 $\text{Im}(T)$ 。也就是说, 像不是问“哪些输入会消失”, 而是问“这个变换最多能到达哪些输出”。

核与像回答的是两个最基本的问题:

1. **核**问的是: 哪些输入被压成了零?
2. **像**问的是: 这个变换最终能到达哪些输出?

如果核很大, 说明很多不同输入都被压到同一个结果, 信息丢失严重。如果像的维数很小, 说明输出被限制在一个低维子空间里, 变换没有覆盖整个目标空间。

例 7.5.1: 投影 $P = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix}$ 把 $(x, y)^T$ 变成 $(x, 0)^T$ 。

先求核。按定义，核里放的是所有满足 $P\vec{v} = \vec{0}$ 的输入。令 $\vec{v} = (x, y)^T$ ，则：

$$P\vec{v} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} x \\ 0 \end{pmatrix} \quad (208)$$

要让它等于零向量，必须有 $x = 0$ ，而 y 可以任意。因此所有这样的输入都是 $(0, y)^T$ 。

它的核为：

$$\text{Ker}(P) = \{(0, y)^T : y \in \mathbb{R}\} \quad (209)$$

也就是 y 轴。因为 y 轴上的所有向量都会被投影成零向量。

再求像。像里放的是所有可能的输出。因为任意输入 $(x, y)^T$ 经过 P 后都会变成 $(x, 0)^T$ ，所以输出只能落在 x 轴上；反过来， x 轴上任意 $(x, 0)^T$ 都可以由输入 $(x, y)^T$ 得到。

它的像为：

$$\text{Im}(P) = \{(x, 0)^T : x \in \mathbb{R}\} \quad (210)$$

也就是 x 轴。因为投影后的结果永远落在 x 轴上。

这里 $\dim(\text{Ker}(P)) = 1$ ， $\dim(\text{Im}(P)) = 1$ ，加起来等于输入空间维数 2。

定理 7.5.1: (秩-零化度定理) 对于线性变换 $T : V \rightarrow W$ ：

$$\dim(V) = \dim(\text{Ker}(T)) + \dim(\text{Im}(T)) \quad (211)$$

或者写成： $\dim(\text{Ker}(T)) + \text{rank}(T) = n$ 。

秩-零化度定理表达的是一个信息守恒关系：输入空间的维数，要么变成可见的输出方向（像），要么在变换中被压掉（核）。对于线性变换来说，不会有第三种去处。

8. 内积空间

8.1. 为什么要引入内积？

前面的向量空间只规定了加法和数乘，因此它能讨论“线性组合”“维数”“基”，但还不能自然地讨论长度、角度、垂直、投影这些几何概念。

例如，在一个普通向量空间里，我们可以问：

1. 一个向量能不能由另一组向量线性表示？
2. 一组向量是否线性无关？
3. 一个线性变换是否可逆？

但我们还不能问：

1. 哪个向量更长？
2. 两个向量夹角是多少？
3. 一个向量在另一个方向上的投影是多少？

4. 两个误差谁更小？

内积就是为了解决这些问题。它给向量空间加上了“测量工具”：通过内积可以定义长度、角度、正交和投影。没有内积，向量空间只是代数对象；有了内积，它才真正带上几何结构。

8.2. 内积的定义

定义 8.2.1: 向量空间 V 上的**内积**是一个二元函数 $\langle \cdot, \cdot \rangle : V \times V \rightarrow F$ ，满足：

1. 共轭对称性： $\langle \vec{u}, \vec{v} \rangle = \overline{\langle \vec{v}, \vec{u} \rangle}$
2. 线性性： $\langle a\vec{u} + b\vec{v}, \vec{w} \rangle = a\langle \vec{u}, \vec{w} \rangle + b\langle \vec{v}, \vec{w} \rangle$
3. 正定性： $\langle \vec{v}, \vec{v} \rangle \geq 0$ ，且等号成立当且仅当 $\vec{v} = \vec{0}$

这些条件不是随便列出来的。它们对应内积作为“长度和角度测量”时必须满足的基本直觉：

1. **共轭对称性**：交换两个向量，几何关系不应该改变。实数空间中就是 $\langle \vec{u}, \vec{v} \rangle = \langle \vec{v}, \vec{u} \rangle$ 。
2. **线性性**：如果一个向量由几个方向叠加而成，它和另一个向量的关系也应该能分解计算。
3. **正定性**：一个向量和自己的内积应该给出长度平方，所以不能为负；只有零向量长度为 0。

因此，内积可以理解作为一种推广后的“点积”。在不同空间里，只要一个运算满足这些性质，它就可以承担测量长度和角度的角色。

8.3. 欧几里得内积

在 $\mathbb{R}^n$ 中，标准内积（点积）为：

$$\langle \vec{u}, \vec{v} \rangle = \vec{u} \cdot \vec{v} = \sum_{i=1}^n u_i v_i \quad (212)$$

为什么点积能反映角度？在二维或三维欧几里得空间中：

$$\vec{u} \cdot \vec{v} = \|\vec{u}\| \|\vec{v}\| \cos \theta \quad (213)$$

其中 θ 是两个向量的夹角。因此：

1. 点积为正：夹角小于 90 度，大体同向
2. 点积为 0：夹角等于 90 度，两个方向正交
3. 点积为负：夹角大于 90 度，大体反向

例 8.3.1: 设 $\vec{u} = (1, 2)^T$ ， $\vec{v} = (2, -1)^T$ 。

则：

$$\vec{u} \cdot \vec{v} = 1 \cdot 2 + 2 \cdot (-1) = 0 \quad (214)$$

所以两个向量正交。几何上看， $(1, 2)^T$ 的斜率是 2， $(2, -1)^T$ 的斜率是 $-\frac{1}{2}$ ，两条直线互相垂直。

内积不一定只能是标准点积。例如在 $\mathbb{R}^n$ 中，可以定义带权内积：

$$\langle \vec{u}, \vec{v} \rangle_W = \vec{u}^T W \vec{v} \quad (215)$$

其中 W 是正定对称矩阵。这个内积表示不同坐标方向的重要性不同。机器学习中的加权误差、统计学中的马氏距离，都可以看成是在使用不同的内积来重新定义“距离”和“角度”。

8.4. 范数与正交性

定义 8.4.1: 向量的**范数**（长度）定义为：

$$\|\vec{v}\| = \sqrt{\langle \vec{v}, \vec{v} \rangle} \quad (216)$$

两个向量**正交**当且仅当 $\langle \vec{u}, \vec{v} \rangle = 0$ 。

范数就是由内积诱导出来的长度。定义

$$\|\vec{v}\| = \sqrt{\langle \vec{v}, \vec{v} \rangle} \quad (217)$$

的原因是： $\langle \vec{v}, \vec{v} \rangle$ 扮演长度平方的角色，所以开平方后得到长度。

正交的意义也不仅是“垂直”。在线性代数和工程中，正交更常被理解为“互不干扰”或“没有重叠成分”。如果两个方向正交，那么一个方向上的变化不会投影到另一个方向上。

定理 8.4.1:（勾股定理）若 $\vec{u}$ 与 $\vec{v}$ 正交，则：

$$\|\vec{u} + \vec{v}\|^2 = \|\vec{u}\|^2 + \|\vec{v}\|^2 \quad (218)$$

这个定理说明正交方向上的能量或误差可以直接相加。在信号处理里，两个正交信号互不干扰；在最小二乘里，残差和拟合空间正交；在 PCA 里，不同主成分方向正交，因此各方向上的方差可以分开解释。

8.5. 投影

内积最常见的用途之一是计算投影。给定非零向量 $\vec{u}$ ，向量 $\vec{v}$ 在 $\vec{u}$ 方向上的投影为：

$$\text{proj}_{\vec{u}} \vec{v} = \frac{\langle \vec{v}, \vec{u} \rangle}{\langle \vec{u}, \vec{u} \rangle} \vec{u} \quad (219)$$

这个公式的含义是：先用内积测量 $\vec{v}$ 在 $\vec{u}$ 方向上有多少分量，再乘以方向向量 $\vec{u}$ 。

例 8.5.1: 设 $\vec{v} = (3, 4)^T$ ， $\vec{u} = (1, 0)^T$ 。

则：

$$\text{proj}_{\vec{u}} \vec{v} = \frac{\langle \vec{v}, \vec{u} \rangle}{\langle \vec{u}, \vec{u} \rangle} \vec{u} = 3\vec{u} = (3, 0)^T \quad (220)$$

这就是把 $(3, 4)^T$ 投影到 x 轴上。

8.6. 格拉姆-施密特正交化

定理 8.6.1: 给定线性无关向量组 $\vec{v}_1, \dots, \vec{v}_k$ ，可以通过以下递推公式构造正交向量组 $\vec{u}_1, \dots, \vec{u}_k$:

$$\vec{u}_1 = \vec{v}_1 \quad (221)$$

$$\vec{u}_j = \vec{v}_j - \sum_{i=1}^{j-1} \frac{\langle \vec{v}_j, \vec{u}_i \rangle}{\langle \vec{u}_i, \vec{u}_i \rangle} \vec{u}_i, \quad j = 2, \dots, k \quad (222)$$

格拉姆-施密特正交化的思想很简单：每拿到一个新向量，就把它在已有正交方向上的投影全部减掉，剩下的部分自然和已有方向正交。

以两个向量为例，先令 $\vec{u}_1 = \vec{v}_1$ 。然后从 $\vec{v}_2$ 中减去它在 $\vec{u}_1$ 上的投影：

$$\vec{u}_2 = \vec{v}_2 - \frac{\langle \vec{v}_2, \vec{u}_1 \rangle}{\langle \vec{u}_1, \vec{u}_1 \rangle} \vec{u}_1 \quad (223)$$

此时 $\vec{u}_2$ 就是 $\vec{v}_2$ 中无法被 $\vec{u}_1$ 解释的剩余部分，所以它和 $\vec{u}_1$ 正交。

这个过程在数值计算中非常重要，因为正交基比普通基更稳定、更容易计算。QR 分解、最小二乘、正交投影、信号分解都依赖这个思想。

9. 矩阵对角化

9.1. 为什么要对角化？

矩阵表示线性变换，但一般矩阵会把各个坐标方向混在一起。例如：

$$A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix} \quad (224)$$

作用到向量 $(x, y)^T$ 上时，输出的两个坐标都是 x 和 y 的混合：

$$A \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 4x + 2y \\ x + 3y \end{pmatrix} \quad (225)$$

这类矩阵直接分析起来不方便。对角矩阵则简单得多：

$$D = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \quad (226)$$

它只做一件事：第一个坐标乘以 λ_1 ，第二个坐标乘以 λ_2 ，各方向互不干扰。

对角化的目标就是：找到一个更合适的坐标系，让原本复杂的矩阵在新坐标系中变成对角矩阵。这个新坐标系通常由特征向量构成。

9.2. 可对角化条件

定义 9.2.1: 矩阵 A 称为**可对角化**的，如果存在可逆矩阵 P 和对角矩阵 D ，使得：

$$A = PDP^{-1} \quad (227)$$

定理 9.2.1: $n \times n$ 矩阵 A 可对角化的充要条件是: A 有 n 个线性无关的特征向量。

此时, P 的列是特征向量, D 的对角元素是对应的特征值。

这个公式应该这样读:

1. P^{-1} : 把向量从原坐标系转换到特征向量坐标系
2. D : 在特征向量坐标系里, 每个方向只做独立缩放
3. P : 把结果从特征向量坐标系转换回原坐标系

因此 $A = PDP^{-1}$ 不是一个纯代数技巧, 而是在说: 矩阵 A 的复杂作用, 可以分解为“换坐标 $\rightarrow$ 独立缩放 $\rightarrow$ 换回来”。

定理 9.2.2: 若 A 有 n 个互不相同的特征值, 则 A 可对角化。

注意这是充分条件, 不是必要条件。也就是说, 有 n 个互不相同的特征值一定可对角化; 但有重复特征值的矩阵也可能可对角化, 关键要看是否仍然能找到 n 个线性无关的特征向量。

例 9.2.1: 前面求过矩阵

$$A = \begin{pmatrix} 4 & 2 \\ 1 & 3 \end{pmatrix} \quad (228)$$

它的特征值为 $\lambda_1 = 5, \lambda_2 = 2$, 对应特征向量可以取:

$$\vec{v}_1 = (2, 1)^T, \quad \vec{v}_2 = (1, -1)^T \quad (229)$$

因此:

$$P = \begin{pmatrix} 2 & 1 \\ 1 & -1 \end{pmatrix}, \quad D = \begin{pmatrix} 5 & 0 \\ 0 & 2 \end{pmatrix} \quad (230)$$

有:

$$A = PDP^{-1} \quad (231)$$

在原坐标系中, A 看起来会混合两个坐标; 但在由 $(2, 1)^T$ 和 $(1, -1)^T$ 构成的新坐标系中, 它只是沿第一个方向放大 5 倍, 沿第二个方向放大 2 倍。

并不是所有矩阵都能对角化。例如:

$$B = \begin{pmatrix} 3 & 1 \\ 0 & 3 \end{pmatrix} \quad (232)$$

它只有一个特征值 $\lambda = 3$, 而且只有一个独立特征方向 $(1, 0)^T$ 。二维空间需要两个独立方向才能构成基, 但这个矩阵只提供一个特征方向, 所以不能对角化。

9.3. 对称矩阵的对角化

定理 9.3.1: (谱定理) 实对称矩阵 A 总可以正交对角化: 存在正交矩阵 Q (即 $Q^T Q = I$), 使得:

$$A = Q D Q^T \quad (233)$$

其中 D 是对角矩阵, 对角元素为 A 的特征值。

对称矩阵的对角化特别重要, 因为这里的换基矩阵 Q 可以取成正交矩阵。正交矩阵表示旋转或反射, 不会拉伸长度, 也不会破坏角度。因此 $Q^T = Q^{-1}$, 计算上更稳定, 几何意义也更清楚。

这也是为什么协方差矩阵、二次型、PCA、图拉普拉斯矩阵都喜欢对称矩阵: 它们不仅可以对角化, 而且可以在一组互相垂直的方向上分解问题。

9.4. 应用: 矩阵的幂

例 9.4.1: 若 $A = P D P^{-1}$, 其中 $D = \text{diag}(\lambda_1, \dots, \lambda_n)$, 则:

$$A^k = P D^k P^{-1} = P \text{diag}(\lambda_1^k, \dots, \lambda_n^k) P^{-1} \quad (234)$$

矩阵的幂是对角化最直接的应用。因为:

$$A^2 = (P D P^{-1})(P D P^{-1}) = P D^2 P^{-1} \quad (235)$$

中间的 $P^{-1} P$ 会抵消。重复 k 次后, 就得到 $A^k = P D^k P^{-1}$ 。而 D^k 很容易算, 只需要把每个对角元素分别取 k 次方。

这对离散动力系统尤其重要:

$$\vec{x}_{k+1} = A \vec{x}_k \quad (236)$$

于是:

$$\vec{x}_k = A^k \vec{x}_0 = P D^k P^{-1} \vec{x}_0 \quad (237)$$

系统长期行为就由特征值的幂决定: 模大的特征值主导增长, 模小于 1 的特征值对应衰减, 负特征值会导致方向交替翻转, 复特征值会带来旋转振荡。

10. 奇异值分解 (Singular Value Decomposition, SVD)

10.1. 为什么需要 SVD?

SVD 是 **Singular Value Decomposition** 的缩写, 中文叫**奇异值分解**。

前面讲过矩阵对角化:

$$A = P D P^{-1} \quad (238)$$

它的思想很漂亮: 换到特征向量坐标系后, 矩阵只是在各个方向上独立缩放。但对角化有两个限制:

1. 只有方阵才谈得上普通特征值分解
2. 即使是方阵，也不一定有足够的特征向量来对角化

实际工程中，我们经常遇到不是方阵的矩阵。例如数据矩阵通常是 $m \times n$ ： m 个样本， n 个特征；图像矩阵可能是高度乘宽度；推荐系统矩阵可能是用户数乘商品数。这些矩阵不一定能做特征值分解，但我们仍然希望把它们拆成简单、可解释的部分。

SVD 的作用就是：给任意矩阵一个稳定、通用的分解方式。它比特征值分解更普适，任意 $m \times n$ 实矩阵都可以做 SVD。

10.2. 定义

定理 10.2.1: 任意 $m \times n$ 矩阵 A 都可以分解为：

$$A = U \Sigma V^T \quad (239)$$

其中 U 是 $m \times m$ 正交矩阵， V 是 $n \times n$ 正交矩阵， Σ 是 $m \times n$ 对角矩阵（对角元素非负，按降序排列）。

这些非负对角元素 $\sigma_1 \geq \sigma_2 \geq \dots \geq 0$ 称为 A 的**奇异值**。

这里说 Σ 是 $m \times n$ 对角矩阵，不是说它必须是方阵，而是说：除了主对角线上的位置，其他位置全是 0。长方形矩阵也可以有主对角线，只是主对角线最多只有 $\min(m, n)$ 个位置。

例如，当 $m = 3, n = 2$ 时， Σ 可以是：

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 \\ 0 & \sigma_2 \\ 0 & 0 \end{pmatrix} \quad (240)$$

这是一个 3×2 矩阵。它不是方阵，但非零元素只出现在 $(1, 1)$ 和 $(2, 2)$ 这两个主对角线位置，所以仍然称为对角矩阵。

当 $m = 2, n = 4$ 时， Σ 可以是：

$$\Sigma = \begin{pmatrix} \sigma_1 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 \end{pmatrix} \quad (241)$$

这也是同一个意思：主对角线上放奇异值，其他位置全为 0。长方形的 Σ 同时承担两个作用：沿主对角线方向缩放，并把维度从 n 维变成 m 维。

这个公式要按从右到左读：

1. V^T ：先在输入空间中旋转或反射，找到一组最适合 A 的输入方向
2. Σ ：沿这些方向做缩放，并且可能改变维度
3. U ：最后在输出空间中旋转或反射，得到最终方向

所以 SVD 的几何含义是：

$$\text{任意线性变换} = \text{输入空间旋转} + \text{坐标轴缩放} + \text{输出空间旋转} \quad (242)$$

这里 U 和 V 都是正交矩阵，所以它们不会改变长度和角度，只是改变观察坐标系。真正改变长度的是中间的 Σ ，它的对角元素就是奇异值。

如果 A 是 $m \times n$ 矩阵，那么：

1. V 的列向量在输入空间 $\mathbb{R}^n$ 中，称为**右奇异向量**
2. U 的列向量在输出空间 $\mathbb{R}^m$ 中，称为**左奇异向量**
3. σ_i 表示第 i 个右奇异方向经过 A 后被拉伸了多少倍

更具体地说，SVD 等价于下面这些关系：

$$A\vec{v}_i = \sigma_i\vec{u}_i \quad (243)$$

其中 $\vec{v}_i$ 是 V 的第 i 列， $\vec{u}_i$ 是 U 的第 i 列。意思是：输入空间中存在一组特殊的正交方向 $\vec{v}_i$ ，矩阵 A 会把这些方向分别变成输出空间中的正交方向 $\vec{u}_i$ ，长度缩放比例是 σ_i 。

例 10.2.1: 若

$$A = \begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix} \quad (244)$$

它已经是一个对角缩放矩阵。它把 x 方向拉伸 3 倍，把 y 方向保持 1 倍。因此它的奇异值是：

$$\sigma_1 = 3, \quad \sigma_2 = 1 \quad (245)$$

此时可以取 $U = I$, $V = I$, $\Sigma = A$ 。也就是说，不需要额外旋转，矩阵本身就已经是“沿正交方向独立缩放”的形式。

更一般的矩阵可以理解为：先把输入坐标系转到合适方向，再做类似 $\begin{pmatrix} 3 & 0 \\ 0 & 1 \end{pmatrix}$ 这样的独立缩放，最后把输出坐标系转到目标方向。

10.3. 与特征值的关系

定理 10.3.1: A 的奇异值是 $A^T A$ （或 AA^T ）的特征值的非负平方根。

这个结论可以从 SVD 直接看出来。若

$$A = U\Sigma V^T \quad (246)$$

则：

$$A^T A = (U\Sigma V^T)^T (U\Sigma V^T) \quad (247)$$

利用正交矩阵 $U^T U = I$ ，得到：

$$A^T A = V\Sigma^T \Sigma V^T \quad (248)$$

$\Sigma^T \Sigma$ 是一个对角矩阵，对角元素是：

$$\sigma_1^2, \sigma_2^2, \dots \quad (249)$$

因此， $A^T A$ 的特征值就是奇异值的平方， V 的列向量就是 $A^T A$ 的特征向量。换句话说：

1. 先求 $A^T A$ 的特征值 λ_i
2. 奇异值就是 $\sigma_i = \sqrt{\lambda_i}$
3. 对应特征向量就是右奇异向量 $\vec{v}_i$

为什么要看 $A^T A$? 因为 A 可能不是方阵，不能直接做普通特征值分解；但 $A^T A$ 一定是 $n \times n$ 方阵，而且是对称半正定矩阵。它的特征值都是非负实数，特征向量可以选成正交基，这正好适合构造 SVD。

10.4. SVD 和特征值分解有什么区别？

特征值分解关注的是：

$$A\vec{v} = \lambda\vec{v} \quad (250)$$

也就是找那些经过 A 作用后仍然留在原方向上的向量。因此它要求输入空间和输出空间是同一个空间，通常只适合方阵。

SVD 关注的是：

$$A\vec{v}_i = \sigma_i\vec{u}_i \quad (251)$$

这里输入方向 $\vec{v}_i$ 和输出方向 $\vec{u}_i$ 可以不在同一个空间。 A 可以把 $\mathbb{R}^n$ 映射到 $\mathbb{R}^m$ ，所以 SVD 能处理长方形矩阵。

可以这样对比：

1. 特征值分解：寻找“不改变方向”的向量
2. SVD：寻找“输入正交方向如何变成输出正交方向”
3. 特征值可能是负数或复数
4. 奇异值永远是非负实数，表示纯粹的拉伸倍数
5. 特征值分解不一定存在完整形式
6. SVD 对任意矩阵都存在

所以在工程计算里，SVD 往往比特征值分解更稳健。

10.5. 低秩近似

SVD 最重要的应用之一是低秩近似。设

$$A = U\Sigma V^T \quad (252)$$

把它展开，可以写成：

$$A = \sigma_1\vec{u}_1\vec{v}_1^T + \sigma_2\vec{u}_2\vec{v}_2^T + \dots + \sigma_r\vec{u}_r\vec{v}_r^T \quad (253)$$

其中 $r = \text{rank}(A)$ 。每一项

$$\sigma_i\vec{u}_i\vec{v}_i^T \quad (254)$$

都是一个秩为 1 的简单矩阵。奇异值越大，说明这一项对原矩阵越重要。

如果只保留最大的前 k 个奇异值，就得到：

$$A_k = \sigma_1\vec{u}_1\vec{v}_1^T + \dots + \sigma_k\vec{u}_k\vec{v}_k^T \quad (255)$$

这个 A_k 是一个秩不超过 k 的矩阵。它用更少的信息近似原矩阵，却保留了最主要的结构。

定理 10.5.1: (Eckart-Young 定理) 在所有秩不超过 k 的矩阵中，由 SVD 截断得到的 A_k 是最接近 A 的近似矩阵。

这就是图像压缩、推荐系统、降噪、潜在语义分析中经常使用 SVD 的原因。大矩阵里往往有很多细节和噪声，最大的几个奇异值对应主要结构，小奇异值对应较弱变化或噪声。丢掉小奇异值，就能用更低维的表示保留主体信息。

例 10.5.1: 一张灰度图像可以看成矩阵 A 。如果图像大小是 1000×1000 ，原始矩阵需要存储一百万个数。

做 SVD 后，如果只保留前 50 个奇异值以及对应的左右奇异向量，就可以用较少参数近似原图。图像的主要轮廓通常还能保留，而高频细节和噪声会被削弱。

10.6. 和 PCA 的关系

PCA 用来解决高维数据中“哪些方向最重要”的问题。假设每个样本有很多特征，例如一张图片可以有几千个像素，一个用户画像可以有上百个行为指标。直接处理这么多维度会带来计算成本，也可能混入大量冗余信息。PCA 的目标是找到少数几个方向，让数据投影到这些方向后仍然保留主要变化。

首先把数据中心化，也就是每个特征减去自己的平均值。然后计算协方差矩阵 C 。协方差矩阵的元素 C_{ij} 描述第 i 个特征和第 j 个特征是否一起变化。若 C_{ij} 很大，说明两个特征经常同向变化；若接近 0，说明它们关系较弱。

更具体地说，假设有 m 个样本，每个样本有 n 个特征，把它们按行放进数据矩阵 $X \in \mathbb{R}^{m \times n}$ 。中心化之后得到 X_c ，其中每一列的均值都是 0。协方差矩阵通常写成：

$$C = \frac{1}{m-1} X_c^T X_c \quad (256)$$

这个公式的含义很直接： $X_c^T X_c$ 的第 i, j 个元素，是第 i 个特征和第 j 个特征在所有样本上的乘积求和；再除以 $m-1$ ，就得到它们的样本协方差。因此 C_{ii} 是第 i 个特征自己的方差， C_{ij} 是两个不同特征之间一起变化的程度。

PCA 的核心就是求：

$$C\vec{v} = \lambda\vec{v} \quad (257)$$

其中 $\vec{v}$ 是一个方向， λ 是数据沿这个方向的方差大小。由于 $C = \frac{1}{m-1} X_c^T X_c$ ，所以 $C^T = C$ ，协方差矩阵一定是对称矩阵。实对称矩阵有一个非常好的性质：不同特征值对应的特征向量天然正交；如果某个特征值有重数，也可以在它的特征空间内部重新选出一组正交基。这个结论的推导见后面的附录。因此 PCA 得到的主成分方向可以取成互相垂直的坐标轴。最大的特征值对应的特征向量，就是数据变化最剧烈的方向，也称为第一主成分；第二大的特征值对应第二主成分，并且和第一主成分正交。

如果只保留前 k 个最大特征值对应的特征向量，就可以把原始高维数据投影到 k 维空间中。这样做的含义不是随便丢掉一些坐标，而是保留最能解释数据变化的方向。图像压缩、特征降维、可视化、噪声过滤都可以用这个思想。

特征值在这里有一个很直接的解释：它衡量对应方向上有多少“信息量”或“变化量”。如果某个特征值很小，说明数据沿该方向几乎没有变化，把它丢掉通常损失不大。特征向量则告诉我们新的坐标轴应该朝哪里放。

PCA 实际上可以通过 SVD 来实现。设中心化后的数据矩阵是 X_c ，协方差矩阵为：

$$C = \frac{1}{m-1} X_c^T X_c \quad (258)$$

如果对 X_c 做 SVD:

$$X_c = U \Sigma V^T \quad (259)$$

则:

$$X_c^T X_c = V \Sigma^T \Sigma V^T \quad (260)$$

所以 V 的列向量就是协方差矩阵的特征向量，也就是 PCA 的主成分方向；奇异值平方除以 $m-1$ ，就是对应方向上的方差：

$$\lambda_i = \frac{\sigma_i^2}{m-1} \quad (261)$$

因此，PCA 可以理解为：对中心化数据矩阵做 SVD，取最大的几个奇异值对应的右奇异向量作为主成分方向。

10.7. 常见误区

1. **奇异值不是特征值。** 奇异值是非负拉伸倍数，来自 $A^T A$ 特征值的平方根；特征值描述的是 A 自己的不变方向。
2. **SVD 不要求矩阵是方阵。** 这是它比普通特征值分解更通用的重要原因。
3. **U 和 V 不是随便两个矩阵。** 它们是正交矩阵，列向量分别给出输出空间和输入空间中的正交方向。
4. **小奇异值不一定完全没用。** 在压缩和降噪中可以丢掉小奇异值，但如果任务依赖细节，小奇异值也可能包含重要信息。
5. **Σ 不一定是方阵。** 当 A 是 $m \times n$ 时， Σ 也是 $m \times n$ ，只是在主对角线上放非负奇异值。

11. 附录：实对称矩阵为什么给出正交特征向量

PCA 中会用到一个关键事实：协方差矩阵是实对称矩阵，因此它的特征向量可以选成一组正交方向。这里给出一个直接推导。

设 A 是实对称矩阵，即 $A^T = A$ 。假设 $\vec{u}$ 和 $\vec{v}$ 是 A 的两个特征向量：

$$A\vec{u} = \lambda\vec{u} \quad (262)$$

$$A\vec{v} = \mu\vec{v} \quad (263)$$

其中 $\lambda \neq \mu$ 。考虑内积 $\langle A\vec{u}, \vec{v} \rangle$ 。一方面，由 $A\vec{u} = \lambda\vec{u}$ 得：

$$\langle A\vec{u}, \vec{v} \rangle = \langle \lambda\vec{u}, \vec{v} \rangle = \lambda \langle \vec{u}, \vec{v} \rangle \quad (264)$$

另一方面，因为 A 是实对称矩阵，矩阵 A 可以在内积中从左边移到右边：

$$\langle A\vec{u}, \vec{v} \rangle = \langle \vec{u}, A\vec{v} \rangle \quad (265)$$

再利用 $A\vec{v} = \mu\vec{v}$ ，得到：

$$\langle \vec{u}, A\vec{v} \rangle = \langle \vec{u}, \mu\vec{v} \rangle = \mu \langle \vec{u}, \vec{v} \rangle \quad (266)$$

于是同一个量有两种表示：

$$\lambda \langle \vec{u}, \vec{v} \rangle = \mu \langle \vec{u}, \vec{v} \rangle \quad (267)$$

移项可得：

$$(\lambda - \mu)\langle \vec{u}, \vec{v} \rangle = 0 \quad (268)$$

由于 $\lambda \neq \mu$ ，所以只能有：

$$\langle \vec{u}, \vec{v} \rangle = 0 \quad (269)$$

这就证明了：实对称矩阵中，不同特征值对应的特征向量相互正交。

如果某个特征值有重数，情况稍微不同。同一个特征值对应的特征空间里，任意两个特征向量不一定天然正交。但这个特征空间本身是一个向量空间，可以在其中使用格拉姆-施密特正交化，选出一组正交基。由于这些新向量仍然留在同一个特征空间内，它们仍然是同一个特征值对应的特征向量。

因此，对实对称矩阵来说，可以把所有特征空间中的基都选成正交的；不同特征空间之间天然正交，同一个特征空间内部再做正交化。最后得到的就是一组正交特征向量。这也是谱定理和 PCA 能成立的核心原因。

12. 矩阵乘法在工程中如何优化？

数学上，矩阵乘法只有一个公式：

$$C = AB \quad (270)$$

其中：

$$A \in \mathbb{R}^{M \times K}, \quad B \in \mathbb{R}^{K \times N}, \quad C \in \mathbb{R}^{M \times N} \quad (271)$$

每个输出元素是一个长度为 K 的点积：

$$C_{ij} = \sum_{k=1}^K A_{ik} B_{kj} \quad (272)$$

如果直接三重循环：

```
for i in 0..M:
    for j in 0..N:
        acc = 0
        for k in 0..K:
            acc += A[i, k] * B[k, j]
        C[i, j] = acc
```

计算量是 $O(MNK)$ 。这件事本身无法消失。高性能矩阵乘法优化的核心不是把乘加次数变没，而是让硬件在做这些乘加时尽量不等待数据。

现代 CPU/GPU 上，矩阵乘法通常叫 GEMM：

$$C = \alpha AB + \beta C \quad (273)$$

GEMM 是 BLAS 里的核心算子，也是神经网络中 linear layer、MLP、attention 投影、卷积展开后的主力算子。很多深度学习性能问题，最后都会落到 GEMM 是否足够快。

12.1. 第一个核心：数据复用

矩阵乘法有大量重复读取。计算 C 的一小块时，同一块 A 会被多个 B 列使用，同一块 B 会被多个 A 行使用。

假设我们计算一个 $m \times n$ 的输出小块：

$$C_{\text{tile}} \in \mathbb{R}^{m \times n} \quad (274)$$

它需要沿 K 方向不断累加：

$$C_{\text{tile}} += A_{\text{tile}} B_{\text{tile}} \quad (275)$$

其中：

$$A_{\text{tile}} \in \mathbb{R}^{m \times k}, \quad B_{\text{tile}} \in \mathbb{R}^{k \times n} \quad (276)$$

读入一次 A_{tile} 和 B_{tile} 后，可以产生 $m \times n \times k$ 次乘加。只要 m, n, k 选得合适，同一份数据就能被反复使用很多次。

这就是分块，也叫 tiling 或 blocking。它是矩阵乘法优化最重要的思想。

没有分块时，程序可能反复从慢速内存读同一批数据。分块后，先把一小块数据搬到更快的存储层级里，然后在这块数据上做尽可能多的计算。

12.2. 分块为什么和原矩阵乘法等价？

分块不是改变矩阵乘法的定义，而是把同一批求和项重新分组。矩阵乘法本来就是：

$$C_{ij} = \sum_{k=1}^K A_{ik} B_{kj} \quad (277)$$

分块只是把 $k = 1 \dots K$ 这段求和拆成几段。例如：

$$\sum_{k=1}^8 A_{ik} B_{kj} = \sum_{k=1}^4 A_{ik} B_{kj} + \sum_{k=5}^8 A_{ik} B_{kj} \quad (278)$$

右边没有少算任何项，也没有多算任何项，只是把 8 项求和分成了前 4 项和后 4 项。因此数学结果完全一样。

看一个 8×8 的具体分块例子。设：

$$C = AB \quad (279)$$

其中：

$$A, B, C \in \mathbb{R}^{8 \times 8} \quad (280)$$

把 A 按行和列都切成两半，每块是 4×4 ：

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad (281)$$

其中：

1. A_{11} 是 A 的第 1 到 4 行、第 1 到 4 列
2. A_{12} 是 A 的第 1 到 4 行、第 5 到 8 列
3. A_{21} 是 A 的第 5 到 8 行、第 1 到 4 列
4. A_{22} 是 A 的第 5 到 8 行、第 5 到 8 列

同样把 B 切成四个 4×4 块：

$$B = \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad (282)$$

那么 C 也可以按同样方式切成四个 4×4 块：

$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix} \quad (283)$$

分块矩阵乘法给出：

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad (284)$$

$$C_{12} = A_{11}B_{12} + A_{12}B_{22} \quad (285)$$

$$C_{21} = A_{21}B_{11} + A_{22}B_{21} \quad (286)$$

$$C_{22} = A_{21}B_{12} + A_{22}B_{22} \quad (287)$$

这个形式看起来像普通 2×2 矩阵乘法，只是里面的元素从标量变成了 4×4 小矩阵。

为什么它等价？拿左上角块 C_{11} 来看。 C_{11} 对应最终矩阵 C 的第 1 到 4 行、第 1 到 4 列。也就是说， C_{11} 中的每个元素都是：

$$C_{ij}, \quad 1 \leq i \leq 4, \quad 1 \leq j \leq 4 \quad (288)$$

按普通矩阵乘法：

$$C_{ij} = \sum_{k=1}^8 A_{ik}B_{kj} \quad (289)$$

把这 8 项按 k 分成两段：

$$C_{ij} = \sum_{k=1}^4 A_{ik}B_{kj} + \sum_{k=5}^8 A_{ik}B_{kj} \quad (290)$$

第一段：

$$\sum_{k=1}^4 A_{ik}B_{kj} \quad (291)$$

正是 $A_{11}B_{11}$ 中第 i, j 个位置的结果。因为 A_{11} 取的是 A 的前 4 列， B_{11} 取的是 B 的前 4 行。

第二段：

$$\sum_{k=5}^8 A_{ik}B_{kj} \quad (292)$$

正是 $A_{12}B_{21}$ 中第 i, j 个位置的结果。因为 A_{12} 取的是 A 的第 5 到 8 列， B_{21} 取的是 B 的第 5 到 8 行。

所以：

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad (293)$$

不是新的规则，而是普通矩阵乘法公式按 k 维度拆成两段后的结果。

再看一个具体元素，例如 $C_{2,3}$ 。它在左上角块 C_{11} 里面。普通算法写成：

$$C_{2,3} = A_{2,1}B_{1,3} + A_{2,2}B_{2,3} + A_{2,3}B_{3,3} + A_{2,4}B_{4,3} + A_{2,5}B_{5,3} + A_{2,6}B_{6,3} + A_{2,7}B_{7,3} + A_{2,8}B_{8,3} \quad (294)$$

分块算法只是把它写成：

$$C_{2,3} = (A_{2,1}B_{1,3} + A_{2,2}B_{2,3} + A_{2,3}B_{3,3} + A_{2,4}B_{4,3}) + (A_{2,5}B_{5,3} + A_{2,6}B_{6,3} + A_{2,7}B_{7,3} + A_{2,8}B_{8,3})$$

第一组来自 $A_{11}B_{11}$ ，第二组来自 $A_{12}B_{21}$ 。八个乘积项完全一样，只是括号位置变了。

因此，分块矩阵乘法成立的数学原因很简单：

1. 矩阵乘法的每个元素本质上是沿 K 维求和
2. 分块只是把 K 维求和拆成几个连续区间
3. 加法满足结合律，所以先分段求和再相加，和一次性求完整个和相同
4. 工程优化利用这个等价性，把每一段数据放到更快的存储层级中复用

所以 tiling 在数学上没有近似误差；它只是改变计算顺序和数据访问方式。真正可能引入细微差异的是浮点数加法的舍入顺序，因为浮点加法不完全满足结合律。但在实数数学里，分块乘法和原始乘法完全等价。

12.3. 分块如何降低拷贝？

继续用上面的 8×8 例子。我们只看左上角输出块：

$$C_{11} \in \mathbb{R}^{4 \times 4} \quad (296)$$

它由两部分累加得到：

$$C_{11} = A_{11}B_{11} + A_{12}B_{21} \quad (297)$$

这里每个小块都是 4×4 。

如果不分块，直接逐元素计算 C_{11} ，它有 16 个输出元素。每个输出元素 C_{ij} 都要做一个长度为 8 的点积：

$$C_{ij} = \sum_{k=1}^8 A_{ik}B_{kj} \quad (298)$$

对一个输出元素来说，需要读取：

1. A 的 8 个元素： $A_{i,1}, \dots, A_{i,8}$
2. B 的 8 个元素： $B_{1,j}, \dots, B_{8,j}$

也就是 16 次标量读取。 C_{11} 有 16 个输出元素，所以如果完全不复用，读取次数大约是：

$$16 \text{ 个输出元素} \times (8 + 8) = 256 \text{ 次标量读取} \quad (299)$$

但这里有大量重复。例如计算 $C_{2,1}, C_{2,2}, C_{2,3}, C_{2,4}$ 时，都会用到同一行：

$$A_{2,1}, A_{2,2}, \dots, A_{2,8} \quad (300)$$

如果每算一个输出元素都重新从慢速内存读取这一行， A 的同一批数据就被反复搬运了。

同理，计算 $C_{1,3}, C_{2,3}, C_{3,3}, C_{4,3}$ 时，都会用到同一列：

$$B_{1,3}, B_{2,3}, \dots, B_{8,3} \quad (301)$$

如果每算一个输出元素都重新读取这一列， B 的同一批数据也被反复搬运。

分块的做法是：先把当前需要的小块搬到更快的存储中，例如 GPU shared memory 或 CPU cache，然后在快存储里反复使用。

对 C_{11} 的第一半：

$$A_{11}B_{11} \quad (302)$$

只需要先搬入：

1. A_{11} ：16 个元素
2. B_{11} ：16 个元素

一共 32 个元素。搬进来之后，它们可以产生一个 4×4 的部分结果，也就是 16 个输出元素的部分累加。

关键在于复用次数：

1. A_{11} 中的每个元素会被 B_{11} 的 4 个列方向使用，所以通常会参与 4 次乘法
2. B_{11} 中的每个元素会被 A_{11} 的 4 个行方向使用，所以通常也会参与 4 次乘法

也就是说，搬一次 A_{11} 和 B_{11} ，不是只服务一个输出元素，而是服务整个 4×4 输出块。

对 C_{11} 的第二半：

$$A_{12}B_{21} \quad (303)$$

同理只需要再搬入：

1. A_{12} ：16 个元素
2. B_{21} ：16 个元素

也是 32 个元素。

所以计算整个 C_{11} 时，分块后从慢速内存搬到快速存储的核心数据量是：

$$(16 + 16) + (16 + 16) = 64 \text{ 个标量元素} \quad (304)$$

而逐元素、不复用的方式大约会读取：

$$256 \text{ 个标量元素} \quad (305)$$

这个小例子里，慢速数据读取量从 256 降到 64，减少为原来的四分之一。乘加次数没有减少，仍然要做：

$$4 \times 4 \times 8 = 128 \text{ 次乘加} \quad (306)$$

减少的是数据搬运。

可以用下面的表看得更直观：

计算 C_{11} ，一个 4×4 输出块

不分块：

16 个输出元素
每个元素读取 8 个 A + 8 个 B
约 $16 \times 16 = 256$ 次慢速读取

分块：

阶段 1：读取 $A_{11}(16) + B_{11}(16) = 32$
阶段 2：读取 $A_{12}(16) + B_{21}(16) = 32$
总共约 64 次慢速读取

这就是分块降低拷贝的本质：**不是少算，而是同一份数据搬进快存储后被多次使用。**

把这个思想推广到更大的矩阵也一样。假设计算一个 $m \times n$ 的输出 tile，并且每次沿 K 方向处理 k 个元素，那么每个阶段需要搬入：

$$A_{\text{tile}} \in \mathbb{R}^{m \times k}, \quad B_{\text{tile}} \in \mathbb{R}^{k \times n} \quad (307)$$

搬运量是：

$$mk + kn \quad (308)$$

但产生的乘加数量是：

$$mnk \quad (309)$$

所以每搬入一个数据块，就能产生很多计算。 m 和 n 越大，单次搬入的数据被复用越多。当然 tile 不能无限变大，因为 shared memory、cache 和寄存器容量有限。

在 GPU 上，这里的“拷贝”通常指：

1. 从 global memory 把 A_{tile} 和 B_{tile} 读到 shared memory
2. 从 shared memory 再读到寄存器或 Tensor Core 需要的 fragment
3. 在寄存器里累加 C_{tile}
4. 最后把 C_{tile} 写回 global memory

如果没有分块，很多 A 和 B 元素会被不同线程反复从 global memory 读取。global memory 带宽虽然高，但延迟和能耗都远高于 shared memory 和寄存器。分块以后，一个 CTA 内的线程协作把 A, B 小块搬进 shared memory，随后多个线程、多条 Tensor Core 指令反复读取这份 shared memory 数据，global memory 的重复读取就少了。

再换成 CUTLASS 的说法：

1. **global memory tile**：从显存读一块 A 和一块 B
2. **shared memory tile**：让一个 CTA 内的 warp 共享这批数据
3. **warp/register tile**：把 shared memory 中的数据继续切成更小片，送进 Tensor Core
4. **accumulator tile**：结果暂存在寄存器里，多轮 K block 累加完成后再写回

所以 CUTLASS 里的 tile shape 和 pipeline，本质上都在围绕一个目标：让每一次从慢速内存搬来的数据，在被丢弃之前尽可能多地参与乘加。

12.4. 第二个核心：存储层级

硬件上的内存不是一个东西，而是很多层：

1. 全局内存或主存：容量大，但延迟高、带宽相对低
2. L2/L1 cache 或 shared memory：容量较小，但更快
3. 寄存器：容量最小，但最快
4. Tensor Core 或 SIMD/FMA 单元：真正执行乘加的计算单元

高性能 GEMM 的目标是让数据沿着这些层级逐步流动：

```
global memory
-> shared memory / cache
    -> registers
        -> FMA / Tensor Core
```

每往下一层，容量更小，但速度更快。优化的关键是：昂贵的数据搬运尽量少做，便宜的寄存器级计算尽量多做。

这也解释了为什么矩阵乘法特别适合优化。相比逐元素操作，GEMM 的算术强度很高。算术强度可以粗略理解为：

$$\text{arithmetic intensity} = \frac{\text{计算次数}}{\text{数据搬运量}} \quad (310)$$

一次读入的矩阵块可以参与很多乘加，所以只要分块做得好，硬件计算单元就能长时间保持忙碌。

12.5. 第三个核心：并行粒度

在 GPU 上，GEMM 会被拆成多层任务：

1. 整个矩阵乘法被拆成很多 thread block 或 CTA
2. 每个 CTA 负责计算一个较大的 C tile
3. 一个 CTA 内部有多个 warp
4. 每个 warp 负责更小的 tile
5. 每个线程持有若干寄存器中的 accumulator

可以把它想成分工：

```
整个 GEMM
-> CTA tile
    -> warp tile
        -> instruction tile
            -> register accumulator
```

每一级 tile 大小都会影响性能。tile 太小，数据复用不够；tile 太大，shared memory 和寄存器不够用，occupancy 会下降。

所以矩阵乘法优化不是简单地“开更多线程”。真正的问题是：

1. 每个线程算多少输出元素
2. 每个 warp 负责多大的输出块
3. 每个 CTA 负责多大的输出块
4. 沿 K 方向每次推进多大
5. shared memory 和寄存器是否够用
6. 并发度是否足以隐藏内存延迟

12.6. 第四个核心：访存模式

GPU 对连续、对齐、合并的访存最友好。如果一个 warp 中的线程读取连续地址，硬件可以把它合并成更高效的内存事务。反过来，如果每个线程都读很散的地址，带宽利用率会很差。

因此 GEMM 会非常关心 layout：

1. A 是 row-major 还是 column-major
2. B 是否需要转置视图
3. 每次 load 是否对齐
4. shared memory 中的布局是否会产生 bank conflict
5. Tensor Core 指令期望的数据排布是什么

很多时候，优化矩阵乘法不是改数学公式，而是在改数据怎么摆、线程怎么读、读进来以后怎么重排。

12.7. Tensor Core 在做什么？

现代 NVIDIA GPU 上，深度学习 GEMM 通常会使用 Tensor Core。Tensor Core 不是让矩阵乘法少算，而是提供更高吞吐的矩阵乘加指令。

普通 CUDA core 更像是在做标量或向量 FMA：

$$a \times b + c \quad (311)$$

Tensor Core 则一次处理一个小矩阵乘加：

$$D = AB + C \quad (312)$$

例如某些指令会处理类似 $16 \times 8 \times 16$ 或其他固定形状的小块。具体形状取决于 GPU 架构、数据类型和指令形式。

这带来两个要求：

1. 数据必须被组织成 Tensor Core 指令喜欢的小 tile
2. 线程协作必须把这些小 tile 正确送进矩阵乘加指令

所以 Tensor Core GEMM 的代码通常不是直接写三重循环，而是围绕硬件矩阵指令组织数据搬运、shared memory 排布和寄存器 accumulator。

12.8. CUTLASS 中最核心考虑的点

CUTLASS 可以理解为 NVIDIA 官方提供的一套 CUDA GEMM 模板库。它的目标不是只写一个 GEMM kernel，而是把高性能 GEMM 拆成可组合的层次结构，让不同矩阵形状、数据类型、硬件架构和融合需求都能生成合适的 kernel。

它最核心考虑的点可以概括为一句话：

用层次化 tile 把数据搬运、数据复用和硬件矩阵指令匹配起来，让计算单元尽量不等内存。

更具体地说，CUTLASS 关注这些问题。

第一，tile shape 怎么选。

典型 GEMM 会有多级 shape：

ThreadblockShape: CTA 负责的 $M \times N \times K$ 块
WarpShape: 每个 warp 负责的子块
InstructionShape: Tensor Core 指令的最小矩阵乘加形状

例如可以粗略想成：

CTA 算一个 128×128 的 C tile
每个 warp 算其中一部分
每条 Tensor Core 指令算更小的 mma tile

这些数字不是越大越好。更大的 tile 提高复用，但会消耗更多 shared memory 和寄存器。寄存器太多会降低 occupancy，shared memory 太多会减少同一个 SM 上能同时驻留的 CTA 数量。

第二，pipeline 怎么设计。

GEMM 沿 K 方向一块一块推进：

$$C_{\text{tile}} += A_{K \text{ block}} B_{K \text{ block}} \quad (313)$$

如果等当前块算完再去加载下一块，计算单元会等待内存。高性能 kernel 会做流水线：

1. 当前 stage 正在用 shared memory 里的数据做 mma
2. 下一个 stage 的数据同时从 global memory 预取进 shared memory
3. 等当前 stage 算完，下一块数据已经准备好

在较新的 NVIDIA 架构上，这类流水线会结合异步拷贝，例如 `cp.async` 或 TMA，让数据搬运和计算重叠。

第三，shared memory layout 怎么设计。

数据搬到 shared memory 后，还要被 warp 高效读取给 Tensor Core。shared memory 有 bank，如果多个线程同时访问同一个 bank，可能产生 bank conflict。CUTLASS 会使用特定 layout 或 swizzle，让访问模式更均匀，减少冲突。

第四，register accumulator 怎么安排。

每个线程通常会在寄存器里保存一小片 C 的累加结果。寄存器 accumulator 越多，单个线程能做更多计算，数据复用更好；但寄存器用太多，会让同一个 SM 上能同时运行的 warp 变少。

这就是常见取舍：

1. 更多寄存器：单线程工作更多，复用更好
2. 更少寄存器：occupancy 更高，更容易隐藏延迟

最优点取决于矩阵形状、数据类型和具体 GPU。

第五，epilogue 怎么融合。

GEMM 算完后，真实模型里往往不只是写回 C 。还可能要做：

1. 加 bias
2. 乘 alpha、加 beta C
3. 激活函数，例如 ReLU、GELU
4. 量化或反量化
5. residual add

如果先把 GEMM 结果写回显存，再启动另一个 kernel 做 bias 或激活，就会多一次读写显存。CUTLASS 的 epilogue 机制允许在写回 C 前把这些操作融合进去，减少内存带宽浪费。

第六，特殊形状怎么处理。

真实模型中的矩阵不一定是漂亮的 M, N, K 都很大且整除 tile size。可能会出现：

1. M 很小、 N, K 很大
2. batch 很多但单个矩阵很小
3. K 特别大，需要 split- K 并行
4. 边界 tile 不完整，需要 predication
5. 不同数据类型混合，例如 FP16 输入、FP32 累加、FP16 输出

CUTLASS 的模板参数和 kernel 组织方式，就是为了覆盖这些变化。

12.9. CPU 上也是同一套思想

CPU GEMM 和 GPU GEMM 的硬件不同，但核心思想相同：

1. 使用 cache blocking，让 A, B, C 的小块留在 cache 中复用
2. 使用 register blocking，让一小块 C accumulator 留在寄存器中
3. 使用 SIMD 指令，一条指令处理多个乘加

4. 使用多线程，把不同输出 tile 分给不同 CPU core
5. 对齐和预取数据，减少 cache miss

所以不管是 OpenBLAS、MKL、BLIS，还是 GPU 上的 cuBLAS、CUTLASS，本质问题都类似：

矩阵乘法的数学公式很简单，但性能取决于能不能把数据复用组织到硬件最擅长的层级上。

12.10. 一个总结视角

优化矩阵乘法时，最重要的不是先背某个库的 API，而是抓住几个问题：

1. **算什么：** $C = AB$ ，核心是大量点积
2. **数据在哪里：**global memory、shared memory、cache、register
3. **数据复用几次：**tile 越合理，复用越高
4. **谁来算：**CTA、warp、thread、Tensor Core 或 SIMD
5. **算和搬能否重叠：**pipeline 是否把内存延迟藏起来
6. **写回前能否融合：**epilogue 是否减少额外 kernel 和显存读写

因此，CUTLASS 的核心不是发明新的矩阵乘法定义，而是把同一个矩阵乘法公式映射到 GPU 的层次结构上：全局内存负责大容量，shared memory 负责块级复用，寄存器负责累加，Tensor Core 负责高吞吐矩阵乘加，pipeline 负责让这些环节尽量同时工作。

13. 参考资源

1. Strang, G. (2016). **Introduction to Linear Algebra**. 5th ed. Wellesley-Cambridge Press.
2. Axler, S. (2015). **Linear Algebra Done Right**. 3rd ed. Springer.
3. Gilbert Strang 麻省理工学院线性代数公开课 (Bilibili)
4. 《The Art of Linear Algebra》：线性代数的可视化解