基与秩综合练习题

一、题目背景

我们希望通过一个 5×4 的矩阵来理解 **秩（rank）**、**基（basis）** 与 **线性相关性（linear dependence）** 之间的关系，
并用 Python 进行验证与计算。

二、数据定义

已知四个列向量

$$x_1, x_2, x_3, x_4 \in \mathbb{R}^5$$

，构造矩阵：

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ -1 \\ -1 \\ -1 \end{bmatrix}, \quad x_2 = \begin{bmatrix} 2 \\ -1 \\ 1 \\ 2 \\ -2 \end{bmatrix}, \quad x_3 = \begin{bmatrix} 3 \\ -4 \\ 3 \\ 5 \\ -3 \end{bmatrix}, \quad x_4 = \begin{bmatrix} -1 \\ 8 \\ -5 \\ -6 \\ 1 \end{bmatrix}$$

$$A = [x_1\ x_2\ x_3\ x_4]$$

三、任务要求

1. **求矩阵的秩**
   - 使用 `np.linalg.matrix_rank(A)`
   - 理解“秩”代表矩阵中**最大线性无关列数**。
2. **确定列空间的基**
   - 用 `sympy.Matrix(A).rref()` 求行最简形（RREF）。
   - 记录主元列索引（pivot columns）。
   - 从 x_1, x_2, x_3, x_4 中选出相应列，构成矩阵的**列空间基**。
3. **验证线性关系**
   - 假设 $x_3 = ax_1 + bx_2 + cx_4$，
     用 `np.linalg.lstsq(A, x3)` 或选取主元列矩阵解出 $[a, b, c]$。
   - 判断哪些列可以被其它列线性表示。
4. **构造正交基（QR分解）**
   - 用 `np.linalg.qr(A)` 得到 (Q, R)。

- 取 Q 的前 `rank(A)` 列组成正交基 Q_r。
- 验证 $Q_r^T Q_r = I$。

5. **计算正交投影**

- 给定向量 $y = [1, 0, 2, -1, 0]^T$，
  求其在 $\mathrm{Col}(A)$ 上的正交投影：
  $\hat{y} = Q_r Q_r^T y$。
- 计算残差 $r = y - \hat{y}$，并验证 $Q_r^T r = 0$。

6. **求坐标表示**

- 若列空间基为 $B = [b_1, b_2, b_3]$，
  解方程 $B\alpha = y$ 得到坐标向量 $[y]_B = \alpha$。
- 可用 `np.linalg.lstsq(B, y, rcond=None)` 实现。

🧠 四、思考提示

步骤	思考方向	Python 提示
1️⃣ 秩	代表"空间维数"	`matrix_rank(A)`
2️⃣ 基	选出"最小无关组"	`sp.Matrix(A).rref()`
3️⃣ 线性关系	判断多余列	解 `A[:,piv] * α ≈ x3`
4️⃣ 正交化	保留方向但使独立	`Q, R = qr(A)`
5️⃣ 投影	找"影子"	`y_hat = Qr @ Qr.T @ y`
6️⃣ 坐标	在基下的表示	`alpha, *_ = lstsq(B, y)`

In [2]:
```python
import numpy as np
from numpy.linalg import matrix_rank, lstsq, qr
import sympy as sp


# 构造矩阵 A
x1 = np.array([ 1,  2, -1, -1, -1.], dtype=float)
x2 = np.array([ 2, -1,  1,  2, -2.], dtype=float)
x3 = np.array([ 3, -4,  3,  5, -3.], dtype=float)
x4 = np.array([-1,  8, -5, -6,  1.], dtype=float)
A  = np.c_[x1, x2, x3, x4]  # 5x4

# 1) 求矩阵的秩
rank_A = matrix_rank(A)
print(f"1) 矩阵A的秩: {rank_A}")
print(f"   这表示矩阵A中最大线性无关列数为 {rank_A}\n")

# 2) 确定列空间的基
M = sp.Matrix(A)
RREF, piv = M.rref()  # piv 是主元列索引（从0起）
print("2) 行最简形的主元列索引 (0-based):", piv)

# 从A中选出主元列作为列空间的基
B = A[:, piv]
```

```
print("   列空间的基B:")
for i, col in enumerate(B.T, 1):
    print(f"   基向量b{i}: {col}")  # 修正了这里的变量名，将b_i改为b{i}
print()

# 3) 验证线性关系
# 找出非主元列（可被主元列线性表示的列）
non_piv = [i for i in range(A.shape[1]) if i not in piv]
print(f"3) 非主元列（可被线性表示的列）: {non_piv}")

# 以x3为例，用主元列表示
if 2 in non_piv:  # 检查x3是否为非主元列
    # 构建主元列矩阵
    A_piv = A[:, piv]
    # 求解x3 = a*x1 + b*x2 + c*x4
    coef, residuals, _, _ = lstsq(A_piv, x3, rcond=None)

    # 显示系数
    print(f"   x3用主元列表示的系数: {coef}")

    # 验证结果
    x3_hat = A_piv @ coef
    print(f"   验证x3 ≈ 线性组合: {np.allclose(x3, x3_hat)}")
print()

# 4) 构造正交基（QR分解）
Q, R = qr(A)
# 取Q的前rank(A)列组成正交基Qr
Qr = Q[:, :rank_A]
print("4) QR分解得到的正交基Qr:")
for i, col in enumerate(Qr.T, 1):
    print(f"   正交基向量q{i}: {col}")

# 验证Qr^T Qr = I
QTQ = Qr.T @ Qr
print("   验证Qr^T Qr ≈ I:")
print(QTQ)
print(f"   是否接近单位矩阵: {np.allclose(QTQ, np.eye(rank_A))}\n")

# 5) 计算正交投影
y = np.array([1., 0., 2., -1., 0.])
y_hat = Qr @ (Qr.T @ y)
resid = y - y_hat
print("5) 向量y在Col(A)上的正交投影:")
print(f"   y = {y}")
print(f"   投影y_hat = {y_hat}")
print(f"   残差r = {resid}")

# 验证Qr^T r = 0
QT_resid = Qr.T @ resid
print(f"   验证Qr^T r ≈ 0: {QT_resid}")
print(f"   是否接近零向量: {np.allclose(QT_resid, np.zeros(rank_A))}\n")

# 6) 求坐标表示
alpha, residuals, _, _ = lstsq(B, y, rcond=None)
print(f"6) y在基B下的坐标表示 [y]_B: {alpha}")

# 验证坐标表示的正确性
y_recon = B @ alpha
print(f"   用坐标重建y: {y_recon}")
```

```
print(f"   重建误差: {np.linalg.norm(y - y_recon)}")
print(f"   重建是否准确: {np.allclose(y, y_recon)}\n")

# 附：验证课堂给出的关系 x3 = -x1 + 2x2（若主元列是 {x1,x2,x4}）
coef_12, *_ = lstsq(np.c_[x1, x2], x3, rcond=None)
print("附加验证：")
print(f"x3用x1和x2表示的系数: {coef_12}")
print(f"x3 ≈ {coef_12[0]:.4f}*x1 + {coef_12[1]:.4f}*x2: {np.allclose(x3, coef_12
```

```
1) 矩阵A的秩: 3
   这表示矩阵A中最大线性无关列数为 3

2) 行最简形的主元列索引 (0-based): (0, 1, 3)
   列空间的基B:
   基向量b1: [ 1.  2. -1. -1. -1.]
   基向量b2: [ 2. -1.  1.  2. -2.]
   基向量b3: [-1.  8. -5. -6.  1.]

3) 非主元列（可被线性表示的列）: [2]
   x3用主元列表示的系数: [-1.00000000e+00  2.00000000e+00  1.89798485e-15]
   验证x3 ≈ 线性组合: True

4) QR分解得到的正交基Qr:
   正交基向量q1: [-0.35355339 -0.70710678  0.35355339  0.35355339  0.35355339]
   正交基向量q2: [ 0.57048265 -0.20134682  0.23490462  0.50336704 -0.57048265]
   正交基向量q3: [-0.26683576 -0.54348719 -0.41731202 -0.36637498 -0.57012315]
   验证Qr^T Qr ≈ I:
[[ 1.00000000e+00  6.22462318e-17 -4.94878455e-17]
 [ 6.22462318e-17  1.00000000e+00 -1.04187512e-16]
 [-4.94878455e-17 -1.04187512e-16  1.00000000e+00]]
   是否接近单位矩阵: True

5) 向量y在Col(A)上的正交投影:
   y = [ 1.  0.  2. -1.  0.]
   投影y_hat = [0.50245322 0.29140108 0.43288586 0.53958695 0.11278256]
   残差r = [ 0.49754678 -0.29140108  1.56711414 -1.53958695 -0.11278256]
   验证Qr^T r ≈ 0: [-1.02144513e-17  2.78748265e-16  5.07021962e-17]
   是否接近零向量: True

6) y在基B下的坐标表示 [y]_B: [ 5.92753623 -3.66666667 -2.04347826]
   用坐标重建y: [ 0.63768116 -0.82608696  0.62318841 -1.         -0.63768116]
   重建误差: 1.7652031126716008
   重建是否准确: False

附加验证:
x3用x1和x2表示的系数: [-1.  2.]
x3 ≈ -1.0000*x1 + 2.0000*x2: True
```

🔢 线性映射与基变换练习题

📘 一、题目背景

已知线性映射

$T : \mathbb{R}^3 \to \mathbb{R}^3$,

在标准基下的矩阵为：

$$A = \begin{bmatrix} 2 & 1 & -1 \\ 0 & 1 & 3 \\ 0 & 0 & 2 \end{bmatrix}.$$

定义新基：

$$B = (b_1, b_2, b_3), \quad b_1 = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \; b_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}, \; b_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix},$$

基矩阵：

$$P = [b_1 \; b_2 \; b_3].$$

二、问题

（1）计算矩阵的秩与线性映射特性

Q1. 求 $\mathrm{rank}(A)$。

💡 *提示：行阶梯形后非零行数即为秩。*

Q2. 判断 T 是否为可逆变换（双射）。

💡 *提示：若 $\mathrm{rank}(A) = 3$，则 A 可逆；若小于 3，则不是。*

Q3. 解释秩的含义。

💡 *提示：秩表示线性变换保留的"独立方向"数量，反映信息维度是否被压缩。*

（2）向量映射与坐标变换

设

$$x = \begin{bmatrix} 2 \\ -1 \\ 1 \end{bmatrix}.$$

Q4. 求 $T(x) = Ax$。

💡 *提示：直接矩阵乘法计算即可。*

Q5. 求 $[x]_B = P^{-1}x$。

💡 *提示：用公式 $[x]_B = P^{-1}x$，可通过计算或 `numpy.linalg.solve()` 得到。*

Q6. 验证

$$[T(x)]_B = A_B[x]_B.$$

💡 *提示：先计算 $[T(x)]_B = P^{-1}(Ax)$，再比较两边是否相等。*

（3）基变换矩阵的计算

Q7. 求线性映射在新基下的矩阵

$$A_B = P^{-1}AP.$$

💡 *提示：注意乘法顺序！P 在右表示输入基变换，P^{-1} 在左表示输出基还原。*

Q8. 思考：
为什么 A 与 A_B 不同，但它们都表示同一个线性映射 T?
💡 *提示：基变换改变了坐标的表示方式，而非变换本身。它们描述的是同一几何映射。*

🧠 三、辅助公式与思路

步骤	关键公式	含义说明
1	$\text{rank}(A)$	独立列/行的数量
2	$T(x) = Ax$	映射公式
3	$[x]_B = P^{-1}x$	坐标变换公式
4	$[T(x)]_B = P^{-1}(Ax)$	映射结果在新基下的坐标
5		线性

步骤	关键公式	含义说明
	$A_B = P^{-1}AP$	映射在新基下的矩阵
6	$[T(x)]_B = A_B [x]_B$	一致性验证

四、总结要点

- 秩越大 → 保留的维度越多；
- 若 $\text{rank}(A) = 3 \rightarrow T$ 为**可逆线性映射**；
- 基变换只是"坐标语言"的切换，不改变映射本质；
- $A_B = P^{-1}AP$ 是**同一映射在不同坐标系下的表达。**

> ✅ **结论：**
>
> - 秩（rank）反映了映射中信息的维度；
> - 基变换是"换语言不换规律"的过程；
> - 数学上，A 与 A_B 描述的是**同一个线性规律 T。**

In [4]:
```python
import sympy as sp

# 数据：矩阵A，新基B的列向量b1,b2,b3，基矩阵P，向量x
A = sp.Matrix([[2,1,-1],
               [0,1, 3],
               [0,0, 2]])
b1 = sp.Matrix([1,1,0])
b2 = sp.Matrix([1,0,1])
b3 = sp.Matrix([0,1,1])
P  = sp.Matrix.hstack(b1,b2,b3)
x  = sp.Matrix([2,-1,1])

# Q1 计算矩阵A的秩
rank_A = A.rank()
print(f"Q1. 矩阵A的秩: {rank_A}")

# Q2 判断T是否为可逆变换（通过行列式是否为0）
det_A = A.det()
is_invertible = det_A != 0
print(f"Q2. 矩阵A的行列式: {det_A}, T是可逆变换: {is_invertible}")
```

```
# Q4 计算T(x) = A x
Tx = A * x
print(f"Q4. T(x) = A x = {Tx}")

# Q5 计算[x]_B = P^{-1} x （LUsolve比直接求逆更稳健）
x_B = P.LUsolve(x)
print(f"Q5. [x]_B = {x_B}")

# Q7 计算新基下矩阵：A_B = P^{-1} A P
A_B = P.LUsolve(A * P)
print(f"Q7. 新基B下的矩阵A_B = \n{A_B}")

# Q6 验证一致性：[T(x)]_B ?= A_B [x]_B
Tx_B = P.LUsolve(Tx)  # [T(x)]_B = P^{-1}(A x)
A_B_x_B = A_B * x_B   # A_B [x]_B
is_consistent = Tx_B == A_B_x_B
print(f"Q6. [T(x)]_B = {Tx_B}")
print(f"Q6. A_B [x]_B = {A_B_x_B}")
print(f"Q6. 一致性验证结果：{is_consistent}")
```

```
Q1. 矩阵A的秩：3
Q2. 矩阵A的行列式：4，T是可逆变换：True
Q4. T(x) = A x = Matrix([[2], [2], [2]])
Q5. [x]_B = Matrix([[0], [2], [-1]])
Q7. 新基B下的矩阵A_B =
Matrix([[2, 1, 1], [1, 0, -1], [-1, 2, 3]])
Q6. [T(x)]_B = Matrix([[1], [1], [1]])
Q6. A_B [x]_B = Matrix([[1], [1], [1]])
Q6. 一致性验证结果：True
```

应用题

一、背景

在机器学习中，一个线性层就是一个线性映射：

$$T(x) = Ax$$

当我们**更换输入特征的基**（重新编码特征）时，线性层的**矩阵表达会随之改变**，但**几何作用不变**。本题让你通过计算与代码体会这一原理。

我们使用如下线性层（从 $\mathbb{R}^4$ 到 $\mathbb{R}^3$）：

$$A = \begin{bmatrix} 1 & 0 & 2 & -1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 3 & 0 \end{bmatrix}.$$

定义新的输入基：

$$b_1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix},\ b_2 = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix},\ b_3 = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix},\ b_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 1 \end{bmatrix}.$$

基矩阵为：

$$P = [b_1\ b_2\ b_3\ b_4].$$

二、问题与提示

1 计算 $\mathrm{rank}(A)$

它表示该层“**有效利用的独立输入方向数**”。

提示：

- 用 `A.rank()` 求秩；
- 若某一行能由其他行线性表示，则输出落在低维空间；
- $\mathrm{rank}(A) \le \min(3,4) = 3$。

---

2 判断是否有维度压缩

若 $\mathrm{rank}(A) < 4$，则有 $(4 - \mathrm{rank}(A))$ 个输入方向被“压平”。

提示：

- 线性层的输出维度 = $\mathrm{rank}(A)$；
- 维度差值即丢失信息量。

---

3 在新基 B 下的矩阵

只更换输入基时：

$$A_B = AP.$$

这表示：**先把新坐标还原为旧坐标（乘 P），再通过线性变换 A。**

---

4 验证 $T(x) = Ax = A_B[x]_B$

取 $x = \begin{bmatrix} 2 \\ 1 \\ 0 \\ 1 \end{bmatrix}$。

提示：

- 用 `P.LUsolve(x)` 求 $[x]_B = P^{-1}x$；
- 比较两边是否相等；
- 若相等，则说明换基后映射保持一致。

---

5 验证输出是否落在二维平面

若某行是其它行之和（如 $r_3 = r_1 + r_2$），
则对所有 x，有：

$$(Ax)_3 = (Ax)_1 + (Ax)_2,$$

输出落在平面

$$\{y \in \mathbb{R}^3 \mid y_3 = y_1 + y_2\}.$$

In [2]:
```python
import sympy as sp

# ==== 0. 数据设置 ====
A = sp.Matrix([
    [1, 0, 2, -1],
    [0, 1, 1,  1],
    [1, 1, 3,  0]
])
b1 = sp.Matrix([1,0,0,0])
b2 = sp.Matrix([1,1,0,0])
b3 = sp.Matrix([0,1,1,0])
b4 = sp.Matrix([0,0,1,1])
P  = sp.Matrix.hstack(b1,b2,b3,b4)
x  = sp.Matrix([2,1,0,1])

print("A ="); sp.pprint(A)
print("\nP ="); sp.pprint(P)
print("\nx =", x.T)

# ==== 1. 秩与“压缩维度” ====
rank_A = A.rank()
print("\n1. rank(A) =", rank_A)

# 2. 判断是否有维度压缩
if rank_A < 4:
    compression = 4 - rank_A
    print(f"2. 存在维度压缩，压缩维度为 {compression}")
else:
    print("2. 不存在维度压缩")

# ==== 3. 新基下的矩阵 A_B ====
A_B = A * P
print("\n3. A_B = A * P =")
sp.pprint(A_B)

# ==== 4. 坐标与一致性验证 ====
Tx = A * x
x_B = P.LUsolve(x)        # [x]_B = P^{-1} x
Tx_via_B = A_B * x_B      # A_B [x]_B

print("\n4. 变换一致性验证：")
print("Tx = A * x =")
sp.pprint(Tx)
print("[x]_B = P^{-1} x =")
sp.pprint(x_B)
print("A_B * [x]_B =")
```

```
sp.pprint(Tx_via_B)
print("Tx 与 A_B * [x]_B 是否相等？", Tx == Tx_via_B)

# ==== 5. 说明“输出落在二维平面上”的性质 ====
# 因为 row3 = row1 + row2，所以 (Ax)_3 = (Ax)_1 + (Ax)_2 对任意 x 成立
r1, r2, r3 = A.row(0), A.row(1), A.row(2)
print("\n5. 行关系检查： row3 == row1 + row2 ? ->", (r3 == r1 + r2))

# 用符号变量验证 (Ax)_3 - (Ax)_1 - (Ax)_2 == 0
x_sym = sp.symbols('x1 x2 x3 x4')
x_vec = sp.Matrix(x_sym)
expr = (A * x_vec)[2] - (A * x_vec)[0] - (A * x_vec)[1]
print("符号验证 (Ax)_3 - (Ax)_1 - (Ax)_2 = ", sp.simplify(expr))
print("这表明所有输出都落在平面 y3 = y1 + y2 上")
```

```
A =
⎡1  0  2  -1⎤
⎢           ⎥
⎢0  1  1  1 ⎥
⎢           ⎥
⎣1  1  3  0 ⎦

P =
⎡1  1  0  0⎤
⎢          ⎥
⎢0  1  1  0⎥
⎢          ⎥
⎢0  0  1  1⎥
⎢          ⎥
⎣0  0  0  1⎦

x = Matrix([[2, 1, 0, 1]])

1. rank(A) = 2
2. 存在维度压缩，压缩维度为 2

3. A_B = A * P =
⎡1  1  2  1⎤
⎢          ⎥
⎢0  1  2  2⎥
⎢          ⎥
⎣1  2  4  3⎦

4. 变换一致性验证：
Tx = A * x =
⎡1⎤
⎢ ⎥
⎢2⎥
⎢ ⎥
⎣3⎦
[x]_B = P^{-1} x =
⎡0 ⎤
⎢  ⎥
⎢2 ⎥
⎢  ⎥
⎢-1⎥
⎢  ⎥
⎣1 ⎦
A_B * [x]_B =
⎡1⎤
⎢ ⎥
⎢2⎥
⎢ ⎥
⎣3⎦
Tx 与 A_B * [x]_B 是否相等？ True

5. 行关系检查： row3 == row1 + row2 ? -> True
符号验证 (Ax)_3 - (Ax)_1 - (Ax)_2 =  0
这表明所有输出都落在平面 y3 = y1 + y2 上
```