

实验 2 NumPy 数组运算与矩阵特征分析实践实习指导书

一、实验基本信息

项目	内容
实验名称	NumPy 数组运算与矩阵特征分析实践
授课课程	大数据分析
授课班级	工业工程 2241 班
实验学时	2 学时（90 分钟）
实验类型	操作性 + 分析性实验
实验目标	1. 掌握 NumPy 数组创建、运算、切片及连接的细节操作，养成参数检查与结果验证的习惯；2. 掌握矩阵生成、转置及线性代数运算的实操流程，理解矩阵特征分析的数学逻辑；3. 能独立排查数组 / 矩阵操作中的常见错误（如形状不匹配、奇异矩阵），培养科研问题的细致排查能力。

二、实验原理

1. NumPy 数组 (ndarray) 核心原理

NumPy 的核心是 **N 维数组 (ndarray)**，其元素需为**相同数据类型**（如 int32、float64），通过 **shape**（维度）和 **dtype**（数据类型）属性描述数组特征。相较于 Python 列表，ndarray

支持批量数值运算（无需循环），大幅提升计算效率，是工业数据（如设备传感器数据、生产流程数据）批量处理的基础。

2. 数组运算原理

1. **元素级运算**：数组间加、减、乘、除（`+`、`-`、`*`、`/`）需满足“广播机制”——若数组形状不同，NumPy 会自动扩展较小数组至匹配形状（如 `(3,1)` 与 `(1,3)` 可广播为 `(3,3)`），但需确保扩展逻辑合法（如 `(2,3)` 与 `(2,2)` 无法广播）。
2. **数学函数运算**：NumPy 提供 `np.sqrt()`（平方根）、`np.sin()`（正弦）、`np.exp()`（指数）等向量化函数，直接作用于数组所有元素，避免 Python 循环的低效率。

1. 数组切片与连接原理

1. **切片**：通过 `[行索引, 列索引]` 实现数据提取，支持整数索引（`[1,2]`）、切片索引（`[1:3, :2]`）、逻辑索引（`[arr>5]`）及 `np.ix_()` 函数（构造行 / 列交叉索引，如 `arr[np.ix_([0,2], [1,3])]`）。
2. **连接**：`np.hstack()`（水平连接，需行数相同）、`np.vstack()`（垂直连接，需列数相同），本质是对数组维度的扩展与拼接，需严格匹配维度条件，否则触发 `ValueError`。

1. 矩阵与线性代数原理

1. 矩阵是特殊的二维数组，支持转置（`.T`）、共轭转置（`.H`）、逆矩阵（`.I`）等特有操作，与线性代数定义完全一致。
2. 核心线性代数运算依赖 `numpy.linalg` 模块：
 1. 逆矩阵（`np.linalg.inv(mat)`）：仅满秩矩阵（行列式 $\neq 0$ ）存在逆矩阵，满足 `mat @ mat.I ≈ 单位矩阵`；
 2. 特征值 / 特征向量（`np.linalg.eig(mat)`）：满足 `mat @ vec = val * vec`（`val` 为特征值，`vec` 为对应特征向量）；
 3. 线性方程组求解（`np.linalg.solve(A, b)`）：求解 `Ax = b`，要求 `A` 为满秩方阵。

三、实验环境准备

1. 硬件要求

计算机内存 $\geq 4\text{GB}$ ，硬盘剩余空间 $\geq 10\text{GB}$ （确保 Anaconda 及临时数据存储）。

2. 软件要求

1. 操作系统：Windows 10/11（64 位）；

2. 软件套件: Anaconda3 (建议 2024.02 及以上版本, 预装 NumPy 1.24+) ;
3. 开发工具: Spyder (Anaconda 自带, 确保启动正常) 。

1. 预检查步骤 (关键细节, 避免实验中报错)

1. 打开 Spyder, 在 IPython 控制台输入以下代码, 验证 NumPy 是否正常导入及版本:

```
import numpy as np
print("NumPy 版本: ", np.__version__) # 需≥1.24, 低于此版本需更新
```

1. 若提示 `ModuleNotFoundError`: 打开 Anaconda Prompt, 输入 `conda install numpy` 重新安装;
2. 若版本过低: 输入 `conda update numpy` 更新至最新兼容版本。

四、实验内容与操作步骤 (含细节要求)

本实验分 **5 个核心模块**, 每个模块需严格遵循“操作→验证→记录→反思”流程, 培养细致观察与严谨验证的科研习惯。

模块 1: NumPy 数组创建与属性查看

操作目标

掌握多种数组创建方法, 能准确解读数组 `shape`、`dtype` 属性, 排查数据类型不匹配问题。

操作步骤

1. 基于列表 / 元组创建数组 (验证数据类型一致性)

1. 在 Spyder 编辑器中新建文件, 命名为 `student_id_numpy_exp2.py` (如 `22010101_Li_exp2.py`, 禁止含中文 / 空格) ;
2. 输入以下代码 (需加注释), 创建不同数据类型的数组:

```
# 任务 1-1: 列表/元组转数组, 观察 dtype 变化
import numpy as np
# 列表转数组 (混合整数与浮点数)
list1 = [1, 2, 3.5, 4]
```

```

arr1 = np.array(list1)
# 元组转数组（纯整数）
tup1 = (5, 6, 7, 8)
arr2 = np.array(tup1, dtype=np.float64) # 指定 dtype 为 float64
# 嵌套列表转二维数组
nested_list = [[1, 2, 3], [4, 5, 6]]
arr3 = np.array(nested_list)
# 查看数组属性（关键：记录 shape 和 dtype）
print("arr1 属性: ", arr1.shape, arr1.dtype) # 预期: (4,) float64（因含 3.5）
print("arr2 属性: ", arr2.shape, arr2.dtype) # 预期: (4,) float64（手动指定）
print("arr3 属性: ", arr3.shape, arr3.dtype) # 预期: (2,3) int32（纯整数）

```

1. 运行代码，在 IPython 控制台查看输出，对比预期结果与实际结果；
 2. 细节要求：若 `arr1.dtype` 为 `int32`，需排查列表 `list1` 是否误写为纯整数（如 `3` 而非 `3.5`），理解“数组元素类型统一”的特性——只要有一个浮点数，整体转为浮点型。
2. 基于内置函数创建特殊数组（验证维度正确性）
1. 继续添加代码，用 `ones`、`zeros`、`arange` 创建数组：

```

# 任务 1-2：内置函数创建数组
# 3 行 2 列全 1 数组（指定 dtype=int32）
arr_ones = np.ones((3, 2), dtype=np.int32)
# 2 行 4 列全 0 数组（默认 float64）
arr_zeros = np.zeros((2, 4))
# 从 2 开始，步长 3，到 15 结束（左闭右开，故最大为 14）
arr_arange = np.arange(2, 15, 3)
# 验证数组维度与值
print("\narr_ones: \n", arr_ones)
print("arr_zeros 形状: ", arr_zeros.shape) # 预期: (2,4)
print("arr_arange 元素: ", arr_arange) # 预期: [ 2  5  8 11 14]

```

1. 运行代码，双击 Spyder “变量资源管理器” 中的数组（如 `arr_ones`），查看表格形式的元素分布；
2. 错误排查练习：故意将 `np.ones((3, 2))` 写为 `np.ones(3, 2)`（少括号），观察 IPython 控制台的 `TypeError`，记录错误信息 “shape must be an integer or tuple of integers”，并修改正确 —— 培养主动排查语法错误的习惯。

模块 2：NumPy 数组运算

操作目标

掌握元素级运算、广播运算规则，能验证运算结果正确性，排查维度不匹配错误。

操作步骤

1. 元素级运算（手动验证结果）

1. 基于模块 1 创建的 `arr1` (`[1.,2.,3.5,4.]`) 和 `arr2` (`[5.,6.,7.,8.]`)，进行加减乘除运算：

```
# 任务 2-1：元素级运算（对应元素计算）
arr_add = arr1 + arr2  # 加法
arr_mul = arr1 * arr2  # 乘法（元素级）
arr_div = arr2 / arr1  # 除法
# 手动验证前 2 个元素，确保运算正确
print("arr1+arr2 前 2 元素：", arr_add[0], arr_add[1]) # 预期：6.0 8.0
print("arr2/arr1 前 2 元素：", arr_div[0], arr_div[1]) # 预期：5.0 3.0
```

1. 运行代码，对比手动计算结果与程序输出，若不一致，检查数组创建时的元素是否正确（如 `arr1` 是否含 `3.5`）。
2. 广播运算（理解维度扩展逻辑）
 1. 创建形状不同的数组，测试广播兼容性：

```
# 任务 2-2：广播运算（形状不同但可扩展）
arr_a = np.array([[1, 2], [3, 4]]) # 2×2
```

```

arr_b = np.array([10, 20])      # 1×2 (可广播为 2×2)
arr_broadcast = arr_a + arr_b    # 广播运算
# 验证广播结果 (预期: 每行加[10,20])
print("\n 广播运算结果: \n", arr_broadcast)
# 测试不兼容广播 (故意触发错误)
try:
    arr_c = np.array([5, 6, 7]) # 1×3 (与 2×2 无法广播)
    arr_error = arr_a + arr_c
except ValueError as e:
    print("广播错误信息: ", e) # 记录错误: operands could not be broadcast together

```

1. 运行代码，观察兼容广播的结果及不兼容的错误信息，理解“广播需满足：较小数组的维度与较大数组的后缘维度一致”（如 `(1,2)` 与 `(2,2)` 的后缘维度均为 2，可广播；`(1,3)` 与 `(2,2)` 后缘维度 $3 \neq 2$ ，不可广播）。

3. 数学函数运算（验证函数效果）

1. 对 `arr_arange` (`[2,5,8,11,14]`) 进行平方根、正弦运算：

```

# 任务 2-3: 数学函数运算
arr_sqrt = np.sqrt(arr_arange) # 平方根
arr_sin = np.sin(arr_arange)   # 正弦值
print("\narr_arange 平方根: ", np.round(arr_sqrt, 2)) # 保留 2 位小数, 便于查看
print("arr_arange 正弦值: ", np.round(arr_sin, 2))   # 预期: 部分值为负 (如 sin(11)≈-0.99)

```

1. 运行代码，用 `np.round()` 简化输出，验证 `np.sqrt(4)=2`、`np.sqrt(9)=3` 等直观结果，确保函数调用正确。

模块 3：数组切片与连接

操作目标

掌握多种切片方法及数组连接规则，能精准提取目标数据，验证连接维度匹配性。

操作步骤

1. 数组切片（多方式提取数据）

1. 基于模块 1 的 `arr3`（ 2×3 数组 `[[1,2,3],[4,5,6]]`），进行基础切片、逻辑切片、`ix_()`切片：

```
# 任务 3-1: 基础切片（行、列控制）
# 提取第 2 行（索引 1）所有列
row_slice = arr3[1, :]
# 提取所有行的第 1-2 列（索引 0-1，左闭右开）
col_slice = arr3[:, 0:2]
# 任务 3-2: 逻辑切片（按条件提取）
logic_slice = arr3[arr3 > 3] # 提取所有大于 3 的元素
# 任务 3-3: ix_()函数切片（交叉提取行和列）
arr4 = np.arange(12).reshape(4, 3) #  $4 \times 3$  数组: [[0,1,2],[3,4,5],[6,7,8],[9,10,11]]
ix_slice = arr4[np.ix_([0, 3], [1, 2])] # 提取行 0/3、列 1/2 的交叉元素
# 验证所有切片结果
print("第 2 行切片: ", row_slice)      # 预期: [4 5 6]
print("第 1-2 列切片: \n", col_slice)  # 预期: [[1,2],[4,5]]
print("大于 3 的元素: ", logic_slice)  # 预期: [4 5 6]
print("ix_()切片结果: \n", ix_slice)   # 预期: [[1,2],[10,11]]
```

1. 运行代码，逐一核对输出结果，若 `ix_slice` 错误（如出现 `[[1,2],[3,5]]`），需检查 `np.ix_()` 的参数是否为列表（如 `[0,3]` 而非 `0,3`）。

2. 数组连接（验证维度匹配）

1. 用 `hstack`（水平）和 `vstack`（垂直）连接数组，测试维度匹配要求：

```
# 任务 3-4: 数组连接
arr_h1 = np.array([[1,2],[3,4]]) #  $2 \times 2$ 
arr_h2 = np.array([[5,6],[7,8]]) #  $2 \times 2$ （行数相同，可水平连接）
arr_hstack = np.hstack((arr_h1, arr_h2)) # 水平连接  $\rightarrow 2 \times 4$ 
arr_v1 = np.array([[1,2],[3,4]]) #  $2 \times 2$ 
```

```

arr_v2 = np.array([[5,6]])    # 1×2 (列数相同, 可垂直连接)
arr_vstack = np.vstack((arr_v1, arr_v2)) # 垂直连接→3×2
# 验证连接结果维度
print("\n 水平连接形状: ", arr_hstack.shape) # 预期: (2,4)
print("垂直连接形状: ", arr_vstack.shape) # 预期: (3,2)
# 测试不匹配连接 (故意错误)
try:
    arr_h3 = np.array([[9]]) # 1×1 (与 2×2 行数不同)
    np.hstack((arr_h1, arr_h3))
except ValueError as e:
    print("连接错误信息: ", e) # 记录错误: all the input array dimensions...

```

1. 运行代码, 确认连接后的数组形状, 理解“水平连接需行数相同, 垂直连接需列数相同”的规则。

模块 4: 矩阵创建与基础操作

操作目标

掌握矩阵创建方法, 区分矩阵乘法与元素乘法, 理解矩阵转置的作用。

操作步骤

1. 矩阵创建 (对比数组与矩阵的区别)

1. 用 `np.mat` 和 `np.matrix` 创建矩阵, 查看其特有属性:

```

# 任务 4-1: 创建矩阵
# 方法 1: 用字符串创建 (分号分隔行, 空格分隔列)
mat1 = np.mat("1 2 3; 4 5 6; 7 8 9")
# 方法 2: 用列表创建
mat2 = np.matrix([[10, 20], [30, 40]])
# 查看矩阵属性 (矩阵有 T/I/H 属性, 数组无)
print("mat1 转置: \n", mat1.T) # 转置矩阵

```



```
print("mat2 形状: ", mat2.shape) # 预期: (2,2)
print("mat1 类型: ", type(mat1)) # 预期: numpy.matrixlib.defmatrix.matrix
```

1. 对比“变量资源管理器”中矩阵 (**mat1**) 与数组 (**arr3**) 的类型差异，理解“矩阵是特殊的二维数组，含线性代数特有属性”。
2. 矩阵运算（区分乘法类型）
 1. 进行矩阵乘法（线性代数）与元素乘法，对比结果：

```
# 任务 4-2: 矩阵乘法 vs 元素乘法
mat_a = np.mat("1 2; 3 4") # 2×2
mat_b = np.mat("5 6; 7 8") # 2×2
# 矩阵乘法（线性代数：行×列）
mat_mul = mat_a * mat_b
# 元素乘法（对应元素相乘，需用 np.multiply）
mat_elem_mul = np.multiply(mat_a, mat_b)
# 验证两种乘法结果
print("\n 矩阵乘法结果: \n", mat_mul)    # 预期: [[19,22],[43,50]]
print("元素乘法结果: \n", mat_elem_mul) # 预期: [[5,12],[21,32]]
```

1. 运行代码，手动计算矩阵乘法（如 $1 \times 5 + 2 \times 7 = 19$ ），验证 **mat_mul[0,0]** 是否正确，避免混淆两种乘法逻辑——这是工业数据分析中常见的计算错误点。

模块 5：矩阵特征分析

操作目标

掌握逆矩阵、特征值 / 特征向量计算及线性方程组求解，能验证结果有效性，理解线性代数的实际意义。

操作步骤

1. 逆矩阵计算（验证满秩条件）
 1. 创建满秩矩阵与奇异矩阵，测试逆矩阵存在性：

```

# 任务 5-1：逆矩阵计算（仅满秩矩阵可用）
# 满秩矩阵（行列式≠0）
mat_full = np.mat("1 2; 3 4")
mat_inv = np.linalg.inv(mat_full) # 求逆
# 验证：原矩阵×逆矩阵≈单位矩阵（浮点误差允许）
mat_identity = mat_full @ mat_inv # 矩阵乘法
print("原矩阵×逆矩阵：\n", np.round(mat_identity, 6)) # 预期：接近单位矩阵
# 奇异矩阵（行列式=0，无逆矩阵）
mat_singular = np.mat("1 2; 2 4") # 第 2 行=2×第 1 行
try:
    np.linalg.inv(mat_singular)
except np.linalg.LinAlgError as e:
    print("奇异矩阵求逆错误：", e) # 记录错误：Singular matrix

```

1. 运行代码，观察 `mat_identity` 是否接近 `[[1,0],[0,1]]`（因浮点误差，可能为 `[[1., 0.], [0., 1.]]`），理解“逆矩阵是矩阵的‘逆运算’工具”。
2. 特征值与特征向量（验证定义）
 1. 计算矩阵的特征值与特征向量，验证 `mat @ vec = val * vec`：

```

# 任务 5-2：特征值与特征向量
mat_eig = np.mat("1 0 2; 0 3 0; 2 0 1") # 对称矩阵
val, vec = np.linalg.eig(mat_eig) # val: 特征值, vec: 特征向量（列向量）
# 验证第 1 个特征值和特征向量 (mat_eig @ vec[:,0] ≈ val[0] * vec[:,0])
left = mat_eig @ vec[:, 0] # 左式：矩阵×特征向量
right = val[0] * vec[:, 0] # 右式：特征值×特征向量
print("\n 特征值验证（左≈右）：", np.allclose(left, right)) # 预期：True
print("特征值：", np.round(val, 2)) # 预期：[3., 3., -1.]
print("特征向量矩阵：\n", np.round(vec, 2))

```

1. 运行代码，`np.allclose(left, right)` 返回 `True` 表示验证通过，理解“特征向量是矩阵线性变换后方向不变的向量”。

3. 线性方程组求解（验证解的正确性）

1. 求解 $Ax = b$ ，验证 $A@x \approx b$ ：

```
# 任务 5-3：求解线性方程组 Ax = b
A = np.mat("1 -1 1; 2 1 0; 2 1 -1") # 系数矩阵（满秩）
b = np.array([4, 3, -1])           # 常数项
x = np.linalg.solve(A, b)          # 求解 x
# 验证解：A@x ≈ b
b_pred = A @ x
print("\n 方程组解 x: ", np.round(x, 2)) # 预期：[1., 1., 4.]
print("验证 A@x ≈ b: ", np.allclose(b_pred, b)) # 预期：True
```

1. 运行代码，若 x 为 $[1., 1., 4.]$ ，代入方程 $1 \times 1 - 1 \times 1 + 1 \times 4 = 4$ ，验证解的正确性，理解“线性方程组求解在工业优化（如生产调度）中的应用价值”。

五、实验注意事项

1. 数据类型一致性

1. 数组创建时注意 `dtype`，如 `np.array([1,2])` 为 `int32`，`np.array([1.,2.])` 为 `float64`，混合类型会自动转为更精确的类型（如 `int+float→float`）；
2. 矩阵运算中若含整数和浮点数，结果会转为浮点数，需注意后续分析是否允许浮点误差（如工业计数数据需整数，需用 `np.round()` 转换）。

1. 维度匹配检查

1. 数组切片时，索引不可超出 `shape` 范围（如 `arr3.shape=(2,3)`，不可用 `arr3[2, :]`，索引最大为 1）；
2. 连接数组或矩阵乘法前，先打印 `shape`（如 `print(arr1.shape, arr2.shape)`），确认符合规则（如 `hstack` 需行数相同）。

1. 线性代数运算前提

1. 求逆矩阵前，先用 `np.linalg.det(mat)` 计算行列式，若行列式 ≈ 0 （如 `abs(det) < 1e-6`），则为奇异矩阵，无逆矩阵；
2. 求解线性方程组时， A 必须为方阵且满秩，否则 `np.linalg.solve()` 报错，需检查系数矩阵是否正确。

1. 结果验证习惯

1. 每步运算后，通过“手动计算部分元素”“打印关键参数”“用 `np.allclose()` 验证”三种方式确认结果，避免因代码笔误（如 `np.linalg.inv` 写成 `np.linalg.in`）导致错误。

六、实验报告要求报告结构

需包含“实验目的”“实验原理（简述核心公式，如矩阵乘法、逆矩阵定义）”“实验步骤与结果（附关键截图）”“问题与解决方法”“实验总结”5个部分。

1. 关键截图要求

1. 必附截图：数组属性（`shape/dtype`）输出、矩阵乘法与元素乘法对比结果、特征值验证（`np.allclose` 返回 `True`）、线性方程组解的验证；
2. 截图需标注清晰（如“图 1-arr3 数组属性”“图 5 - 线性方程组解验证”），包含代码片段与对应输出，证明为本人操作。

1. “问题与解决方法”要求

需记录至少 2 个实验中遇到的问题及解决过程，示例：

1. “问题 1：广播运算报错‘operands could not be broadcast together’；解决：检查数组形状，发现 `arr_c` 为 `(1,3)`，`arr_a` 为 `(2,2)`，后缘维度不匹配，修改 `arr_c` 为 `(1,2)`后正常运行”；
2. “问题 2：奇异矩阵求逆报错‘Singular matrix’；解决：用 `np.linalg.det(mat_singular)` 计算行列式，结果为 0，确认是奇异矩阵，更换为满秩矩阵 `mat_full` 后成功求逆”。

1. “实验总结”要求

1. 技术总结：提炼 NumPy 数组与矩阵的核心区别（如矩阵含 `T/I` 属性、乘法逻辑不同）、线性代数运算的实际意义（如逆矩阵用于解方程组，特征值用于设备状态诊断）；
2. 品质反思：反思实验中因细节疏忽导致的错误（如未检查数组 `shape` 导致连接失败），提出改进措施（如后续操作前先打印 `shape` 和 `dtype`）。

七、实验考核标准

考核维度	考核要点（总分 100 分）	分值
环境准备（10 分）	NumPy 版本验证正确（5 分）；Spyder 无启动异	10

	常，代码文件命名规范 (5 分)	
代码完成 (40 分)	模块 1-5 代码完整且运行正确 (每模块 8 分，语法错误扣 3 分 / 处，逻辑错误扣 5 分 / 处)	40
结果验证 (20 分)	数组运算、矩阵乘法、特征值验证等关键步骤有手动核对或 <code>np.allclose()</code> 验证 (每处 4 分)	20
报告质量 (20 分)	结构完整 (5 分) ; 截图清晰标注 (5 分) ; “问题与解决方法” 真实详细 (10 分)	20
科研品质 (10 分)	代码注释规范 (3 分) ; 错误排查记录完整 (4 分) ; 实验总结体现细节反思 (3 分)	10